# Table of Contents

**Getting started** ......................................................................................................................... 16  
  About this guide ........................................................................................................................ 16
  New features ............................................................................................................................. 18
  Key features ............................................................................................................................ 19

**Release notes** ............................................................................................................................ 23
  DSE release notes ...................................................................................................................... 23
    DSE 6.7.3 release notes .......................................................................................................... 23
    DSE 6.7.2 release notes .......................................................................................................... 31
    DSE 6.7.1 release notes .......................................................................................................... 32
    DSE 6.7.0 release notes .......................................................................................................... 38
  Bulk loader release notes ......................................................................................................... 56
  Studio release notes ................................................................................................................... 56

**Installing** ....................................................................................................................................... 57

**Configuration** .............................................................................................................................. 58
  Recommended production settings ............................................................................................ 58
  YAML and configuration properties .......................................................................................... 64
    cassandra.yaml ..................................................................................................................... 64
    dse.yaml .............................................................................................................................. 103
    remote.yaml ......................................................................................................................... 149
    cassandra-rackdc.properties ............................................................................................... 152
    cassandra-topology.properties ............................................................................................ 153
  Cloud provider snitches ............................................................................................................ 154
    Amazon EC2 single-region snitch ....................................................................................... 154
    Amazon EC2 multi-region snitch ......................................................................................... 156
    Google Cloud Platform ......................................................................................................... 157
    Apache CloudStack snitch .................................................................................................... 158
  JVM system properties .............................................................................................................. 158
### DSE 6.7 Developer Guide (Latest version)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>159</td>
</tr>
<tr>
<td>JMX</td>
<td>162</td>
</tr>
<tr>
<td>DSE Search</td>
<td>163</td>
</tr>
<tr>
<td>TPC</td>
<td>163</td>
</tr>
<tr>
<td>LDAP</td>
<td>164</td>
</tr>
<tr>
<td>Kerberos</td>
<td>165</td>
</tr>
<tr>
<td>NodeSync</td>
<td>165</td>
</tr>
<tr>
<td>Choosing a compaction strategy</td>
<td>166</td>
</tr>
<tr>
<td>NodeSync service</td>
<td>167</td>
</tr>
<tr>
<td>About NodeSync</td>
<td>167</td>
</tr>
<tr>
<td>Starting and stopping the NodeSync service</td>
<td>169</td>
</tr>
<tr>
<td>Enabling NodeSync validation</td>
<td>170</td>
</tr>
<tr>
<td>Tuning validations</td>
<td>170</td>
</tr>
<tr>
<td>Manually starting validation</td>
<td>172</td>
</tr>
<tr>
<td>Configuring Virtual Nodes</td>
<td>173</td>
</tr>
<tr>
<td>Virtual node (vnode) configuration</td>
<td>173</td>
</tr>
<tr>
<td><strong>DSE advanced functionality</strong></td>
<td>176</td>
</tr>
<tr>
<td>DSE Analytics</td>
<td>176</td>
</tr>
<tr>
<td>About DSE Analytics</td>
<td>176</td>
</tr>
<tr>
<td>Setting the replication factor for analytics keyspaces</td>
<td>177</td>
</tr>
<tr>
<td>DSE Analytics and Search integration</td>
<td>178</td>
</tr>
<tr>
<td>About DSE Analytics Solo</td>
<td>180</td>
</tr>
<tr>
<td>Analyzing data using Spark</td>
<td>182</td>
</tr>
<tr>
<td>DSEFS (DataStax Enterprise file system)</td>
<td>285</td>
</tr>
<tr>
<td>DSE Search</td>
<td>312</td>
</tr>
<tr>
<td>About DSE Search</td>
<td>312</td>
</tr>
<tr>
<td>Configuring DSE Search</td>
<td>317</td>
</tr>
<tr>
<td>DSE Search operations</td>
<td>343</td>
</tr>
</tbody>
</table>
Managing audit logs

Solr interfaces............................................................................................................. 361
HTTP API SolrJ and other Solr clients......................................................................... 372
DSE Graph.................................................................................................................. 373
About DSE Graph....................................................................................................... 373
DSE Graph Terminology............................................................................................. 376
DSE Graph QuickStart............................................................................................... 378
DSE Graph, OLTP, and OLAP...................................................................................... 416
Graph anti-patterns....................................................................................................... 432
DSE Graph data modeling............................................................................................ 434
Using DSE Graph......................................................................................................... 442
DSE Graph Analysis with DSE Analytics..................................................................... 620
DSE Graph Tools......................................................................................................... 637
Starting the Gremlin console...................................................................................... 638
DSE Graph Reference.................................................................................................. 641
NodeSync service........................................................................................................ 756
DSE Advanced Replication......................................................................................... 756
About DSE Advanced Replication............................................................................... 756
Architecture................................................................................................................ 757
Traffic between the clusters......................................................................................... 760
Terminology................................................................................................................ 762
Getting started............................................................................................................ 763
Keypaces...................................................................................................................... 774
Data types.................................................................................................................... 775
Operations.................................................................................................................... 776
CQL queries................................................................................................................. 795
Metrics......................................................................................................................... 797
Managing invalid messages....................................................................................... 805
Managing audit logs................................................................................................. 806
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dse advrep commands</td>
<td>808</td>
</tr>
<tr>
<td>Developing applications with DataStax drivers</td>
<td>853</td>
</tr>
<tr>
<td>Best practices for DataStax drivers</td>
<td>855</td>
</tr>
<tr>
<td>Connecting to DSE clusters using DSE drivers</td>
<td>858</td>
</tr>
<tr>
<td>Authentication in DSE drivers</td>
<td>858</td>
</tr>
<tr>
<td>Using SSL in DSE drivers</td>
<td>860</td>
</tr>
<tr>
<td>Load balancing with DSE drivers</td>
<td>861</td>
</tr>
<tr>
<td>Connection pooling</td>
<td>863</td>
</tr>
<tr>
<td>Retry policies</td>
<td>863</td>
</tr>
<tr>
<td>Reconnection policies</td>
<td>865</td>
</tr>
<tr>
<td>Execution profiles</td>
<td>867</td>
</tr>
<tr>
<td>Submitting queries with DataStax drivers</td>
<td>868</td>
</tr>
<tr>
<td>Working with multi-workload clusters</td>
<td>868</td>
</tr>
<tr>
<td>Using DSE Search with the DataStax drivers</td>
<td>869</td>
</tr>
<tr>
<td>Submitting DSE Graph queries with the DataStax drivers</td>
<td>871</td>
</tr>
<tr>
<td>Result paging with DataStax drivers</td>
<td>873</td>
</tr>
<tr>
<td>Synchronous and asynchronous query execution</td>
<td>874</td>
</tr>
<tr>
<td>Speculative query execution</td>
<td>875</td>
</tr>
<tr>
<td>Query idempotence</td>
<td>879</td>
</tr>
<tr>
<td>Driver metrics</td>
<td>881</td>
</tr>
<tr>
<td>Object mappers in DSE drivers</td>
<td>882</td>
</tr>
<tr>
<td>Query timestamps</td>
<td>884</td>
</tr>
<tr>
<td>Error handling</td>
<td>886</td>
</tr>
<tr>
<td>Server errors</td>
<td>886</td>
</tr>
<tr>
<td>Client errors</td>
<td>892</td>
</tr>
<tr>
<td>Tools</td>
<td>893</td>
</tr>
<tr>
<td>DSE Metrics Collector</td>
<td>893</td>
</tr>
<tr>
<td>Enabling and disabling DSE Metrics Collector</td>
<td>893</td>
</tr>
</tbody>
</table>
Configuring DSE Metrics Collector................................................................. 895
Filtering metrics................................................................................................. 897
Exporting and visualizing metrics with Prometheus and Docker.................. 899
Manually exporting and visualizing metrics with Prometheus..................... 902

nodetool.............................................................................................................. 906
About the nodetool utility.................................................................................. 906
abortrebuild........................................................................................................ 906
assassinate.......................................................................................................... 908
bootstrap............................................................................................................. 909
cfhistograms....................................................................................................... 911
cfstats................................................................................................................ 911
cleanup................................................................................................................. 911
clearsnapshot..................................................................................................... 913
compact............................................................................................................... 915
compactionhistory............................................................................................. 918
compactionstats................................................................................................. 922
decommission.................................................................................................... 924
describecluster................................................................................................... 926
describerering.................................................................................................... 928
disableautocompaction...................................................................................... 930
disablebackup.................................................................................................... 932
disablebinary...................................................................................................... 934
disablegossip..................................................................................................... 935
disablehandoff.................................................................................................. 937
disablehintsfordc............................................................................................ 938
drain..................................................................................................................... 940
enableautocompaction....................................................................................... 942
enablebackup..................................................................................................... 943
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalidatekeycache</td>
<td>1001</td>
</tr>
<tr>
<td>invalidaterowcache</td>
<td>1002</td>
</tr>
<tr>
<td>join</td>
<td>1004</td>
</tr>
<tr>
<td>listendpointpendinghints</td>
<td>1006</td>
</tr>
<tr>
<td>listsnapshots</td>
<td>1007</td>
</tr>
<tr>
<td>mark_unrepaired</td>
<td>1009</td>
</tr>
<tr>
<td>move</td>
<td>1011</td>
</tr>
<tr>
<td>netstats</td>
<td>1013</td>
</tr>
<tr>
<td>nodesyncservice</td>
<td>1015</td>
</tr>
<tr>
<td>pausehandoff</td>
<td>1030</td>
</tr>
<tr>
<td>proxyhistograms</td>
<td>1032</td>
</tr>
<tr>
<td>rangekeysample</td>
<td>1034</td>
</tr>
<tr>
<td>rebuild</td>
<td>1036</td>
</tr>
<tr>
<td>rebuild_index</td>
<td>1039</td>
</tr>
<tr>
<td>rebuild_view</td>
<td>1041</td>
</tr>
<tr>
<td>refresh</td>
<td>1043</td>
</tr>
<tr>
<td>refreshsizeestimates</td>
<td>1045</td>
</tr>
<tr>
<td>reloadseeds</td>
<td>1046</td>
</tr>
<tr>
<td>reloadtriggers</td>
<td>1048</td>
</tr>
<tr>
<td>relocatesstables</td>
<td>1050</td>
</tr>
<tr>
<td>removenode</td>
<td>1052</td>
</tr>
<tr>
<td>repair</td>
<td>1054</td>
</tr>
<tr>
<td>replaybatchlog</td>
<td>1059</td>
</tr>
<tr>
<td>resetlocalschema</td>
<td>1060</td>
</tr>
<tr>
<td>resume</td>
<td>1062</td>
</tr>
<tr>
<td>resumehandoff</td>
<td>1063</td>
</tr>
<tr>
<td>ring</td>
<td>1064</td>
</tr>
<tr>
<td>scrub</td>
<td>1067</td>
</tr>
</tbody>
</table>
sequence.................................................................................................................. 1069
setbatchlogreplaythrottle......................................................................................... 1072
setcachecapacity..................................................................................................... 1074
setcachekeystosave.................................................................................................. 1076
setcompactionthreshold.......................................................................................... 1078
setcompactionthroughput......................................................................................... 1080
setconcurrentcompactors......................................................................................... 1081
setconcurrentviewbuilders....................................................................................... 1082
sethintedhandoffthrottlekb..................................................................................... 1084
setinterdcstreamthroughput...................................................................................... 1086
setlogginglevel......................................................................................................... 1088
setmaxhintwindow.................................................................................................... 1090
setstreamthroughput................................................................................................ 1092
settimeout................................................................................................................ 1094
settraceprobability.................................................................................................... 1096
sjk............................................................................................................................... 1098
snapshot..................................................................................................................... 1100
status.......................................................................................................................... 1104
statusbackup............................................................................................................. 1107
statusbinary............................................................................................................... 1109
statusgossip............................................................................................................... 1110
statushandoff............................................................................................................. 1112
stop.............................................................................................................................. 1114
stopdaemon............................................................................................................... 1115
tablehistograms........................................................................................................ 1117
tablestats................................................................................................................... 1118
toppartitions.............................................................................................................. 1127
tpstats........................................................................................................................ 1130
DSE 6.7 Developer Guide (Latest version)

dse commands........................................................................................................ 1147
  About dse commands....................................................................................... 1147
  dse connection options.................................................................................... 1147
  add-node.......................................................................................................... 1149
  advrep.............................................................................................................. 1153
  beeline.............................................................................................................. 1198
  cassandra......................................................................................................... 1198
  cassandra-stop............................................................................................... 1201
  exec.................................................................................................................. 1202
  fs...................................................................................................................... 1204
  gremlin-console............................................................................................. 1205
  list-nodes........................................................................................................ 1207
  pyspark............................................................................................................ 1208
  remove-node................................................................................................... 1209
  spark................................................................................................................ 1211
  spark-jobserver.............................................................................................. 1214
  spark-history-server....................................................................................... 1216
  spark-sql......................................................................................................... 1218
  spark-sql-thriftserver..................................................................................... 1219
  spark-submit.................................................................................................... 1221
  SparkR.............................................................................................................. 1223
  -v.................................................................................................................... 1224
  dse client-tool............................................................................................... 1225
About dse client-tool ........................................................................................................ 1225
client-tool connection options ...................................................................................... 1225
cassandra ..................................................................................................................... 1228
configuration export .................................................................................................. 1230
configuration byos-export ......................................................................................... 1231
configuration import .................................................................................................. 1233
spark ............................................................................................................................ 1235
always-on-sql ............................................................................................................. 1238
nodesync ..................................................................................................................... 1240
disable ......................................................................................................................... 1243
enable .......................................................................................................................... 1247
tracing .......................................................................................................................... 1251
validation ..................................................................................................................... 1266
dsefs shell commands ............................................................................................... 1269
append ......................................................................................................................... 1269
cat ................................................................................................................................. 1270
cd ................................................................................................................................. 1272
chgrp ............................................................................................................................ 1274
chmod ............................................................................................................................ 1276
chown ........................................................................................................................... 1278
cp ................................................................................................................................. 1279
df ................................................................................................................................. 1281
echo ............................................................................................................................... 1283
exit ................................................................................................................................. 1284
fsck ............................................................................................................................... 1286
get ................................................................................................................................. 1287
ls ................................................................................................................................. 1289
mkdir ............................................................................................................................ 1291
mv........................................................................................................1293
put...............................................................................................................1295
pwd............................................................................................................1297
realpath.................................................................................................1299
rename.................................................................................................1300
rm.............................................................................................................1302
rmdir.......................................................................................................1303
stat...........................................................................................................1305
truncate.................................................................................................1306
umount.................................................................................................1308
dsetool......................................................................................................1309
About dsetool.........................................................................................1309
Connection options..............................................................................1311
core_indexing_status...........................................................................1313
create_core.........................................................................................1315
createsystemkey..................................................................................1320
encryptconfigvalue.............................................................................1322
get_core_config..................................................................................1322
get_core_schema................................................................................1324
help..........................................................................................................1326
index_checks........................................................................................1327
infer_sorl_schema...............................................................................1330
inmemrystatus......................................................................................1330
insights_config.....................................................................................1333
insights_filters.....................................................................................1336
list_index_files.....................................................................................1338
list_core_properties.............................................................................1340
list_subranges......................................................................................1341
listj.................................................................1343
managekmip list................................................1344
managekmip expirekey........................................1346
managekmip revoke............................................1347
managekmip destroy..........................................1349
node_health.......................................................1350
partitioner........................................................1352
perf...............................................................1353
read_resource..................................................1356
rebuild_indexes.................................................1358
reload_core......................................................1359
ring.................................................................1361
set_core_property.............................................1362
sparkmaster cleanup..........................................1364
sparkworker restart..........................................1366
status............................................................1367
stop_core_reindex.............................................1368
tieredtablestats..............................................1369
tsrload..........................................................1372
unload_core....................................................1373
upgrade_index_files.........................................1374
write_resource...............................................1376
Stress tools......................................................1378
cassandra-stress tool........................................1378
Interpreting the output of cassandra-stress..............1390
fs-stress tool...................................................1392
SSTable utilities...............................................1393
About SSTable tools..........................................1393
sstabledowngrade................................................................. 1394
sstabledump........................................................................... 1396
sstableexpiredblockers......................................................... 1404
sstablelevelreset................................................................. 1406
sstableloader....................................................................... 1408
sstablemetadata................................................................... 1411
sstableofflinerelevel............................................................. 1415
sstablepartitions.................................................................. 1417
sstablerepairedset................................................................. 1421
sstablescrub.......................................................................... 1424
sstablesplit........................................................................... 1426
sstableupgrade...................................................................... 1428
sstableutil.............................................................................. 1430
sstableverify.......................................................................... 1432
DataStax tools........................................................................... 1434
Preflight check tool............................................................... 1434
cluster_check and yaml_diff tools.......................................... 1435
Operations................................................................................. 1437
Starting and stopping DSE...................................................... 1437
Starting as a service.............................................................. 1437
Starting as a stand-alone process.......................................... 1440
Stopping a node................................................................. 1442
Clearing data from DSE......................................................... 1443
CQL...................................................................................... 1445
Getting started with DataStax Enterprise 6.7

Information about using this guide, plus new and key features in DataStax Enterprise 6.7.

About the DataStax Enterprise 6.7 Developer Guide

The Developer Guide provides information for creating enterprise-level applications that require real-time always available storage, search, and analytics. DataStax Enterprise seamlessly integrates your code, allowing applications to utilize a breadth of techniques to produce a mobile app or online applications.

Tip: Developing applications requires a basic understanding of how DataStax Enterprise works and how it differs from a relational database. In conjunction with this guide, you should refer to the Architecture Guide for background information. This will save you a lot of time when developing your data models, applications, and using the features in DataStax Enterprise. To get started, be sure to read the DataStax Enterprise 6.7 FAQ and Architecture in brief.

As a developer, you must be familiar with data modeling and CQL.

To ensure that you get the best experience in using this document, take a moment to look at the Tips for using DataStax documentation. This page provides information on search, navigational aids, and providing feedback.

DataStax supplies a number of drivers so that CQL statements and search commands can be passed from client to cluster and back.

Key topics for developers include:

- Connecting to DSE clusters using DSE drivers (page 858)
- YAML and configuration properties (page 64)
- Setting system properties during startup (page 158)
- Choosing a compaction strategy (page 166)
- Using DataStax Enterprise advanced functionality (page 176)
- DataStax Enterprise tools (page 893)
- Starting and stopping DSE
- Clearing the data from DataStax Enterprise (page 1443)

Other information sources

<table>
<thead>
<tr>
<th>Install Guide</th>
<th>Various ways for installing DataStax Enterprise.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture Guide</td>
<td>How the DataStax Enterprise database works.</td>
</tr>
<tr>
<td>Administrators Guide</td>
<td>Information about capacity planning, configuration, migration, performance monitoring, security, backup, data recovery and more.</td>
</tr>
</tbody>
</table>
## CQL for DSE 6.7

Cassandra Query Language (CQL) is a query language for the DataStax Enterprise database. You can interact with the database using the CQL shell, `cqlsh`, and About DataStax Studio 6.7.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataStax Bulk Loader (dbulk)</td>
<td>Load and unload DSE data in either CSV or JSON format.</td>
</tr>
<tr>
<td>Kafka Connector</td>
<td>Synchronizes records from a Kafka topic with rows in one or more DSE database tables.</td>
</tr>
<tr>
<td>Docker</td>
<td>Production ready Docker images for DataStax Enterprise (DSE), OpsCenter, and Studio.</td>
</tr>
<tr>
<td>Kerberos</td>
<td>Enable Kerberos authentication in DSE and OpsCenter.</td>
</tr>
<tr>
<td>DSE Graph Loader (page 484)</td>
<td>A customizable, highly tunable command line utility for loading graph datasets into DSE Graph from various input sources.</td>
</tr>
<tr>
<td>Landing pages</td>
<td>Topics on supported platforms, product compatibility, third-party software, recommended settings, and links to current and archived documentation.</td>
</tr>
<tr>
<td>DSE OpsCenter</td>
<td>OpsCenter is a visual management and monitoring solution for DSE.</td>
</tr>
<tr>
<td>Lifecycle Manager</td>
<td>Lifecycle Manager can create, configure, and manage clusters.</td>
</tr>
<tr>
<td>DSE drivers</td>
<td>C/C++ driver, C# driver, Java driver, Node.js driver, PHP driver, Python driver, and Ruby driver.</td>
</tr>
<tr>
<td>Planning and testing DSE deployments</td>
<td>Includes hardware selection, estimating disk capacity, anti-patterns, and cluster testing.</td>
</tr>
<tr>
<td>DataStax Studio</td>
<td>Studio features include:</td>
</tr>
<tr>
<td></td>
<td>- <strong>CQL (Cassandra Query Language):</strong> visually create and navigate database objects, create complex queries, and tune CQL queries. The schema helps you visualize the keyspaces in a tree-like view.</td>
</tr>
<tr>
<td></td>
<td>- <strong>DSE Graph:</strong> explore and view large datasets in DSE Graph. The code for DSE Graph is written in the Gremlin graph traversal language and is executed by the Gremlin Server that is a part of the DSE Graph component. The graph schema view helps you visualize the graph organization and connections.</td>
</tr>
<tr>
<td></td>
<td>- <strong>DSE Analytics:</strong> write, test, and run Spark SQL queries against DSE clusters.</td>
</tr>
</tbody>
</table>
DataStax Enterprise 6.7 new features

DataStax Enterprise (DSE) version 6.7 is the industry's best distributed cloud database designed for hybrid cloud. Easily deploy the only active-everywhere database platform that runs wherever needed: on-premises, across regions or clouds. Benefit from all the capabilities of the best distribution of Apache Cassandra™ with enterprise tooling and expert support required for production cloud applications.

Be sure to read the DataStax Enterprise 6.7 release notes (page 23) and the blog Announcing DataStax Enterprise 6.7 (And More!)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version numbers</td>
<td>Starting with 6.7, the version numbers for DataStax Enterprise, DSE OpsCenter, and DataStax Studio are now synchronized. This correspondence makes it easier to know which product versions belong together.</td>
</tr>
<tr>
<td>DataStax Apache Kafka Connector</td>
<td>Uses the Kafka Connect framework to automatically sync records from a Kafka topic to a DataStax Enterprise database table. This capability allows you act on events as they arrive, including fraud/anomaly detection, financial systems, IoT time series, and similar use cases. You install the connector as a plug-in on the Kafka platform. Be sure to read the Introducing the DataStax Apache Kafka™ Connector blog.</td>
</tr>
</tbody>
</table>
| DSE Analytics (page 176)       | New features in DSE Analytics include:  
  • The Apache Spark™ connectivity layer is enhanced to improve performance, deliver greater stability, and decrease resource overhead on a DSE cluster.
  • Improvements to DSEFS include:
    # Improved Kerberos support.
    # Simplified working with many files and nested folders stored in DSEFS.
  • Faster graph database analytical performance and a better high-availability Spark engine.

Be sure to read the What’s New for DataStax Enterprise Analytics 6.7https://www.datastax.com/2018/12/whats-new-for-datastax-enterprise-analytics-6-7 blog.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geospatial Search</td>
<td>Geospatial Search makes building applications for geospatial search use cases, such as logistics, IoT, and financial network management much easier. DSE 6.7 includes automatic handling of key geospatial data types, automated installation of geospatial software on every DSE node, and support for geospatial heatmaps in CQL. Be sure to read the What’s New for Search in DSE 6.7 blog.</td>
</tr>
<tr>
<td>DSE Search</td>
<td>Improved diagnostic monitoring of search performance and better speed and reliability for deployments of DSE Search that use vnodes. Be sure to read the Backup Service in DSE OpsCenter 6.7—Data Protection At Scale blog.</td>
</tr>
<tr>
<td>Data protection</td>
<td>New capabilities in OpsCenter 6.7 add backups to Microsoft Azure and generic S3 platforms. Additionally, improvements in the restore engine get a database back on its feet magnitudes faster than prior releases. Be sure to read the Backup Service in DSE OpsCenter 6.7—Data Protection At Scale blog.</td>
</tr>
<tr>
<td>Security</td>
<td>Support for PKCS11 keystores for SSL is now available when you add your own libraries and update java.security.</td>
</tr>
<tr>
<td>DSE Metrics Collector (page 893)</td>
<td>When DSE 6.7 starts, it automatically begins sending metrics and other structured events to DSE Metrics Collector. Aggregates DSE metrics and integrates with existing monitoring solutions to facilitate problem resolution and remediation. It is built on collectd, a popular, well-supported, open source metric collection agent. With over 90 plugins, you can tailor the solution to collect metrics most important to your organization. To disable, see Disabling DSE Metrics Collector (page 894). Be sure to read the Improved Performance Diagnostics With DataStax Metrics Collector blog.</td>
</tr>
<tr>
<td>Production Docker images</td>
<td>DataStax Docker images are certified for production usage. Be sure to read the DSE and Docker: From Development to Production blog.</td>
</tr>
<tr>
<td>Drivers</td>
<td>DataStax drivers are updated for DSE 6.7. Additionally, a new guide has been created for Connecting to DSE clusters using DSE drivers (page 858).</td>
</tr>
</tbody>
</table>

**DataStax Enterprise 6.7 key features**

DataStax Enterprise (DSE) version 6.7 is the industry's best distributed cloud database designed for hybrid cloud. Easily deploy the only active-everywhere database platform that runs wherever needed: on-premises, across regions or clouds. Benefit from all the capabilities
of the best distribution of Apache Cassandra™ with enterprise tooling and expert support required for production cloud applications.

For an overview of what's new in this release, see DataStax Enterprise 6.7 new features (page 18).

A note about terminology

In the DataStax Enterprise 6.7 documentation, the Cassandra database and related commercial-only features are referred to cumulatively as the DataStax Enterprise database, the database, DataStax database, or DataStax Enterprise depending on the context.

DSE database

The DSE database is a partitioned row store database. It is a massively scalable NoSQL database that provides automatic data distribution across all nodes in a cluster. There is nothing programmatic that a developer or administrator needs to do or code to distribute data across a cluster.

The database provides built-in and customizable replication, which stores redundant copies of data across the cluster. This means that if any node in a cluster goes down, one or more copies of that node’s data are available on other nodes. Replication can be configured to work across one datacenter, many datacenters, and multiple cloud availability zones.

DataStax Enterprise advanced functionality

DSE Analytics (page 176)

Built on a production-certified version of the industry standard Apache Spark™, DSE Analytics allows you to easily stream data to or from existing databases, including Apache Hadoop™, HDFS, S3, Oracle, MySQL, IBM DB2, and external Apache Spark clusters. DSE Analytics builds on Apache Spark’s ability to eliminate single points of failure, deliver faster performance over open source, and enable DSE’s real-time search, analytics, and graph capabilities.

DSE Graph (page 373)

Handles large, complex, relationship-heavy data sets through a highly-scalable graph database, capable of executing both transactional and analytical workloads in an always-on, horizontally scalable data platform.

DSE Search (page 312)

Integrated with Apache Solr™ 6.0 to provide continuously available search. Index management CQL and cqlsh commands streamline operations and development.

DSE Advanced Security

A feature suite for protecting data in enterprise environments. It includes advanced mechanisms for authentication and authorization, encryption of data in-flight and at-rest, data auditing, and row-level access control (RLAC).

NodeSync service (page 167)

Easy to use continuous repair service with low overhead that provides consistent performance and virtually eliminates manual efforts to run repair operations in a DataStax cluster.

DSE Advanced Replication (page 756)

DSE Advanced Replication lets you have a single cluster that can define a primary hub with multiple spokes, allowing configurable distributed data replication from
Getting started with DataStax Enterprise 6.7

source clusters to destination clusters bi-directionally. It’s designed to support microservice analytics commonly found in retail environments and tolerate sporadic connectivity that can occur in constrained environments, such as oil-and-gas remote sites and cruise ships.

Advanced Replication solves the complex setup and limitations when using Cassandra in these environments. Moreover, any type of workload is supported at both the edge and the hub, allowing for advanced Search and Analytic use cases at remote and central locations.

**DSE Tiered Storage**
Part of the multiple storage options offered in DataStax Enterprise for optimizing performance and cost goals. It automates the smart movement of data across different types of storage media to improve performance, lower costs, and reduce manual processes.

**DSE Multi-Instance**
Provides multi-tenancy to run multiple DataStax Enterprise nodes on a single host machine to leverage large server capabilities. This allows you to utilize the price-performance sweet spot in the contemporary hardware market and ensures that cost saving goals are met without compromising performance and availability.

**DSE In-Memory**
Part of the multiple storage options offered in DataStax Enterprise for optimizing performance and cost goals. It provides the ability to set which parts (some or all) of a database to reside fully in RAM. DSE in-memory provides lightning-fast performance for read-intensive situations.

Other DataStax Enterprise docs

**Planning and testing DataStax Enterprise deployments**
Information on choosing hardware, capacity planning, estimating disk capacity, anti-patterns, planning for the cloud, and testing your cluster before deployment.

**Troubleshooting DataStax Enterprise**
Troubleshooting for installing and starting DSE, Linux settings, security, DSE Graph, DSE Analytics, DSE Search, DataStax Studio, and more.

Development and production tools

**Integrated DataStax products**
cqlsh, Gremlin console (page 638)

**Demos**
You can install the DataStax Demos as described in the Installation Guides.

**Available tools**
- DataStax Studio
- DSE OpsCenter
- Lifecycle Manager
- nodetool (page 906)
- dsetool (page 1309)
- DSE Graph Loader (page 484)
- DataStax Bulk Loader
• Datastax Apache Kafka Connector

DataStax Drivers

DataStax drivers come in two types: DataStax drivers for DataStax Enterprise 5.0 and later and DataStax drivers for Apache Cassandra™.

The Connecting to DSE clusters using DSE drivers (page 858) documentation provides information for developing, connecting to clusters, submitting queries, handling errors, and best practices for using DataStax drivers.

Download drivers from DataStax Academy. For version compatibility, see the DataStax drivers page.

DataStax Drivers

These drivers can only be used with DataStax Enterprise 6.7 and support its advanced functionality:

• C/C++ driver
• C# driver
• Java driver
• Node.js driver
• PHP driver
• Python driver
• Ruby driver

DataStax drivers for Apache Cassandra

These drivers can be used with DataStax Enterprise 6.7 but do not support its advanced functionality:

• C/C++ driver
• C# driver
• Java driver
• Node.js driver
• PHP driver
• Python driver
• Ruby driver
DataStax Enterprise release notes

Release notes for:

DataStax Enterprise 6.7 release notes

DataStax Enterprise release notes cover cluster requirements, upgrade guidance, components, security updates, changes and enhancements, issues, and resolved issues for DataStax Enterprise (DSE) 6.7.x.

Requirement for Uniform Licensing

All nodes in each cluster must be uniformly licensed to use the same subscription. For example, if a cluster contains 5 nodes, all 5 nodes within that cluster must be either DataStax Distribution of Apache Cassandra™, or all 5 nodes must be DataStax Enterprise. Mixing different subscriptions within a cluster is not permitted. The DataStax Advanced Workloads Pack may be added to any DataStax Enterprise (not DataStax Distribution of Apache Cassandra) cluster in an incremental fashion. For example, a 10-node DSE cluster may be extended to include 3 nodes of the Advanced Workloads Pack. “Cluster” means a collection of nodes running the software which communicate with one another using gossip. See Enterprise Terms.

Before you upgrade

<table>
<thead>
<tr>
<th>Upgrade advice</th>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before you upgrade to a later major version, upgrade to the latest patch release (6.7.3) on your current version. Be sure to read the relevant upgrade documentation.</td>
<td>Upgrades to DSE 6.7 are supported from:</td>
</tr>
<tr>
<td></td>
<td>• DSE 6.0</td>
</tr>
<tr>
<td></td>
<td>• DSE 5.1</td>
</tr>
<tr>
<td></td>
<td>• DSE 5.0</td>
</tr>
<tr>
<td>Check the compatibility page for your products.</td>
<td>DSE 6.7 product compatibility:</td>
</tr>
<tr>
<td></td>
<td>• OpsCenter 6.7</td>
</tr>
<tr>
<td></td>
<td>• OpsCenter 6.5</td>
</tr>
<tr>
<td></td>
<td>• Studio 6.7</td>
</tr>
<tr>
<td>See Upgrading DataStax drivers.</td>
<td>DataStax Drivers: You may need to recompile your client application code.</td>
</tr>
<tr>
<td>Use DataStax Bulk Loader for loading and unloading data.</td>
<td>Loads data into DSE 5.0 or later and unloads data from any Apache Cassandra™ 2.1 or later data source.</td>
</tr>
</tbody>
</table>

DSE 6.7.3 release notes

23 April 2019
In this section:

- 6.7.3 Components (page 24)
- 6.7.3 Highlights (page 24)
- Cassandra enhancements for DSE 6.7.3 (page 30)
- General upgrade advice for DSE 6.7.3 (page 30)
- TinkerPop changes for DSE 6.7.3 (page 31)

Table 1: DSE functionality

<table>
<thead>
<tr>
<th>6.7.3 DSE core (page 25)</th>
<th>6.7.3 DSE Graph (page 29)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7.3 DSE Analytics (page 28)</td>
<td>6.7.3 DSE Search (page 30)</td>
</tr>
<tr>
<td>6.7.3 DSEFS (page 28)</td>
<td></td>
</tr>
</tbody>
</table>

6.7.3 Components

- Apache Solr™ 6.0.1.1.2408 (updated)
- Apache Spark™ 2.2.3.4 (updated)
- Apache TinkerPop™ 3.3.6 with additional production-certified changes (page 31)
- Apache Tomcat® 8.0.53
- DSE Java Driver 1.7.0
- Netty 4.1.25.4.dse
- Spark Jobserver 0.8.0.45 DSE custom version

DSE 6.7.3 is compatible with Apache Cassandra™ 3.11 and adds additional production-certified changes (page 30).

DSE 6.7.3 Highlights

High-value benefits of upgrading to DSE 6.7.3 include these highlights:

DSE Database (DSE core) highlights

- Compaction performance improvement with new cassandra.yaml pick_level_on_streaming (page 76) option. (DB-1658)
- The sstableloader downgrade from DSE to OSS Apache Cassandra is supported with new sstabledowngrade (page 1394) tool. (DB-2756)
- Fix NodeSync failing when validating MV row with empty partition key. (DB-2823)
- nodesync (page 1247) can be enabled and disabled on materialized views (MV). (DB-3008)
- Fixed anti-compaction transaction for atomicity and index building. (DB-3016)
- Remedy deadlock during node startup when calculating disk boundaries. (DB-3028)
- Correct handling of dropped UDT columns in SSTables. (DB-3031)
Workaround: If issues with UDTs in SSTables exist after upgrade from DSE 6.0.x or DSE 5.0.x, run `sstablescrub -e fix-only offline` on the SSTables that have or had UDTs that were created in DSE 5.0.x.

- Fix unclosed range tombstones in read response. See Cassandra enhancements for DSE 6.7.3 (page 30).
- The frame decoding off-heap queue size is configurable and smaller by default. (DB-3047)

**DSE Analytics highlights**

- AlwaysOn SQL (AOSS) log files, including `service.log`, are consolidated in `system.log`. (DSP-18261)
- Support configuration to connect to multiple hosts from BYOS connector with multiple hostnames in `DseByosAuthConfFactory`. (DSP-18231)
- Fixed a leak in `BulkTableWriter`. (DSP-18513)

**DSE Graph highlights**

- Time, date, inet, and duration data types (page 754) are now supported in graph search indexes (page 587). (DSP-17694)
- Some minor DSE GraphFrame code fixes. (DSP-18215)
- Improved usability with simplified vertex and edge loading for single label update. (DSP-18404)
- Operations through gremlin-console run with anonymous permissions. (DSP-18471)

**DSE Search highlights**

Upgrade if:

- You use timestamps as primary keys or primary key elements. (DSP-18223)
- You are hitting LUCENE-8262 and having to reload Solr cores. (DSP-18211)
- You use `queryExecutorThreads` with facet queries. (DSP-18237)

**6.7.3 DSE core**

Changes and enhancements:

- Compaction performance improvement with new `cassandra.yaml pick_level_on_streaming` option. (DB-1658)

  Streamed-in SSTables of tables using LCS (leveled compaction strategy) are placed in the same level as the source node, with possible up-leveling. Set to true to save compaction work for operations like nodetool refresh and replacing a node.

- The `sstableloader downgrade` from DSE to OSS Apache Cassandra is supported with new `sstabledowngrade` (page 1394) tool. (DB-2756)
- Unused memory in buffer pool. (DB-2788)
- The memory in use in the buffer pool is not identical to the memory allocated. (DB-2904)
• Support for using `sstableloader (page 1408)` to stream OSS Cassandra 3.x and DSE 5.x data to DSE 6.0 and later. (DB-2909)
• Improved user tools for SSTable upgrades (`sstableupgrade (page 1428)`) and downgrades (`sstabledowngrade (page 1394)`). (DB-2950)
• Memory improvements with these supported changes:
  # Configurable memory is supported for offline sstable tools (page 1393). (DB-2955)

You can use these environment variables tools:

# MAX_HEAP_SIZE - defaults to 256 MB
# MAX_DIRECT_MEMORY - defaults to ((system_memory - heap_size) / 4) with a minimum of 1 GB and a maximum of 8 GB.

To specify memory on the command line:

```
$ MAX_HEAP_SIZE=2g MAX_DIRECT_MEMORY=10g sstabledowngrade keyspace table
```

# Buffer pool, and metrics for the buffer pool, are now in two pools. In cassandra.yaml, file_cache_size_in_mb (page 76) option sets the file cache (or chunk cache) and new direct_reads_size_in_mb (page 77) option for all other short-lived read operations. (DB-2958)

To retrieve the buffer pool metrics:

```
$ nodetool sjk mxdump -q 
 "org.apache.cassandra.metrics:type=CachedReadsBufferPool,name=*"

$ nodetool sjk mxdump -q 
 "org.apache.cassandra.metrics:type=DirectReadsBufferPool,name=*"
```

For legacy compatibility, `org.apache.cassandra.metrics:type=BufferPool` still exists and is the same as `org.apache.cassandra.metrics:type=CachedReadsBufferPool`.

# cassandra-env.sh respect heap and direct memory values set in jvm.options (page 158) or as environment variables. (DB-2973)

The precedence for heap and direct memory is:

# Environment variables
# jvm.options
# calculations in cassandra-env.sh

• AIO is automatically disabled if the chunk cache size is small enough: less or equal to system RAM / 8. (DB-2997)
• `nodesync (page 1247)` cannot be enabled or disabled on materialized views (MV). (DB-3008)
- Optimized memory usage for direct reads pool when using a high number of LWTs. (DB-3124)

  When not set in cassandra.yaml, the default calculated size of `direct_reads_size_in_mb` (page 77) changed from 128 MB to 2 MB per TPC core thread, plus 2 MB shared by non-TPC threads, with a maximum value of 128 MB.

Resolved issues:

- Native server Message.Dispatcher.Flusher task stalls under heavy load. (DB-1814)
- Race in CommitLog can cause failed force-flush-all. (DB-2542)
- Unclosed range tombstones in read response. (DB-2601)
- The sstableloader downgrade from DSE to OSS Apache Cassandra is not supported. New sstabledowngrade (page 1394) tool is required. (DB-2756)
- NodeSync fails when validating MV row with empty partition key. (DB-2823)
- TupleType values with null fields NPE when being made byte-comparable. (DB-2872)
- Nodes in a cluster continue trying to connect to a decommissioned node. (DB-2886)
- Reference leak in SSTableRewriter in sstableupgrade (page 1428) when keepOriginals is true. (DB-2944)
- Hint-dispatcher file-channel not closed, if open() fails with OOM. (DB-2947)
- Hints and metadata should not use buffer pool. (DB-2958)
- Lightweight transactions contention may cause IO thread exhaustion. (DB-2965)
- DIRECT_MEMORY is being calculated using 25% of total system memory if -Xmx is set in jvm.options. (DB-2973)
- Netty direct buffers can potentially double the -XX:MaxDirectMemorySize limit. (DB-2993)
- Increased NIO direct memory because the buffers are not cleaned until GC is run. (DB-2996)
- Unable to upgrade SSTables from DSE 5.0.14 to DSE 6.0.5. (DB-3014)
- Anti-compaction transaction causes temporary data loss. (DB-3016)
- Deadlock during node startup when calculating disk boundaries. (DB-3028)
- Dropped UDT columns in SSTables deserialization are broken after upgrading from DSE 5.0. (DB-3031)
- Limit off-heap frame queues by configurable number of frames and total number of bytes. (DB-3047)
- Mishandling of frozen in complex nested types. (DB-3081)
- cqlsh EXECUTE AS command does not work. (DB-3098)
- 32-bit int overflow in StreamingTombstoneHistogramBuilder during compaction. (DB-3108)
- Too many NotInCacheExceptions (12) in trie index flow. (DB-3120)
- Counters in memtable allocators and buffer pool metrics can be incorrect when out of memory (OOM) failures occur. (DB-3126)
- Memory leak occurs when a read from disk times out. (DB-3127)
• RpcExecutionException does not print the user who is not authorized to perform a certain action. (DSP-15895)
• Leak in BulkTableWriter. (DSP-18513)
• Make the remote host visible in the error message for failed magic number verification. (DSP-18645)

6.7.3 DSE Analytics

Changes and enhancements:

• AlwaysOn SQL (AOSS) log files, including service.log, are consolidated in system.log. (DSP-18261)
• Support configuration to connect to multiple hosts from BYOS connector. (DSP-18231)
• CQL syntax error when single quote is not correctly escaped before including in save cache query to AOSS cache table. (DSP-18418)
• Improved error messaging for AlwaysOn SQL (AOSS) client tool. (DSP-18409)
• dse spark-submit --status <driver_ID> command fails. (DSP-18616)

Resolved issues:

• After client-to-node SSL is enabled, all Spark nodes must also listen on port 7480. (DSP-15744)
• dse client-tool configuration byos-export does not export required Spark properties. (DSP-15938)
• Issue with viewing information for completed jobs when authentication is enabled. (DSP-17854)
• Downloaded Spark JAR files are executable for all users. (DSP-17692)
• Cassandra Spark Connector rejects nested UDT when null. (DSP-17965)
• CassandraHiveMetastore does not unquote predicates for server side filtering. (DSP-18017)
• Spark Cassandra Connector does properly cache manually prepared RegularStatements, see SPARKC-558. (DSP-18075)
• Apache Spark local privilege escalation vulnerability: CVE-2018-11760. (DB-18225)
• Can’t access AlwaysOn SQL (AOSS) UI when authorization is enabled. (DSP-18236)
• Invalid options show for dse spark-submit command line help. (DSP-18293)
• Remove class DGFCleanerInterceptor from byos.jar. (DSP-18445)
• GBTClassifier in Spark ML fails when periodic checkpointing is on. (DSP-18450)

6.7.3 DSEFS

Resolved issues:

• Change dsefs:// default port when the DSEFS setting public_port (page 137) is changed in dse.yaml. (DSP-17962)
The shortcut `dsefs:///` now automatically resolves to `broadcastaddress:dsefs.public_port` (page 137), instead of incorrectly using `broadcastaddress:5598` regardless of the configured port.

- **weather_sensors** (page 274) demo is updated to use native DSEFS commands instead of `dse fs hadoop`. (DSP-17708)
- Fix handling of path alternatives in DSEFS shell to provide wildcard (page 294) support for `mkdir` (page 1291) and `ls` (page 1289) commands. (DSP-17768)

For example, to make several subdirectories with a single command:

```bash
$ dse fs mkdir -p /datastax/demos/weather_sensors/{byos-daily,byos-monthly,byos-station}
$ dse fs mkdir -p {path1,path2}/dir
```

- Problem with change group ownership of files using the `fileSystem.setOwner` method. (DSP-18052)

### 6.7.3 DSE Graph

Changes and enhancements:

- The default for the `spark.cassandra.output.ignoreNulls` parameter is now true for DSE Graph Frames edge updates. To override this setting, set the `spark.cassandra.output.ignoreNulls` property to false. (DSP-17377)
- Vertex and especially edge loading is simplified. `idColumn` function is no longer required. (DSP-18404)

Known issues:

- Improved error reporting for errors during cache-based vertex lookup. `AssertionError: Should not happen` errors are properly reported depending on the root cause error, for example as a timeout exception. (DSP-18254)
- Potential performance drop involving large table scans with DSE Analytics. DSE Graph OLAP operations such as `V().count...` and `V().groupCount...` may be trivially affected (< 10%). (DSP-18683)

Resolved issues:

- NPE when dropping a graph with an alias in gremlin console. (DSP-13387)
- OLAP traversal duplicates the partition key properties: OLAP `g.V().properties()` prints ‘first’ vertex `n` times with custom ids. (DSP-15688)
- Time, date, inet, and duration data types (page 754) are not supported in graph search indexes (page 587). (DSP-17694)
- AND operator is ignored in combination with OR operator in graph searches. (DSP-18061)
- Should prevent sharing Gremlin Groovy closures between scripts that are submitted through session-less connections, like DSE drivers. (DSP-18146)
Some minor DSE GraphFrame code fixes. (DSP-18215)

Reduce probability of hitting `max_concurrent_sessions` limit for OLAP workloads with BYOS. (DSP-18280)

**Tip:** For OLAP workloads with BYOS, DataStax recommends increasing the `max_concurrent_sessions` using this formula as a guideline:

\[
\text{max\_concurrent\_sessions} = \text{spark\_executors\_threads\_per\_node} \times \text{reliability\_coefficient}
\]

where `reliability\_coefficient` must be greater than 1, with a minimum `reliability\_coefficient` value between 2 and RF x 2.

Operations through gremlin-console run with system permissions, but should run with anonymous permissions. (DSP-18471)

### 6.7.3 DSE Search

Resolved issues:

- SASI should discard stale static row. (DB-2956)
- Edges are inserted with tombstone values set when inserting a recursive edge with multiple cardinality. When calling `g.updateEdges(df)`, any null entries in the provided data frame should be ignored. (DSP-17377)
- Solr HTTP request for CSV output is blank. The CSVResponseWriter returns only stored fields if a field list is not provided in the URL. (DSP-18029)

To workaround, specify a field list with the URL:

```
/select?q=*%3A*&sort=lst_updt_gdtm+desc&rows=10&fl=field1,field2&wt=csv&indent=true
```

- Avoid interrupting request threads when an internode handshake fails so that the Lucene file channel lock cannot be interrupted. (DSP-18211)
- Timestamp PK routing on solr_query fails. (DSP-18223)
- Facets and stats queries broken when using queryExecutorThreads. (DSP-18237)

### Cassandra enhancements for DSE 6.7.3

DataStax Enterprise 6.7.3 is compatible with Apache Cassandra™ 3.11 and adds these production-certified enhancements:

- Fix unclosed range tombstones in read response. Always close RT markers returned by `ReadCommand#executeLocally()` (CASSANDRA-14515)
- Severe concurrency issues in STCS,DTCS,TWCS,TMD.Topology,TypeParser (CASSANDRA-14781)

### General upgrade advice for DSE 6.7.3

DataStax Enterprise 6.7.3 is compatible with Apache Cassandra™ 3.11. All upgrade advice from previous versions applies. Carefully reviewing the DataStax Enterprise [upgrade](#)
planning and upgrade instructions can ensure a smooth upgrade and avoid pitfalls and frustrations.

**TinkerPop changes for DSE 6.7.3**

DataStax Enterprise (DSE) 6.7.3 includes all changes from previous DSE releases plus these production-certified changes that are in addition to Apache TinkerPop™ 3.3.6. See TinkerPop upgrade documentation for all changes.

- Disables the ScriptEngine global function cache which can hold on to references to "g" along with some other minor bug fixes/enhancements.

**DSE 6.7.2 release notes**

**Important:** DataStax recommends the latest patch release for most environments.

27 February 2019

6.7.2 Components

All components from DSE 6.7.2 are listed.

- Apache Solr™ 6.0.1.1.2381
- Apache Spark™ 2.2.2.8
- Apache TinkerPop™ 3.3.5 with additional production-certified changes (page 32)
- Apache Tomcat® 8.0.53
- DSE Java Driver 1.7.0
- Netty 4.1.25.4.dse
- Spark Jobserver 0.8.0.45 DSE custom version

DSE 6.7.2 is compatible with Apache Cassandra™ 3.11 and includes all production-certified changes from earlier versions.

6.7.2 Resolved issue

- DSE 5.0 SSTables with UDTs will be corrupted after migrating to DSE 5.1, DSE 6.0, and DSE 6.7. (DB-2954, CASSANDRA-15035)
If the DSE 5.0.x schema contains user-defined types (UDTs), the SSTable serialization headers are fixed when DSE is started with DSE 6.7.2 or later.

### Cassandra enhancements for DSE 6.7.2

DataStax Enterprise 6.7.2 is compatible with Apache Cassandra™ 3.11 and includes all production-certified enhancements from previous versions.

### General upgrade advice for DSE 6.7.2

DataStax Enterprise 6.7.2 is compatible with Apache Cassandra™ 3.11. All upgrade advice from previous versions applies. Carefully reviewing the DataStax Enterprise upgrade planning and upgrade instructions can ensure a smooth upgrade and avoid pitfalls and frustrations.

### TinkerPop changes for DSE 6.7.2

DataStax Enterprise (DSE) 6.7.2 includes Apache TinkerPop™ 3.3.5 and all production-certified changes from previous DSE releases. See TinkerPop upgrade documentation for all changes.

### DSE 6.7.1 release notes

11 February 2019

- 6.7.1 Components (page 32)
- 6.7.1 Highlights (page 33)
- Cassandra enhancements for DSE 6.7.1 (page 37)
- General upgrade advice for DSE 6.7.1 (page 38)
- TinkerPop changes for DSE 6.7.1 (page 38)

#### Table 2: DSE functionality

<table>
<thead>
<tr>
<th>6.7.1 DSE database (page 34)</th>
<th>6.7.1 DSE Graph (page 36)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7.1 DSE Analytics (page 35)</td>
<td>6.7.1 DSE Search (page 37)</td>
</tr>
<tr>
<td>6.7.1 DSEFS (page 36)</td>
<td></td>
</tr>
</tbody>
</table>

#### 6.7.1 Components

All components from DSE 6.7.1 are listed. Components that are updated for DSE 6.7.1 are indicated.

- Apache Solr™ 6.0.1.1.2381 (updated)
- Apache Spark™ 2.2.2.8 (updated)
- Apache TinkerPop™ 3.3.5 with additional production-certified changes (page 38)
- Apache Tomcat® 8.0.53 (updated)
• DSE Java Driver 1.7.0 (updated)
• Netty 4.1.25.4.dse (updated)
• Spark Jobserver 0.8.0.45 DSE custom version (updated)

DSE 6.7.1 is compatible with Apache Cassandra™ 3.11 and adds additional production-certified changes (page 37).

DSE 6.7.1 Highlights

High-value benefits of upgrading to DSE 6.7.1 include these highlights:

DSE Database (DSE core) highlights

Improvements:
• DSE Metrics Collector aggregates DSE metrics and integrates with existing monitoring solutions to facilitate problem resolution and remediation. (DSP-17319)

Important bug fixes:
• Fixed an issue where heap memory usage seems higher with default file cache settings. (DB-2865)
• Fixed resource leak related to streaming operations that affects tiered storage users. Excessive number of TieredRowWriter threads causing java.lang.OutOfMemoryError. (DB-2463)

DSE Analytics highlights

Upgrade if:
• You want improved error reporting during Spark job submission. (DSP-16359)
• You are having issues with search improvements for analytics when search is not enabled. (DSP-16465)
• You are moving directories in DSEFS. (DSP-17347)

DSE Graph highlights

Upgrade if:
• You want to run the DSEFS auth demo. (DSP-17700)
• You want to disable and configure DSEFS internode (node-to-node) authentication. (DSP-17721)
• You have slow gremlin script compilation times. (DSP-14132)
• You get errors for OLAP traversals after dropping schema elements. (DSP-15884)
• You want new JMX operations for graph MBeans. (DSP-15928)
• You want server side error messages for remote exceptions reported in Gremlin console. (DSP-16375)
• You occasionally get inconsistent query results. (DSP-18005)
• Use graph OLAP and want secret tokens redacted in log files. (DSP-18074)
You want to build fuzzy-text search indexes on string properties that form part of a vertex label ID. (DSP-17386)

DSE Search highlights

Upgrade if:
- You index timestamp partition keys. (DSP-17761)
- You do a lot of reindexing. (DSP-17975)
- You use frozen maps in a table with DSE Search. (DSP-18073)
- You have non-ASCII characters in your indexed columns. (DSP-17816, DSP-17961)

6.7.1 DSE core

Changes and enhancements:
- The overuse bounds for token allocation is improved when RF = 1 or RF = number of racks. (DB-1552)
- New nodetool rebuild_view command rebuilds materialized views for local data. Existing view data is not cleared. (DB-2451)
- Improved messages for nodetool nodesyncservice ratesimulator (page 1022) command include explanation for single node clusters and when no tables have NodeSync enabled. (DB-2468)
- New environment variable MAX_DIRECT_MEMORY overrides cassandra.yaml value for how much direct memory (NIO direct buffers) that the JVM can use. (DB-2919)
- New JMX operations for graph MBeans. (DSP-15928)

Resolved issues:
- Running the nodetool nodesyncservice enable command reports the error NodeSyncRecord constructor assertion failed. (DB-2280)
  Workaround: Before DSE 6.7.1, restart DSE to resolve the issue so that you can execute the command and enable NodeSync without error.
- Read and compaction errors with levelled compaction strategy (LCS). (DB-2446)
- Excessive number of TieredRowWriter threads causing java.lang.OutOfMemoryError (DB-2463)
- The nodetool nodesyncservice ratesimulator (page 1022) -deadline-overrides option is not supported. (DB-2468)
• The nodetool gcstats command output incorrectly reports the GC reclaimed metric in bytes, instead of the expected MB. (DB-2598)
• TypeParser is not thread safe. (DB-2602)
• Possible corruption in compressed files with uncompressed chunks. (DB-2634)
• Incorrect order of application of nodetool garbagecollect leaves tombstones that should be deleted. (DB-2658)
• DSE does not start with Unable to gossip with any peers error if cross_node_timeout (page 86) is true. (DB-2670)
• Heap and CPU quota exceeded detection for asynchronous JavaScript UDFs is not reliable. (DB-2645)
• Exception should occur when user with no permissions returns no rows on restricted table. (DB-2668)
• User-defined aggregates (UDAs) that instantiate user-defined types (UDTs) break after restart. (DB-2771)
• Memory leak on unfetched continuous paging requests. (DB-2851)
• Batch replay is interrupted and good batches are skipped when a mutation of an unknown table is found. (DB-2855)
• Late continuous paging errors can leave unreleased buffers behind. (DB-2862)
• Heap memory usage is higher with default file cache settings. (DB-2865)
• Prepared statement cache issues when using row-level access control (RLAC) permissions. Existing prepared statements are not correctly invalidated. (DB-2867)
• dsetool does not work when native_transport_interface (page 88) is set in cassandra.yaml. (DSP-16796)

To workaround for DSE 6.7.0: Use native_transport_interface_prefer_ipv6 (page 88) instead.

• Security: java-xmlbuilder is vulnerable to XML external entities (XXE). (DSP-13962)
• Kerberos protocol and QoP parameters are not correctly propagated. (DSP-15455)

Known issue:

• DSE 5.0 SSTables with UDTs will be corrupted after migrating to DSE 5.1, DSE 6.0, and DSE 6.7. (DB-2954, CASSANDRA-15035)

  Important: If the DSE 5.0.x schema contains user-defined types (UDTs), upgrade to at least DSE 5.1.13, DSE 6.0.6, or DSE 6.7.2. The SSTable serialization headers are fixed when DSE is started with the upgraded versions.

6.7.1 DSE Analytics

Changes and enhancements:

• Memory leak in Spark Thrift Server. (DSP-17433)
• Structured Streaming support for (Bring Your Own Spark) BYOS Spark 2.3. (DSP-17593)
• Add the ability to disable and configure DSEFS internode (node-to-node) authentication. (DSP-17721)

Resolved issues:
• Improved error handling: only submission-related error exceptions from Spark submitted applications are wrapped in a Dse Spark Submit Bootstrapper Failed to Submit error. (DSP-16359)
• Search optimizations for search analytics Spark SQL queries are applied to a datacenter that no longer has search enabled. Queries launched from a search-enabled datacenter cause search optimizations even when the target datacenter does not have search enabled. (DSP-16465)
• Submission in client mode does not support specifying remote jars (DSEFS) for main application resource (main jar) and jars specified with --jars / spark.jars. (DSP-17382)
• Incorrect conversions in DirectJoin Spark SQL operations for timestamps, UDTs, and collections. (DSP-17444)
• dse spark-sql-metastore-migrate does not work with DSE Unified Authentication and internal authentication. (DSP-17632)
• Spark Web UI redirection drops path component. (DSP-17877)
• AlwaysOn SQL (AOSS) shutdown service is not called so the driver processes are not killed. (DSP-18039)

6.7.1 DSEFS

Resolved issues:
• DSEFS does not support listen_on_broadcast_address (page 80) as configured in cassandra.yaml. (DSP-17363)
• Moving a directory under itself causes data loss and orphan data structures. (DSP-17347)
• DSEFS retries resolving corrupted paths. (DSP-17379)
• DSEFS auth demo does not work. (DSP-17700)

6.7.1 DSE Graph

Changes and enhancements:
• New tool fixes inconsistencies in graph data that are caused by schema changes, like label delete, or improper data loading. (DSP-15884)
  # DSE Graph Gremlin console: graph.cleanUp()
  # Spark: spark.dseGraph("name").cleanUp()
• Server side error messages for remote exceptions are reported in Gremlin console. (DSP-16375)

Resolved issues:
• Properties unattached to vertex show up with null values. (DSP-12300)
• Graph OLTP: Slow gremlin script compilation times. (DSP-14132)
• Graph/Search escaping fixes. (DSP-17216, DSP-17277, DSP-17816)
• Search indexes on key fields work only with non-tokenized queries. (DSP-17386)
• Graph OLTP: Potential ThreadLocal resource leak. (DSP-17808)
• DseGraphFrame fail to read properties with symbols, like period (.), in names. (DSP-17818)
• DSE GraphFrame operations cache but do not explicitly uncache. (DSP-17870)
• g.V().repeat(...).until(...).path() returns incomplete path without edges. (DSP-17933)
• gf.V().id().next() causes data to get mismatched with properties in legacy DseGraphFrame. (DSP-17979)
• Inconsistent results when using gremlin on static data. (DSP-18005)
• Graph OLAP: secret tokens are unmasked in log files. (DSP-18074)

6.7.1 DSE Search

Changes and enhancements:
• The default for auto-generated schemas is useJtsMulti="false". See Spatial queries with polygons require JTS. (DSP-17764)
• Requesting a core reindex with dsetool reload_core (page 1359) or REBUILD SEARCH INDEX no longer builds up a queue of reindexing tasks on a node. Instead, a single starting reindexing task handles all reindex requests that are already submitted to that node. (DSP-17045, DSP-13030)
• CQL timestamp field can be part of a Solr unique key. (DSP-17761)

Resolved issues:
• java.lang.AssertionError: rtDocValues.maxDoc=5230 maxDoc=4488 error is thrown in the system.log during indexing and reindexing. (DSP-17529)
• Unexpected search index errors occur when non-ASCII characters, like the U+3000 (ideographic space) character, are in indexed columns. (DSP-17816, DSP-17961)
• TextField type in search index schema should be case-sensitive if created when using copyField. (DSP-17817)
• Strong self-ref loop detected after reindex is detected. (DSP-17975)
• Loading frozen map columns fails during search read-before-write. (DSP-18073)

Cassandra enhancements for DSE 6.7.1

DataStax Enterprise 6.7.1 is compatible with Apache Cassandra™ 3.11 and adds these production-certified enhancements:
• Pad uncompressed chunks when they would be interpreted as compressed (CASSANDRA-14892)
• Correct SSTable sorting for garbagecollect and levelled compaction (CASSANDRA-14870)
• Avoid calling iter.next() in a loop when notifying indexers about range tombstones (CASSANDRA-14794)
• Fix purging semi-expired RT boundaries in reversed iterators (CASSANDRA-14672)
• DESC order reads can fail to return the last Unfiltered in the partition (CASSANDRA-14766)
• Fix corrupted collection deletions for dropped columns in 3.0 <-> 2.{1,2} messages (CASSANDRA-14568)
• Fix corrupted static collection deletions in 3.0 <-> 2.{1,2} messages (CASSANDRA-14568)
• Handle failures in parallelAllSSTableOperation (cleanup/upgradesstables/etc) (CASSANDRA-14657)
• Improve TokenMetaData cache populating performance avoid long locking (CASSANDRA-14660)
• Fix static column order for SELECT * wildcard queries (CASSANDRA-14638)
• sstableloader should use discovered broadcast address to connect intra-cluster (CASSANDRA-14522)
• Fix reading columns with non-UTF names from schema (CASSANDRA-14468)

General upgrade advice for DSE 6.7.1

DataStax Enterprise 6.7.1 is compatible with Apache Cassandra™ 3.11. All upgrade advice from previous versions applies. Carefully reviewing the DataStax Enterprise upgrade planning and upgrade instructions can ensure a smooth upgrade and avoid pitfalls and frustrations.

TinkerPop changes for DSE 6.7.1

DataStax Enterprise (DSE) 6.7.1 includes all changes from previous DSE releases plus these production-certified changes that are in addition to Apache TinkerPop™ 3.3.5. See TinkerPop upgrade documentation for all changes.

Resolved issues:
• Masked sensitive configuration options in the KryoShimServiceLoader logs.
• Fixed a concurrency issue in TraverserSet.

DSE 6.7.0 release notes

5 December 2018

• 6.7.0 New features (page 39)
• 6.7.0 Components (page 39)
• Cassandra enhancements for DSE 6.7.0 (page 44)
• General upgrade advice for DSE 6.7.0 (page 49)
• TinkerPop changes for DSE 6.7.0 (page 56)

Table 3: DSE functionality

<table>
<thead>
<tr>
<th>6.7.0 DSE database (page 39)</th>
<th>6.7.0 DSE Graph (page 43)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7.0 DSE Advanced Replication (page 42)</td>
<td>6.7.0 DSE Search (page 43)</td>
</tr>
</tbody>
</table>
## 6.7.0 Components

- Apache Solr™ 6.0.1.1.2356
- Apache Spark™ 2.2.2.5
- Apache TinkerPop™ 3.3.3 with additional production-certified changes (page 56)
- Apache Tomcat® 8.0.53
- DSE Java Driver 1.7.0
- Netty 4.1.25.4.dse
- Spark Jobserver 0.8.0.44 (DSE custom version)

DSE 6.7.0 is compatible with Apache Cassandra™ 3.11 and adds additional production-certified changes (page 44).

## 6.7.0 New features

See DataStax Enterprise 6.7 new features (page 18).

## 6.7.0 DSE database

Experimental features. These features are experimental and are not supported for production:

- SASI indexes.
- DSE OpsCenter Labs features in OpsCenter.

Changes and enhancements:

- Improved Java user-defined functions (UDF). (DB-1049)
- Improvements with new engine for materialized views (MV). (DB-1060)
  
  # Supports multiple non-base primary key in view clustering key when partition key is the same as base table.
  
  # Supports multiple filter expressions on non-primary key columns.
  
  # Supports out-of-order modification on columns not selected in MV, see CASSANDRA-11500.
  
  # Allow dropping base columns that are not part of MV primary key and not filtered.
  
  **Important:** Base column data and corresponding MV column data will be dropped.
  
  # Only MVs created with DSE 6.7 or later use the new MV format. To upgrade legacy MVs, create and build a new MV with the same schema and then point applications to use the new MV. See Known limitations of materialized views.

- Ability to read the TTL and WRITE TIME of an element in a collection. (DB-1289)
• Centralized handling of system properties to allow for logging of final values. Print all configuration flags values in system log. (DB-1556)

• Support is added for cryptographic token interface standard PKCS#11 keystores. New cassandra.yaml and dse.yaml options for server and client encryption. (DB-1629)

• CQL CAST function supports INSERT INTO and UPDATE statements, and can be used in WHERE clause. (DB-1837)

• Reduced allocations when using offheap objects. (DB-2095, DSP-17054)

• Improved protocol version presentation and setting in cqlsh. (DB-2096)

• When DSE 6.7 starts, it automatically begins sending metrics and other structured events to DSE Metrics Collector (page 893). DSE Metrics Collector is enabled by default. To disable, see Disabling DSE Metrics Collector (page 894). (DSP-15910)

• JTS (Java Topology Suite) is distributed with DSE. Remove any previously installed JTS JAR files from DSE installation classpath. (DSP-16086)

• Changes in cassandra.yaml and dse.yaml. (DB-2095, DSP-17054)

Upgrade impact: Make changes to configuration files after the upgrade and before restarting with 6.7.0. As always, carefully review and follow the Upgrading DataStax Enterprise recommendations.

After the upgrade and before restarting with DSE 6.7.0, remove deprecated settings and use new settings.

• cassandra.yaml changes

<table>
<thead>
<tr>
<th>Memtable settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deprecated cassandra.yaml settings</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>memtable_heap_space_in_mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>memtable_offheap_space_in_mb</td>
</tr>
</tbody>
</table>

Replace with this setting

| memtable_space_in_mb |

Governs heap and offheap space allocation to set a threshold for automatic memtable flush. The calculated default is 1/4 of the heap size.

| Changed setting |

| memtable_allocation_type: offheap_objects |

The default method the database uses to allocate and manage memtable memory is offheap_objects.

<table>
<thead>
<tr>
<th>User-defined functions (UDF) settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deprecated cassandra.yaml settings</td>
</tr>
</tbody>
</table>
user_defined_function_warn_timeout
user_defined_function_fail_timeout

Replace with these settings

user_defined_function_warn_micros: 500
user_defined_function_fail_micros: 10000
user_defined_function_warn_heap_mb: 200
user_defined_function_fail_heap_mb: 500
user_function_timeout_policy: die

Settings are in microseconds since Java UDFs run faster. The new timeouts are not equivalent to the deprecated settings.

Internode encryption settings

Deprecated cassandra.yaml setting

server_encryption_options:
  store_type: JKS

Replace with these settings

server_encryption_options:
  keystore_type: JKS
  truststore_type: JKS

Valid type options are JKS, JCEKS, PKCS12, or PKCS11.

Client-to-node encryption settings

Deprecated cassandra.yaml setting

client_encryption_options:
  store_type: JKS

Replace with these settings

client_encryption_options:
  keystore_type: JKS
  truststore_type: JKS

Valid type options are JKS, JCEKS, PKCS12, or PKCS11.

dse.yaml changes

Spark resource and encryption options

Deprecated dse.yaml setting
spark_ui_options:
  server_encryption_options:
    store_type: JKS

Replace with these settings

spark_ui_options:
  server_encryption_options:
    keystore_type: JKS
    truststore_type: JKS

Valid options are JKS, JCEKS, PKCS12, or PKCS11.

Known issues:

- DSE 5.0 SSTables with UDTs will be corrupted after migrating to DSE 5.1, DSE 6.0, and DSE 6.7. (DB-2954, CASSANDRA-15035)
  
  **Important:** If the DSE 5.0.x schema contains user-defined types (UDTs), **upgrade** to at least DSE 5.1.13, DSE 6.0.6, or DSE 6.7.2. The SSTable serialization headers are fixed when DSE is started with the upgraded versions.

- dsetool does not work when native_transport_interface (page 88) is set in cassandra.yaml. (DSP-16796)

  To workaround: Use native_transport_interface_prefer_ip6 (page 88) instead.

### 6.7.0 DSE Advanced Replication

No updates.

### 6.7.0 DSE Analytics

Experimental features. These features are experimental and are not supported for production:

- Importing graphs using DseGraphFrame (page 632).

Changes and enhancements:

- The default logging behavior of the Spark-SQL shell does not log to STDOUT. All information is in the spark-shell log file. (DSP-16969)

- DSEFS REST interface supports Kerberos authentication (page 301) with SPNEGO and Kerberos delegation token authentication. (DSP-13102)

- Spark Cassandra Connector: New parameter to set a read throttle per task to manage resources when multiple jobs are running in parallel. (DSP-14523)

- Improved logging messages with recommended resolutions for AlwaysOn SQL (AOSS). (DSP-17326, DSP 17358, DSP-17533)

- AlwaysOn SQL (AOSS): Set default for spark.sql.thriftServer.incrementalCollect to true. (DSP-17428)
• Provide a way for clients to determine if AlwaysOn SQL (AOSS) is enabled in DSE. (DSP-17180)

Resolved issues:
• Unresolved dependency with dse-core when running Spark Application tests with dse-connector. (DSP-17232)
• AlwaysOn SQL (AOSS) should attempt to auto start again on datacenter restart, regardless of the previous status. (DSP-17359)

6.7.0 DSEFS

Changes and enhancements:
• The dsefs `cp -r` (page 1279) shell command adds support for recursive copying. (DSP-10579)
• The dse hadoop fs command is removed. Use the dsefs commands (page 1269) instead. (DSP-16063, DSP-16594)

6.7.0 DSE Graph

Changes and enhancements:
• New DSE start-up parameter `-Dcassandra.consistent_replace` (page 159) improves LOCAL_QUORUM and QUORUM consistency on new node after node replacement. (DB-1577)

6.7.0 DSE Search

Experimental features. These features are experimental and are not supported for production:
• The dsetool `index_checks` (page 1327) use an Apache Lucene® experimental feature.

Changes and enhancements:
• Search index schema auto generation supports PolygonType. Lenient (page 341) mode is no longer required. (DSP-16480)
• The stored flag in search index schemas is deprecated and is no longer added to auto-generated schemas. If the flag exists in custom schemas, it is ignored. (DSP-14425)

  Workaround: Because the `stored=false` flag is ignored, queries return more columns than expected. To ensure queries return expected results, specify the fields to return with `fl=field1,field2` and so on.

Resolved issues:
• Search indexes automatically configure geospatial fields of Point or LineString types. (DSP-15811)

DataStax Studio
DataStax Enterprise release notes

- See DataStax Studio 6.7 release notes.

DataStax Bulk Loader

- See DataStax Bulk Loader release notes.

Cassandra enhancements for DSE 6.7.0

DataStax Enterprise 6.7.0 is compatible with Apache Cassandra™ 3.11 and adds these production-certified enhancements:

- Add DEFAULT, UNSET, MBEAN and MBEANS to `ReservedKeywords`.
  (CASSANDRA-14205)
- Add Unittest for schema migration fix (CASSANDRA-14140)
- Print correct snitch info from nodetool describecluster (CASSANDRA-13528)
- Close socket on error during connect on OutboundTcpConnection
  (CASSANDRA-9630)
- Enable CDC unittest (CASSANDRA-14141)
- Split CommitLogStressTest to avoid timeout (CASSANDRA-14143)
- Improve commit log chain marker updating (CASSANDRA-14108)
- Fix updating base table rows with TTL not removing view entries
  (CASSANDRA-14071)
- Reduce garbage created by DynamicSnitch (CASSANDRA-14091)
- More frequent commitlog chained markers (CASSANDRA-13987)
- RPM package spec: fix permissions for installed jars and config files
  (CASSANDRA-14181)
- More PEP8 compliance for cqlsh (CASSANDRA-14021)
- Fix support for SuperColumn tables (CASSANDRA-12373)
- Fix missing original update in TriggerExecutor (CASSANDRA-13894)
- Improve short read protection performance (CASSANDRA-13794)
- Fix counter application order in short read protection (CASSANDRA-12872)
- Fix MV timestamp issues (CASSANDRA-11500)
- Fix AssertionError in short read protection (CASSANDRA-13747)
- Gossip thread slows down when using batch commit log (CASSANDRA-12966)
- Allow native function calls in CQLSSTableWriter (CASSANDRA-12606)
- Copy session properties on cqlsh.py do_login (CASSANDRA-13847)
- Fix load over calculated issue in IndexSummaryRedistribution (CASSANDRA-13738)
- Obfuscate password in stress-graphs (CASSANDRA-12233)
- ReverseIndexedReader may drop rows during 2.1 to 3.0 upgrade
  (CASSANDRA-13525)
- Avoid reading static row twice from old format sstables (CASSANDRA-13236)
- Fix possible NPE on upgrade to 3.0/3.X in case of IO errors (CASSANDRA-13389)
- Add duration data type (CASSANDRA-11873)
- Properly report LWT contention (CASSANDRA-12626)
• Stress daemon help is incorrect (CASSANDRA-12563)
• Remove ALTER TYPE support (CASSANDRA-12443)
• Fix assertion for certain legacy range tombstone pattern (CASSANDRA-12203)
• Remove support for non-JavaScript UDFs (CASSANDRA-12883)
• Better handle invalid system roles table (CASSANDRA-12700)
• Upgrade netty version to fix memory leak with client encryption (CASSANDRA-13114)
• Fix trivial log format error (CASSANDRA-14015)
• Allow SSTabledump to do a JSON object per partition (CASSANDRA-13848)
• Remove unused and deprecated methods from AbstractCompactionStrategy (CASSANDRA-14081)
• Fix Distribution.average in cassandra-stress (CASSANDRA-14090)
• Presize collections (CASSANDRA-13760)
• Add GroupCommitLogService (CASSANDRA-13530)
• Parallelize initial materialized view build (CASSANDRA-12245)
• Fix flaky SecondaryIndexManagerTest.assert[Not]MarkedAsBuilt (CASSANDRA-13965)
• Make LWTs send resultset metadata on every request (CASSANDRA-13992)
• Fix flaky indexWithFailedInitializationIsNotQueryableAfterPartialRebuild (CASSANDRA-13963)
• Introduce leaf-only iterator (CASSANDRA-9988)
• Allow only one concurrent call to StatusLogger (CASSANDRA-12182)
• Refactoring to specialised functional interfaces (CASSANDRA-13982)
• Speculative retry should allow more friendly parameters (CASSANDRA-13876)
• Throw exception if we send/receive repair messages to incompatible nodes (CASSANDRA-13944)
• Replace usages of MessageDigest with Guava’s Hasher (CASSANDRA-13291)
• Add nodetool command to print hinted handoff window (CASSANDRA-13728)
• Fix some alerts raised by static analysis (CASSANDRA-13799)
• Checksum SSTable metadata (CASSANDRA-13321, CASSANDRA-13593)
• Add result set metadata to prepared statement MD5 hash calculation (CASSANDRA-10786)
• Add incremental repair support for --hosts, --force, and subrange repair (CASSANDRA-13818)
• Refactor GcCompactionTest to avoid boxing (CASSANDRA-13941)
• Expose recent histograms in JmxHistograms (CASSANDRA-13642)
• Add SERIAL and LOCAL_SERIAL support for cassandra-stress (CASSANDRA-13925)
• LCS needlessly checks for L0 STCS candidates multiple times (CASSANDRA-12961)
• Correctly close netty channels when a stream session ends (CASSANDRA-13905)
• Update lz4 to 1.4.0 (CASSANDRA-13741)
• Throttle base partitions during MV repair streaming to prevent OOM (CASSANDRA-13299)
• Improve short read protection performance (CASSANDRA-13794)
• Fix AssertionError in short read protection (CASSANDRA-13747)
• Use compaction threshold for STCS in L0 (CASSANDRA-13861)
• Fix problem with min_compress_ratio: 1 and disallow ratio < 1 (CASSANDRA-13703)
• Add extra information to SASI timeout exception (CASSANDRA-13677)
• Rework CompactionStrategyManager.getScanners synchronization (CASSANDRA-13786)
• Add additional unit tests for batch behavior, TTLs, Timestamps (CASSANDRA-13846)
• Add keyspace and table name in schema validation exception (CASSANDRA-13845)
• Emit metrics whenever we hit tombstone failures and warn thresholds (CASSANDRA-13771)
• Allow changing log levels via nodetool for related classes (CASSANDRA-12696)
• Add stress profile yaml with LWT (CASSANDRA-7960)
• Reduce memory copies and object creations when acting on ByteBuffers (CASSANDRA-13789)
• simplify mx4j configuration (Cassandra-13578)
• Fix trigger example on 4.0 (CASSANDRA-13796)
• force minumum timeout value (CASSANDRA-9375)
• Add bytes repaired/unrepaired to nodetool tablestats (CASSANDRA-13774)
• Don't delete incremental repair sessions if they still have sstables (CASSANDRA-13758)
• Fix pending repair manager index out of bounds check (CASSANDRA-13769)
• Don't use RangeFetchMapCalculator when RF=1 (CASSANDRA-13576)
• Don't optimise trivial ranges in RangeFetchMapCalculator (CASSANDRA-13664)
• Use an ExecutorService for repair commands instead of new Thread(..).start() (CASSANDRA-13594)
• Fix race / ref leak in anticompaction (CASSANDRA-13688)
• Fix race / ref leak in PendingRepairManager (CASSANDRA-13751)
• Enable ppc64le runtime as unsupported architecture (CASSANDRA-13615)
• Improve sstablemetadata output (CASSANDRA-11483)
• Support for migrating legacy users to roles has been dropped (CASSANDRA-13371)
• Introduce error metrics for repair (CASSANDRA-13387)
• Refactoring to primitive functional interfaces in AuthCache (CASSANDRA-13732)
• Update metrics to 3.1.5 (CASSANDRA-13648)
• batch_size_warn_threshold_in_kb can now be set at runtime (CASSANDRA-13699)
• Avoid always rebuilding secondary indexes at startup (CASSANDRA-13725)
• Upgrade JMH from 1.13 to 1.19 (CASSANDRA-13727)
• Upgrade SLF4J from 1.7.7 to 1.7.25 (CASSANDRA-12996)
• Default for start_native_transport now true if not set in config (CASSANDRA-13656)
• Don't add localhost to the graph when calculating where to stream from (CASSANDRA-13583)
• Allow skipping equality-restricted clustering columns in ORDER BY clause (CASSANDRA-10271)
• Use common nowInSec for validation compactions (CASSANDRA-13671)
• Improve handling of IR prepare failures (CASSANDRA-13672)
• Send IR coordinator messages synchronously (CASSANDRA-13673)
• Flush system.repair table before IR finalize promise (CASSANDRA-13660)
• Fix column filter creation for wildcard queries (CASSANDRA-13650)
• Add ‘nodetool getbatchlogreplaythrottle’ and ‘nodetool setbatchlogreplaythrottle’ (CASSANDRA-13614)
• Fix race condition in PendingRepairManager (CASSANDRA-13659)
• Allow noop incremental repair state transitions (CASSANDRA-13658)
• Run repair with down replicas (CASSANDRA-10446)
• Added started & completed repair metrics (CASSANDRA-13598)
• Added started & completed repair metrics (CASSANDRA-13598)
• Improve secondary index (re)build failure and concurrency handling (CASSANDRA-10130)
• Improve calculation of available disk space for compaction (CASSANDRA-13068)
• Change the accessibility of RowCacheSerializer for third party row cache plugins (CASSANDRA-13579)
• Allow sub-range repairs for a preview of repaired data (CASSANDRA-13570)
• NPE in IR cleanup when columnfamily has no sstables (CASSANDRA-13585)
• Fix Randomness of stress values (CASSANDRA-12744)
• Allow selecting Map values and Set elements (CASSANDRA-7396)
• Fast and garbage-free Streaming Histogram (CASSANDRA-13444)
• Update repairTime for keyspaces on completion (CASSANDRA-13539)
• Add configurable upper bound for validation executor threads (CASSANDRA-13521)
• Bring back maxHintTTL proprety (CASSANDRA-12982)
• Add testing guidelines (CASSANDRA-13497)
• Add more repair metrics (CASSANDRA-13531)
• RangeStreamer should be smarter when picking endpoints for streaming (CASSANDRA-4650)
• Avoid rewrapping an exception thrown for cache load functions (CASSANDRA-13367)
• Log time elapsed for each incremental repair phase (CASSANDRA-13498)
• Add multiple table operation support to cassandra-stress (CASSANDRA-8780)
• Fix incorrect cqlsh results when selecting same columns multiple times (CASSANDRA-13262)
• Fix WriteResponseHandlerTest is sensitive to test execution order (CASSANDRA-13421)
• Improve incremental repair logging (CASSANDRA-13468)
• Start compaction when incremental repair finishes (CASSANDRA-13454)
• Add repair streaming preview (CASSANDRA-13257)
• Cleanup isIncremental/repairedAt usage (CASSANDRA-13430)
• Change protocol to allow sending key space independent of query string (CASSANDRA-10145)
- Make gc_log and gc_warn settable at runtime (CASSANDRA-12661)
- Take number of files in L0 in account when estimating remaining compaction tasks (CASSANDRA-13354)
- Skip building views during base table streams on range movements (CASSANDRA-13065)
- Improve error messages for +/- operations on maps and tuples (CASSANDRA-13197)
- Remove deprecated repair JMX APIs (CASSANDRA-11530)
- Fix version check to enable streaming keep-alive (CASSANDRA-12929)
- Make it possible to monitor an ideal consistency level separate from actual consistency level (CASSANDRA-13289)
- Outbound TCP connections ignore internode authenticator (CASSANDRA-13324)
- Upgrade junit from 4.6 to 4.12 (CASSANDRA-13360)
- Cleanup ParentRepairSession after repairs (CASSANDRA-13359)
- Upgrade snappy-java to 1.1.2.6 (CASSANDRA-13336)
- Incremental repair not streaming correct sstables (CASSANDRA-13328)
- Upgrade the JNA version to 4.3.0 (CASSANDRA-13300)
- Add the currentTimestamp, currentDate, currentTime and currentTimeUUID functions (CASSANDRA-13132)
- Remove config option index_interval (CASSANDRA-10671)
- Reduce lock contention for collection types and serializers (CASSANDRA-13271)
- Make it possible to override MessagingService.Verb ids (CASSANDRA-13283)
- Avoid synchronized on prepareForRepair in ActiveRepairService (CASSANDRA-9292)
- Adds the ability to use uncompressed chunks in compressed files (CASSANDRA-10520)
- Don't flush sstables when streaming for incremental repair (CASSANDRA-13226)
- Remove unused method (CASSANDRA-13227)
- Fix minor bugs related to #9143 (CASSANDRA-13217)
- Output warning if user increases RF (CASSANDRA-13079)
- Remove pre-3.0 streaming compatibility code for 4.0 (CASSANDRA-13081)
- Add support for + and - operations on dates (CASSANDRA-11936)
- Fix consistency of incrementally repaired data (CASSANDRA-9143)
- Increase commitlog version (CASSANDRA-13161)
- Make TableMetadata immutable, optimize Schema (CASSANDRA-9425)
- Refactor ColumnCondition (CASSANDRA-12981)
- Parallelize streaming of different keyspaces (CASSANDRA-4663)
- Improved compactions metrics (CASSANDRA-13015)
- Speed-up start-up sequence by avoiding un-needed flushes (CASSANDRA-13031)
- Use Caffeine (W-TinyLFU) for on-heap caches (CASSANDRA-10855)
- Thrift removal (CASSANDRA-11115)
- Remove pre-3.0 compatibility code for 4.0 (CASSANDRA-12716)
- Add column definition kind to dropped columns in schema (CASSANDRA-12705)
- Add (automate) Nodetool Documentation (CASSANDRA-12672)
• Update bundled cqlsh python driver to 3.7.0 (CASSANDRA-12736)
• Reject invalid replication settings when creating or altering a keyspace (CASSANDRA-12681)
• Clean up the SSTableReader#getScanner API wrt removal of RateLimiter (CASSANDRA-12422)
• Use new token allocation for non bootstrap case as well (CASSANDRA-13080)
• Avoid byte-array copy when key cache is disabled (CASSANDRA-13084)
• Require forceful decommission if number of nodes is less than replication factor (CASSANDRA-12510)
• Allow IN restrictions on column families with collections (CASSANDRA-12654)
• Log message size in trace message in OutboundTcpConnection (CASSANDRA-13028)
• Add timeUnit Days for cassandra-stress (CASSANDRA-13029)
• Add mutation size and batch metrics (CASSANDRA-12649)
• Add method to get size of endpoints to TokenMetadata (CASSANDRA-12999)
• Expose time spent waiting in thread pool queue (CASSANDRA-8398)
• Conditionally update index built status to avoid unnecessary flushes (CASSANDRA-12969)
• cqlsh auto completion: refactor definition of compaction strategy options (CASSANDRA-12946)
• Add support for arithmetic operators (CASSANDRA-11935)
• Add histogram for delay to deliver hints (CASSANDRA-13234)
• Fix cqlsh automatic protocol downgrade regression (CASSANDRA-13307)
• Changing `max_hint_window_in_ms` at runtime (CASSANDRA-11720)
• Nodetool repair can hang forever if we lose the notification for the repair completing/failing (CASSANDRA-13480)
• Anticompaction can cause noisy log messages (CASSANDRA-13684)
• Switch to client init for sstabledump (CASSANDRA-13683)
• CQLSH: Don't pause when capturing data (CASSANDRA-13743)

**General upgrade advice for DSE 6.7.0**

DataStax Enterprise 6.7.0 is compatible with Apache Cassandra™ 3.11. All upgrade advice from previous versions applies. Carefully reviewing the DataStax Enterprise upgrade planning and upgrade instructions can ensure a smooth upgrade and avoid pitfalls and frustrations.

DataStax Enterprise 6.7.0 is compatible with Apache Cassandra™ 3.11 and adds Cassandra enhancements for DSE 6.7.0 (page 44).

Additional advice for upgrading between versions of Apache Cassandra™ includes:

**Cassandra 4.0 changes**

• Support for COMPACT STORAGE tables is removed. Follow the DataStax Enterprise upgrade documentation for upgrades from DSE 5.1 to DSE 6.7 and DSE 5.0 to DSE
6.7 for instructions on migrating all tables with COMPACT STORAGE to CQL table format.

- Fixed a problem with incremental repair which caused repaired data to be inconsistent between nodes. The fix changes the behavior of both full and incremental repairs. For full repairs, data is no longer marked repaired. For incremental repairs, anticompacktion is run at the beginning of the repair, instead of at the end. If incremental repair was being used prior to upgrading, a full repair should be run after upgrading to resolve any inconsistencies. The DataStax Enterprise upgrade documentation includes instructions to run nodetool repair.

- Deprecated config option index_interval is removed (it was deprecated since 2.0)
- Deprecated repair JMX APIs are removed.
- The version of snappy-java has been upgraded to 1.1.2.6.
- Config option commitlog_sync_batch_window_in_ms (page 73) is deprecated. Batch mode remains a valid commit log mode, however.
- A new commit log sync mode (page 73), group, is similar to batch mode but blocks for up to a configurable number of milliseconds between disk flushes.
- Due to the parallelization of the initial build of materialized views, the per token range view building status is stored in the new table `system.view_builds_in_progress`. The old table `system.views_builds_in_progress` is no longer used and can be removed. See CASSANDRA-12245 for more details.
- `nodetool clearsnapshot` (page 913) now requires the --all flag to remove all snapshots. Previous behavior would delete all snapshots by default.
- Background read repair has been deprecated. dclocal_read_repair_chance and read_repair_chance table options have been deprecated, and will be removed entirely in 4.0. See CASSANDRA-13910 for details.

Cassandra 3.11.2 changes

- Cassandra is now relying on the JVM options to properly shutdown on OutOfMemoryError. By default it will rely on the OnOutOfMemoryError option as the ExitOnOutOfMemoryError and CrashOnOutOfMemoryError options are not supported by the older 1.7 and 1.8 JVMs. A warning will be logged at startup if none of those JVM options are used. See CASSANDRA-13006 for more details.

Cassandra 3.11.2 upgrade considerations

- Creating Materialized View with filtering on non-primary-key base column (added in CASSANDRA-10368) is disabled, because the liveness of view row is depending on multiple filtered base non-key columns and base non-key column used in view primary-key. This semantic cannot be supported without storage format change, see CASSANDRA-13826. For append-only use case, you may still use this feature with a startup flag: 
  
  `-Dcassandra.mv.allow_filtering_nonkey_columns_unsafe=true`

- The NativeAccessMBean isAvailable method will only return true if the native library has been successfully linked. Previously it was returning true if JNA could be found but was not taking into account link failures.

- Primary ranges in the system.size_estimates table are now based on the keyspace replication settings and adjacent ranges are no longer merged (CASSANDRA-9639).
• In 2.1, the default for otc_coalescing_strategy was 'DISABLED'. In 2.2 and 3.0, it was changed to 'TIMEHORIZON', but that value was shown to be a performance regression. The default for 3.11.0 and newer has been reverted to 'DISABLED'. Users upgrading from Cassandra 2.2 or 3.0 should be aware that the default has changed.

• The StorageHook interface has been modified to allow to retrieve read information from SSTableReader (CASSANDRA-13120).

• Materialized Views for upgrades from DSE 5.1.1 or 5.1.2 or any version DSE 5.0.10 or later:
  
  # Cassandra will no longer allow dropping columns on tables with Materialized Views.

  # A change was made in the way the Materialized View timestamp is computed, which may cause an old deletion to a base column which is view primary key (PK) column not to be reflected in the view when repairing the base table post-upgrade. This condition is only possible when a column deletion to an MV primary key (PK) column not present in the base table PK (via UPDATE base SET view_pk_col = null or DELETE view_pk_col FROM base) is missed before the upgrade and received by repair after the upgrade. If such column deletions are done on a view PK column which is not a base PK, it's advisable to run repair on the base table of all nodes prior to the upgrade. Alternatively it's possible to fix potential inconsistencies by running repair on the views after upgrade or drop and re-create the views. See CASSANDRA-11500 for more details.

  # Removal of columns not selected in the Materialized View (via UPDATE base SET unselected_column = null or DELETE unselected_column FROM base) may not be properly reflected in the view in some situations so we advise against doing deletions on base columns not selected in views until this is fixed on CASSANDRA-13826.

Cassandra 3.10 changes

• Runtime modification of concurrent_compactors is now available via nodetool concurrent_compactors (page 1081).

• Support for the assignment operators +=/-= has been added for update queries.

• An Index implementation may now provide a task which runs prior to joining the ring. See CASSANDRA-12039

• Filtering on partition key columns is now also supported for queries without secondary indexes.

• A slow query log has been added: slow queries will be logged at DEBUG level. For more details refer to CASSANDRA-12403 and slow_query_log_timeout_in_ms in cassandra.yaml.

• Support for GROUP BY queries has been added.

• A new compaction-stress tool has been added to test the throughput of compaction for any cassandra-stress user schema. see compaction-stress help for how to use.

• Prepared statements are now persisted in the table prepared_statements in the system keyspace. Upon startup, this table is used to preload all previously prepared statements - i.e. in many cases clients do not need to re-prepare statements against restarted nodes.
• cqlsh can now connect to older Cassandra versions by downgrading the native protocol version. Please note that this is currently not part of our release testing and, as a consequence, it is not guaranteed to work in all cases. See CASSANDRA-12150 for more details.

• Snapshots that are automatically taken before a table is dropped or truncated will have a "dropped" or "truncated" prefix on their snapshot tag name.

• Metrics are exposed for successful and failed authentication attempts. These can be located using the object names org.apache.cassandra.metrics:type=Client,name=AuthSuccess and org.apache.cassandra.metrics:type=Client,name=AuthFailure respectively.

• Add support to "unset" JSON fields in prepared statements by specifying DEFAULT UNSET. See CASSANDRA-11424 for details

• Allow TTL with null value on insert and update. It will be treated as equivalent to inserting a 0.

• Removed outboundBindAny configuration property. See CASSANDRA-12673 for details.

Cassandra 3.10 upgrade considerations

• Support for alter types of already defined tables and of UDTs fields has been disabled. If it is necessary to return a different type, please use casting instead. See CASSANDRA-12443 for more details.

• Specifying the default_time_to_live option when creating or altering a materialized view was erroneously accepted (and ignored). It is now properly rejected.

• Only Java and JavaScript are now supported UDF languages. The sandbox in 3.0 already prevented the use of script languages except Java and JavaScript.

• Compaction now correctly drops sstables out of CompactionTask when there isn't enough disk space to perform the full compaction. This should reduce pending compaction tasks on systems with little remaining disk space.

• Request timeouts in cassandra.yaml (read_request_timeout_in_ms, etc) now apply to the "full" request time on the coordinator. Previously, they only covered the time from when the coordinator sent a message to a replica until the time that the replica responded. Additionally, the previous behavior was to reset the timeout when performing a read repair, making a second read to fix a short read, and when subranges were read as part of a range scan or secondary index query. In 3.10 and higher, the timeout is no longer reset for these "subqueries". The entire request must complete within the specified timeout. As a consequence, your timeouts may need to be adjusted to account for this. See CASSANDRA-12256 for more details.

• Logs written to stdout are now consistent with logs written to files. Time is now local (it was UTC on the console and local in files). Date, thread, file and line info where added to stdout. (see CASSANDRA-12004)

• The 'clientutil' jar, which has been somewhat broken on the 3.x branch, is not longer provided. The features provided by that jar are provided by any good java driver and we advise relying on drivers rather on that jar, but if you need that jar for backward compatibility until you do so, you should use the version provided on previous Cassandra branch, like the 3.0 branch (by design, the functionality provided by that jar
are stable accross versions so using the 3.0 jar for a client connecting to 3.x should work without issues).

- (Tools development) DatabaseDescriptor no longer implicitly startups components/services like commit log replay. This may break existing 3rd party tools and clients. In order to startup a standalone tool or client application, use the DatabaseDescriptor.toolInitialization() or DatabaseDescriptor.clientInitialization() methods. Tool initialization sets up partitioner, snitch, encryption context. Client initialization just applies the configuration but does not setup anything. Instead of using Config.setClientMode() or Config.isClientMode(), which are deprecated now, use one of the appropriate new methods in DatabaseDescriptor.

- Application layer keep-alives were added to the streaming protocol to prevent idle incoming connections from timing out and failing the stream session (CASSANDRA-11839). This effectively deprecates the streaming_socket_timeout_in_ms property in favor of streaming_keep_alive_period_in_secs. See cassandra.yaml for more details about this property.

- Duration literals support the ISO 8601 format. By consequence, identifiers matching that format (e.g P2Y or P1MT6H) will not be supported anymore (CASSANDRA-11873).

Cassandra 3.8 changes

- Shared pool threads are now named according to the stage they are executing tasks for. Thread names mentioned in traced queries change accordingly.

- A new option has been added to cassandra-stress "-rate fixed={number}/s" that forces a scheduled rate of operations/sec over time. Using this, stress can accurately account for coordinated omission from the stress process.

- The cassandra-stress "-rate limit=" option has been renamed to "-rate throttle="

- hdr histograms have been added to stress runs, it's output can be saved to disk using: "-log hdrfile=" option. This histogram includes response/service/wait times when used with the fixed or throttle rate options. The histogram file can be plotted on http://hdrhistogram.github.io/HdrHistogram/plotFiles.html

- TimeWindowCompactionStrategy has been added. This has proven to be a better approach to time series compaction and new tables should use this instead of DTCS. See CASSANDRA-9666 for details.

- DateTieredCompactionStrategy has been deprecated - new tables should use TimeWindowCompactionStrategy. Note that migrating an existing DTCS-table to TWCS might cause increased compaction load for a while after the migration so make sure you run tests before migrating. Read CASSANDRA-9666 for background on this.

- Change-Data-Capture is now available. See cassandra.yaml and for cdc-specific flags and a brief explanation of on-disk locations for archived data in CommitLog form. This can be enabled via ALTER TABLE ... WITH cdc=true. Upon flush, CommitLogSegments containing data for CDC-enabled tables are moved to the data/cdc_raw directory until removed by the user and writes to CDC-enabled tables will be rejected with a WriteTimeoutException once cdc_total_space_in_mb is reached between unflushed CommitLogSegments and cdc_raw. NOTE: CDC is disabled by default in the .yaml file. Do not enable CDC on a mixed-version cluster as it will lead to
exceptions which can interrupt traffic. Once all nodes have been upgraded to 3.8 it is safe to enable this feature and restart the cluster.

Cassandra 3.10 upgrade considerations

- The ReversedType behaviour has been corrected for clustering columns of BYTES type containing empty value. Scrub should be run on the existing SSTables containing a descending clustering column of BYTES type to correct their ordering. See CASSANDRA-12127 for more details.
- Ec2MultiRegionSnitch will no longer automatically set broadcast_rpc_address to the public instance IP if this property is defined on cassandra.yaml.
- The name "json" and "distinct" are not valid anymore a user-defined function names (they are still valid as column name however). In the unlikely case where you had defined functions with such names, you will need to recreate those under a different name, change your code to use the new names and drop the old versions, and this _before_ upgrade (see CASSANDRA-10783 for more details).

Cassandra 3.7 upgrade considerations

- A maximum size for SSTables values has been introduced, to prevent out of memory exceptions when reading corrupt SSTables. This maximum size can be set via max_value_size_in_mb (page 78) in cassandra.yaml. The default is 256MB, which matches the default value of native_transport_max_frame_size_in_mb. SSTables will be considered corrupt if they contain values whose size exceeds this limit. See CASSANDRA-9530 for more details.

Cassandra 3.6 changes

- JMX connections can now use the same auth mechanisms as CQL clients. New options in cassandra-env.(sh|ps1) enable JMX authentication and authorization to be delegated to the IAuthenticator and IAuthorizer configured in cassandra.yaml. The default settings still only expose JMX locally, and use the JVM's own security mechanisms when remote connections are permitted. For more details on how to enable the new options, see the comments in cassandra-env.sh. A new class of IResource, JMXResource, is provided for the purposes of GRANT/REVOKE via CQL. See CASSANDRA-10091 for more details. Also, directly setting JMX remote port via the com.sun.management.jmxremote.port system property at startup is deprecated. See CASSANDRA-11725 for more details.
- JSON timestamps are now in UTC and contain the timezone information, see CASSANDRA-11137 for more details.
- Collision checks are performed when joining the token ring, regardless of whether the node should bootstrap. Additionally, replace_address can legitimately be used without bootstrapping to help with recovery of nodes with partially failed disks. See CASSANDRA-10134 for more details.
- Key cache will only hold indexed entries up to the size configured by column_index_cache_size_in_kb in cassandra.yaml in memory. Larger indexed entries will never go into memory. See CASSANDRA-11206 for more details.
- For tables having a default_time_to_live specifying a TTL of 0 will remove the TTL from the inserted or updated values.
• Startup is now aborted if corrupted transaction log files are found. The details of the affected log files are now logged, allowing the operator to decide how to resolve the situation.
• Filtering expressions are made more pluggable and can be added programatically via a QueryHandler implementation. See CASSANDRA-11295 for more details.

Cassandra 3.4 changes
• Internal authentication now supports caching of encrypted credentials. Reference cassandra.yaml:credentials_validity_in_ms
• Remote configuration of auth caches via JMX can be disabled using the the system property cassandra.disable_auth_caches_remote_configuration
• sstabledump tool is added to be 3.0 version of former sstable2json. The tool only supports v3.0+ SSTables. See tool's help for more detail.
• The mbean interfaces org.apache.cassandra.auth.PermissionsCacheMBean and org.apache.cassandra.auth.RolesCacheMBean are deprecated in favor of org.apache.cassandra.auth.AuthCacheMBean. This generalized interface is common across all caches in the auth subsystem. The specific mbean interfaces for each individual cache will be removed in a subsequent major version.

Cassandra 3.2 changes
• We now make sure that a token does not exist in several data directories. This means that we run one compaction strategy per data_file_directory and we use one thread per directory to flush. Use nodetool relocatesstables to make sure your tokens are in the correct place, or just wait and compaction will handle it. See CASSANDRA-6696 for more details.
• bound maximum in-flight commit log replay mutation bytes to 64 megabytes tunable via cassandra.commitlog_max_outstanding_replay_bytes
• Support for type casting has been added to the selection clause.
• Hinted handoff now supports compression. Reference cassandra.yaml:hints_compression. Note: hints compression is currently disabled by default.
• The Thrift API is deprecated and will be removed in Cassandra 4.0.

Cassandra 3.2 upgrade considerations
• The compression ratio metrics computation has been modified to be more accurate.
• Running Cassandra as root is prevented by default.
• JVM options are moved from cassandra-env.(sh|ps1) to jvm.options file

Cassandra 3.1 upgrade considerations
• The return value of SelectStatement::getLimit as been changed from DataLimits to int.
• Custom index implementation should be aware that the method Indexer::indexes() has been removed as its contract was misleading and all custom implementation should have almost surely returned true incondionally for that method.
• GC logging is now enabled by default (you can disable it in the jvm.options file if you prefer).

**TinkerPop changes for DSE 6.7.0**

DataStax Enterprise (DSE) 6.7.0 includes all changes from previous DSE releases plus these production-certified changes that are in addition to Apache TinkerPop™ 3.3.3:

• Upgrade to Groovy 2.4.15 - resolves a Groovy bug preventing Lambda creation in GLVs in some cases. (TINKERPOP-1953)
• Implement TraversalSelectStep - expands the capability of the select() step by allowing nesting as in select("a").select(select("n")) which thus allows for dynamic keys for select() (6.0.1+). (TINKERPOP-1628)
• Performance enhancement to Bytecode deserialization. (TINKERPOP-1936)
• Traversal construction performance enhancements. (TINKERPOP-1950)
• Path history isn't preserved for keys in mutations. (TINKERPOP-1947)
• Profile step and iterate do not play nicely with each other (6.0.1+). (TINKERPOP-1869)

**DataStax Bulk Loader release notes**

Release notes for DataStax Bulk Loader.

DataStax Bulk Loader can migrate data in CSV or JSON format into DataStax Enterprise (DSE) from another DSE or Apache Cassandra™ cluster.

• Can unload data from any Cassandra 2.1 or later
• Can load data into DSE 5.0 or later

**DataStax Studio release notes**

Release notes for DataStax Studio.

See the DataStax Studio 6.7 release notes in the DataStax Studio guide.
Installing DataStax Enterprise 6.7

Installation information is located in the Installation Guide.
Configuration

Recommended production settings

The following sections provide recommendations for optimizing your DataStax Enterprise installation on Linux:

Use the latest Java Virtual Machine

Configure your operating system to use the latest build of a Technology Compatibility Kit (TCK) Certified OpenJDK version 8. For example, OpenJDK 8 (1.8.0_151 minimum).

Tip: Although Oracle JRE/JDK 8 is supported, DataStax does more extensive testing on OpenJDK 8. This change is due to the end of public updates for Oracle JRE/JDK 8. Java 9 is not supported.

Synchronize clocks

Synchronize the clocks on all nodes and application servers. Use NTP (Network Time Protocol) or other methods.

This is required because DataStax Enterprise (DSE) overwrites a column only if there is another version whose timestamp is more recent, which can happen when machines in are different locations.

DSE timestamps are encoded as microseconds since UNIX epoch without timezone information. The timestamp for all writes in DSE is UTC (Universal Time Coordinated). DataStax recommends converting to local time only when generating output to be read by humans.

TCP settings

During low traffic intervals, a firewall configured with an idle connection timeout can close connections to local nodes and nodes in other data centers. To prevent connections between nodes from timing out, set the following network kernel settings:

```bash
$ sudo sysctl -w net.ipv4.tcp_keepalive_time=60 
net.ipv4.tcp_keepalive_probes=3 
net.ipv4.tcp_keepalive_intvl=10
```

These values set the TCP keepalive timeout to 60 seconds with 3 probes, 10 seconds gap between each. The settings detect dead TCP connections after 90 seconds (60 + 10 + 10 + 10). There is no need to be concerned about the additional traffic as it’s negligible; permanently leaving these settings isn’t an issue. See Securing DataStax Enterprise ports.

To the handle thousands of concurrent connections used by the database, change these Linux kernel settings:
Configuration

```
$ sudo sysctl -w \
net.core.rmem_max=16777216 \nnet.core.wmem_max=16777216 \nnet.core.rmem_default=16777216 \nnet.core.wmem_default=16777216 \nnet.core.optmem_max=40960 \nnet.ipv4.tcp_rmem=4096 87380 16777216 \nnet.ipv4.tcp_wmem=4096 65536 16777216
```

To persist the TCP settings:

1. **Add the following to the** /etc/sysctl.conf **file:**

```
net.ipv4.tcp_keepalive_time=60
net.ipv4.tcp_keepalive_probes=3
net.ipv4.tcp_keepalive_intvl=10
net.core.rmem_max=16777216
net.core.wmem_max=16777216
net.core.rmem_default=16777216
net.core.wmem_default=16777216
net.core.optmem_max=40960
net.ipv4.tcp_rmem=4096 87380 16777216
net.ipv4.tcp_wmem=4096 65536 16777216
```

2. **Load the settings using one of the following commands:**

```
$ sudo sysctl -p /etc/sysctl.conf
$ sudo sysctl -p /etc/sysctl.d/filename.conf
```

Disable CPU frequency scaling

Recent Linux systems include a feature called *CPU frequency scaling* or *CPU speed scaling*. It allows a server's clock speed to be dynamically adjusted so that the server can run at lower clock speeds when the demand or load is low. This reduces the server's power consumption and heat output (which significantly impacts cooling costs). Unfortunately, this behavior has a detrimental effect on servers running DataStax Enterprise because throughput can get capped at a lower rate.

On most Linux systems, a CPUfreq governor manages the scaling of frequencies based on defined rules and the default ondemand governor switches the clock frequency to maximum when the demand is high and switches to the lowest frequency when the system is idle.

Do not use governors that lower the CPU frequency. To ensure optimal performance, reconfigure all CPUs to use the performance governor, which locks the frequency at maximum. This governor will not switch frequencies, which means there will be no power savings but the servers will always run at maximum throughput. On most systems, set the governor as follows:

```
for CPUFREQ in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```
do
  [ -f $CPUFREQ ] || continue
  echo -n performance > $CPUFREQ
done

For more information, see High server load and latency when CPU frequency scaling is enabled in the DataStax Help Center.

Make sure that new settings persist after reboot

Caution: Depending on your environment, some of the following settings may not be persisted after reboot. Check with your system administrator to ensure they are viable for your environment.

Optimize SSDs

The default SSD configurations on most Linux distributions are not optimal. Follow these steps to ensure the best settings for SSDs:

1. Ensure that the SysFS rotational flag is set to false (zero).
   
   This overrides any detection by the operating system to ensure the drive is considered an SSD.

2. Apply the same rotational flag setting for any block devices created from SSD storage, such as mdarrays.

3. Set the IO scheduler to either deadline or noop:
   
   • The noop scheduler is the right choice when the target block device is an array of SSDs behind a high-end IO controller that performs IO optimization.
   
   • The deadline scheduler optimizes requests to minimize IO latency. If in doubt, use the deadline scheduler.

4. Set the readahead value for the block device to 8 KB.
   
   This setting tells the operating system not to read extra bytes, which can increase IO time and pollute the cache with bytes that weren’t requested by the user.

For example, if the SSD is /dev/sda, in /etc/rc.local:

```bash
echo deadline > /sys/block/sda/queue/scheduler
#OR...
#echo noop > /sys/block/sda/queue/scheduler
touch /var/lock/subsys/local
echo 0 > /sys/class/block/sda/queue/rotational
echo 8 > /sys/class/block/sda/queue/read_ahead_kb
```
Use the optimum --setra setting for RAID on SSD

The optimum readahead setting for RAID on SSDs (in Amazon EC2) is 8 KB, the same as it is for non-RAID SSDs. For details, see Optimizing SSDs (page 60).

Disable zone_reclaim_mode on NUMA systems

The Linux kernel can be inconsistent in enabling/disabling zone_reclaim_mode. This can result in odd performance problems.

To ensure that zone_reclaim_mode is disabled:

```
$ echo 0 > /proc/sys/vm/zone_reclaim_mode
```

For more information, see Peculiar Linux kernel performance problem on NUMA systems.

Set user resource limits

Use the ulimit -a command to view the current limits. Although limits can also be temporarily set using this command, DataStax recommends making the changes permanent:

Package installations:

Ensure that the following settings are included in the /etc/security/limits.d/cassandra.conf file:

```
<cassandra_user> - memlock unlimited
<cassandra_user> - nofile 1048576
<cassandra_user> - nproc 32768
<cassandra_user> - as unlimited
```

Tarball installations:

In RHEL version 6.x, ensure that the following settings are included in the /etc/security/limits.conf file:

```
<cassandra_user> - memlock unlimited
<cassandra_user> - nofile 1048576
<cassandra_user> - nproc 32768
<cassandra_user> - as unlimited
```

If you run DataStax Enterprise as root, some Linux distributions, such as Ubuntu, require setting the limits for root explicitly instead of using cassandra_user:

```
root - memlock unlimited
root - nofile 1048576
root - nproc 32768
root - as unlimited
```

For RHEL 6.x-based systems, also set the nproc limits in /etc/security/limits.d/90-nproc.conf:
**Configuration**

```
cassandra_user - nproc 32768
```

For all installations, add the following line to `/etc/sysctl.conf`:

```
vm.max_map_count = 1048575
```

For installations on Debian and Ubuntu operating systems, the `pam_limits.so` module is not enabled by default. Edit the `/etc/pam.d/su` file and uncomment this line:

```
session  required  pam_limits.so
```

This change to the PAM configuration file ensures that the system reads the files in the `/etc/security/limits.d` directory.

To make the changes take effect, reboot the server or run the following command:

```
$ sudo sysctl -p
```

For more information, see [Insufficient user resource limits errors](#).

**Disable swap**

DataStax strongly recommends disabling swap entirely (`sudo swapoff --all`). Because the database has multiple replicas and transparent failover, it is preferable for a replica to be killed immediately when memory is low rather than go into swap. This allows traffic to be immediately redirected to a functioning replica instead of continuing to hit the replica that has high latency due to swapping. If your system has a lot of DRAM, swapping still lowers performance significantly because the OS swaps out executable code so that more DRAM is available for caching disks.

```
$ sudo swapoff --all
```

To make this change permanent, remove all swap file entries from `/etc/fstab`.

**Note:** If you insist on using swap, you can set `vm.swappiness=1`. This allows the kernel swap out the absolute least used parts.

To make this change permanent, remove all swap file entries from `/etc/fstab`.

For more information, see [Nodes seem to freeze after some period of time](#).

**Check the Java Hugepages setting**

Many modern Linux distributions ship with Transparent Hugepages enabled by default. When Linux uses Transparent Hugepages, the kernel tries to allocate memory in large chunks (usually 2MB), rather than 4K. This can improve performance by reducing the number of pages the CPU must track. However, some applications still allocate memory based on 4K pages. This can cause noticeable performance problems when Linux tries to defrag 2MB pages. For more information, see the [Cassandra Java Huge Pages](#) blog and this [RedHat bug report](#).

To solve this problem, disable `defrag` for hugepages. Enter:
echo never | sudo tee /sys/kernel/mm/transparent_hugepage/defrag

For more information, including a temporary fix, see No DataStax Enterprise processing but high CPU usage.

Set the heap size for optional Java garbage collection in DataStax Enterprise

The default JVM garbage collection (GC) is G1.

**Note:** DataStax does not recommend using G1 when using Java 7. This is due to a problem with class unloading in G1. In Java 7, PermGen fills up indefinitely until a full GC is performed.

Heap size is usually between \( \frac{1}{4} \) and \( \frac{1}{2} \) of system memory. Do not devote all memory to heap because it is also used for offheap cache and file system cache.

The easiest way to determine the optimum heap size for your environment is:

1. Set the MAX_HEAP_SIZE in the cassandra-env.sh file to a high arbitrary value on a single node.
2. View the heap used by that node:
   - Enable GC logging and check the logs to see trends.
   - Use List View in OpsCenter.
3. Use the value for setting the heap size in the cluster.

   **Note:** This method decreases performance for the test node, but generally does not significantly reduce cluster performance.

If you don't see improved performance, contact the DataStax Services team for additional help in tuning the JVM.

Determining the heap size when using Concurrent-Mark-Sweep (CMS) garbage collection in DataStax Enterprise

There are many nuances for tuning CMS. It requires time, expertise, and repeated testing to get the best results. DataStax recommends contacting the DataStax Services team instead. Tuning Java Virtual Machine provides the basic information to get you started.

Set the heap size for optimal Java garbage collection

See Tuning Java resources.

Apply optimum blockdev --setra settings for RAID on spinning disks

Typically, a readahead of 128 is recommended.

Check to ensure setra is not set to 65536:
$ sudo blockdev --report /dev/spinning_disk

To set setra:

$ sudo blockdev --setra 128 /dev/spinning_disk

**Note:** The recommended setting for RAID on SSDs is the same as that for SSDs that are not being used in a RAID installation. For details, see Optimizing SSDs (page 60).

### YAML and configuration properties

#### cassandra.yaml configuration file

The cassandra.yaml file is the main configuration file for DataStax Enterprise. The dse.yaml (page 103) file is the primary configuration file for security, DSE Search, DSE Graph, and DSE Analytics.

**Important:** After changing properties in the cassandra.yaml file, you must restart the node for the changes to take effect.

#### Syntax

For the properties in each section, the parent setting has zero spaces. Each child entry requires at least two spaces. Adhere to the YAML syntax and retain the spacing.

- Literal default values are shown as *literal*.
- Calculated values are shown as *calculated*.
- Default values that are not defined are shown as Default: none.
- Internally defined default values are described.

**Note:** Default values can be defined internally, commented out, or have implementation dependencies on other properties in the cassandra.yaml file. Additionally, some commented-out values may not match the actual default values. The commented out values are recommended alternatives to the default values.

#### Organization

The configuration properties are grouped into the following sections:

- **Quick start (page 65)**
  
  The minimal properties needed for configuring a cluster.

- **Default directories (page 66)**
If you have changed any of the default directories during installation, set these properties to the new locations. Make sure you have root access.

- **Commonly used properties (page 67)**
  Properties most frequently used when configuring DataStax Enterprise.

- **Performance tuning (page 72)**
  Tuning performance and system resource utilization, including commit log, compaction, memory, disk I/O, CPU, reads, and writes.

- **Advanced properties (page 79)**
  Properties for advanced users or properties that are less commonly used.

- **Security properties (page 92)**
  DSE Unified Authentication provides authentication, authorization, and role management.

- **User-defined functions (UDF) properties (page 100)**
- **Continuous paging options (page 101)** Properties configure memory, threads, and duration when pushing pages continuously to the client.

### Quick start properties

The minimal properties needed for configuring a cluster.

```plaintext
cluster_name: 'Test Cluster'
listen_address: localhost
# listen_interface: wlan0
# listen_interface_prefer_ipv6: false
```

**Tip:** See Initializing a DataStax Enterprise cluster.

- **cluster_name**
  The name of the cluster. This setting prevents nodes in one logical cluster from joining another. All nodes in a cluster must have the same value.
  Default: 'Test Cluster'

- **listen_address**
  The IP address or hostname that the database binds to for connecting this node to other nodes.

  **Warning:**
  - Never set listen_address to 0.0.0.0.
  - Set listen_address or listen_interface, do not set both.

  Default: localhost

- **listen_interface**
The interface that the database binds to for connecting to other nodes. Interfaces must correspond to a single address. IP aliasing is not supported.

**Warning:** Set `listen_address` or `listen_interface`, not both.

Default: commented out (wlan0)

`listen_interface_prefer_ipv6`

Use IPv4 or IPv6 when interface is specified by name.

- false - use first IPv4 address.
- true - use first IPv6 address.

When only a single address is used, that address is selected without regard to this setting.

Default: commented out (false)

### Default directories

```yaml
- data_file_directories:
  - /var/lib/cassandra/data
- commitlog_directory: /var/lib/cassandra/commitlog
- cdc_raw_directory: /var/lib/cassandra/cdc_raw
- hints_directory: /var/lib/cassandra/hints
- saved_caches_directory: /var/lib/cassandra/saved_caches
```

If you have changed any of the default directories during installation, set these properties to the new locations. Make sure you have root access.

**data_file_directories**

The directory where table data is stored on disk. The database distributes data evenly across the location, subject to the granularity of the configured compaction strategy. If not set, the directory is `$DSE_HOME/data/data`.

**Tip:** For production, DataStax recommends RAID 0 and SSDs.

Default: `/var/lib/cassandra/data`

**commitlog_directory**

The directory where the commit log is stored. If not set, the directory is `$DSE_HOME/data/commitlog`.

For optimal write performance, place the commit log on a separate disk partition, or ideally on a separate physical device from the data file directories. Because the commit log is append only, a hard disk drive (HDD) is acceptable.

Default: `/var/lib/cassandra/commitlog`

**cdc_raw_directory**

The directory where the change data capture (CDC) commit log segments are stored on flush. DataStax recommends a physical device that is separate from the data directories. If not set, the directory is `$DSE_HOME/data/cdc_raw`.

Default: `/var/lib/cassandra/cdc_raw`

**hints_directory**

The directory in which hints are stored. If not set, the directory is `$CASSANDRA_HOME/data/hints`.

Default: `/var/lib/cassandra/hints`
**saved_caches_directory**
The directory location where table key and row caches are stored. If not set, the directory is $DSE_HOME/data/saved_caches.
Default: /var/lib/cassandra/saved_caches

Commonly used properties

Properties most frequently used when configuring DataStax Enterprise.

Before starting a node for the first time, DataStax recommends that you carefully evaluate your requirements.

- **Common initialization properties** ([page 67](#))
- **Common compaction settings** ([page 70](#))
- **Memtable settings** ([page 70](#))
- **Common automatic backup settings** ([page 72](#))

**Common initialization properties**

```
commit_failure_policy: stop
prepared_statements_cache_size_mb:
# disk_optimization_strategy: ssd
disk_failure_policy: stop
endpoint_snitch: com.datastax.bdp.snitch.DseSimpleSnitch
seed_provider:
  - org.apache.cassandra.locator.SimpleSeedProvider
  - seeds: "127.0.0.1"
enable_user_defined_functions: false
enable_scripted_user_defined_functions: false
enable_user_defined_functions_threads: true
user_defined_function_warn_micros: 500
user_defined_function_fail_micros: 10000
user_defined_function_warn_heap_mb: 200
user_defined_function_fail_heap_mb: 500
user_function_timeout_policy: die
```

**Note:** Be sure to set the properties in the Quick start section ([page 65](#)) as well.

**commit_failure_policy**
Policy for commit disk failures:

- **die** - Shut down the node and kill the JVM, so the node can be replaced.
- **stop** - Shut down the node, leaving the node effectively dead, available for inspection using JMX.
- **stop_commit** - Shut down the commit log, letting writes collect but continuing to service reads.
- **ignore** - Ignore fatal errors and let the batches fail.

Default: **stop**

**prepared_statements_cache_size_mb**
Maximum size of the native protocol prepared statement cache. Change this value only if there are more prepared statements than fit in the cache.
Generally, the calculated default value is appropriate and does not need adjusting. DataStax recommends contacting the DataStax Services team before changing this value.

**Note:** Specifying a value that is too large results in long running GCs and possibly out-of-memory errors. Keep the value at a small fraction of the heap.

Constantly re-preparing statements is a performance penalty. When not set, the default is automatically calculated to heap / 256 or 10 MB, whichever is greater. Default: calculated

### disk_optimization_strategy
The strategy for optimizing disk reads.
- ssd - solid state disks
- spinning - spinning disks

When commented out, the default is ssd.
Default: commented out (ssd)

### disk_failure_policy
Sets how the database responds to disk failure. Recommend settings: stop or best_effort. Valid values:
- die - Shut down gossip and client transports, and kill the JVM for any file system errors or single SSTable errors, so the node can be replaced.
- stop_paranoid - Shut down the node, even for single SSTable errors.
- stop - Shut down the node, leaving the node effectively dead, but available for inspection using JMX.
- best_effort - Stop using the failed disk and respond to requests based on the remaining available SSTables. This setting allows obsolete data at consistency level of ONE.
- ignore - Ignore fatal errors and lets the requests fail; all file system errors are logged but otherwise ignored.

**Tip:** See Recovering from a single disk failure using JBOD.

Default: stop

### endpoint_snitch
A class that implements the IEndpointSnitch interface. The database uses the snitch to locate nodes and route requests.

**Important:** Use only snitch implementations bundled with DSE.

- DseSimpleSnitch
  
  Appropriate only for development deployments. Proximity is determined by DSE workload, which places transactional, analytics, and search nodes into their separate datacenters. Does not recognize datacenter or rack information.

- GossipingPropertyFileSnitch
Recommended for production. Reads rack and datacenter for the local node in cassandra-rackdc.properties file and propagates these values to other nodes via gossip. For migration from the PropertyFileSnitch, uses the cassandra-topology.properties file if it is present.

- **PropertyFileSnitch**
  
  Determines proximity by rack and datacenter that are explicitly configured in cassandra-topology.properties file.

- **Ec2Snitch**
  
  For EC2 deployments in a single region. Loads region and availability zone information from the Amazon EC2 API. The region is treated as the datacenter, the availability zone is treated as the rack, and uses only private IP addresses. For this reason, Ec2Snitch does not work across multiple regions.

- **Ec2MultiRegionSnitch**
  
  Uses the public IP as the broadcast_address (page 80) to allow cross-region connectivity. This means you must also set seed (page 69) addresses to the public IP and open the storage_port (page 86) or ssl_storage_port (page 99) on the public IP firewall. For intra-region traffic, the database switches to the private IP after establishing a connection.

- **RackInferringSnitch**
  
  Proximity is determined by rack and datacenter, which are assumed to correspond to the 3rd and 2nd octet of each node’s IP address, respectively. Best used as an example for writing a custom snitch class (unless this happens to match your deployment conventions).

- **GoogleCloudSnitch**
  
  Use for deployments on Google Cloud Platform across one or more regions. The region is treated as a datacenter and the availability zones are treated as racks within the datacenter. All communication occurs over private IP addresses within the same logical network.

- **CloudstackSnitch**
  
  Use the CloudstackSnitch for Apache Cloudstack environments.

**Tip:** See Snitches.

**Default:** com.datastax.bdp.snitch.DseSimpleSnitch

**seed_provider**

The addresses of hosts that are designated as contact points in the cluster. A joining node contacts one of the nodes in the -seeds list to learn the topology of the ring.
**Important:** Use only seed provider implementations bundled with DSE.

- **class_name** - The class that handles the seed logic. It can be customized, but this is typically not required.
  Default: `org.apache.cassandra.locator.SimpleSeedProvider`

- **seeds** - A comma delimited list of addresses that are used by **gossip** for bootstrapping new nodes joining a cluster. If your cluster includes multiple nodes, you must change the list from the default value to the IP address of one of the nodes.
  Default: "127.0.0.1"

  **Attention:** Making every node a seed node is not recommended because of increased maintenance and reduced gossip performance. Gossip optimization is not critical, but it is recommended to use a small seed list (approximately three nodes per datacenter).

**Tip:** See Initializing a single datacenter per workload type and Initializing multiple datacenters per workload type.

Default: `org.apache.cassandra.locator.SimpleSeedProvider`

### Common compaction settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compaction_throughput_mb_per_sec</code></td>
<td>16</td>
</tr>
<tr>
<td><code>compaction_large_partition_warning_threshold_mb</code></td>
<td>100</td>
</tr>
</tbody>
</table>

**compaction_throughput_mb_per_sec**

The MB per second to throttle compaction for the entire system. The faster the database inserts data, the faster the system must compact in order to keep the SSTable count down.

- 16 to 32 x rate of write throughput in MB/second, recommended value.
- 0 - disable compaction throttling

**Tip:** See Configuring compaction.

Default: 16

**compaction_large_partition_warning_threshold_mb**

The partition size threshold before logging a warning.

Default: 100

### Memtable settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code># memtable_space_in_mb</code></td>
<td>2048</td>
</tr>
<tr>
<td><code>memtable_allocation_type</code></td>
<td>heap_buffers</td>
</tr>
<tr>
<td><code># memtable_cleanup_threshold</code></td>
<td>0.2</td>
</tr>
<tr>
<td><code>memtable_flush_writers</code></td>
<td>4</td>
</tr>
</tbody>
</table>

**memtable_space_in_mb**
Total permitted memory to use for memtables. When this threshold is exceeded, writes are not accepted until a flush completes. A flush is triggered based on **memtable_cleanup_threshold** *(page 71)*. When not set:

- If the deprecated settings are not present, the calculated default is 1/4 of the heap size.
- If the deprecated memtable_heap_space_in_mb or memtable_offheap_space_in_mb settings are present, an error is logged and the appropriate value is used based on memtable_allocation_type. Remove the deprecated settings.

**Tip:** See **Tuning the Java heap**.

Default: commented out 2048

**memtable_cleanup_threshold**
Ratio used for automatic memtable flush.

Generally, the calculated default value is appropriate and does not need adjusting. DataStax recommends contacting the DataStax Services team before changing this value.

When not set, the calculated default is \( \frac{1}{(\text{memtable_flush_writers} (page 71) + 1)} \)
Default: commented out (0.2)

**memtable_allocation_type**
The method the database uses to allocate and manage memtable memory.

- offheap_objects - Native memory, eliminating NIO buffer heap overhead.
- heap_buffers - On heap NIO (non-blocking I/O) buffers.
- offheap_buffers - Off heap (direct) NIO buffers.

Default: offheap_objects

**memtable_flush_writers**
The number of memtable flush writer threads per disk and the total number of memtables that can be flushed concurrently, generally a combination of computer and I/O bound. Memtable flushing is more CPU efficient than memtable ingest. A single thread can keep up with the ingest rate of a server on a single fast disk, until the server temporarily becomes IO bound under contention, typically with compaction. Generally, the default value is appropriate and does not need adjusting.
Default for SSDs: 4
Default for HDDs: 2

**memtable_heap_space_in_mb** *(deprecated)*

**Notice:** This setting is deprecated. Use **memtable_space_in_mb** *(page 70)* instead.

The amount of on-heap memory allocated for memtables. The database uses the total of this amount and the value of memtable_offheap_space_in_mb to set a threshold for automatic memtable flush.

Default: calculated 1/4 of heap size (2048)

**memtable_offheap_space_in_mb** *(deprecated)*
Notice: This setting is deprecated. Use `memtable_space_in_mb` (page 70) instead.

Replaced by `memtable_space_in_mb` (page 70). The amount of off-heap memory allocated for memtables. The database uses the total of this amount and the value of `memtable_heap_space_in_mb` to set a threshold for automatic memtable flush.

Default: calculated 1/4 of heap size (2048)

Common automatic backup settings

```
incremental_backups: false
snapshot_before_compaction: false
```

**incremental_backups**

Whether to enable incremental backups.

- true - Enable incremental backups to create a hard link to each SSTable flushed or streamed locally in a backups subdirectory of the keyspace data. Incremental backups enable storing backups off site without transferring entire snapshots.

  Important: The database does not automatically clear incremental backup files. DataStax recommends setting up a process to clear incremental backup hard links each time a new snapshot is created.

- false - Do not enable incremental backups.

  Tip: See Enabling incremental backups.

Default: `false`

**snapshot_before_compaction**

Whether to take a snapshot before each compaction. A snapshot is useful to back up data when there is a data format change.

  Important: Be careful using this option, the database does not clean up older snapshots automatically.

  Tip: See Configuring compaction.

Default: `false`

Performance tuning properties

Tuning performance and system resource utilization, including commit log, compaction, memory, disk I/O, CPU, reads, and writes.

Performing tuning properties include:

- Commit log settings (page 73)
- Change-data-capture (CDC) space settings (page 74)
- Common compaction settings (page 70)
- Memtable settings (page 70)
- Cache and index settings (page 76)
• **Disk settings (page 77)**

## Commit log settings

```plaintext
commitlog_sync: periodic
commitlog_sync_period_in_ms: 10000
# commitlog_sync_group_window_in_ms: 1000
# commitlog_sync_batch_window_in_ms: 2  //deprecated
commitlog_segment_size_in_mb: 32
# commitlog_total_space_in_mb: 8192
# commitlog_compression:
#   - class_name: LZ4Compressor
#     parameters:
#   -
```

**commitlog_sync**

The method that the database uses to acknowledge writes in milliseconds.

- **periodic** - Send ACK signal for writes immediately. Commit log is synced every `commitlog_sync_period_in_ms`.
- **group** - Send ACK signal for writes after the commit log has been flushed to disk. Wait up to `commitlog_sync_group_window_in_ms` between flushes.
- **batch** - Send ACK signal for writes after the commit log has been flushed to disk. Each incoming write triggers the flush task.

Default: periodic

**commitlog_sync_period_in_ms**

Use with `commitlog_sync: periodic`. Time interval between syncing the commit log to disk. Periodic syncs are acknowledged immediately.

Default: 10000

**commitlog_sync_group_window_in_ms**

Use with `commitlog_sync: group`. The time that the database waits between flushing the commit log to disk. DataStax recommends using `group` instead of `batch`.

Default: commented out (1000)

**commitlog_sync_batch_window_in_ms**

Deprecated. Use with `commitlog_sync: batch`. The maximum length of time that queries may be batched together.

Default: commented out (2)

**commitlog_segment_size_in_mb**

The size of an individual commitlog file segment. A commitlog segment may be archived, deleted, or recycled after all its data has been flushed to SSTables. This data can potentially include commitlog segments from every table in the system.

The default size is usually suitable, but for commitlog archiving you might want a finer granularity; 8 or 16 MB is reasonable.

**Restriction:**

If you set `max_mutation_size_in_kb (page 74)` explicitly, then you must set `commitlog_segment_size_in_mb` to:
2 * max_mutation_size_in_kb / 1024

The value must be positive and less than 2048.

**Tip:** See Commit log archive configuration.

Default: 32

**max_mutation_size_in_kb**

The maximum size of a mutation before the mutation is rejected. Before increasing the commitlog segment size of the commitlog segments, investigate why the mutations are larger than expected. Look for underlying issues with access patterns and data model, because increasing the commitlog segment size is a limited fix. When not set, the default is calculated as \( \frac{\text{commitlog_segment_size_in_mb} \times 1024}{2} \).

Default: calculated

**commitlog_total_space_in_mb**

Total space for commit logs. If the total space used by all commit logs exceeds this threshold, the database rounds up to the next nearest segment multiple and flushes memtables to disk for the oldest commitlog segments, removing those log segments from the commit log. This flushing reduces the amount of data to replay on start-up, and prevents infrequently updated tables from keeping commitlog segments indefinitely. If the `commitlog_total_space_in_mb` is small, the result is more flush activity on less-active tables.

**Tip:** See Configuring memtable thresholds.

Default for 64-bit JVMs: calculated \((8192 \text{ or } 25\% \text{ of the total space of the commit log value, whichever is smaller})\)

Default for 32-bit JVMs: calculated \((32 \text{ or } 25\% \text{ of the total space of the commit log value, whichever is smaller})\)

**commitlog_compression**

The compressor to use if commit log is compressed. To make changes, uncomment the `commitlog_compression` section and make changes to these options:

```
# commitlog_compression:
#   - class_name: LZ4Compressor
#     parameters:
#       -
```

- class_name: LZ4Compressor, Snappy, or Deflate
- parameters: optional parameters for the compressor

When not set, the default compression for the commit log is uncompressed.

Default: commented out

**Change-data-capture (CDC) space settings**

```
cdc_enabled: false
cdc_total_space_in_mb: 4096
cdc_free_space_check_interval_ms: 250
```
See also `cdc_raw_directory (page 66)`.

**cdc_enabled**

Whether to enable change data capture (CDC) functionality on a per-node basis. This modifies the logic used for write path allocation rejection.

- true - use cdc functionality to reject mutations that contain a CDC-enabled table if at space limit threshold in cdc_raw_directory
- false - standard behavior, never reject.

Default: `false`

**cdc_total_space_in_mb**

Total space to use for change-data-capture (CDC) logs on disk. If space allocated for CDC exceeds this value, the database throws WriteTimeoutException on mutations, including CDC-enabled tables. A CDCCompactor (a consumer) is responsible for parsing the raw CDC logs and deleting them when parsing is completed.

Default: `calculated (4096 or 1/8th of the total space of the drive where the cdc_raw_directory (page 66) resides)`

**cdc_free_space_check_interval_ms**

Interval between checks for new available space for CDC-tracked tables when the cdc_total_space_in_mb (page 75) threshold is reached and the CDCCompactor is running behind or experiencing back pressure. When not set, the default is 250.

Default: commented out (250)

### Compaction settings

```
# concurrent_compactors: 1
# concurrent_validations: 0
concurrent_materialized_view_builders: 2
sstable_preemptive_open_interval_in_mb: 50
# pick_level_on_streaming: false
```

**Tip:** See also `compaction_throughput_mb_per_sec (page 70)` in the common compaction settings section and Configuring compaction.

**concurrent_compactors**

The number of concurrent compaction processes allowed to run simultaneously on a node, not including validation compactions for anti-entropy repair. Simultaneous compactions help preserve read performance in a mixed read-write workload by limiting the number of small SSTables that accumulate during a single long-running compaction. If your data directories are backed by SSDs, increase this value to the number of cores. If compaction running too slowly or too fast, adjust `compaction_throughput_mb_per_sec (page 70)` first.

**Important:** Increasing concurrent compactors leads to more use of available disk space for compaction, because concurrent compactions happen in parallel, especially for STCS. Ensure that adequate disk space is available before increasing this configuration.
Generally, the calculated default value is appropriate and does not need adjusting. DataStax recommends contacting the DataStax Services team before changing this value.

**Default:** calculated The fewest number of disks or number of cores, with a minimum of 2 and a maximum of 8 per CPU core.

**concurrent_validations**

Number of simultaneous repair validations to allow. When not set, the default is unbounded. Values less than one are interpreted as unbounded.

**Default:** commented out (0) unbounded

**concurrent_materialized_view_builders**

Number of simultaneous materialized view builder tasks allowed to run concurrently. When a view is created, the node ranges are split into (num_processors * 4) builder tasks and submitted to this executor.

**Default:** 2

**sstable_preemptive_open_interval_in_mb**

The size of the SSTables to trigger preemptive opens. The compaction process opens SSTables before they are completely written and uses them in place of the prior SSTables for any range previously written. This process helps to smoothly transfer reads between the SSTables by reducing cache churn and keeps hot rows hot.

**Important:** A low value has a negative performance impact and will eventually cause heap pressure and GC activity. The optimal value depends on hardware and workload.

**Default:** 50

**pick_level_on_streaming**

The compaction level for streamed-in SSTables.

- true - streamed-in SSTables of tables using LeveledCompactionStrategy (LCS) are placed on the same level as the source node. For operational tasks like nodetool refresh or replacing a node, true improves performance for compaction work.
- false - streamed-in SSTables are placed in level 0.

When not set, the default is false.

**Default:** commented out (false)

**Cache and index settings**

```
column_index_size_in_kb: 16
# file_cache_size_in_mb: 4096
# direct_reads_size_in_mb: 128
```

**column_index_size_in_kb**

Granularity of the index of rows within a partition. For huge rows, decrease this setting to improve seek time. Lower density nodes might benefit from decreasing this value to 4, 2, or 1.

**Default:** 16

**file_cache_size_in_mb**
DSE 6.7.0-6.7.2: Maximum memory for buffer pooling and SSTable chunk cache. 32 MB is reserved for pooling buffers, the remaining memory is the cache for holding recent or frequently used index pages and uncompressed SSTable chunks. This pool is allocated off heap and is in addition to the memory allocated for heap. Memory is allocated only when needed.

DSE 6.7.3 and later: Buffer pool is split into two pools, this setting defines the maximum memory to use file buffers that are stored in the file cache, also known as chunk cache. Memory is allocated only when needed but is not released. The other buffer pool is direct_reads_size_in_mb (page 77).

See Tuning Java Virtual Machine.
Default: calculated \((0.5 \text{ of } -\text{XX:MaxDirectMemorySize})\)

**direct_reads_size_in_mb**
- **DSE 6.7.3 and later:** Buffer pool is split into two pools, this setting defines the buffer pool for transient read operations. A buffer is typically used by a read operation and then returned to this pool when the operation is finished so that it can be reused by other operations. The other buffer pool is file_cache_size_in_mb (page 76). When not set, the default calculated as 2 MB per TPC core thread, plus 2 MB shared by non-TPC threads, with a maximum value of 128 MB.
- Default: calculated

Disk settings

```plaintext
# stream_throughput_outbound_megabits_per_sec: 200
# inter_dc_stream_throughput_outbound_megabits_per_sec: 200
# streaming_keep_alive_period_in_secs: 300
# streaming_connections_per_host: 1
```

**stream_throughput_outbound_megabits_per_sec**
- Throttle for the throughput of all outbound streaming file transfers on a node. The database does mostly sequential I/O when streaming data during bootstrap or repair which can saturate the network connection and degrade client (RPC) performance. When not set, the value is 200 Mbps.
- Default: commented out \((200)\)

**inter_dc_stream_throughput_outbound_megabits_per_sec**
- Throttle for all streaming file transfers between datacenters, and for network stream traffic as configured with stream_throughput_outbound_megabits_per_sec (page 77). When not set, the value is 200 Mbps.
- **Note:** Should be set to a value less than or equal to stream_throughput_outbound_megabits_per_sec since it is a subset of total throughput.
- Default: commented out \((200)\)

**streaming_keep_alive_period_in_secs**
- Interval to send keep-alive messages. The stream session fails when a keep-alive message is not received for 2 keep-alive cycles. When not set, the default is 300 seconds (5 minutes) so that a stalled stream times out in 10 minutes.
- Default: commented out \((300)\)
**Configuration**

Maximum number of connections per host for streaming. Increase this value when you notice that joins are CPU-bound, rather than network-bound. For example, a few nodes with large files. When not set, the default is 1. Default: commented out (1)

**Fs Cross settings**

```yaml
trickle_fsync: true
trickle_fsync_interval_in_kb: 10240
```

**trickle_fsync**
When set to true, causes fsync to force the operating system to flush the dirty buffers at the set interval trickle_fsync_interval_in_kb. Enable this parameter to prevent sudden dirty buffer flushing from impacting read latencies. Recommended for use with SSDs, but not with HDDs.

Default: false

**trickle_fsync_interval_in_kb**
The size of the fsync in kilobytes.

Default: 10240

**max_value_size_in_mb**
The maximum size of any value in SSTables. SSTables are marked as corrupted when the threshold is exceeded.

Default: 256

**Thread Per Core (TPC) parameters**

```yaml
#tpc_cores:
# tpc_io_cores:
io_global_queue_depth: 128
```

**tpc_cores**
The number of TPC event loops. If not set, the default is the number of cores (processors on the machine) minus one. For most workloads: Do not tune. DataStax recommends contacting the DataStax Services team before changing this value.

**Attention:** DSE Search workloads only: change tpc_cores (page 78) from the default to the number of physical CPUs. See Configuring and tuning indexing performance.

Default: commented out (number of cores -1)

**tpc_io_cores**
The number of tpc_cores (page 78) to use for asynchronous disk reads.

**Attention:** Do not tune. DataStax recommends contacting the DataStax Services team before changing this value.

Default: commented out (min(io_global_queue_depth/4, tpc_cores)

**io_global_queue_depth**
Global IO queue depth used for reads when AIO is enabled (the default for SSDs). The optimal queue depth as found with the fio tool for a given disk setup.
Attention: Do not tune. DataStax recommends contacting the DataStax Services team before changing this value.

Default: 128

NodeSync parameters

```
nodesync:
  rate_in_kb: 1024
```

By default, the NodeSync service runs on every node.

**Tip:** Manage the NodeSync service using the `nodetool nodesyncservice (page 1015)` command.

**Tip:** See Setting the rate (page 171).

**rate_in_kb**

The maximum bytes per second for data validation on the local node. The optimum validation rate for each node may vary.

Default: 1024

Advanced properties

Properties for advanced users or properties that are less commonly used.

**Advanced initialization properties**

```
batch_size_warn_threshold_in_kb: 64
batch_size_fail_threshold_in_kb: 640
unlogged_batch_across_partitions_warn_threshold: 10
# broadcast_address: 1.2.3.4
# listen_on_broadcast_address: false
# initial_token:
# num_tokens: 128
# allocate_tokens_for_local_replication_factor: 3
partitioner: org.apache.cassandra.dht.Murmur3Partitioner
trace_type_query_ttl: 86400
trace_type_repair_ttl: 604800
```

**auto_bootstrap**

This setting has been removed from default configuration.

- true - causes new (non-seed) nodes migrate the right data to themselves automatically
- false - When initializing a fresh cluster without data

**Tip:** See Initializing a DataStax Enterprise cluster.

When not set, the internal default is true.

Default: not present

**batch_size_warn_threshold_in_kb**
Threshold to log a warning message when any multiple-partition batch size exceeds this value in kilobytes.

**Caution:** Increasing this threshold can lead to node instability.

Default: 64

**batch_size_fail_threshold_in_kb**

Threshold to fail and log WARN on any multiple-partition batch whose size exceeds this value. The default value is 10X the value of batch_size_warn_threshold_in_kb.

Default: 640

**unlogged_batch_across_partitions_warn_threshold**

Threshold to log a WARN message on any batches not of type LOGGED that span across more partitions than this limit.

Default: 10

**broadcast_address**

The public IP address this node uses to broadcast to other nodes outside the network or across regions in multiple-region EC2 deployments. If this property is commented out, the node uses the same IP address or hostname as listen_address. A node does not need a separate broadcast_address in a single-node or single-datacenter installation, or in an EC2-based network that supports automatic switching between private and public communication. It is necessary to set a separate broadcast_address and broadcast_address on a node with multiple physical network interfaces or other topologies where not all nodes have access to other nodes by their private IP addresses. For specific configurations, see the instructions for **listen_address** (page 65).

Default: **listen_address** (page 65)

**listen_on_broadcast_address**

Whether to enable the node to communicate on both interfaces.

- true - If this node uses multiple physical network interfaces, set a unique IP address for broadcast_address (page 80)
- false - If this node is on a network that automatically routes between public and private networks, like Amazon EC2 does

**Tip:** See **listen_address** (page 65).

Default: false

**initial_token**

The token to start the contiguous range. Set this property for single-node-per-token architecture, in which a node owns exactly one contiguous range in the ring space. Setting this property overrides **num_tokens** (page 80).

If your installation is not using vnodes or this node's **num_tokens** (page 80) is set to 1 or is commented out, you should always set an initial_token value when setting up a production cluster for the first time, and when adding capacity. See Generating tokens.

Use this parameter only with **num_tokens** (page 80) (vnodes) in special cases such as Restoring from a snapshot.

Default: 1 (disabled)

**num_tokens**

Define virtual node (vnode) token architecture.
Note: All other nodes in the datacenter must have the same token architecture.

- 1 - disable vnodes and use 1 token for legacy compatibility.
- a number between 2 and 128 - the number of token ranges to assign to this virtual node (page 173) (vnode). A higher value increases the probability that the data and workload are evenly distributed.

Restriction: DataStax recommends not using vnodes with DSE Search. However, if you decide to use vnodes with DSE Search, do not use more than 8 vnodes and ensure that allocate_tokens_for_local_replication_factor (page 81) option in cassandra.yaml is correctly configured for your environment.

Caution: Using vnodes can impact performance for your cluster. DataStax recommends testing the configuration before enabling vnodes in production environments.

When the token number varies between nodes in a datacenter, the vnode logic assigns a proportional number of ranges relative to other nodes in the datacenter. In general, if all nodes have equal hardware capability, each node should have the same num_tokens value.

Default: 1 (disabled)

To migrate an existing cluster from single node per token range to vnodes, see Enabling virtual nodes on an existing production cluster.

allocate_tokens_for_local_replication_factor

- RF of keyspaces in datacenter - triggers the recommended algorithmic allocation for the RF and num_tokens (page 80) for this node.

The allocation algorithm optimizes the workload balance using the target keyspace replication factor. DataStax recommends setting the number of tokens to 8 to distribute the workload with ~10% variance between nodes. The allocation algorithm attempts to choose tokens in a way that optimizes replicated load over the nodes in the datacenter for the specified RF. The load assigned to each node is close to proportional to the number of vnodes.

Note: The allocation algorithm is supported only for the Murmur3Partitioner and RandomPartitioner partitioners. The Murmur3Partitioner is the default partitioning strategy for new clusters and the right choice for new clusters in almost all cases.

- commented out - uses the random selection algorithm to assign token ranges randomly.

Note: Over time, loads in a datacenter using the random selection algorithm become unevenly distributed. DataStax recommends using only the allocation algorithm.
Configuration

Default: commented out (use random selection algorithm)

See Virtual node (vnode) configuration (page 173), and for set up instructions see
Adding nodes to vnode-enabled cluster or Adding a datacenter to a cluster.

**partitioner**
The class that distributes rows (by partition key) across all nodes in the cluster. Any
IPartitioner may be used, including your own as long as it is in the class path.
For new clusters use the default partitioner.

DataStax Enterprise provides the following partitioners for backward compatibility:

- RandomPartitioner
- ByteOrderedPartitioner (deprecated)
- OrderPreservingPartitioner (deprecated)

**Important:** Use only partitioner implementations bundled with DSE.

**Tip:** See Partitioners.

Default: org.apache.cassandra.dht.Murmur3Partitioner

**tracetype_query_ttl**
TTL for different trace types used during logging of the query process.
Default: 86400

**tracetype_repair_ttl**
TTL for different trace types used during logging of the repair process.
Default: 604800

**Advanced automatic backup setting**

```
auto_snapshot: true
```

**auto_snapshot**
Whether to enable snapshots of the data before truncating a keyspace or dropping
a table. To prevent data loss, DataStax strongly advises using the default setting. If
you set auto_snapshot to false, you lose data on truncation or drop.
Default: true

**Global row properties**

```
column_index_cache_size_in_kb: 2
# row_cache_class_name: org.apache.cassandra.cache.OHCProvider
row_cache_size_in_mb: 0
row_cache_save_period: 0
# row_cache_keys_to_save: 100
```

When creating or modifying tables, you can enable or disable the row cache for that table by
setting the caching parameter. Other row cache tuning and configuration options are set at
the global (node) level. The database uses these settings to automatically distribute memory
for each table on the node based on the overall workload and specific table usage. You can
also configure the save periods for these caches globally.
Tip: See Configuring caches.

column_index_cache_size_in_kb
(Only applies to BIG format SSTables) Threshold for the total size of all index entries for a partition that the database stores in the partition key cache. If the total size of all index entries for a partition exceeds this amount, the database stops putting entries for this partition into the partition key cache.
Default: 2

row_cache_class_name
The classname of the row cache provider to use. Valid values:

- org.apache.cassandra.cache.OHCP provider - fully off-heap
- org.apache.cassandra.cache.SerializingCacheProvider - partially off-heap, available in earlier releases

Important: Use only row cache provider implementations bundled with DSE.

When not set, the default is org.apache.cassandra.cache.OHCP provider (fully off-heap)
Default: commented out (org.apache.cassandra.cache.OHCP provider)

row_cache_size_in_mb
Maximum size of the row cache in memory. The row cache can save time, but it is space-intensive because it contains the entire row. Use the row cache only for hot rows or static rows. If you reduce the size, you may not get you hottest keys loaded on start up.

- 0 - disable row caching
- MB - Maximum size of the row cache in memory

Default: 0 (disabled)

row_cache_save_period
The number of seconds that rows are kept in cache. Caches are saved to saved_caches_directory (page 67). This setting has limited use as described in row_cache_size_in_mb.
Default: 0 (disabled)

row_cache_keys_to_save
The number of keys from the row cache to save. All keys are saved.
Default: commented out (100)

Counter caches properties

counter_cache_size_in_mb: 7200
counter_cache_save_period: 7200
# counter_cache_keys_to_save: 100

Counter cache helps to reduce counter locks' contention for hot counter cells. In case of RF = 1 a counter cache hit causes the database to skip the read before write entirely. With RF > 1 a counter cache hit still helps to reduce the duration of the lock hold, helping with hot counter cell updates, but does not allow skipping the read entirely. Only the local (clock,
count) tuple of a counter cell is kept in memory, not the whole counter, so it is relatively cheap.

**Note:** If you reduce the counter cache size, the database may load the hottest keys start-up.

**counter_cache_size_in_mb**
When no value is set, the database uses the smaller of minimum of 2.5% of heap or 50 megabytes (MB). If your system performs counter deletes and relies on low gc_grace_seconds, you should disable the counter cache. To disable, set to 0.
Default: calculated

**counter_cache_save_period**
The time, in seconds, after which the database saves the counter cache (keys only). The database saves caches to saved_caches_directory (page 67).
Default: 7200 (2 hours)

**counter_cache_keys_to_save**
Number of keys from the counter cache to save. When not set, the database saves all keys.
Default: commented out (disabled, saves all keys)

**Tombstone settings**

| tombstone_warn_threshold: 1000 | tombstone_failure_threshold: 100000 |

When executing a scan, within or across a partition, the database must keep tombstones in memory to allow them to return to the coordinator. The coordinator uses tombstones to ensure that other replicas know about the deleted rows. Workloads that generate numerous tombstones may cause performance problems and exhaust the server heap. Adjust these thresholds only if you understand the impact and want to scan more tombstones. You can adjust these thresholds at runtime using the StorageServiceMBean.

**Tip:** See the DataStax Developer Blog post *Cassandra anti-patterns: Queues and queue-like datasets*.

**tombstone_warn_threshold**
The database issues a warning if a query scans more than this number of tombstones.
Default: 1000

**tombstone_failure_threshold**
The database aborts a query if it scans more than this number of tombstones.
Default: 100000

**Network timeout settings**

| read_request_timeout_in_ms: 5000 | range_request_timeout_in_ms: 10000 |
| aggregated_request_timeout_in_ms: 120000 | write_request_timeout_in_ms: 2000 |
| counter_write_request_timeout_in_ms: 5000 |
read_request_timeout_in_ms
   Default: 5000. How long the coordinator waits for read operations to complete
   before timing it out.

range_request_timeout_in_ms
   Default: 10000. How long the coordinator waits for sequential or index scans to
   complete before timing it out.

aggregated_request_timeout_in_ms
   How long the coordinator waits for sequential or index scans to complete. Lowest
   acceptable value is 10 ms. This timeout does not apply to aggregated queries such
   as SELECT, COUNT(\*), MIN(x), and so on.
   Default: 120000 (2 minutes)

write_request_timeout_in_ms
   How long the coordinator waits for write requests to complete with at least one
   node in the local datacenter. Lowest acceptable value is 10 ms.
   Tip: See Hinted handoff: repair during write path.
   Default: 2000 (2 seconds)

counter_write_request_timeout_in_ms
   How long the coordinator waits for counter writes to complete before timing it out.
   Default: 5000 (5 seconds)

cas_contention_timeout_in_ms
   How long the coordinator continues to retry a CAS (compare and set) operation that
   contends with other proposals for the same row. If the coordinator cannot complete
   the operation within this timespan, it aborts the operation.
   Default: 1000 (1 second)

truncate_request_timeout_in_ms
   How long the coordinator waits for a truncate (the removal of all data from a table)
   to complete before timing it out. The long default value allows the database to take
   a snapshot before removing the data. If auto_snapshot (page 82) is disabled
   (not recommended), you can reduce this time.
   Default: 60000 (1 minute)

request_timeout_in_ms
   The default timeout value for other miscellaneous operations. Lowest acceptable
   value is 10 ms.
   Tip: See Hinted handoff: repair during write path.
   Default: 10000

cross_dc_rtt_in_ms
   How much to increase the cross-datacenter timeout
   (write_request_timeout_in_ms + cross_dc_rtt_in_ms) for requests that
   involve only nodes in a remote datacenter. This setting is intended to reduce hint
   pressure.
   Tip: DataStax recommends using LOCAL_* consistency levels (CL) for
   read and write requests in multi-datacenter deployments to avoid timeouts.
that may occur when remote nodes are chosen to satisfy the CL, such as QUORUM.

Default: commented out (0)

**slow_query_log_timeout_in_ms**
Default: 500. How long before a node logs slow queries. Select queries that exceed this value generate an aggregated log message to identify slow queries. To disable, set to 0.

### Inter-node settings

```
storage_port: 7000
cross_node_timeout: false
# internode_send_buff_size_in_bytes:
# internode_recv_buff_size_in_bytes:
internode_compression: dc
inter_dc_tcp_nodelay: false
```

**storage_port**
The port for inter-node communication. Follow security best practices, do not expose this port to the internet. Apply firewall rules.

**Default:** 7000

**cross_node_timeout**
Whether to enable operation timeout information exchange between nodes to accurately measure request timeouts. If this property is disabled, the replica assumes any requests are forwarded to it instantly by the coordinator. During overload conditions this means extra time is required for processing already-timed-out requests.

**Caution:** Before enabling this property make sure NTP (network time protocol) is installed and the times are synchronized among the nodes.

**Default:** false

**internode_send_buff_size_in_bytes**
The sending socket buffer size, in bytes, for inter-node calls.

**Tip:** See TCP settings (page 58).

The sending socket buffer size and **internode_recv_buff_size_in_bytes** (page 86) is limited by `net.core.wmem_max`. If this property is not set, `net.ipv4.tcp_wmem` determines the buffer size. For more details run `man tcp` and refer to:

- `/proc/sys/net/core/wmem_max`
- `/proc/sys/net/core/rmem_max`
- `/proc/sys/net/ipv4/tcp_wmem`
- `/proc/sys/net/ipv4/tcp_wmem`

**Default:** not set

**internode_recv_buff_size_in_bytes**
The receiving socket buffer size in bytes for inter-node calls.
Default: not set

**internode_compression**
Controls whether traffic between nodes is compressed. Valid values:
- **all** - Compresses all traffic
- **dc** - Compresses traffic between datacenters only
- **none** - No compression.
Default: dc

**inter_dc_tcp_nodelay**
Whether to enable tcp_nodelay for inter-datacenter communication. When disabled, the network sends larger, but fewer, network packets. This reduces overhead from the TCP protocol itself. However, disabling `inter_dc_tcp_nodelay` may increase latency by blocking cross datacenter responses.
Default: false

## Native transport (CQL Binary Protocol)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_native_transport:</td>
<td>true</td>
</tr>
<tr>
<td>native_transport_port:</td>
<td>9042</td>
</tr>
<tr>
<td>native_transport_port_ssl:</td>
<td>9142</td>
</tr>
<tr>
<td>native_transport_max_frame_size_in_mb:</td>
<td>256</td>
</tr>
<tr>
<td>native_transport_max_concurrent_connections:</td>
<td>-1</td>
</tr>
<tr>
<td>native_transport_max_concurrent_connections_per_ip:</td>
<td>-1</td>
</tr>
<tr>
<td>native_transport_address:</td>
<td>localhost</td>
</tr>
<tr>
<td>native_transport_interface:</td>
<td>eth0</td>
</tr>
<tr>
<td>native_transport_interface_prefer_ipv6:</td>
<td>false</td>
</tr>
<tr>
<td>native_transport_broadcast_address:</td>
<td>1.2.3.4</td>
</tr>
<tr>
<td>native_transport_keepalive:</td>
<td>true</td>
</tr>
</tbody>
</table>

**Tip:** See also `native_transport_port_ssl (page 99)` in SSL Ports (page 99).

**start_native_transport**
Enables or disables the native transport server.
Default: true

**native_transport_port**
The port where the CQL native transport listens for clients. For security reasons, do not expose this port to the internet. Firewall it if needed.
Default: 9042

**native_transport_max_frame_size_in_mb**
The maximum allowed size of a frame. Frame (requests) larger than this are rejected as invalid.
Default: 256

**native_transport_max_concurrent_connections**
The maximum number of concurrent client connections.
Default: -1 (unlimited)

**native_transport_max_concurrent_connections_per_ip**
The maximum number of concurrent client connections per source IP address.
Default: -1 (unlimited)
When left blank, uses the configured hostname of the node. Unlike the listen_address, this value can be set to 0.0.0.0, but you must set the native_transport_broadcast_address to a value other than 0.0.0.0.

**Note:** Set native_transport_address OR native_transport_interface, not both.

_default: localhost

**native_transport_interface**

IP aliasing is not supported.

**Note:** Set native_transport_address OR native_transport_interface, not both.

_default: eth0

**native_transport_interface_prefer_ipv6**

Use IPv4 or IPv6 when interface is specified by name.

- false - use first IPv4 address.
- true - use first IPv6 address.

When only a single address is used, that address is selected without regard to this setting.

_default: commented out (false)

**native_transport_broadcast_address**

Native transport address to broadcast to drivers and other DSE nodes. This cannot be set to 0.0.0.0.

- blank - will be set to the value of native_transport_address
- _IP_address_ - when native_transport_address is set to 0.0.0.0

_default: commented out (1.2.3.4)

**native_transport_keepalive**

Whether to enable keepalive on native connections.
_default: true

Advanced fault detection settings

Settings to handle poorly performing or failing components.

```
# gc_log_threshold_in_ms: 200
# gc_warn_threshold_in_ms: 1000
# otc_coalescing_strategy: DISABLED
# otc_coalescing_window_us: 200
# otc_coalescing_enough_coalesced_messages: 8
```

**gc_log_threshold_in_ms**

The threshold for log messages at the INFO level. Adjust to minimize logging.
_default: commented out (200)

**gc_warn_threshold_in_ms**

Threshold for GC pause. Any GC pause longer than this interval is logged at the WARN level. By default, the database logs any GC pause greater than 200 ms at the INFO level.

**Tip:** See Configuring logging.
Default: commented out (1000)

**otc_coalescing_strategy**

Strategy to combine multiple network messages into a single packet for outbound TCP connections to nodes in the same data center. See the DataStax Developer Blog post Performance doubling with message coalescing.

**Important:** Use only strategy implementations bundled with DSE.

Supported strategies are:

- FIXED
- MOVINGAVERAGE
- TIMEHORIZON
- DISABLED

Default: commented out (DISABLED)

**otc_coalescing_window_us**

How many microseconds to wait for coalescing messages to nodes in the same datacenter.

- For FIXED strategy - the amount of time after the first message is received before it is sent with any accompanying messages.
- For MOVING average - the maximum wait time and the interval that messages must arrive on average to enable coalescing.

Default: commented out (200)

**otc_coalescing_enough_coalesced_messages**

The threshold for the number of messages to nodes in the same data center. Do not coalesce messages when this value is exceeded. Should be more than 2 and less than 128.

Default: commented out (8)

**seed_gossip_probability**

The percentage of time that gossip messages are sent to a seed node during each round of gossip. Decreases the time to propagate gossip changes across the cluster.

Default: 1.0 (100%)

**Backpressure settings**

```
back_pressure_enabled: false
  back_pressure_strategy:
    - class_name: org.apache.cassandra.net.RateBasedBackPressure
      parameters:
        - high_ratio: 0.90
          factor: 5
          flow: FAST
```

**back_pressure_enabled**

Whether to enable for the coordinator to apply the specified back pressure strategy to each mutation that is sent to replicas.

Default: false

**back_pressure_strategy**
To add new strategies, implement org.apache.cassandra.net.BackpressureStrategy and provide a public constructor that accepts a `Map<String, Object>`.

**Important:** Use only strategy implementations bundled with DSE.

### class_name
The default class name uses the ratio between incoming mutation responses and outgoing mutation requests.

**Default:** `org.apache.cassandra.net.RateBasedBackPressure`

### high_ratio
When outgoing mutations are below this value, they are rate limited according to the incoming rate decreased by the factor (described below). When above this value, the rate limiting is increased by the factor.

**Default:** 0.90

### factor
A number between 1 and 10. When backpressure is below high ratio, outgoing mutations are rate limited according to the incoming rate decreased by the given factor; if above high ratio, the rate limiting is increased by the given factor.

**Default:** 5

### flow
The flow speed to apply rate limiting:

- **FAST** - rate limited to the speed of the fastest replica
- **SLOW** - rate limit to the speed of the slowest replica

**Default:** `FAST`

### dynamic_snitch_badness_threshold
The performance threshold for dynamically routing client requests away from a poorly performing node. Specifically, it controls how much worse a poorly performing node has to be before the `dynamic snitch` prefers other replicas. A value of 0.2 means the database continues to prefer the static snitch values until the node response time is 20% worse than the best performing node. Until the threshold is reached, incoming requests are statically routed to the closest replica as determined by the snitch.

**Default:** 0.1

### dynamic_snitch_reset_interval_in_ms
Time interval after which the database resets all node scores. This allows a bad node to recover.

**Default:** 600000

### dynamic_snitch_update_interval_in_ms
The time interval, in milliseconds, between the calculation of node scores. Because score calculation is CPU intensive, be careful when reducing this interval.

**Default:** 100

### Hinted handoff options

```
hinted_handoff_enabled: true
# hinted_handoff_disabled_datacenters:
#   - DC1
#   - DC2
max_hint_window_in_ms: 10800000 # 3 hours
```
<table>
<thead>
<tr>
<th>Configuration Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>hinted_handoff_throttle_in_kb</td>
<td>1024</td>
</tr>
<tr>
<td>max_hints_delivery_threads</td>
<td>2</td>
</tr>
<tr>
<td>hints_directory</td>
<td>/var/lib/cassandra/hints</td>
</tr>
<tr>
<td>hints_flush_period_in_ms</td>
<td>10000</td>
</tr>
<tr>
<td>max_hints_file_size_in_mb</td>
<td>128</td>
</tr>
<tr>
<td>hints_compression</td>
<td></td>
</tr>
<tr>
<td>batchlog_replay_throttle_in_kb</td>
<td>1024</td>
</tr>
<tr>
<td>batchlog_endpoint_strategy</td>
<td>random_remote</td>
</tr>
</tbody>
</table>

**Tip:** See Hinted handoff: repair during write path.

### hinted_handoff_enabled
Enables or disables hinted handoff. A hint indicates that the write needs to be replayed to an unavailable node. The database writes the hint to a hints file on the coordinator node.

- **false** - do not enable hinted handoff
- **true** - globally enable hinted handoff, except for datacenters specified for `hinted_handoff_disabled_datacenters`

Default: **true**

### hinted_handoff_disabled_datacenters
A blacklist of datacenters that will not perform hinted handoffs. To disable hinted handoff on a certain datacenter, add its name to this list.

Default: commented out

### max_hint_window_in_ms
Maximum amount of time during which the database generates hints for an unresponsive node. After this interval, the database does not generate any new hints for the node until it is back up and responsive. If the node goes down again, the database starts a new interval. This setting can prevent a sudden demand for resources when a node is brought back online and the rest of the cluster attempts to replay a large volume of hinted writes.

**Tip:** See About failure detection and recovery.

Default: 10800000 (3 hours)

### hinted_handoff_throttle_in_kb
Maximum amount of traffic per delivery thread in kilobytes per second. This rate reduces proportionally to the number of nodes in the cluster. For example, if there are two nodes in the cluster, each delivery thread uses half the maximum rate. If there are three, each node throttles to half of the maximum, since the two nodes are expected to deliver hints simultaneously.

**Note:** When applying this limit, the calculated hint transmission rate is based on the uncompressed hint size, even if internode_compression (page 87) or hints_compression (page 92) is enabled.

Default: 1024

### hints_flush_period_in_ms
The time, in milliseconds, to wait before flushing hints from internal buffers to disk.
**max_hints_delivery_threads**
Number of threads the database uses to deliver hints. In multiple datacenter deployments, consider increasing this number because cross datacenter handoff is generally slower.
Default: 2

**max_hints_file_size_in_mb**
The maximum size for a single hints file, in megabytes.
Default: 128

**hints_compression**
The compressor for hint files. Supported compressors: LZ, Snappy, and Deflate.
When not set, the database does not compress hints files.
Default: LZ4Compressor

**batchlog_replay_throttle_in_kb**
Total maximum throttle, in KB per second, for replaying hints. Throttling is reduced proportionally to the number of nodes in the cluster.
Default: 1024

**batchlog_endpoint_strategy**
Strategy to choose the batchlog storage endpoints.
- random_remote - Default, purely random. Prevents the local rack, if possible. Same behavior as earlier releases.
- dynamic_remote - Uses DynamicEndpointSnitch to select batchlog storage endpoints. Prevents the local rack, if possible. This strategy offers the same availability guarantees as random_remote, but selects the fastest endpoints according to the DynamicEndpointSnitch. DynamicEndpointSnitch tracks reads but not writes. Write-only, or mostly-write, workloads might not benefit from this strategy. Note: this strategy will fall back to random_remote if dynamic_snitch is not enabled.
- dynamic - Mostly the same as dynamic_remote, except that local rack is not excluded, which offers lower availability guarantee than random_remote or dynamic_remote. Note: this strategy will fall back to random_remote if dynamic_snitch is not enabled.
Default: random_remote

Security properties

DSE Advanced Security fortifies DataStax Enterprise (DSE) databases against potential harm due to deliberate attack or user error. Configuration properties include authentication and authorization, permissions, roles, encryption of data in-flight and at-rest, and data auditing. DSE Unified Authentication provides authentication, authorization, and role management. Enabling DSE Unified Authentication requires additional configuration in dse.yaml, see Configuring DSE Unified Authentication.

```yaml
authenticator: com.datastax.bdp.cassandra.auth.DseAuthenticator
# internode_authenticator: org.apache.cassandra.auth.AllowAllInternodeAuthenticator
authorizer: com.datastax.bdp.cassandra.auth.DseAuthorizer
role_manager: com.datastax.bdp.cassandra.auth.DseRoleManager
```
**system_keyspaces_filtering**: false
**roles_validity_in_ms**: 120000
# **roles_update_interval_in_ms**: 120000
**permissions_validity_in_ms**: 120000
# **permissions_update_interval_in_ms**: 120000

**authenticator**
The authentication backend. The only supported authenticator is DseAuthenticator for external authentication with multiple authentication schemes such as Kerberos, LDAP, and internal authentication. Authenticators other than DseAuthenticator are deprecated and not supported. Some security features might not work correctly if other authenticators are used. See authentication_options (page 104) in dse.yaml.

**Important:** Use only authentication implementations bundled with DSE.

Default: com.datastax.bdp.cassandra.auth.DseAuthenticator

**internode_authenticator**
Internode authentication backend to enable secure connections from peer nodes.

**Important:** Use only authentication implementations bundled with DSE.

Default: org.apache.cassandra.auth.AllowAllInternodeAuthenticator

**authorizer**
The authorization backend. Authorizers other than DseAuthorizer are not supported. DseAuthorizer supports enhanced permission management of DSE-specific resources. Authorizers other than DseAuthorizer are deprecated and not supported. Some security features might not work correctly if other authorizers are used. See Authorization options (page 107) in dse.yaml.

**Important:** Use only authorization implementations bundled with DSE.

Default: com.datastax.bdp.cassandra.auth.DseAuthorizer

**system_keyspaces_filtering**
Whether to enable system keyspace filtering so that users can access and view only schema information for rows in the system and system_schema keyspaces to which they have access.

**Attention:** Security requirements and user permissions apply. Enable this feature only after appropriate user permissions are granted.

**Tip:** See Controlling access to keyspaces and tables and Configuring the security keyspaces replication factors.

Default: false

**role_manager**
The DSE Role Manager supports LDAP roles and internal roles supported by the CassandraRoleManager. Role options are stored in the dse_security keyspace. When using the DSE Role Manager, increase the replication factor of the dse_security keyspace. Role managers other than DseRoleManager are deprecated and not supported. Some security features might not work correctly if other role managers are used.

**Important:** Use only role manager implementations bundled with DSE.
Default: com.datastax.bdp.cassandra.auth.DseRoleManager

**roles_validity_in_ms**
Validity period for roles cache in milliseconds. Determines how long to cache the list of roles assigned to the user; users may have several roles, either through direct assignment or inheritance (a role that has been granted to another role). Adjust this setting based on the complexity of your role hierarchy, tolerance for role changes, the number of nodes in your environment, and activity level of the cluster.

Fetching permissions can be an expensive operation, so this setting allows flexibility. Granted roles are cached for authenticated sessions in AuthenticatedUser. After the specified time elapses, role validity is rechecked. Disabled automatically when internal authentication is not enabled when using DseAuthenticator.

- 0 - disable role caching
- milliseconds - how long to cache the list of roles assigned to the user

Default: 120000 (2 minutes)

**roles_update_interval_in_ms**
Refresh interval for roles cache. After this interval, cache entries become eligible for refresh. On next access, the database schedules an async reload, and returns the old value until the reload completes. If roles_validity_in_ms is non-zero, then this value must also be non-zero. When not set, the default is the same value as roles_validity_in_ms.

Default: commented out (120000)

**permissions_validity_in_ms**
How long permissions in cache remain valid to manage performance impact of permissions queries. Fetching permissions can be resource intensive. Set the cache validity period to your security tolerances. The cache is used for the standard authentication and the row-level access control (RLAC) cache. The cache is quite effective at small durations.

- 0 - disable permissions cache
- milliseconds - time, in milliseconds

**Caution:** REVOKE does not automatically invalidate cached permissions. Permissions are invalidated the next time they are refreshed.

Default: 120000 (2 minutes)

**permissions_update_interval_in_ms**
Sets refresh interval for the standard authentication cache and the row-level access control (RLAC) cache. After this interval, cache entries become eligible for refresh. On next access, the database schedules an async reload and returns the old value until the reload completes. If permissions_validity_in_ms is non-zero, the value for roles_update_interval_in_ms must also be non-zero. When not set, the default is the same value as permissions_validity_in_ms (page 94).

Default: commented out (2000)

**permissions_cache_max_entries**
The maximum number of entries that are held by the standard authentication cache and row-level access control (RLAC) cache. With the default value of 1000,
the RLAC permissions cache can have up to 1000 entries in it, and the standard authentication cache can have up to 1000 entries. This single option applies to both caches. To size the permissions cache for use with Setting up Row Level Access Control (RLAC), use this formula:

\[ \text{numRlacUsers} \times \text{numRlacTables} + 100 \]

If this option is not present in cassandra.yaml, manually enter it to use a value other than 1000. See Enabling DSE Unified Authentication.
Default: not set (1000)

Inter-node encryption options

Node-to-node (internode) encryption protects data that is transferred between nodes in a cluster using SSL.

```yaml
server_encryption_options:
  internode_encryption: none
  keystore: resources/dse/conf/.keystore
  keystore_password: cassandra
  truststore: resources/dse/conf/.truststore
  truststore_password: cassandra
  # More advanced defaults below:
  # protocol: TLS
  # algorithm: SunX509
  # keytruststore_type: JKS
  # truststore_type: JKS
  # cipher_suites:
    # [TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA]
  # require_client_auth: false
  # require_endpoint_verification: false
```

**server_encryption_options**

Inter-node encryption options. If enabled, you must also generate keys and provide the appropriate key and truststore locations and passwords. No custom encryption options are supported.

**Tip:** The passwords used in these options must match the passwords used when generating the keytruststore and truststore. For instructions on generating these files, see Creating a Keystore to Use with JSSE.

**Tip:** See Configuring SSL for node-to-node connections.

**internode_encryption**

Encryption options for of inter-node communication using the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite for authentication, key exchange, and encryption of data transfers. Use the DHE/ECDHE ciphers, such as TLS_DHE_RSA_WITH_AES_128_CBC_SHA if running in (Federal Information Processing Standard) FIPS 140 compliant mode.

- all - Encrypt all inter-node communications
- none - No encryption
- dc - Encrypt the traffic between the datacenters (server only)
• rack - Encrypt the traffic between the racks (server only)
  Default: none

keystore
  Relative path from DSE installation directory or absolute path to the Java keystore (JKS) suitable for use with Java Secure Socket Extension (JSSE), which is the Java version of the Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols. The keystore contains the private key used to encrypt outgoing messages.
  Default: resources/dse/conf/.keystore

keystore_password
  Password for the keystore. This must match the password used when generating the keystore and truststore.
  Default: cassandra

truststore
  Relative path from DSE installation directory or absolute path to truststore containing the trusted certificate for authenticating remote servers.
  Default: resources/dse/conf/.truststore

truststore_password
  Password for the truststore.
  Default: cassandra

protocol
  Default: commented out (TLS)

algorithm
  Default: commented out (SunX509)

keystore_type
  Valid types are JKS, JCEKS, PKCS12, or PKCS11. For file-based keystores, use PKCS12.
  Default: commented out (JKS)

truststore_type
  Valid types are JKS, JCEKS, PKCS12, or PKCS11.
  Default: commented out (JKS)

cipher_suites
  Supported ciphers:
  • TLS_RSA_WITH_AES_128_CBC_SHA
  • TLS_RSA_WITH_AES_256_CBC_SHA
  • TLS_DHE_RSA_WITH_AES_128_CBC_SHA
  • TLS_DHE_RSA_WITH_AES_256_CBC_SHA
  • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
  • TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
  Default: commented out

require_client_auth
  Whether to enable certificate authentication for node-to-node (internode) encryption. When not set, the default is false.
  Default: commented out (false)

require_endpoint_verification
Whether to verify the connected host and the host name in the certificate match. When not set, the default is false. Default: commented out (false)

Client-to-node encryption options
Client-to-node encryption protects in-flight data from client machines to a database cluster using SSL (Secure Sockets Layer) and establishes a secure channel between the client and the coordinator node.

```
client_encryption_options:
  enabled: false
    # If enabled and optional is set to true, encrypted and unencrypted connections over native transport are handled.
    optional: false
    keystore: resources/dse/conf/.keystore
    keystore_password: cassandra
    # require_client_auth: false
    # Set trustore and truststore_password if require_client_auth is true
    # truststore: resources/dse/conf/.truststore
    # truststore_password: cassandra
    # More advanced defaults below:
    # protocol: TLS
    # algorithm: SunX509
    # keystore_type: JKS
    # truststore_type: JKS
    # cipher_suites:
    [TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_DHE_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_RSA_WITH_AES_256_CBC_SHA]
```

Tip: See Configuring SSL for client-to-node connections.

**client_encryption_options**
Whether to enable client-to-node encryption. You must also generate keys and provide the appropriate key and truststore locations and passwords. There are no custom encryption options enabled for DataStax Enterprise.

Advanced settings:

**enabled**
Whether to enable client-to-node encryption. Default: false

**optional**
Whether to allow unsecured connections when client encryption is enabled. Default: false

**keystore**
Relative path from DSE installation directory or absolute path to the Java keystore (JKS) suitable for use with Java Secure Socket Extension (JSSE), which is the Java version of the Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols. The keystore contains the private key used to encrypt outgoing messages.
Default: resources/dse/conf/.keystore

**keystore_password**
Configuration

Password for the keystore.
Default: *cassandra*

**require_client_auth**
Whether to enable certificate authentication for client-to-node encryption. When not set, the default is false.
Default: commented out (*false*)

**truststore**
Relative path from DSE installation directory or absolute path to truststore containing the trusted certificate for authenticating remote servers.

*Note:* Truststore password and path is only required when `require_client_auth` (*page 98*) is set to *true.*
Default: `resources/dse/conf/.truststore`

**truststore_password**
Password for the truststore. This must match the password used when generating the keystore and truststore.

*Note:* Truststore password and path is only required when `require_client_auth` (*page 98*) is set to *true.*
Default: *cassandra*

**protocol**
Default: commented out (*TLS*)

**algorithm**
Default: commented out (*SunX509*)

**keystore_type**
Valid types are JKS, JCEKS, PKCS12, or PKCS11. For file-based keystores, use PKCS12.
Default: commented out (*JKS*)

**truststore_type**
Valid types are JKS, JCEKS, PKCS12, or PKCS11.
Default: commented out (*JKS*)

**cipher_suites**
Supported ciphers:
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

Default: commented out

Transparent data encryption options

**transparent_data_encryption_options**
DataStax Enterprise supports this option only for backward compatibility. When using DSE, configure data encryption options (*page 114*) in the dse.yaml; see Transparent data encryption.
TDE properties:

- enabled: (Default: false)
- chunk_length_kb: (Default: 64)
- cipher: options:
  # AES
  # CBC
  # PKCS5Padding
- key_alias: testing:1
- iv_length: 16

  **Note:** iv_length is commented out in the default `cassandra.yaml` file. Uncomment only if cipher is set to AES. The value must be 16 (bytes).

- key_provider:
  # class_name: org.apache.cassandra.security.JKSKeyProvider
  parameters:
    # keystore: conf/.keystore
    # keystore_password: cassandra
    # store_type: JCEKS
    # key_password: cassandra

**SSL Ports**

- ssl_storage_port: 7001
- native_transport_port_ssl: 9142

**Tip:** See [Securing DataStax Enterprise ports](#).

**ssl_storage_port**

The SSL port for encrypted communication. Unused unless enabled in `encryption_options`. Follow security best practices, do not expose this port to the internet. Apply firewall rules.

Default: 7001

**native_transport_port_ssl**

Dedicated SSL port where the CQL native transport listens for clients with encrypted communication. For security reasons, do not expose this port to the internet. Firewall it if needed.

- commented out (disabled) - the native_transport_port will encrypt all traffic
- port number different than native_transport_port - use encryption for native_transport_port_ssl, keep native_transport_port unencrypted to use both unencrypted and encrypted traffic

Default: 9142
User-defined functions (UDF) properties

```plaintext
enable_user_defined_functions: false
enable_scripted_user_defined_functions: false
enable_user_defined_functions_threads: true
user_defined_function_warn_micros: 500
user_defined_function_fail_micros: 10000
user_defined_function_warn_heap_mb: 200
user_defined_function_fail_heap_mb: 500
user_function_timeout_policy: die
```

**enable_user_defined_functions**
Whether to enable user defined functions (UDFs), code that is executed inside Cassandra daemons. UDFs can present a security risk, since they are executed on the server side. UDFs are executed in a sandbox to control what code can be executed. See the DataStax blog post [User Defined Functions](https://datastax.com/blog/2016/06/user-defined-functions).

- true - Enabled. Supports Java as the code language. Detect endless loops and unintended memory leaks.
- false - Disabled.

Default: false (disabled)

**enable_scripted_user_defined_functions**
Whether to enable the use of JavaScript language in UDFs. Scripted UDFs are less performant than UDFs and produce more garbage on the heap.

- true - Enabled. Allow JavaScript in addition to Java as a code language.
- false - Disabled. Allow only Java as a code language.

**Note:** If `enable_user_defined_functions (page 100)` is false, this setting has no impact.

Default: false

**enable_user_defined_functions_threads**
Whether to enable sandbox for asynchronous JavaScript UDF execution. Does not apply to Java UDFs.

- true - Enabled. Only one instance of a function can run at one time. Asynchronous execution prevents UDFs from running too long or forever and destabilizing the cluster.
- false - Disabled. Allows multiple instances of the same function to run simultaneously.

**Caution:** Disabling asynchronous UDF execution implicitly disables the Java security manager. You must monitor the read timeouts for JavaScript UDFs that run too long or forever, which can cause the cluster to destabilize.

Default: true

**user_defined_function_warn_micros**
Threshold in microseconds (CPU time). When a UDF runs too long and this threshold is exceeded, a warning is logged and sent to the client.
Java UDFs always issue a warning. Scripted UDFs log a warning only if `enable_user_defined_functions_threads (page 100)` is set to true.
Default: 500

**user_defined_function_fail_micros**
Threshold in microseconds (CPU time). When a fatal UDF run-time situation is detected and this threshold is exceeded, the UDF is stopped. Java UDFs always throw an exception and stop. Scripted UDFs throw an exception and stop only if `enable_user_defined_functions_threads (page 100)` is set to true.
Default: 10000

**user_defined_function_warn_heap_mb**
Threshold in MB for heap allocations. When this threshold is exceeded, a warning is logged and sent to the client. Java UDFs always issue a warning. Scripted UDFs log a warning only if `enable_user_defined_functions_threads (page 100)` is set to true.
Default: 200

**user_defined_function_fail_heap_mb**
Threshold in MB for heap allocations. When this threshold is exceeded, the UDF is stopped.
- Java UDFs fail and are safely stopped. Java UDFs always throw an exception.
- Scripted UDFs stop and throw an exception only if `enable_user_defined_functions_threads (page 100)` is set to true.
Default: 500

**user_function_timeout_policy**
Defines action when a scripted UDF exceeds `user_defined_function_fail_micros (page 101)` threshold. Applies only when `enable_user_defined_functions_threads (page 100)` is set to true.

- **die** - issue a warning to the client before the Cassandra daemon shuts down
- **die_immediate** - shut down Cassandra daemon immediately, effectively prevents the client from receiving a warning
- **ignore** - log the warning, but do not take any action. DataStax does not recommend this option for production environments.
Default: die

Continuous paging options

```
continuous_paging:
  max_concurrent_sessions: 60
  max_session_pages: 4
  max_page_size_mb: 8
  max_local_query_time_ms: 5000
  client_timeout_sec: 600
  cancel_timeout_sec: 5
  paused_check_interval_ms: 1
```

**continuous_paging**
Options to tune continuous paging that pushes pages, when requested, continuously to the client:
• Maximum memory used:

\[
\text{max\_concurrent\_sessions} \times \text{max\_session\_pages} \times \text{max\_page\_size\_mb}
\]

Default: \(\text{calculated}(60 \times 4 \times 8 = 1920 \text{ MB})\)

**Guidance**

• Because memtables and SSTables are used by the continuous paging query, you can define the maximum period of time during which memtables cannot be flushed and compacted SSTables cannot be deleted.

• If fewer threads exist than sessions, a session cannot execute until another one is swapped out.

• Distributed queries (\(\text{CL} > \text{ONE}\) or non-local data) are swapped out after every page, while local queries at \(\text{CL} = \text{ONE}\) are swapped out after \(\text{max\_local\_query\_time\_ms}\).

**max\_concurrent\_sessions**

The maximum number of concurrent sessions. Additional sessions are rejected with an unavailable error.
Default: 60

**max\_session\_pages**

The maximum number of pages that can be buffered for each session. If the client is not reading from the socket, the producer thread is blocked after it has prepared \(\text{max\_session\_pages}\).
Default: 4

**max\_page\_size\_mb**

The maximum size of a page, in MB. If an individual CQL row is larger than this value, the page can be larger than this value.
Default: 8

**max\_local\_query\_time\_ms**

The maximum time for a local continuous query to run. When this threshold is exceeded, the session is swapped out and rescheduled. Swapping and rescheduling ensures the release of resources that prevent the memtables from flushing and ensures fairness when \(\text{max\_threads} < \text{max\_concurrent\_sessions}\). Adjust when high write workloads exist on tables that have continuous paging requests.
Default: 5000

**client\_timeout\_sec**

How long the server will wait, in seconds, for clients to request more pages if the client is not reading and the server queue is full.
Default: 600

**cancel\_timeout\_sec**

How long to wait before checking if a paused session can be resumed. Continuous paging sessions are paused because of backpressure or when the client has not request more pages with backpressure updates.
Default: 5

**paused\_check\_interval\_ms**
How long to wait, in milliseconds, before checking if a continuous paging sessions can be resumed, when that session is paused because of backpressure. Default: 1

Fault detection setting

```yaml
# phi_convict_threshold: 8

phi_convict_threshold
The sensitivity of the failure detector on an exponential scale. Generally, this setting does not need adjusting.

Tip: See About failure detection and recovery.
When not set, the internal value is 8. Default: commented out (8)
```

**dse.yaml configuration file**

The `dse.yaml` file is the primary configuration file for security, DSE Search, DSE Graph, and DSE Analytics.

**Important:** After changing properties in the `dse.yaml` file, you must restart the node for the changes to take effect.

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
</table>
| Tarball installations       | installation_location/
|                             | resources/dse/conf/dse.yaml |

The `cassandra.yaml (page 64)` file is the primary configuration file for the DataStax Enterprise database.

**Syntax**

For the properties in each section, the parent setting has zero spaces. Each child entry requires at least two spaces. Adhere to the YAML syntax and retain the spacing. For example, no spaces before the parent `node_health_options` entry, and at least two spaces before the child settings:

```
node_health_options:
  refresh_rate_ms: 50000
  uptime_ramp_up_period_seconds: 10800
  dropped Mutation window_minutes: 30
```

**Organization**

The DataStax Enterprise configuration properties are grouped into the following sections:

- Security and authentication options *(page 104)*
- DSE In-Memory *(page 116)*
- Node health *(page 117)*
- Health-based routing *(page 117)*
• Lease metrics *(page 118)*
• DSE Search options *(page 118)*
• DSE Analytics options *(page 130)*
• Performance Service options *(page 122)*
• DSE Metrics Collector options *(page 140)*
• Audit logging for database activities *(page 140)*
• Cassandra audit writer options *(page 140)*
• DSE Tiered Storage *(page 143)*
• DSE Advanced Replication *(page 144)*
• Inter-node messaging *(page 145)*
• DSE Multi-Instance *(page 146)*
• DSE Graph options *(page 146)*

Security and authentication options

• Authentication options *(page 104)*
• Role management options *(page 106)*
• Authorization options *(page 107)*
• Kerberos options *(page 108)*
• LDAP options *(page 109)*
• Encrypt sensitive system resources *(page 113)*
• Encrypted configuration properties settings *(page 114)*
• KMIP encryption options *(page 115)*
• DSE Search index encryption settings *(page 116)*

Authentication options

Authentication options for the DSE Authenticator that allows you to use multiple schemes for authentication in a DataStax Enterprise cluster. Additional authenticator *(page 93)* configuration is required in cassandra.yaml.

**Note:** Internal and LDAP schemes can also be used for role management, see role_management_options *(page 106).*

**Tip:** See Enabling DSE Unified Authentication.

```yaml
# authentication_options:
  # enabled: false
  # default_scheme: internal
  # other_schemes:
  #   - ldap
  #   - kerberos
  # scheme_permissions: false
  # transitional_mode: disabled
  # allow_digest_with_kerberos: true
  # plain_text_without_ssl: warn
```

authentication_options
Options for the DseAuthenticator to authenticate users when the authenticator option in cassandra.yaml is set to com.datastax.bdp.cassandra.auth.DseAuthenticator. Authenticators other than DseAuthenticator are not supported.

**enabled**

Enables user authentication.

- **true** - The DseAuthenticator authenticates users.
- **false** - The DseAuthenticator does not authenticate users and allows all connections.

When not set, the default is false.
Default: commented out `false`

**default_scheme**

Sets the first scheme to validate a user against when the driver does not request a specific scheme.

- **internal** - Plain text authentication using the internal password authentication.
- **ldap** - Plain text authentication using pass-through LDAP authentication.
- **kerberos** - GSSAPI authentication using the Kerberos authenticator.

Default: commented out (`internal`)

**other_schemes**

List of schemes that are also checked if validation against the first scheme fails and no scheme was specified by the driver. Same scheme names as `default_scheme`.

**scheme_permissions**

Whether roles need to have permission granted to them in order to use specific authentication schemes. These permissions can be granted only when the DseAuthorizer is used. Set to one of the following values:

- **true** - Use multiple schemes for authentication. Every role requires permissions to a scheme in order to be assigned.
- **false** - Do not use multiple schemes for authentication. Prevents unintentional role assignment that might occur if user or group names overlap in the authentication service.

**Tip:** See [Binding a role to an authentication scheme](#).

When not set, the default is false.
Default: commented out (`false`)

**allow_digest_with_kerberos**

Controls whether DIGEST-MD5 authentication is also allowed with Kerberos. The DIGEST-MD5 mechanism is not directly associated with an authentication scheme, but is used by Kerberos to pass credentials between nodes and jobs.

- **true** - DIGEST-MD5 authentication is also allowed with Kerberos. In analytics clusters, set to true to use Hadoop inter-node authentication with Hadoop and Spark jobs.
- **false** - DIGEST-MD5 authentication is not used with Kerberos.
Analytics nodes require true to use internode authentication with Hadoop and Spark jobs. When not set, the default is true. Default: commented out (true)

**plain_text_without_ssl**
Controls how the DseAuthenticator responds to plain text authentication requests over unencrypted client connections. Set to one of the following values:
- block - Block the request with an authentication error.
- warn - Log a warning about the request but allow it to continue.
- allow - Allow the request without any warning.
Default: commented out (warn)

**transitional_mode**
Whether to enable transitional mode for temporary use during authentication setup in an already established environment.

Transitional mode allows access to the database using the anonymous role, which has all permissions except AUTHORIZE.
- disabled - Transitional mode is disabled. All connections must provide valid credentials and map to a login-enabled role.
- permissive - Only super users are authenticated and logged in. All other authentication attempts are logged in as the anonymous user.
- normal - Allow all connections that provide credentials. Maps all authenticated users to their role AND maps all other connections to anonymous.
- strict - Allow only authenticated connections that map to a login-enabled role OR connections that provide a blank username and password as anonymous.

Important: Credentials are required for all connections after authentication is enabled; use a blank username and password to login with anonymous role in transitional mode.
Default: commented out (disabled)

**Role management options**

```
#role_management_options:
#  mode: internal
```

Tip: See Enabling DSE Unified Authentication.

**role_management_options**
Options for the DSE Role Manager. To enable role manager, set:
- **authorization_options (page 104)** enabled to true
- **role_manager (page 93)** in cassandra.yaml to com.datastax.bdp.cassandra.auth.DseRoleManager

Tip: See Setting up logins and users.
When `scheme_permissions (page 105)` is enabled, all roles must have permission to execute on the authentication scheme, see Binding a role to an authentication scheme.

**mode**

Set to one of the following values:

- **internal** - Scheme that manages roles per individual user in the internal database. Allows nesting roles for permission management.
- **ldap** - Scheme that assigns roles by looking up the user name in LDAP and mapping the group attribute (`ldap_options (page 109)`) to an internal role name. To configure an LDAP scheme, complete the steps in Defining an LDAP scheme.

**Attention:** Internal role management allows nesting roles for permission management; when using LDAP mode role, nesting is disabled. Using `GRANT role_name TO role_name` results in an error.

Default: commented out (internal)

**Authorization options**

```yaml
#authorization_options:
#    enabled: false
#    transitional_mode: disabled
#    allow_row_level_security: false
```

**Tip:** See Enabling DSE Unified Authentication.

**authorization_options**

Options for the DSE Authorizer.

**enabled**

Whether to use the DSE Authorizer for role-based access control (RBAC).

- **true** - use the DSE Authorizer for role-based access control (RBAC)
- **false** - do not use the DSE Authorizer

When not set, the default is false.

Default: commented out (false)

**transitional_mode**

Allows the DSE Authorizer to operate in a temporary transitional mode during setup of authorization in a cluster. Set to one of the following values:

- **disabled** - Transitional mode is disabled.
- **normal** - Permissions can be passed to resources, but are not enforced.
- **strict** - Permissions can be passed to resources, and are enforced on authenticated users. Permissions are not enforced against anonymous users.

Default: commented out (disabled)

**allow_row_level_security**

Whether to enable row-level access control (RLAC) permissions; use the same setting on all nodes.
• true - use row-level security
• false - do not use row-level

When not set, the default is false.
Default: commented out (false)

Kerberos options

```
kerberos_options:
  keytab: resources/dse/conf/dse.keytab
  service_principal: dse/_HOST@REALM
  http_principal: HTTP/_HOST@REALM
  qop: auth
```

Tip: See Defining a Kerberos scheme.

kerberos_options
Options to configure security for a DataStax Enterprise cluster using Kerberos.

keytab
The file path of `dse.keytab`.

service_principal
The service_principal that the DataStax Enterprise process runs under must use the form `dse_user/_HOST@REALM`, where:

• dse_user is the name of the user that starts the DataStax Enterprise process.
• _HOST is converted to a reverse DNS lookup of the broadcast address.
• REALM is the name of your Kerberos realm. In the Kerberos principal, REALM must be uppercase.

http_principal
The http_principal is used by the Tomcat application container to run DSE Search. The Tomcat web server uses the GSSAPI mechanism (SPNEGO) to negotiate the GSSAPI security mechanism (Kerberos). Set REALM to the name of your Kerberos realm. In the Kerberos principal, REALM must be uppercase.

qop
A comma-delimited list of Quality of Protection (QOP) values that clients and servers can use for each connection. The client can have multiple QOP values, while the server can have only a single QOP value. The valid values are:

• auth - Authentication only.
• auth-int - Authentication plus integrity protection for all transmitted data.
• auth-conf - Authentication plus integrity protection and encryption of all transmitted data.

Encryption using auth-conf is separate and independent of whether encryption is done using SSL. If both auth-conf and SSL are enabled, the transmitted data is encrypted twice. DataStax recommends choosing only one method and using it for both encryption and authentication.
LDAP options

Define LDAP options to authenticate users against an external LDAP service and/or for Role Management using LDAP group look up.

**Tip:** See Enabling DSE Unified Authentication.

```yaml
# ldap_options:
#     server_host:
#     server_port: 389
#     search_dn:
#     search_password:
#     use_ssl: false
#     use_tls: false
#     truststore_path:
#     truststore_password:
#     truststore_type: jks
#     user_search_base:
#     user_search_filter: (uid={0})
#     user_memberof_attribute: memberof
#     group_search_type: directory_search
#     group_search_base:
#     group_search_filter: (uniquemember={0})
#     group_name_attribute: cn
#     credentials_validity_in_ms: 0
#     search_validity_in_seconds: 0
#     connection_pool:
#         max_active: 8
#         max_idle: 8
```

Microsoft Active Directory (AD) example, for both authentication and role management:

```bash
ldap_options:
    server_host: win2012ad_server.mycompany.lan
    server_port: 389
    search_dn: cn=lookup_user,cn=users,dc=win2012domain,dc=mycompany,dc=lan
    search_password: lookup_user_password
    use_ssl: false
    use_tls: false
    truststore_path:
    truststore_password:
    truststore_type: jks
    #group_search_type: directory_search
    #group_search_base:
    #group_search_filter:
    #group_name_attribute: cn
    user_search_base: cn=users,dc=win2012domain,dc=mycompany,dc=lan
    user_search_filter: (sAMAccountName={0})
    user_memberof_attribute: memberOf
    connection_pool:
        max_active: 8
```
max_idle: 8

Tip: See Defining an LDAP scheme.

ldap_options
Options to configure LDAP security. When not set, LDAP authentication is not used.
Default: commented out

server_host
The host name of the LDAP server.

Important: LDAP on the same host (localhost) is appropriate only in single node test or development environments.

Default: commented out

server_port
The port on which the LDAP server listens.

- 389 - the default port for unencrypted connections
- 636 - typically used for encrypted connections; the default SSL port for LDAP is 636

Default: commented out (389)

search_dn
Distinguished name (DN) of an account with read access to the user_search_base and group_search_base. For example:

- OpenLDAP: uid=lookup,ou=users,dc=springsource,dc=com
- Microsoft Active Directory (AD): cn=lookup, cn=users, dc=springsource, dc=com

Warning: Do not create/use an LDAP account or group called cassandra. The DSE database comes with a default login role, cassandra, that has access to all database objects and uses the consistency level QUOROM.

When not set, an anonymous bind is used for the search on the LDAP server.
Default: commented out

search_password
The password of the search_dn account.
Default: commented out

use_ssl
Whether to use an SSL-encrypted connection.

- true - use an SSL-encrypted connection, set server_port (page 110) to the LDAP port for the server (typically port 636)
- false - do not enable SSL connections to the LDAP server

Default: commented out (false)

use_tls
Whether to enable TLS connections to the LDAP server.
true - enable TLS connections to the LDAP server, set \texttt{server\_port (page 110)} to the TLS port of the LDAP server.

false - do not enable TLS connections to the LDAP server

Default: commented out (false)

\texttt{truststore\_path}

The path to the truststore for SSL certificates.

Default: commented out

\texttt{truststore\_password}

The password to access the trust store.

Default: commented out

\texttt{truststore\_type}

The type of truststore.

Default: commented out (jks)

\texttt{user\_search\_base}

Distinguished name (DN) of the object to start the recursive search for user entries for authentication and role management memberof searches. For example to search all users in example.com, \texttt{ou=users,dc=example,dc=com}.

- For your LDAP domain, set the \texttt{ou} and \texttt{dc} elements. Typically set to \texttt{ou=users,dc=domain,dc=top\_level\_domain}. For example, \texttt{ou=users,dc=example,dc=com}.
- Active Directory uses a different search base, typically \texttt{CN=search,CN=Users,DC=ActDir\_domname,DC=internal}. For example, \texttt{CN=search,CN=Users,DC=example-sales,DC=internal}.

Default: commented out

\texttt{user\_search\_filter}

Attribute that identifies the user that the search filter uses for looking up user names.

- \texttt{uid={0}} - when using LDAP
- \texttt{samAccountName={0}} - when using AD (Microsoft Active Directory). For example, \texttt{(sAMAccountName={0})}

Default: commented out (uid={0})

\texttt{user\_memberof\_attribute}

Attribute that contains a list of group names; role manager assigns DSE roles that exactly match any group name in the list. Required when managing roles using \texttt{group\_search\_type: memberof\_search} with LDAP (\texttt{role\_manager.mode:ldap (page 106)}). The directory server must have memberof support, which is a default user attribute in Microsoft Active Directory (AD).

Default: commented out (memberof)

\texttt{group\_search\_type}

Required when managing roles with LDAP (\texttt{role\_manager.mode: ldap (page 106)}). Define how group membership is determined for a user. Choose from one of the following values:
• directory_search - Filters the results by doing a subtree search of 
group_search_base (page 112) to find groups that contain the user name in 
the attribute defined in the group_search_filter (page 112). (Default)

• memberof_search - Recursively search for user entries using the 
user_search_base and user_search_filter. Get groups from the user 
attribute defined in user_memberof_attribute. The directory server must 
have memberof support.

Default: commented out (directory_search)

**group_search_base**

The unique distinguished name (DN) of the group record from which to start the 
group membership search on.
Default: commented out

**group_search_filter**

Set to any valid LDAP filter.
Default: commented out (uniquemember={0})

**group_name_attribute**

The attribute in the group record that contains the LDAP group name. Role names 
are case-sensitive and must match exactly on DSE for assignment. Unmatched 
groups are ignored.
Default: commented out (cn)

**credentials_validity_in_ms**

The duration period of the credentials cache.

- 0 - disable credentials cache
- duration period in milliseconds - enable a search cache and improve 
performance by reducing the number of requests that are sent to the LDAP 
server

When not set, the default is 0 (disabled).
Default: commented out (0)

**search_validity_in_seconds**

The duration period for the search cache.

- 0 - disable search credentials cache
- duration period in seconds - enables a search cache and improves 
performance by reducing the number of requests that are sent to the LDAP 
server

Default: commented out (0, disabled)

**connection_pool**

The configuration settings for the connection pool for making LDAP requests.

**max_active**

The maximum number of active connections to the LDAP server.
Default: commented out (8)

**max_idle**

The maximum number of idle connections in the pool awaiting requests.
Default: commented out (8)
Encrypt sensitive system resources

Options to encrypt sensitive system resources using a local encryption key or a remote KMIP key.

```yaml
system_info_encryption:
  enabled: false
  cipher_algorithm: AES
  secret_key_strength: 128
  chunk_length_kb: 64
  key_provider: KmipKeyProviderFactory
  kmip_host: kmip_host_name
```

**Note:** DataStax recommends using a remote encryption key from a KMIP provider when using Transparent Data Encryption (TDE) features. Use a local encryption key only if a KMIP server is not available.

**system_info_encryption**
Options to set encryption settings for system resources that might contain sensitive information, including the `system.batchlog` and `system.paxos` tables, hint files, and the database commit log.

**enabled**
Whether to enable encryption of system resources. See Encrypting system resources.

**Note:** The `system_trace` keyspace is NOT encrypted by enabling the `system_information_encryption` section. In environments that also have tracing enabled, manually configure encryption with compression on the `system_trace` keyspace. See Transparent data encryption.

Default: `false`

**cipher_algorithm**
The name of the JCE cipher algorithm used to encrypt system resources.

**Table 4: Supported cipher algorithms names**

<table>
<thead>
<tr>
<th>cipher_algorithm</th>
<th>secret_key_strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>128, 192, or 256</td>
</tr>
<tr>
<td>DES</td>
<td>56</td>
</tr>
<tr>
<td>DESede</td>
<td>112 or 168</td>
</tr>
<tr>
<td>Blowfish</td>
<td>32-448</td>
</tr>
<tr>
<td>RC2</td>
<td>40-128</td>
</tr>
</tbody>
</table>

Default: AES

**secret_key_strength**
Length of key to use for the system resources. See Table 1 (page 113).
Note: DSE uses a matching local key or requests the key type from the KMIP server. For KMIP, if an existing key does not match, the KMIP server automatically generates a new key.

Default: 128

chunk_length_kb
Optional. Size of SSTable chunks when data from the system.batchlog or system.paxos are written to disk.

Note: To encrypt existing data, run `nodetool upgradesstables (page 1140)` -a system batchlog paxos on all nodes in the cluster.

Default: 64

key_provider
KMIP key provider to enable encrypting sensitive system data with a KMIP key. Comment out if using a local encryption key.

Default: commented out (KmipKeyProviderFactory)

kmip_host
The KMIP key server host. Set to the `kmip_group_name` that defines the KMIP host in `kmip_hosts (page 115)` section. DSE requests a key from the KMIP host and uses the key generated by the KMIP provider.

Default: commented out

Encrypted configuration properties settings
Settings for using encrypted passwords in sensitive configuration file properties.

```
system_key_directory: /etc/dse/conf
cfg_encryption_active: false
cfg_encryption_key_name: (key_filename | KMIP_key_URL )
```

system_key_directory
Path to the directory where local encryption/decryption key files are stored, also called system keys. Distribute the system keys to all nodes in the cluster. Ensure that the DSE account is the folder owner and has read/write (600) permissions.

See Setting up local encryption keys.

Note: This directory is not used for KMIP keys.

Default: /etc/dse/conf

config_encryption_active
Whether to enable decryption of configuration property values using the specified `config_encryption_key_name (page 114)`. When set to true, the configuration values must be encrypted or commented out. See Encrypting configuration file properties

Default: false

config_encryption_key_name
Set to the local encryption key filename or KMIP key URL to use for configuration file property value decryption.
**Note:** Use `dsetool dsetool encryptconfigvalue (page 1322)` to generate encrypted values for the configuration file properties.

Default: system_key. The default name is not configurable.

### KMIP encryption options

Options for KMIP encryption keys and communication between the DataStax Enterprise node and the KMIP key server or key servers. Enables DataStax Enterprise encryption features to use encryption keys that stored on a server that is not running DataStax Enterprise.

```yaml
kmip_hosts:
  your_kmip_groupname:
    hosts: kmip1.yourdomain.com, kmip2.yourdomain.com
    keystore_path: path/to/kmip/keystore.jks
    keystore_type: jks
    keystore_password: password
    truststore_path: path/to/kmip/truststore.jks
    truststore_type: jks
    truststore_password: password
```

**kmip_hosts**

Connection settings for key servers that support the KMIP protocol.

**kmip_groupname**

The unique name of the KMIP host/cluster that is specified in the table schema. A user-defined name for a group of options to configure a KMIP server or servers, key settings, and certificates. Configure options for a `kmip_groupname` section for each KMIP key server or group of KMIP key servers. Using separate key server configuration settings allows use of different key servers to encrypt table data, and eliminates the need to enter key server configuration information in DDL statements and other configurations. Multiple KMIP hosts are supported.

Default: commented out

**hosts**

A comma-separated list KMIP hosts (`host[:port]`) using the FQDN (Fully Qualified Domain Name). DSE queries the host in the listed order, so add KMIP hosts in the intended failover sequence.

For example, if the host list contains `kmip1.yourdomain.com, kmip2.yourdomain.com`, DSE tries `kmip1.yourdomain.com` and then `kmip2.yourdomain.com`.

**keystore_path**

The path to a Java keystore created from the KMIP agent PEM files.

Default: commented out (`/etc/dse/conf/KMIP_keystore.jks`)

**keystore_type**

The type of keystore.

Default: commented out (`jks`)

**keystore_password**

The password to access the keystore.

Default: commented out (`password`)
truststore_path
The path to a Java truststore that was created using the KMIP root certificate.
Default: commented out (/etc/dse/conf/KMIP_truststore.jks)

truststore_type
The type of truststore.
Default: commented out (jks)

truststore_password
The password to access the truststore.
Default: commented out (password)

key_cache_millis
Milliseconds to locally cache the encryption keys that are read from the KMIP hosts. The longer the encryption keys are cached, the fewer requests are made to the KMIP key server, but the longer it takes for changes, like revocation, to propagate to the DataStax Enterprise node. DataStax Enterprise uses concurrent encryption, so multiple threads fetch the secret key from the KMIP key server at the same time. DataStax recommends using the default value.
Default: commented out (300000)

timeout
Socket timeout in milliseconds.
Default: commented out (1000)

DSE Search index encryption settings

# solr_encryption_options:
#    decryption_cache_offheap_allocation: true
#    decryption_cache_size_in_mb: 256

solr_encryption_options
Settings to tune encryption of search indexes.

decryption_cache_offheap_allocation
Whether to allocate shared DSE Search decryption cache off JVM heap.
- true - allocate shared DSE Search decryption cache off JVM heap
- false - do not allocate shared DSE Search decryption cache off JVM heap
When not set, the default is true.
Default: commented out (true)

decryption_cache_size_in_mb
The maximum size of shared DSE Search decryption cache in megabytes (MB).
Default: commented out (256)

DSE In-Memory options

To use the DSE In-Memory, choose one of these options to specify how much system memory to use for all in-memory tables: fraction or size.

# max_memory_to_lock_fraction: 0.20
# max_memory_to_lock_mb: 10240

max_memory_to_lock_fraction
A fraction of the system memory. The default value of 0.20 specifies to use up to 20% of system memory. This `max_memory_to_lock_fraction` value is ignored if `max_memory_to_lock_mb` is set to a non-zero value. To specify a fraction, use instead of `max_memory_to_lock_mb`.
Default: commented out (0.20)

**max_memory_to_lock_mb**
A maximum amount of memory in megabytes (MB).
- not set - use the fraction specified with `max_memory_to_lock_fraction`
- number greater than 0 - maximum amount of memory in megabytes (MB)
Default: commented out (10240)

Node health options

```yaml
node_health_options:
  refresh_rate_ms: 50000
  uptime_ramp_up_period_seconds: 10800
  dropped_mutation_window_minutes: 30
```

**node_health_options**
Node health options are always enabled.

**refresh_rate_ms**
Default: 60000

**uptime_ramp_up_period_seconds**
The amount of continuous uptime required for the node's uptime score to advance the node health score from 0 to 1 (full health), assuming there are no recent dropped mutations. The health score is a composite score based on dropped mutations and uptime.

**Tip:** If a node is repairing after a period of downtime, you might want to increase the uptime period to the expected repair time.
Default: commented out (10800 3 hours)

**dropped_mutation_window_minutes**
The historic time window over which the rate of dropped mutations affect the node health score.
Default: 30

Health-based routing

```yaml
enable_health_based_routing: true
```

**enable_health_based_routing**
Whether to consider node health for replication selection for distributed DSE Search queries. Health-based routing enables a trade-off between index consistency and query throughput.
- true - consider node health when multiple candidates exist for a particular token range.
- false - ignore node health for replication selection. When the primary concern is performance, do not enable health-based routing.
Lease metrics

```
lease_metrics_options:
  enabled: false
  ttl_seconds: 604800
```

**lease_metrics_options**
Lease holder statistics help monitor the lease subsystem for automatic management *(page 225)* of Job Tracker and Spark Master nodes.

**enabled**
Enables (true) or disables (false) log entries related to lease holders. Most of the time you do not want to enable logging.
Default: false

**ttl_seconds**
Defines the time, in milliseconds, to persist the log of lease holder changes. Logging of lease holder changes is always on, and has a very low overhead.
Default: 604800

DSE Search options

- Scheduler settings for DSE Search indexes *(page 118)*
- async_bootstrap_reindex *(page 119)*
- CQL Solr paging *(page 119)*
- Solr CQL query option *(page 119)*
- DSE Search resource upload limit *(page 120)*
- Shard transport options *(page 120)*
- DSE Search indexing settings *(page 120)*

**Scheduler settings for DSE Search indexes**
To ensure that records with TTLs are purged from search indexes when they expire, the search indexes are periodically checked for expired documents.

```
ttl_index_rebuild_options:
  fixed_rate_period: 300
  initial_delay: 20
  max_docs_per_batch: 4096
  thread_pool_size: 1
```

**ttl_index_rebuild_options**
Section of options to control the schedulers in charge of querying for and removing expired records, and the execution of the checks.

**fixed_rate_period**
Time interval to check for expired data in seconds.
Default: 300

**initial_delay**
The number of seconds to delay the first TTL check to speed up start-up time.
Default: 20

**max_docs_per_batch**
The maximum number of documents to check and delete per batch by the TTL rebuild thread. All documents determined to be expired are deleted from the index during each check, to avoid memory pressure, their unique keys are retrieved and deletes issued in batches.
Default: 4096

**thread_pool_size**
The maximum number of cores that can execute TTL cleanup concurrently. Set the thread_pool_size to manage system resource consumption and prevent many search cores from executing simultaneous TTL deletes.
Default: 1

---

**Reindexing of bootstrapped data**

**async_bootstrap_reindex**
For DSE Search, configure whether to asynchronously reindex bootstrapped data. Default: false

- If enabled, the node joins the ring immediately after bootstrap and reindexing occurs asynchronously. Do not wait for post-bootstrap reindexing so that the node is not marked down.
- If disabled, the node joins the ring after reindexing the bootstrapped data.

---

**CQL Solr paging**
Options to specify the paging behavior.

**cql_solr_query_paging**

- **driver** - Respects driver paging settings. Specifies to use Solr pagination (cursors) only when the driver uses pagination. Enabled automatically for DSE SearchAnalytics workloads.
- **off** - Paging is off. Ignore driver paging settings for CQL queries and use normal Solr paging unless:
  
  ```
  # The current workload is an analytics workload, including SearchAnalytics. SearchAnalytics nodes always use driver paging settings.
  # The cqlsh query parameter paging is set to driver.
  ```

  Even when cql_solr_query_paging: off, paging is dynamically enabled with the "paging" :"driver" parameter in JSON queries.

When not set, the default is off.
Default: commented out (off)

---

**Solr CQL query option**
Available option for CQL Solr queries.
cql_solr_query_row_timeout: 10000

**cql_solr_query_row_timeout**

The maximum time in milliseconds to wait for each row to be read from the database during CQL Solr queries.

Default: commented out (10000 10 seconds)

### DSE Search resource upload limit

**solr_resource_upload_limit_mb: 10**

**solr_resource_upload_limit_mb**

Option to disable or configure the maximum file size of the search index config or schema. Resource files can be uploaded, but the search index config and schema are stored internally in the database after upload.

- 0 - disable resource uploading
- upload size - The maximum upload size limit in megabytes (MB) for a DSE Search resource file (search index config or schema).

Default: 10

### Shard transport options

**shard_transport_options:

netty_client_request_timeout: 60000**

**shard_transport_options**

Fault tolerance option for inter-node communication between DSE Search nodes.

**netty_client_request_timeout**

Timeout behavior during distributed queries. The internal timeout for all search queries to prevent long running queries. The client request timeout is the maximum cumulative time (in milliseconds) that a distributed search request will wait idly for shard responses.

Default: 60000 (1 minute)

### DSE Search indexing settings

```bash
# back_pressure_threshold_per_core: 1024
# flush_max_time_per_core: 5
# load_max_time_per_core: 5
# enable_index_disk_failure_policy: false
# solr_data_dir: /MyDir
# solr_field_cache_enabled: false
# ram_buffer_heap_space_in_mb: 1024
# ram_buffer_offheap_space_in_mb: 1024
```

**Tip:** See Configuring and tuning indexing performance.

**back_pressure_threshold_per_core**
The maximum number of queued partitions during search index rebuilding and
reindexing. This maximum number safeguards against excessive heap use by the
indexing queue. If set lower than the number of threads per core (TPC), not all TPC
threads can be actively indexing.
Default: commented out (1024)

flush_max_time_per_core
The maximum time, in minutes, to wait for the flushing of asynchronous index
updates that occurs at DSE Search commit time or at flush time. Expert level
knowledge is required to change this value. Always set the value reasonably high to
ensure flushing completes successfully to fully sync DSE Search indexes with the
database data. If the configured value is exceeded, index updates are only partially
committed and the commit log is not truncated which can undermine data durability.

Note: When a timeout occurs, it usually means this node is being
overloaded and cannot flush in a timely manner. Live indexing increases the
time to flush asynchronous index updates.
Default: commented out (5)

load_max_time_per_core
The maximum time, in minutes, to wait for each DSE Search index to load on
startup or create/reload operations. This advanced option should be changed only
if exceptions happen during search index loading. When not set, the default is 5
minutes.
Default: commented out (5)

enable_index_disk_failure_policy
Whether to apply the configured disk failure policy if IOExceptions occur during
index update operations.
• true - apply the configured Cassandra disk failure policy to index write failures
• false - do not apply the disk failure policy

When not set, the default is false.
Default: commented out (false)

solr_data_dir
The directory to store index data. By default, each DSE Search index is saved in
solrconfig_data_dir/keyspace_name.table_name, or as specified by the
dse.solr.data.dir system property.

Tip: See Managing the location of DSE Search data.
Default: commented out (/MyDir)

solr_field_cache_enabled
The Apache Lucene® field cache is deprecated. Instead, for fields that are sorted,
faceted, or grouped by, set docValues="true" on the field in the search index
schema. Then reload the search index and reindex. When not set, the default is
false.
Default: commented out (false)

ram_buffer_heap_space_in_mb
Global Lucene RAM buffer usage threshold for heap to force segment flush. Setting
too low might induce a state of constant flushing during periods of ongoing write
activity. For NRT, forced segment flushes also de-schedule pending auto-soft
commit to avoid potentially flushing too many small segments. When not set, the
default is 1024.
Default: commented out (1024)

**ram_buffer_offheap_space_in_mb**
Global Lucene RAM buffer usage threshold for offheap to force segment flush.
Setting too low might induce a state of constant flushing during periods of ongoing
write activity. For NRT, forced segment flushes also de-schedule pending auto-soft
commit to avoid potentially flushing too many small segments. When not set, the
default is 1024.
Default: commented out (1024)

Performance Service options

- Global Performance Service options (page 122)
- Performance Service options (page 122)
- DSE Search Performance Service options (page 126)
- Spark Performance Service options (page 129)

Global Performance Service options

Available options to configure the thread pool that is used by most plug-ins. A dropped
task warning is issued when the performance service requests more tasks than
performance_max_threads + performance_queue_capacity. When a task is dropped,
collected statistics might not be current.

```
# performance_core_threads: 4
# performance_max_threads: 32
# performance_queue_capacity: 32000
```

**performance_core_threads**
Number of background threads used by the performance service under normal
conditions. Default: 4

**performance_max_threads**
Maximum number of background threads used by the performance service.

**performance_queue_capacity**
The number of queued tasks in the backlog when the number of
performance_max_threads are busy. Default: 32000

Performance Service options

These settings are used by the Performance Service to configure collection of performance
metrics on transactional nodes. Performance metrics are stored in the dse_perf keyspace
and can be queried with CQL using any CQL-based utility, such as cqlsh or any application
using a CQL driver. To temporarily make changes for diagnostics and testing, use the
dsetool perf (page 1353) subcommands.

**Tip:** See Collecting system level diagnostics.

**graph_events**
Graph event information.
graph_events:
  ttl_seconds: 600

ttl_seconds
The TTL in milliseconds.
Default: 600

cql_slow_log_options
Options to configure reporting distributed sub-queries for search (query executions on individual shards) that take longer than a specified period of time.

# cql_slow_log_options:
#   enabled: true
#   threshold: 200.0
#   minimum_samples: 100
#   ttl_seconds: 259200
#   skip_writing_to_db: true
#   num_slowest_queries: 5

Tip: See Collecting slow queries.

enabled
Enables (true) or disables (false) log entries for slow queries. When not set, the default is true.
Default: commented out (true)

threshold
The threshold in milliseconds or as a percentile.

- A value greater than 1 is expressed in time and will log queries that take longer than the specified number of milliseconds.
- A value of 0 to 1 is expressed as a percentile and will log queries that exceed this percentile.

Default: commented out (200.0 0.2 seconds)

minimum_samples
The initial number of queries before activating the percentile filter.
Default: commented out (100)

ttl_seconds
Time, in milliseconds, to keep the slow query log entries.
Default: commented out (259200)

skip_writing_to_db
Whether to keep slow queries in-memory only and not write data to database.
- false - write slow queries to the database; the threshold must be >= 2000 ms to prevent a high load on the database
- true - skip writing to database, keep slow queries only in memory

Default: commented out (true)

num_slowest_queries
The number of slow queries to keep in-memory.
Default: commented out (5)
cql_system_info_options
Options to configure collection of system-wide performance information about a cluster.

cql_system_info_options:
  enabled: false
  refresh_rate_ms: 10000

enabled
Whether to collect system-wide performance information about a cluster.
• false - do not collect metrics
• true - enable collection of metrics
Default: false
refresh_rate_ms
The length of the sampling period in milliseconds; the frequency to update the statistics.
Default: 10000 (10 seconds)

resource_level_latency_tracking_options
Options to configure collection of object I/O performance statistics.

resource_level_latency_tracking_options:
  enabled: false
  refresh_rate_ms: 10000

Tip: See Collecting system level diagnostics.

enabled
Whether to collect object I/O performance statistics.
• false - do not collect metrics
• true - enable collection of metrics
Default: false
refresh_rate_ms
The length of the sampling period in milliseconds; the frequency to update the statistics.
Default: 10000 (10 seconds)

db_summary_stats_options
Options to configure collection of summary statistics at the database level.

db_summary_stats_options:
  enabled: false
  refresh_rate_ms: 10000

Tip: See Collecting database summary diagnostics.

enabled
Whether to collect database summary performance information.
• false - do not collect metrics
• true - enable collection of metrics
Default: false

**refresh_rate_ms**
The length of the sampling period in milliseconds; the frequency to update the statistics.
Default: 10000 (10 seconds)

**cluster_summary_stats_options**
Options to configure collection of statistics at a cluster-wide level.

```
cluster_summary_stats_options:
    enabled: false
    refresh_rate_ms: 10000
```

**Tip:** See Collecting cluster summary diagnostics.

**enabled**
Whether to collect statistics at a cluster-wide level.
• false - do not collect metrics
• true - enable collection of metrics
Default: false

**refresh_rate_ms**
The length of the sampling period in milliseconds; the frequency to update the statistics.
Default: 10000 (10 seconds)

**spark_cluster_info_options**
Options to configure collection of data associated with Spark cluster and Spark applications.

```
spark_cluster_info_options:
    enabled: false
    refresh_rate_ms: 10000
```

**Tip:** See Monitoring Spark with Spark Performance Objects.

**enabled**
Whether to collect Spark performance statistics.
• false - do not collect metrics
• true - enable collection of metrics
Default: false

**refresh_rate_ms**
The length of the sampling period in milliseconds; the frequency to update the statistics.
Default: 10000 (10 seconds)

**histogram_data_options**
Histogram data for the dropped mutation metrics are stored in the dropped_messages table in the dse_perf keyspace.

```
histogram_data_options:
```
Configuration

```yaml
enabled: false
refresh_rate_ms: 10000
retention_count: 3
```

**Tip:** See [Collecting histogram diagnostics](#).

### enabled

- false - do not collect metrics
- true - enable collection of metrics

Default: false

### refresh_rate_ms

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 10000 (10 seconds)

### retention_count

Default: 3

### user_level_latency_tracking_options

User-resource latency tracking settings.

```yaml
user_level_latency_tracking_options:
  enabled: false
  refresh_rate_ms: 10000
  top_stats_limit: 100
  quantiles: false
```

**Tip:** See [Collecting user activity diagnostics](#).

### enabled

- false - do not collect metrics
- true - enable collection of metrics

Default: false

### refresh_rate_ms

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 10000 (10 seconds)

### top_stats_limit

Limit the number of individual metrics.

Default: 100

### quantiles

Default: false

---

**DSE Search Performance Service options**

These settings are used by the [DataStax Enterprise Performance Service](#).

```yaml
solr_slow_sub_query_log_options:
  enabled: false
  ttl_seconds: 604800
  threshold_ms: 3000
```
### async_writers
- Value: 1

### solr_update_handler_metrics_options
- **enabled**: false
- **ttl_seconds**: 604800
- **refresh_rate_ms**: 60000

### solr_request_handler_metrics_options
- **enabled**: false
- **ttl_seconds**: 604800
- **refresh_rate_ms**: 60000

### solr_index_stats_options
- **enabled**: false
- **ttl_seconds**: 604800
- **refresh_rate_ms**: 60000

### solr_cache_stats_options
- **enabled**: false
- **ttl_seconds**: 604800
- **refresh_rate_ms**: 60000

### solr_latency_snapshot_options
- **enabled**: false
- **ttl_seconds**: 604800
- **refresh_rate_ms**: 60000

---

**solr_slow_sub_query_log_options**

See Collecting slow search queries.

**enabled**
- false - do not collect metrics
- true - enable collection of metrics

**Default**: false

**ttl_seconds**
The length of the sampling period in milliseconds; the frequency to update the statistics.

**Default**: 604800 (about 10 minutes)

**async_writers**
The number of server threads dedicated to writing in the log. More than one server thread might degrade performance.

**Default**: 1

**threshold_ms**

**Default**: 3000

**solr_update_handler_metrics_options**
Options to collect search index direct update handler statistics over time.

**Tip**: See Collecting handler statistics.

**enabled**
false - do not collect metrics
• true - enable collection of metrics

Default: false

ttl_seconds
The length of the sampling period in milliseconds; the frequency to update the
statistics.
Default: 604800 (about 10 minutes)

refresh_rate_ms
The length of the sampling period in milliseconds; the frequency to update the
statistics.
Default: 60000 (1 minute)

solr_index_stats_options
Options to record search index statistics over time.

Tip: See Collecting index statistics.

enabled

• false - do not collect metrics
• true - enable collection of metrics

Default: false

ttl_seconds
The length of the sampling period in milliseconds; the frequency to update the
statistics.
Default: 604800 (about 10 minutes)

refresh_rate_ms
The length of the sampling period in milliseconds; the frequency to update the
statistics.
Default: 60000 (1 minute)

solr_cache_stats_options
See Collecting cache statistics.

enabled

• false - do not collect metrics
• true - enable collection of metrics

Default: false

ttl_seconds
The length of the sampling period in milliseconds; the frequency to update the
statistics.
Default: 604800 (about 10 minutes)

refresh_rate_ms
The length of the sampling period in milliseconds; the frequency to update the
statistics.
Default: 60000 (1 minute)

solr_latency_snapshot_options
See Collecting Apache Solr performance statistics.

enabled
• false - do not collect metrics
• true - enable collection of metrics

Default: false

ttl_seconds
The length of the sampling period in milliseconds; the frequency to update the statistics.
Default: 604800 (about 10 minutes)

refresh_rate_ms
The length of the sampling period in milliseconds; the frequency to update the statistics.
Default: 60000 (1 minute)

Spark Performance Service options

Tip: See Monitoring Spark application information.

```yaml
spark_application_info_options:
  enabled: false
  refresh_rate_ms: 10000
  driver:
    sink: false
    connectorSource: false
    jvmSource: false
    stateSource: false
  executor:
    sink: false
    connectorSource: false
    jvmSource: false

spark_application_info_options
Statistics options.

enabled
• false - do not collect metrics
• true - enable collection of metrics

Default: false

refresh_rate_ms
The length of the sampling period in milliseconds; the frequency to update the statistics.
Default: 10000 (10 seconds)

driver
Options to configure collection of metrics at the Spark Driver.

connectorSource
Whether to collect Spark Cassandra Connector metrics at the Spark Driver.
• false - do not collect metrics
• true - enable collection of metrics

Default: false

jvmSource
Configuration

Whether to collect JVM heap and garbage collection (GC) metrics from the Spark Driver.

- false - do not collect metrics
- true - enable collection of metrics

Default: false

**stateSource**

Whether to collect application state metrics at the Spark Driver.

- false - do not collect metrics
- true - enable collection of metrics

Default: false

**executor**

Options to configure collection of metrics at Spark executors.

**sink**

Whether to write metrics collected at Spark executors.

- false - do not collect metrics
- true - enable collection of metrics

Default: false

**connectorSource**

Whether to collect Spark Cassandra Connector metrics at Spark executors.

- false - do not collect metrics
- true - enable collection of metrics

Default: false

**jvmSource**

Whether to collect JVM heap and GC metrics at Spark executors.

- false - do not collect metrics
- true - enable collection of metrics

Default: false

DSE Analytics options

- Spark (page 130)
- Starting Spark drivers and executors (page 134)
- DSE File System (DSEFS) options (page 135)
- Spark Performance Service (page 129)

Spark resource and encryption options

```
spark_shared_secret_bit_length: 256
spark_security_enabled: false
spark_security_encryption_enabled: false
spark_daemon_readiness_assertion_interval: 1000
resource_manager_options:
```
worker_options:
  cores_total: 0.7
  memory_total: 0.6

workpools:
  - name: alwayson_sql
    cores: 0.25
    memory: 0.25

spark_ui_options:
  encryption: inherit
  encryption_options:
    enabled: false
    keystore: .keystore
    keystore_password: cassandra
    require_client_auth: false
    truststore: .truststore
    truststore_password: cassandra
    # Advanced settings
    # protocol: TLS
    # algorithm: SunX509
    # keystore_type: JKS
    # truststore_type: JKS
    # cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_DHE_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_RSA_WITH_AES_256_CBC_SHA, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA]

spark_shared_secret_bit_length
  The length of a shared secret used to authenticate Spark components and encrypt the connections between them. This value is not the strength of the cipher for encrypting connections. Default: 256

spark_security_enabled
  Enables Spark security based on shared secret infrastructure. Enables mutual authentication and optional encryption between DSE Spark Master and Workers and of communication channels except the web UI. Default: false

spark_security_encryption_enabled
  Enables encryption of Spark connections except the web UI. Uses DIGEST-MD5 SASL-based encryption mechanism. Requires spark_security_enabled: true.

spark_daemon_readiness_assertion_interval
  Time interval, in milliseconds, between subsequent retries by the Spark plugin for Spark Master and Worker readiness to start. Default: 1000

resource_manager_options
  DataStax Enterprise can control the memory and cores offered by particular Spark Workers in semi-automatic fashion. You can define the total amount of physical resources available to Spark Workers, and optionally add named work pools with specific resources dedicated to them.

worker_options
  If the option is not specified, the default value 0.6 is used. The amount of system resources that are made available to the Spark Worker.

cores_total
Configuration

If the option is not specified, the default value 0.7 is used. The number of total system cores available to Spark. This setting can be the exact number of cores or a decimal of the total system cores.

When the value is expressed as a decimal, the available resources are calculated in the following way:

```
Spark Worker cores = cores_total * total system cores
```

The lowest value that you can assign to Spark Worker cores is 1 core. If the results are lower, no exception is thrown and the values are automatically limited.

**Note:** Setting `cores_total` or a workpool’s `cores` to 1.0 is a decimal value, meaning 100% of the available cores will be reserved. Setting `cores_total` or `cores` to 1 (no decimal point) is an explicit value, and one core will be reserved.

**memory_total**

The amount of total system memory available to Spark. This setting can be the exact amount of memory or a decimal of the total system memory. When the value is an absolute value, you can use standard suffixes like M for megabyte and G for gigabyte.

When the value is expressed as a decimal, the available resources are calculated in the following way:

```
Spark Worker memory = memory_total * (total system memory - memory assigned to DataStax Enterprise)
```

The lowest values that you can assign to Spark Worker memory is 64 MB. If the results are lower, no exception is thrown and the values are automatically limited. If the option is not specified, the default value 0.6 is used.

**workpools**

Named work pools that can use a portion of the total resources defined under `worker_options`. A default work pool named `default` is used if no work pools are defined in this section. If work pools are defined, the resources allocated to the work pools are taken from the total amount, with the remaining resources available to the `default` work pool. The total amount of resources defined in the `workpools` section must not exceed the resources available to Spark in `worker_options`. A work pool named `alwayson_sql` is created by default for AlwaysOn SQL. By default, it is configured to use 25% of the resources available to Spark.

**name**

The name of the work pool.

**cores**

The number of system cores to use in this work pool expressed as either an absolute value or a decimal value. This option follows the same rules as `cores_total`.

**memory**

The amount of memory to use in this work pool expressed as either an absolute value or a decimal value. This option follows the same rules as `memory_total`.
spark_ui_options
Specify the source for SSL settings for Spark Master and Spark Worker UIs. The spark_ui_options apply only to Spark daemon UIs, and do not apply to user applications even when the user applications are run in cluster mode.

equipment
- inherit - inherit the SSL settings from the client encryption options.
- custom - use the following encryption_options (page 133) from dse.yaml.

Default: inherit

encryption_options
Set encryption options for HTTPS of Spark Master and Worker UI. The spark_encryption_options are not valid for DSE 5.1 and later.

enabled
Whether to enable Spark encryption for Spark client-to-Spark cluster and Spark internode communication.
Default: false

keystore
The keystore for Spark encryption keys.

The relative file path is the base Spark configuration directory that is defined by the SPARK_CONF_DIR environment variable. The default Spark configuration directory is resources/spark/conf.

Default: resources/dse/conf/.ui-keystore

keystore_password
The password to access the key store.
Default: cassandra

require_client_auth
Whether to require truststore for client authentication. When not set, the default is false.
Default: commented out (false)

truststore
The truststore for Spark encryption keys.

The relative file path is the base Spark configuration directory that is defined by the SPARK_CONF_DIR environment variable. The default Spark configuration directory is resources/spark/conf.

Default: commented out (resources/dse/conf/.ui-truststore)

truststore_password
The password to access the truststore.
Default: commented out (cassandra)

protocol
Defines the encryption protocol.
Default: commented out (TLS)

algorithm
Defines the key manager algorithm.
Default: commented out (TLSunX509SunX509S)

keystore_type
Valid types are JKS, JCEKS, PKCS12, or PKCS11. For file-based keystores, use PKCS12.
Default: commented out (JKS)

**truststore_type**
Valid types are JKS, JCEKS, PKCS12, or PKCS11.
Default: commented out (JKS)

**cipher_suites**
Defines the cipher suites for Spark encryption:
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
Default: commented out

Starting Spark drivers and executors

```
spark_process_runner:
    runner_type: default
    run_as_runner_options:
        user_slots:
            - slot1
            - slot2
```

**spark_process_runner:**
Options to configure how Spark driver and executor processes are created and managed.

**runner_type**
- default - Use the default runner type.
- run_as - Use the run_as_runner_options options. See Running Spark processes as separate users (page 231).

**run_as_runner_options**
The slot users for separating Spark processes users from the DSE service user.
See Running Spark processes as separate users (page 231).
Default: slot1, slot2

AlwaysOn SQL options

Properties to enable and configure AlwaysOn SQL (page 257).

```
# AlwaysOn SQL options
# alwayson_sql_options:
#     enabled: false
#     thrift_port: 10000
#     web_ui_port: 9077
#     reserve_port_wait_time_ms: 100
#     alwayson_sql_status_check_wait_time_ms: 500
```
alwayson_sql_options
The AlwaysOn SQL options enable and configure the server on this node.

enabled
Whether to enable AlwaysOn SQL for this node. The node must be an analytics node. When not set, the default is false.
Default: commented out (false)

thrift_port
The Thrift port on which AlwaysOn SQL listens.
Default: commented out (10000)

web_ui_port
The port on which the AlwaysOn SQL web UI is available.
Default: commented out (9077)

reserve_port_wait_time_ms
The wait time in milliseconds to reserve the thrift_port if it is not available.
Default: commented out (100)

alwayson_sql_status_check_wait_time_ms
The time in milliseconds to wait for a health check status of the AlwaysOn SQL server.
Default: commented out (500)

workpool
The work pool name (page 132) used by AlwaysOn SQL.
Default: commented out (alwayson_sql)

log_dsefs_dir
Location in DSEFS of the AlwaysOn SQL log files.
Default: commented out (/spark/log/alwayson_sql)

auth_user
The role to use for internal communication by AlwaysOn SQL if authentication is enabled. Custom roles must be created with login=true.
Default: commented out (alwayson_sql)

runner_max_errors
The maximum number of errors that can occur during AlwaysOn SQL service runner thread runs before stopping the service. A service stop requires a manual restart.
Default: commented out (10)

DSE File System (DSEFS) options
Properties to enable and configure the DSE File System (DSEFS (page 285)).

Note: DSEFS replaced the Cassandra File System (CFS). DSE version 6.7 does not support CFS.
Configuration

```
#    work_dir: /var/lib/dsefs
#    public_port: 5598
#    private_port: 5599
#    data_directories:
#      - dir: /var/lib/dsefs/data
#        storage_weight: 1.0
#        min_free_space: 5368709120
#    service_startup_timeout_ms: 30000
#    service_close_timeout_ms: 600000
#    server_close_timeout_ms: 2147483647  # Integer.MAX_VALUE
#    compression_frame_max_size: 1048576
#    query_cache_size: 2048
#    query_cache_expire_after_ms: 2000
#    gossip_options:
#      - round_delay_ms: 2000
#      - startup_delay_ms: 5000
#      - shutdown_delay_ms: 10000
#    rest_options:
#      - request_timeout_ms: 330000
#      - connection_open_timeout_ms: 550000
#      - client_close_timeout_ms: 600000
#      - server_request_timeout_ms: 300000
#      - idle_connection_timeout_ms: 600000
#      - internode_idle_connection_timeout_ms: 1200000
#      - core_max_concurrent_connections_per_host: 8
#    transaction_options:
#      - transaction_timeout_ms: 3000
#      - conflict_retry_delay_ms: 200
#      - conflict_retry_count: 40
#      - execution_retry_delay_ms: 1000
#      - execution_retry_count: 3
#      - block_allocator_options:
#        - overflow_margin_mb: 1024
#        - overflow_factor: 1.05
```

dsefs_options

Enable and configure options for DSEFS. See Using DSEFS (page 290).

enabled

Whether to enable DSEFS.

- true - enables DSEFS on this node, regardless of the workload.
- false - disables DSEFS on this node, regardless of the workload.
- blank or commented out (#) - DSEFS will start only if the node is configured to run analytics workloads.

Default: commented out (blank)

keyspace_name

The keyspace where the DSEFS metadata is stored. You can optionally configure multiple DSEFS file systems within a single datacenter by specifying different keyspace names for each cluster.

Default: commented out (dsefs)

work_dir
The local directory for storing the local node metadata, including the node identifier. The volume of data stored in this directory is nominal and does not require configuration for throughput, latency, or capacity. This directory must not be shared by DSEFS nodes.
Default: commented out (/var/lib/dsefs)

**public_port**
The public port on which DSEFS listens for clients.

**Note:** DataStax recommends that all nodes in the cluster have the same value. Firewalls must open this port to trusted clients. The service on this port is bound to the native_transport_address (page 87).
Default: commented out (5598)

**private_port**
The private port for DSEFS inter-node communication.

**Caution:** Do not open this port to firewalls; this private port must be not visible from outside of the cluster.
Default: commented out (5599)

**data_directories**
One or more data locations where the DSEFS data is stored.

- **dir**
  Mandatory attribute to identify the set of directories. DataStax recommends segregating these data directories on physical devices that are different from the devices that are used for DataStax Enterprise. Using multiple directories on JBOD improves performance and capacity.
  Default: commented out (/var/lib/dsefs/data)

**storage_weight**
The weighting factor for this location specifies how much data to place in this directory, relative to other directories in the cluster. This soft constraint determines how DSEFS distributes the data. For example, a directory with a value of 3.0 receives about three times more data than a directory with a value of 1.0.
Default: commented out (1.0)

**min_free_space**
The reserved space, in bytes, to not use for storing file data blocks. You can use a unit of measure suffix to specify other size units. For example: terabyte (1 TB), gigabyte (10 GB), and megabyte (5000 MB).
Default: commented out (5368709120)

**Advanced properties for DSEFS**

**service_startup_timeout_ms**
Wait time, in milliseconds, before the DSEFS server times out while waiting for services to bootstrap.
Default: commented out (30000)

**service_close_timeout_ms**
Wait time, in milliseconds, before the DSEFS server times out while waiting for services to close.
Default: commented out (60000)

**server_close_timeout_ms**
Wait time, in milliseconds, that the DSEFS server waits during shutdown before closing all pending connections.
Default: commented out (2147483647)

**compression_frame_max_size**
The maximum accepted size of a compression frame defined during file upload.
Default: commented out (1048576)

**query_cache_size**
Maximum number of elements in a single DSEFS Server query cache.
Default: commented out (2048)

**query_cache_expire_after_ms**
The time to retain the DSEFS Server query cache element in cache. The cache element expires when this time is exceeded.
Default: commented out (2000)

**gossip options**
Options to configure DSEFS gossip rounds.

**round_delay_ms**
The delay, in milliseconds, between gossip rounds.
Default: commented out (2000)

**startup_delay_ms**
The delay time, in milliseconds, between registering the location and reading back all other locations from the database.
Default: commented out (5000)

**shutdown_delay_ms**
The delay time, in milliseconds, between announcing shutdown and shutting down the node.
Default: commented out (30000)

**rest_options**
Options to configure DSEFS rest times.

**request_timeout_ms**
The time, in milliseconds, that the client waits for a response that corresponds to a given request.
Default: commented out (330000)

**connection_open_timeout_ms**
The time, in milliseconds, that the client waits to establish a new connection.
Default: commented out (55000)

**client_close_timeout_ms**
The time, in milliseconds, that the client waits for pending transfer to complete before closing a connection.
Default: commented out (60000)

**server_request_timeout_ms**
The time, in milliseconds, to wait for the server rest call to complete.
Default: commented out (30000)

**idle_connection_timeout_ms**
The time, in milliseconds, for RestClient to wait before closing an idle connection. If RestClient does not close connection after timeout, the connection is closed after 2*idle_connection_timeout_ms.

- time - wait time to close idle connection
• 0 - disable closing idle connections
Default: commented out (60000)

**internode_idle_connection_timeout_ms**
Wait time, in milliseconds, before closing idle internode connection. The internode connections are primarily used to exchange data during replication. Do not set lower than the default value for heavily utilized DSEFS clusters.
Default: commented out (0) (disabled)

**core_max_concurrent_connections_per_host**
Maximum number of connections to a given host per single CPU core. DSEFS keeps a connection pool for each CPU core.
Default: 120000

**transaction_options**
Options to configure DSEFS transaction times.

**transaction_timeout_ms**
Transaction run time, in milliseconds, before the transaction is considered for timeout and rollback.
Default: 3000

**conflict_retry_delay_ms**
Wait time, in milliseconds, before retrying a transaction that was ended due to a conflict. Default: 200

**conflict_retry_count**
The number of times to retry a transaction before giving up. Default: 40

**execution_retry_delay_ms**
Wait time, in milliseconds, before retrying a failed transaction payload execution.
Default: 1000

**execution_retry_count**
The number of payload execution retries before signaling the error to the application. Default: 3

**block_allocator_options**
Controls how much additional data can be placed on the local coordinator before the local node overflows to the other nodes. The trade-off is between data locality of writes and balancing the cluster. A local node is preferred for a new block allocation, if:

```
used_size_on_the_local_node < average_used_size_per_node * overflow_factor + overflow_margin
```

**overflow_margin_mb**

• **margin_size** - overflow margin size in megabytes
• 0 - disable block allocation overflow
Default: commented out (1024)

**overflow_factor**

• **factor** - overflow factor on an exponential scale
• 1.0 - disable block allocation overflow
Default: commented out (1.05)
## DSE Metrics Collector options

```
# insights_options:
# data_dir: /var/lib/cassandra/insights_data
# log_dir: /var/log/cassandra/
```

Uncomment these options only to change the default directories.

**insights_options**

Options for DSE Metrics Collector.

**data_dir**

Directory to store collected metrics. When not set, the default directory is `/var/lib/cassandra/insights_data`.

**log_dir**

Directory to store logs for collected metrics. When not set, the default directory is `/var/log/cassandra/`.

### Audit logging options for database activities

Track database activity using the audit log feature. To get the maximum information from data auditing, turn on data auditing on every node.

**Tip:** See [Setting up database auditing](#).

**audit_logging_options**

Options to enable and configure database activity logging.

**enabled**

Whether to enable database activity auditing.

- true - enables database activity auditing
- false - disables database activity auditing

Default: false

**logger**

The logger to use for recording events:

- SLF4JAuditWriter - Capture events in a log file.
- CassandraAuditWriter - Capture events in a table, `dse_audit.audit_log`.

**Tip:** Configure logging level, sensitive data masking, and log file name/location in the `logback.xml` file.

Default: SLF4JAuditWriter

**included_categories**

Comma separated list of event categories that are captured, where the category names are:

- QUERY - Data retrieval events.
- DML - (Data manipulation language) Data change events.
- DDL - (Data definition language) Database schema change events.
- DCL - (Data change language) Role and permission management events.
- AUTH - (Authentication) Login and authorization related events.
- ERROR - Failed requests.
- UNKNOWN - Events where the category and type are both UNKNOWN.

Event categories that are not listed are not captured.

**Warning:** Use either included_categories or excluded_categories but not both. When specifying included categories leave excluded_categories (page 141) blank or commented out.

Default: none (include all categories)

`excluded_categories`

Comma separated list of categories to ignore, where the categories are:

- QUERY - Data retrieval events.
- DML - (Data manipulation language) Data change events.
- DDL - (Data definition language) Database schema change events.
- DCL - (Data change language) Role and permission management events.
- AUTH - (Authentication) Login and authorization related events.
- ERROR - Failed requests.
- UNKNOWN - Events where the category and type are both UNKNOWN.

Events in all other categories are logged.

**Warning:** Use either included_categories or excluded_categories but not both. When specifying excluded categories leave included_categories (page 140) blank or commented out.

Default: none (exclude no categories)

`included_keyspaces`

The keyspaces for which events are logged. Specify keyspace names in a comma separated list or use a regular expression to filter on keyspace name.

**Warning:** DSE supports using either included_keyspaces or excluded_keyspaces but not both. When specifying included categories leave excluded_keyspaces (page 141) blank or commented out.

Default: none (include all keyspaces)

`excluded_keyspaces`

Log events for all keyspaces which are not listed. Specify a comma separated list keyspace names or use a regular expression to filter on keyspace name. Only use this option if included_keyspaces is blank or commented out.

Default: none (exclude no keyspaces)

`included_roles`

The roles for which events are logged. Log events for the listed roles. Specify roles in a comma separated list.

**Warning:** DSE supports using either included_roles or excluded_roles but not both. When specifying included_roles leave excluded_keyspaces (page 141) blank or commented it out.

Default: none (include all roles)

`excluded_roles`
Configuration

The roles for which events are not logged. Specify a comma separated list role names. Only use this option if included_roles is blank or commented out. Default: none (exclude no roles)

Cassandra audit writer options

```
retention_time: 0
 cassandra_audit_writer_options:
   mode: sync
   batch_size: 50
   flush_time: 250
   queue_size: 30000
   write_consistency: QUORUM
   # dropped_event_log: /var/log/cassandra/dropped_audit_events.log
   # day_partition_millis: 3600000
```

**retention_time**
The amount of time, in hours, audit events are retained by supporting loggers. Only the CassandraAuditWriter supports retention time.

- 0 - retain events forever
- hours - the number of hours to retain audit events

Default: 0 (retain events forever)

**cassandra_audit_writer_options**
Audit writer options.

**mode**
The mode the writer runs in.

- sync - A query is not executed until the audit event is successfully written.
- async - Audit events are queued for writing to the audit table, but are not necessarily logged before the query executes. A pool of writer threads consumes the audit events from the queue, and writes them to the audit table in batch queries.

**Important:** While async substantially improves performance under load, if there is a failure between when a query is executed, and its audit event is written to the table, the audit table might be missing entries for queries that were executed.

Default: sync

**batch_size**
Available only when mode: async. Must be greater than 0.

The maximum number of events the writer dequeues before writing them out to the table. If warnings in the logs reveal that batches are too large, decrease this value or increase the value of `batch_size_warn_threshold_in_kb (page 79)` in cassandra.yaml.

Default: 50

**flush_time**
Available only when mode: async.
configuration

The maximum amount of time in milliseconds before an event is removed from the queue by a writer before being written out. This flush time prevents events from waiting too long before being written to the table when there are not a lot of queries happening.
Default: 500

queue size

The size of the queue feeding the asynchronous audit log writer threads. When there are more events being produced than the writers can write out, the queue fills up, and newer queries are blocked until there is space on the queue. If a value of 0 is used, the queue size is unbounded, which can lead to resource exhaustion under heavy query load.
Default: 30000

write consistency

The consistency level that is used to write audit events.
Default: QUORUM

dropped event log

The directory to store the log file that reports dropped events. When not set, the default is /var/log/cassandra/dropped_audit_events.log.
Default: commented out (/var/log/cassandra/dropped_audit_events.log)

day partition millis

The interval, in milliseconds, between changing nodes to spread audit log information across multiple nodes. For example, to change the target node every 12 hours, specify 43200000 milliseconds. When not set, the default is 3600000 (1 hour).
Default: commented out (3600000) (1 hour)

DSE Tiered Storage options

Options to define one or more disk configurations for DSE Tiered Storage. Specify multiple disk configurations as unnamed tiers by a collection of paths that are defined in priority order, with the fastest storage media in the top tier. With heterogeneous storage configurations across the cluster, specify each disk configuration with config name: config settings, and then use this configuration in CREATE TABLE or ALTER TABLE statements.

```
# tiered_storage_options:
#     strategy1:
#         tiers:
#             - paths:
#                 - /mnt1
#                 - /mnt2
#             - paths: [ /mnt3, /mnt4 ]
#             - paths: [ /mnt5, /mnt6 ]
#     'another strategy':
```

DSE 6.7 Developer Guide (Latest version)
Configuration

# tiers: [ paths: [ /mnt1 ] ]

tiered_storage_options
Options to configure the smart movement of data across different types of storage media so that data is matched to the most suitable drive type, according to the performance and cost characteristics it requires.

strategy1
The first disk configuration strategy. Create a strategy2, strategy3, and so on. In this example, strategy1 is the configurable name of the tiered storage configuration strategy.

tiers
The unnamed tiers in this section define a storage tier with the paths and file paths that define the priority order.

local_options
Local configuration options overwrite the tiered storage settings for the table schema in the local dse.yaml file. See Testing DSE Tiered Storage configurations.

- paths
The section of file paths that define the data directories for this tier of the disk configuration. Typically list the fastest storage media first. These paths are used only to store data that is configured to use tiered storage. These paths are independent of any settings in the cassandra.yaml file.

- /filepath
The file paths that define the data directories for this tier of the disk configuration.

DSE Advanced Replication configuration settings

DSE Advanced Replication configuration options to replicate data from remote clusters to central data hubs.

# advanced_replication_options:
# enabled: false
# conf_driver_password_encryption_enabled: false
# advanced_replication_directory: /var/lib/cassandra/advrep
# security_base_path: /base/path/to/advrep/security/files/

advanced_replication_options
Options to enable and configure DSE Advanced Replication.

enabled
Whether to enable an edge node to collect data in the replication log.
Default: commented out (false)

conf_driver_password_encryption_enabled
Whether to enable encryption of driver passwords. When enabled, the stored driver password is expected to be encrypted. See Encrypting configuration file properties.
Default: commented out (false)

advanced_replication_directory
The directory for storing advanced replication CDC logs. A directory replication_logs will be created in the specified directory.
Default: commented out (/var/lib/cassandra/advrep)

security_base_path
Configuration

The base path to prepend to paths in the Advanced Replication configuration locations, including locations to SSL keystore, SSL truststore, and so on.
Default: commented out (/base/path/to/advrep/security/files/)

Inter-node messaging options

Configuration options for the internal messaging service used by several components of DataStax Enterprise. All internode messaging requests use this service.

```yaml
internode_messaging_options:
  port: 8609
  # frame_length_in_mb: 256
  # server_acceptor_threads: 8
  # server_worker_threads: 16
  # client_max_connections: 100
  # client_worker_threads: 16
  # handshake_timeout_seconds: 10
  # client_request_timeout_seconds: 60
```

**internode_messaging_options**
Configuration options for inter-node messaging.

**port**
The mandatory port for the inter-node messaging service.
Default: 8609

**frame_length_in_mb**
Maximum message frame length. When not set, the default is 256.
Default: commented out (256)

**server_acceptor_threads**
The number of server acceptor threads. When not set, the default is the number of available processors.
Default: commented out

**server_worker_threads**
The number of server worker threads. When not set, the default is the number of available processors * 8.
Default: commented out

**client_max_connections**
The maximum number of client connections. When not set, the default is 100.
Default: commented out (100)

**client_worker_threads**
The number of client worker threads. When not set, the default is the number of available processors * 8.
Default: commented out

**handshake_timeout_seconds**
Timeout for communication handshake process. When not set, the default is 10.
Default: commented out (10)

**client_request_timeout_seconds**
Configuration

Timeout for non-query search requests like core creation and distributed deletes. When not set, the default is 60.
Default: commented out (60)

DSE Multi-Instance server_id

server_id
In DSE Multi-Instance /etc/dse-nodeId/dse.yaml files, the server_id option is generated to uniquely identify the physical server on which multiple instances are running. The server_id default value is the media access control address (MAC address) of the physical server. You can change server_id when the MAC address is not unique, such as a virtualized server where the host's physical MAC is cloned.

DSE Graph options

- DSE Graph system-level options (page 146)
- DSE Graph Gremlin Server options (page 148)

DSE Graph system-level options
These graph options are system-level configuration options and options that are shared between graph instances. Add an option if it is not present in the provided dse.yaml file.

```yaml
graph:
  # analytic_evaluation_timeout_in_minutes: 10080
  # realtime_evaluation_timeout_in_seconds: 30
  # schema_agreement_timeout_in_ms: 10000
  # system_evaluation_timeout_in_seconds: 180
  # index_cache_size_in_mb: 128
  # max_query_queue: 10000
  # max_query_threads (no explicit default)
  # max_query_params: 16
```

These graph options are system-level configuration options and options that are shared between graph instances.

Option names and values expressed in ISO 8601 format used in earlier DSE 5.0 releases are still valid. The ISO 8601 format is deprecated.

analytic_evaluation_timeout_in_minutes
Maximum time to wait for an OLAP analytic (Spark) traversal to evaluate. When not set, the default is 10080 (168 hours).
Default: commented out (10080)

realtime_evaluation_timeout_in_seconds
Maximum time to wait for an OLTP real-time traversal to evaluate. When not set, the default is 30.
Default: commented out (30)

schema_agreement_timeout_in_ms
Maximum time to wait for the database to agree on schema versions before timing out. When not set, the default is 10000 (10 seconds).
Default: commented out (10000)

**system_evaluation_timeout_in_seconds**
Maximum time to wait for a graph system-based request to execute, like creating a new graph. When not set, the default is 180 (3 minutes).
Default: commented out (180)

**schema_mode**
Controls the way that the schemas are handled.
- Production = Schema must be created before data insertion. Schema cannot be changed after data is inserted. Full graph scans are disallowed unless the option graph.allow_scan is changed to TRUE.
- Development = No schema is required to write data to a graph. Schema can be changed after data is inserted. Full graph scans are allowed unless the option graph.allow_scan is changed to FALSE.

When not set, the default is Production. If this option is not present, manually enter it to use Development.
Default: not present

**index_cache_size_in_mb**
The amount of ram to allocate to the index cache. When not set, the default is 128.
Default: commented out (128)

**max_query_queue**
The maximum number of CQL queries that can be queued as a result of Gremlin requests. Incoming queries are rejected if the queue size exceeds this setting.
When not set, the default is 10000.
Default: commented out (10000)

**max_query_threads**
The maximum number of threads to use for queries to the database. When this option is not set, the default is calculated:
- If gremlinPool is present and nonzero:
  
  \[10 \times \text{the gremlinPool setting}\]
- If gremlinPool is not present in this file or set to zero:

  The number of available CPU cores

See DSE Graph Gremlin connectionPool options *(page 150).*
Default: calculated

**max_query_params**
The maximum number of parameters that can be passed on a graph query request for TinkerPop drivers and drivers using the Cassandra native protocol. Passing very large numbers of parameters on requests is an anti-pattern, because the script evaluation time increases proportionally. DataStax recommends reducing the number of parameters to speed up script compilation times. Before you increase this value, consider alternate methods for parameterizing scripts, like passing a single map. If the graph query request requires many arguments, pass a list.
Default: commented out (16)
DSE Graph Gremlin Server options

The Gremlin Server is configured using Apache TinkerPop specifications.

```yaml
gremlin_server:
  # port: 8182
  # threadPoolWorker: 2
  # gremlinPool: 0
  # scriptEngines:
    # gremlin-groovy:
      # config:
        # sandbox_enabled: false
        # sandbox_rules:
        #   whitelist_packages:
        #     - package.name
        #   whitelist_types:
        #     - fully.qualified.type.name
        #   whitelist_supers:
        #     - fully.qualified.class.name
        # blacklist_packages:
        #     - package.name
        # blacklist_supers:
        #     - fully.qualified.class.name
```

**gremlin_server**
The top-level configurations in Gremlin Server.

**port**
The available communications port for Gremlin Server. When not set, the default is 8182. Default: commented out (8182)

**threadPoolWorker**
The number of worker threads that handle non-blocking read and write (requests and responses) on the Gremlin Server channel, including routing requests to the right server operations, handling scheduled jobs on the server, and writing serialized responses back to the client. When not set, the default is 2. Default: commented out (2)

**gremlinPool**
The number of Gremlin threads available to execute actual scripts in a ScriptEngine. This pool represents the workers available to handle blocking operations in Gremlin Server.

- 0 - the value of the JVM property cassandra.available_processors, if that property is set
- When not set - the value of Runtime.getRuntime().availableProcessors()

Default: commented out (0)

**scriptEngines**
Section to configure gremlin server scripts.

**gremlin-groovy**
Section for gremlin-groovy scripts.

**sandbox_enabled**
Sandbox is enabled by default. To disable the gremlin groovy sandbox entirely, set to false.

**sandbox_rules**
Section for sandbox rules.

**whitelist_packages**
List of packages, one package per line, to whitelist.

- **package.name**
  Retain the hyphen before the fully qualified package name.

**whitelist_types**
List of types, one type per line, to whitelist.

- **fully.qualified.type.name**
  Retain the hyphen before the fully qualified type name.

**whitelist_supers**
List of super classes, one class per line, to whitelist. Retain the hyphen before the fully qualified class name.

- **fully.qualified.class.name**
  Retain the hyphen before the fully qualified class name.

**blacklist_packages**
List of packages, one package per line, to blacklist.

- **package.name**
  Retain the hyphen before the fully qualified package name.

**blacklist_supers**
List of super classes, one class per line, to blacklist. Retain the hyphen before the fully qualified class name.

- **fully.qualified.class.name**
  Retain the hyphen before the fully qualified class name.

See also remote.yaml configuration file (page 149).

**remote.yaml configuration file**

The remote.yaml file is the primary configuration file for DSE Graph Gremlin console connection to the Gremlin Server.

remote.yaml
The location of the remote.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/graph/gremlin-console/conf/remote.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/graph/gremlin-console/conf/remote.yaml</td>
</tr>
</tbody>
</table>

The dse.yaml (page 103) file is the primary configuration file for the DataStax Enterprise Graph configuration, and includes the setting for the Gremlin Server (page 148) options.

**Synopsis**

For the properties in each section, the parent setting has zero spaces. Each child entry requires at least two spaces. Adhere to the YAML syntax and retain the spacing. For
example, no spaces before the parent `node_health_options` entry, and at least two spaces before the child settings:

```yaml
node_health_options:
  refresh_rate_ms: 50000
  uptime_ramp_up_period_seconds: 10800
  dropped_mutation_window_minutes: 30
```

DSE Graph Gremlin basic options

An `Apache TinkerPop` YAML file, `remote.yaml`, is configured with Gremlin Server information: The Gremlin Server is configured using `Apache TinkerPop` specifications.

```yaml
hosts: [localhost]
port: 8182
serializer: { className:
  org.apache.tinkerpop.gremlin.driver.ser.GryoMessageSerializerV3d0,
  config: { ioRegistries:
    [org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerIoRegistryV3d0] }}
```

**hosts**

Identifies a host or hosts running DSE node running Gremlin Server. May need to use `rpc_address` value set in
Default: [localhost]

**Note:** You can also connect to the Spark Master (page 225) node for the datacenter by either running the console from the Spark Master or specifying the Spark Master in the `hosts` field in the `remote.yaml` file.

**port**

Identifies a port on a DSE node running Gremlin Server. The port value needs to match the port value specified (page 148) for `gremlin_server`: in the dse.yaml file.
Default: 8182

**serializer**

Specifies the class and configuration for the serializer used to pass information between the Gremlin console and the Gremlin Server.
Default: { className:
  org.apache.tinkerpop.gremlin.driver.ser.GryoMessageSerializerV3d0,
  config: { ioRegistries:
    [org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerIoRegistryV3d0]
  }
}

DSE Graph Gremlin connectionPool options

The `connectionPool` settings specify a number of options that will be passed between the Gremlin console and the Gremlin Server.

```yaml
connectionPool: {
  enableSsl: false,
  maxContentLength: 65536000,
  maxInProcessPerConnection: 4,
  maxSimultaneousUsagePerConnection: 16,
```
maxSize: 8,
maxWaitForConnection: 3000,
maxWaitForSessionClose: 3000,
minInProcessPerConnection: 1,
minSimultaneousUsagePerConnection: 8,
minSize: 2,
reconnectInterval: 1000,
resultIterationBatchSize: 64
}

enableSsl
Determine if SSL should be enabled or not. If enabled on the server, then it must be enabled on the client.
Default: false

maxContentLength
The maximum length in bytes that a message can be sent to the server. This number can be no greater than the setting of the same name in the server configuration.
Default: 65536000

maxInProcessPerConnection
The maximum number of in-flight requests that can occur on a connection.
Default: 4

maxSimultaneousUsagePerConnection
The maximum number of times that a connection can be borrowed from the pool simultaneously.
Default: 16

maxSize
The maximum size of a connection pool for a host.
Default: 8

maxWaitForConnection
The amount of time in milliseconds to wait for a new connection before timing out.
Default: 3000

maxWaitForSessionClose
The amount of time in milliseconds to wait for a session to close before timing out (does not apply to sessionless connections).
Default: 3000

minInProcessPerConnection
The minimum number of in-flight requests that can occur on a connection.
Default: 1

minSimultaneousUsagePerConnection
The maximum number of times that a connection can be borrowed from the pool simultaneously.
Default: 8

minSize
The minimum size of a connection pool for a host.
Default: 2

reconnectInterval
The amount of time in milliseconds to wait before trying to reconnect to a dead host.
Configuration

Default: 1000

**resultIterationBatchSize**
The override value for the size of the result batches to be returned from the server.
Default: 64

### DSE Graph Gremlin AuthProperties options

Security considerations for authentication between the Gremlin console and the Gremlin server require additional options in the `remote.yaml` file.

```yaml
# jaasEntry:
# protocol:
# username: xxx
# password: xxx
```

- **jaasEntry**
  Sets the `AuthProperties.Property.JAAS_ENTRY` properties for authentication to Gremlin Server.
  Default: commented out (no value)

- **protocol**
  Sets the `AuthProperties.Property.PROTOCOL` properties for authentication to Gremlin Server.
  Default: commented out (no value)

- **username**
  The username to submit on requests that require authentication.
  Default: commented out (xxx)

- **password**
  The password to submit on requests that require authentication.
  Default: commented out (xxx)

### cassandra-rackdc.properties file

The GossipingPropertyFileSnitch, Ec2Snitch, and Ec2MultiRegionSnitch use the cassandra-rackdc.properties configuration file to determine which datacenters and racks nodes belong to. They inform the database about the network topology to route requests efficiently and distribute replicas evenly. Settings for this file depend on the type of snitch:

- GossipingPropertyFileSnitch *(page 152)*
- Configuring the Amazon EC2 single-region snitch *(page 154)*
- Configuring Amazon EC2 multi-region snitch *(page 156)*

This page also includes instructions for migrating *(page 153)* from the PropertyFileSnitch to the GossipingPropertyFileSnitch.

### GossipingPropertyFileSnitch

This snitch is recommended for production. It uses rack and datacenter information for the local node defined in the cassandra-rackdc.properties file and propagates this information to other nodes via gossip.
To configure a node to use GossipingPropertyFileSnitch, edit the `cassandra-rackdc.properties` file as follows:

- Define the datacenter and rack that include this node. The default settings:

  ```
  dc=DC1
  rack=RAC1
  ```

  **Note:** datacenter and rack names are case-sensitive. For examples, see Initializing a single datacenter per workload type and Initializing multiple datacenters per workload type.

- To save bandwidth, add the `prefer_local=true` option. This option tells DataStax Enterprise to use the local IP address when communication is not across different datacenters.

**Migrating from the PropertyFileSnitch to the GossipingPropertyFileSnitch**

To allow migration from the PropertyFileSnitch, the GossipingPropertyFileSnitch uses the `cassandra-topology.properties` file when present. Delete the file after the migration is complete. For more information about migration, see Switching snitches.

  **Note:** The GossipingPropertyFileSnitch always loads `cassandra-topology.properties` when that file is present. Remove the file from each node on any new cluster or any cluster migrated from the PropertyFileSnitch.

**cassandra-topology.properties file**

The PropertyFileSnitch uses the `cassandra-topology.properties` for datacenters and rack names and to determine network topology so that requests are routed efficiently and allows the database to distribute replicas evenly.

  **Note:** The GossipingPropertyFileSnitch (page 152) snitch is recommended for production. See Migrating from the PropertyFileSnitch to the GossipingPropertyFileSnitch (page 153).

**PropertyFileSnitch**

This snitch determines proximity as determined by rack and datacenter. It uses the network details located in the `cassandra-topology.properties` file. When using this snitch, you can define your datacenter names to be whatever you want. Make sure that the datacenter names correlate to the name of your datacenters in the **keyspace definition**. Every node in the cluster should be described in the `cassandra-topology.properties` file, and this file should be exactly the same on every node in the cluster.
Setting datacenters and rack names

If you had non-uniform IPs and two physical datacenters with two racks in each, and a third logical datacenter for replicating analytics data, the `cassandra-topology.properties` file might look like this:

**Note:** Datacenter and rack names are case-sensitive.

```plaintext
# datacenter One
175.56.12.105=DC1:RAC1
175.50.13.200=DC1:RAC1
175.54.35.197=DC1:RAC1

120.53.24.101=DC1:RAC2
120.55.16.200=DC1:RAC2
120.57.102.103=DC1:RAC2

# datacenter Two
110.56.12.120=DC2:RAC1
110.50.13.201=DC2:RAC1
110.54.35.184=DC2:RAC1

50.33.23.120=DC2:RAC2
50.45.14.220=DC2:RAC2
50.17.10.203=DC2:RAC2

# Analytics Replication Group
172.106.12.120=DC3:RAC1
172.106.12.121=DC3:RAC1
172.106.12.122=DC3:RAC1

# default for unknown nodes
default =DC3:RAC1
```

Configuring snitches for cloud providers

Configure a cloud provider snitch that corresponds to the provider.

**Configuring the Amazon EC2 single-region snitch**

Use the `Ec2Snitch` for simple cluster deployments on Amazon EC2 where all nodes in the cluster are within a single region. Because private IPs are used, this snitch does not work across multiple regions.

In EC2 deployments, the region name is treated as the datacenter name and availability zones are treated as racks within a datacenter. For example, if a node is in the `us-east-1` region, its IP would be treated as `us-east-1:RAC1`. The `Ec2Snitch` automatically sets the correct IP addresses and rack names for nodes in a single region.
region, **us-east** is the datacenter name and **1** is the rack location. (Racks are important for distributing replicas, but not for datacenter naming.)

If you are using only a single datacenter, you do not need to specify any properties.

If you need multiple datacenters, set the `dc_suffix` options in the `cassandra-rackdc.properties` file. Any other lines are ignored.

For example, for each node within the **us-east** region, specify the datacenter in its `cassandra-rackdc.properties` file:

**Note:** Datacenter names are case-sensitive.

- **node0**
  
  `dc_suffix=_1_cassandra`

- **node1**
  
  `dc_suffix=_1_cassandra`

- **node2**
  
  `dc_suffix=_1_cassandra`

- **node3**
  
  `dc_suffix=_1_cassandra`

- **node4**
  
  `dc_suffix=_1_analytics`

- **node5**
  
  `dc_suffix=_1_search`

This results in three datacenters for the region:

<table>
<thead>
<tr>
<th>Datacenter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east_1_cassandra</td>
</tr>
<tr>
<td>us-east_1_analytics</td>
</tr>
<tr>
<td>us-east_1_search</td>
</tr>
</tbody>
</table>

**Note:** The datacenter naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200.

**Keypspace strategy options**

When defining your [keyspace strategy options](https://docs.datastax.com/en/dse/6.7/devguide/develop/keyspace-strategy-options.html), use the EC2 region name, such as `us-east`, as your datacenter name.
Configuring Amazon EC2 multi-region snitch

Use the Ec2MultiRegionSnitch for deployments on Amazon EC2 where the cluster spans multiple regions.

You must configure settings in both the cassandra.yaml file and the property file (cassandra-rackdc.properties) used by the Ec2MultiRegionSnitch.

Configuring cassandra.yaml for cross-region communication

The Ec2MultiRegionSnitch uses public IP designated in the broadcast_address to allow cross-region connectivity. Configure each node as follows:

1. In the cassandra.yaml, set the listen_address (page 65) to the private IP address of the node, and the broadcast_address (page 80) to the public IP address of the node.

   This allows DataStax Enterprise nodes in one EC2 region to bind to nodes in another region, thus enabling multiple datacenter support. For intra-region traffic, DataStax Enterprise switches to the private IP after establishing a connection.

2. Set the addresses of the seed nodes in the cassandra.yaml file to that of the public IP. Private IP are not routable between networks. For example:

   ```
   seeds: 50.34.16.33, 60.247.70.52
   ```

   To find the public IP address, from each of the seed nodes in EC2:

   ```
   $ curl http://instance-data/latest/meta-data/public-ipv4
   ```

   **Note:** Do not make all nodes seeds, see Internode communications (gossip).

3. Be sure that the storage_port (page 86) or ssl_storage_port (page 99) is open on the public IP firewall.

Configuring the snitch for cross-region communication

In EC2 deployments, the region name is treated as the datacenter name and availability zones are treated as racks within a datacenter. For example, if a node is in the us-east-1 region, us-east is the datacenter name and 1 is the rack location. (Racks are important for distributing replicas, but not for datacenter naming.)

For each node, specify its datacenter in the cassandra-rackdc.properties. The dc_suffix option defines the datacenters used by the snitch. Any other lines are ignored.

In the example below, there are two DataStax Enterprise datacenters and each datacenter is named for its workload. The datacenter naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200. (datacenter names are case-sensitive.)
### Region: us-east

<table>
<thead>
<tr>
<th>Node and datacenter:</th>
<th>Node and datacenter:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• node0 dc_suffix=_1_transactional</td>
<td>• node0 dc_suffix=_1_transactional</td>
</tr>
<tr>
<td>• node1 dc_suffix=_1_transactional</td>
<td>• node1 dc_suffix=_1_transactional</td>
</tr>
<tr>
<td>• node2 dc_suffix=_2_transactional</td>
<td>• node2 dc_suffix=_2_transactional</td>
</tr>
<tr>
<td>• node3 dc_suffix=_2_transactional</td>
<td>• node3 dc_suffix=_2_transactional</td>
</tr>
<tr>
<td>• node4 dc_suffix=_1_analytics</td>
<td>• node4 dc_suffix=_1_analytics</td>
</tr>
<tr>
<td>• node5 dc_suffix=_1_search</td>
<td>• node5 dc_suffix=_1_search</td>
</tr>
</tbody>
</table>

This results in four **us-east** datacenters:

- us-east_1_transactional
- us-east_2_transactional
- us-east_1_analytics
- us-east_1_search

### Region: us-west

<table>
<thead>
<tr>
<th>Node and datacenter:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• node0 dc_suffix=_1_transactional</td>
</tr>
<tr>
<td>• node1 dc_suffix=_1_transactional</td>
</tr>
<tr>
<td>• node2 dc_suffix=_2_transactional</td>
</tr>
<tr>
<td>• node3 dc_suffix=_2_transactional</td>
</tr>
<tr>
<td>• node4 dc_suffix=_1_analytics</td>
</tr>
<tr>
<td>• node5 dc_suffix=_1_search</td>
</tr>
</tbody>
</table>

This results in four **us-west** datacenters:

- us-west_1_transactional
- us-west_2_transactional
- us-west_1_analytics
- us-west_1_search

### Keyspace strategy options

When defining your keyspace strategy options, use the EC2 region name, such as ``us-east``, as your datacenter name.

### Configuring the Google Cloud Platform snitch

Use the GoogleCloudSnitch for DataStax Enterprise deployments on Google Cloud Platform across one or more regions. The region is treated as a datacenter and the availability zones are treated as racks within the datacenter. All communication occurs over private IP addresses within the same logical network.

The region name is treated as the datacenter name and zones are treated as racks within a datacenter. For example, if a node is in the **us-central1-a** region, **us-central1** is the datacenter name and **a** is the rack location. (Racks are important for distributing replicas, but not for datacenter naming.) This snitch can work across multiple regions without additional configuration.
Configuration

If you are using only a single datacenter, you do not need to specify any properties.

If you need multiple datacenters, set the `dc_suffix` options in the cassandra-rackdc.properties file. Any other lines are ignored.

For example, for each node within the `us-central1` region, specify the datacenter in its cassandra-rackdc.properties file:

<table>
<thead>
<tr>
<th>Node</th>
<th>dc_suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>node0</td>
<td><code>dc_suffix=_a_transactional</code></td>
</tr>
<tr>
<td>node1</td>
<td><code>dc_suffix=_a_transactional</code></td>
</tr>
<tr>
<td>node2</td>
<td><code>dc_suffix=_a_transactional</code></td>
</tr>
<tr>
<td>node3</td>
<td><code>dc_suffix=_a_transactional</code></td>
</tr>
<tr>
<td>node4</td>
<td><code>dc_suffix=_a_analytics</code></td>
</tr>
<tr>
<td>node5</td>
<td><code>dc_suffix=_a_search</code></td>
</tr>
</tbody>
</table>

**Note:** Datacenter names are case-sensitive.

### Configuring the Apache CloudStack snitch

Use the `CloudstackSnitch` for Apache CloudStack environments. Because zone naming is free-form in Apache CloudStack, this snitch uses the widely-used `<country> <az>` notation.

### Setting system properties during startup

Use the system property (`-D`) switch to modify the DataStax Enterprise (DSE) settings during start up.

**Tip:** To automatically pass the settings each time DSE starts, uncomment or add the switch to the `jvm.options` file.

### Synopsis

Change the start up parameters using the following syntax:

- **Command line:**
  
  ```
  dse cassandra -Dparameter_name=value
  ```

- **jvm.options file:**
  
  ```
  -Dparameter_name=value
  ```

- **cassandra-env.sh file:**
  
  ```
  JVM_OPTS= "$JVM_OPTS -Dparameter_name=value"
  ```
Warning: Only pass the parameter to the start-up operation once. If the same switch is passed to the start operation multiple times, for example from both the `jvm.options` file and on the command line, DSE may fail to start or may use the wrong parameter.

Example: Examples

Starting a node without joining the ring:
- Command line:
  ```
  dse cassandra -Dcassandra.join_ring=false
  ```
- jvm.options:
  ```
  -Dcassandra.join_ring=false
  ```

Replacing a dead node:
- Command line:
  ```
  dse cassandra -Dcassandra.replace_address=10.91.176.160
  ```
- jvm.options:
  ```
  -Dcassandra.replace_address=10.91.176.160
  ```

Cassandra system properties

Cassandra native Java Virtual Machine (JVM) system parameters.

- **-Dcassandra.auto_bootstrap**
  Set `auto_bootstrap` (page 79) to `false` during the initial set up of a node to override the default setting in the `cassandra.yaml` file.
  Default: `true`.

- **-Dcassandra.available_processors**
  Number of processors available to DSE. In a multi-instance deployment, each instance independently assumes that all CPU processors are available to it. Use this setting to specify a smaller set of processors.
  Default: `all_processors`.

- **-Dcassandra.config**
  Set to the directory location of the `cassandra.yaml` file.
  Default: depends on the type of installation.

- **-Dcassandra.consistent_rangemovement**
  Set to `true`, makes bootstrapping behavior effective.
  Default: `false`.

- **-Dcassandra.consistent_replace**
  Specify the level of consistency required during a node replacement (ONE, QUORUM, or LOCAL_QUORUM). The default value, `ONE`, may result in possibly stale data but...
uses less system resources. If set to QUORUM or LOCAL QUORUM, the replacement node coordinates repair among a (local) quorum of replicas concurrently with replacement streaming. Repair transfers the differences to the replacement node, ensuring it is consistent with other replicas when the replacement process is finished, assuming data is inserted using either QUORUM or LOCAL QUORUM consistency levels.

**Note:** The value for consistent replace should match the value for application read consistency.

Default: ONE

-Dcassandra.consistent_replace.parallelism
Specify how many ranges will be repaired simultaneously during a consistent replace. The higher the parallelism, the more resources are consumed cluster-wide, which may affect overall cluster performance. Used only in conjunction with Dcassandra.consistent_replace.

Default: 2

-Dcassandra.consistent_replace.retries
Specify how many times a failed repair will be retried during a replace. If all retries fail, the replace fails. Used only in conjunction with Dcassandra.consistent_replace.

Default: 3

-Dcassandra.consistent_replace.whitelist
Specify keyspaces and tables on which to perform a consistent replace. The keyspaces and tables can be specified as: “ks1, ks2.cf1”. The default is blank, in which case all keyspaces and tables are replaced. Used only in conjunction with Dcassandra.consistent_replace.

Default: blank (not set)

-Dcassandra.disable_auth_caches_remote_configuration
Set to true to disable authentication caches, for example the caches used for credentials, permissions, and roles. This will mean those config options can only be set (persistently) in cassandra.yaml and will require a restart for new values to take effect.

Default: false.

-Dcassandra.expiration_date_overflow_policy
Set the policy (REJECT or CAP) for any TTL (time to live) timestamps that exceeds the maximum value supported by the storage engine, 2038-01-19T03:14:06+00:00. The database storage engine can only encode TTL timestamps through January 19 2038 03:14:07 UTC due to the Year 2038 problem.

- **REJECT:** Reject requests that contain an expiration timestamp later than 2038-01-19T03:14:06+00:00.
- CAP: Allow requests and insert expiration timestamps later than 2038-01-19T03:14:06+00:00 as 2038-01-19T03:14:06+00:00.  
- CAP-NOWARN: Allow requests and insert expiration timestamps later than 2038-01-19T03:14:06+00:00 as 2038-01-19T03:14:06+00:00, but do not emit a warning.

Default: REJECT.

-Dcassandra.force_default_indexing_page_size
Set to true to disable dynamic calculation of the page size used when indexing an entire partition during initial index build or a rebuild. Fixes the page size to the default of 10000 rows per page.
Default: false.

-Dcassandra.ignore_dc
Set to true to ignore the datacenter name change on startup. Applies only when using DseSimpleSnitch.
Default: false.

-Dcassandra.initial_token
Use when DSE is not using virtual nodes (vnodes). Set to the initial partitioner token for the node on the first start up.
Default: blank (not set).

Note: Vnodes automatically select tokens.

-Dcassandra.join_ring
Set to false to prevent the node from joining a ring on startup.

Tip: Add the node to the ring afterwards using nodetool join (page 1004) and a JMX call.

Default: true.

-Dcassandra.load_ring_state
Set to false to clear all gossip state for the node on restart.
Default: true.

-Dcassandra.metricsReporterConfigFile
Enables pluggable metrics reporter and configures it from the specified file.
Default: blank (not set).

-Dcassandra.native_transport_port
Set to the port number that CQL native transport listens for clients.
Default: 9042.

-Dcassandra.native_transport_startup_delay_seconds
Set to the number of seconds to delay the native transport server start up.
Default: 0 (no delay).

-Dcassandra.partitioner
Set to the partitioner name.

-Dcassandra.partition_sstables_by_token_range
Set to false to disable JBOD SSTable partitioning by token range to multiple data_file_directories.

Caution: Advanced setting that should only be used with guidance from DataStax Support.
Configuration

Default: true.

-Dcassandra.replace_address
Set to the listen_address (page 65) or the broadcast_address (page 80) when replacing a dead node with a new node. The new node must be in the same state as before bootstrapping, without any data in its data directory.

Note: The broadcast_address defaults to the listen_address except when the ring is using the Configuring Amazon EC2 multi-region snitch (page 156).

-Dcassandra.replace_address_first_boot
Same as -Dcassandra.replace_address but only runs the first time the Cassandra node boots.

Note: This property is preferred over -Dcassandra.replace_address since it has no effect on subsequent boots if it is not removed from jvm.options or cassandra-env.sh.

-Dcassandra.replayList
Allows restoring specific tables from an archived commit log.

-Dcassandra.ring_delay_ms
Set to the number of milliseconds the node waits to hear from other nodes before formally joining the ring.
Default: 30000.

-Dcassandra.ssl_storage_port
Sets the SSL port for encrypted communication.
Default: 7001.

-Dcassandra.start_native_transport
Enables or disables the native transport server. See start_native_transport (page 87) in cassandra.yaml.
Default: true.

-Dcassandra.storage_port
Sets the port for inter-node communication.
Default: 7000.

-Dcassandra.write_survey
Set to true to enable a tool for testing new compaction and compression strategies. write_survey allows you to experiment with different strategies and benchmark write performance differences without affecting the production workload. See Testing compaction and compression.
Default: false.

Java Management Extension system properties

DataStax Enterprise exposes metrics and management operations via Java Management Extensions (JMX). JConsole and the nodetool utility are JMX-compliant management tools.

-Dcom.sun.management.jmxremote.port
Sets the port number on which the database listens for JMX connections.

Note: By default, you can interact with DataStax Enterprise using JMX on port 7199 without authentication.
-Dcom.sun.management.jmxremote.ssl
  Change to true to enable SSL for JMX.
  Default: false

-Dcom.sun.management.jmxremote.authenticate
  True enables remote authentication for JMX.
  Default: false

-Djava.rmi.server.hostname
  Sets the interface hostname or IP that JMX should use to connect. Uncomment and set if you are having trouble connecting.

**Search system properties**

DataStax Enterprise (DSE) Search system properties.

-Ddse.search.client.timeout.secs
  Set the timeout in seconds for native driver search core management calls using the dsetool search-specific commands.
  Default: 600 (10 minutes).

-Ddse.search.query.threads
  Sets the number of Search queries that can execute in parallel. Consider increasing this value or reducing client/driver requests per connection if EnqueuedRequestCount (page 359) does not stabilize near zero.
  Default: The default is two times the number of CPUs (including hyperthreading).

-Ddse.solr.data.dir
  Set the path to store DSE Search data. See Set the location of search indexes.

-Dsolr.offheap.enable
  The DSE Search per-segment filter cache is moved off-heap by using native memory to reduce on-heap memory consumption and garbage collection overhead. The off-heap filter cache is enabled by default. To disable, set to false to pass the offheap JVM system property at startup time. When not set, the default is true.
  Default: true

**Threads per core system properties**

Tune TPC using the Netty system parameters.

-Ddse.io.aio.enable
  Set to false to have all read operations use the AsynchronousFileChannel regardless of the operating system or disk type.

  The default setting true allows dynamic switching of libraries for read operations as follows:
  - LibAIO on solid state drives (SSD) and EXT4/XFS
  - AsynchronousFileChannel for read operations on hard disk drives and all non-Linux operating systems
Caution: Use this advanced setting only with guidance from DataStax Support.
Default: true

-Ddse.io.aio.force
Set to true to force all read operations to use LibAIO regardless of the disk type or operating system.
Caution: Use this advanced setting only with guidance from DataStax Support.
Default: false

-Dnetty.epoll_check_interval_nanos
Sets the granularity for calling an epoll select in nanoseconds, which is a system call. Setting the value too low impacts performance because by making too many system calls. Setting the value too high, impacts performance by delaying the discovery of new events.
Default: 2000

-Dnetty.eventloop.busy_extra_spins=N
Set to the number of iterations in the epoll event loops that are performed when queues are empty before moving on to the next backoff stage. Increasing the value reduces latency while increasing CPU usage when the loops are idle.
Default: 10

-Dnetty.schedule_check_interval_nanos
Set the granularity for checking if scheduled events are ready to execute in nanoseconds. Specifying a value below 1 nanosecond is not productive. Too high a values delays scheduled tasks.
Default: 1000

LDAP system properties for DataStax Enterprise Authentication

-Ddse.ldap.connection.timeout.ms
The number of milliseconds before the connection times out.
Default:

-Ddse.ldap.pool.min.idle
Finer control over the connection pool for DataStax Enterprise LDAP authentication connector. The min idle settings determines the minimum number of connections allowed in the pool before the evictor thread will create new connections. This setting has no effect if the evictor thread isn't configured to run.
Default:

-Ddse.ldap.pool.exhausted.action
Determines what the pool does when it is full. It can be one of:
- fail - the pool with throw an exception
- block - the pool will block for max wait ms (default)
- grow - the pool will just keep growing (not recommended)
Default: block

-Ddse.ldap.pool.max.wait
When the `dse.ldap.pool.exhausted.action` is `block`, sets the number of milliseconds to block the pool before throwing an exception.
Default:

`-Ddse.ldap.pool.test.borrow`
Tests a connection when it is borrowed from the pool.
Default:

`-Ddse.ldap.pool.test.return`
Tests a connection returned to the pool.
Default:

`-Ddse.ldap.pool.test.idle`
Tests any connections in the eviction loop that are not being evicted. Only works if the time between eviction runs is greater than 0ms.
Default:

`-Ddse.ldap.pool.time.between.evictions`
Determines the time in ms (milliseconds) between eviction runs. When run with the `dse.ldap.pool.test.idle` this becomes a basic keep alive for connections.
Default:

`-Ddse.ldap.pool.num.tests.per.eviction`
Number of connections in the pool that are tested each connection run. If this is set the same as max active (the pool size) then all connections will be tested each eviction run.
Default:

`-Ddse.ldap.pool.min.evictable.idle.time.ms`
Determines the minimum time in ms (milliseconds) that a connection can sit in the pool before it becomes available for eviction.
Default:

`-Ddse.ldap.pool.soft.min.evictable.idle.time.ms`
Determines the minimum time in ms (milliseconds) that a connection can sit the pool before it becomes available for eviction with the proviso that the number of connections doesn't fall below `dse.ldap.pool.min.evictable.idle.time.ms`.
Default:

### Kerberos system properties

`-Ddse.sasl.protocol`
Kerberos principal name, `user@realm`. For example, `dse_admin@EXAMPLE.com`.

`-Djava.security.auth.login.config`
The path to the JAAS configuration file for `DseClient`.

### NodeSync system parameters

`-Ddse.nodesync.controller_update_interval_sec`
Set the frequency to execute NodeSync auto-tuning process in seconds.
Default: 300 (5 minutes).

`-Ddse.nodesync.log_reporter_interval_sec`
Set the frequency of short INFO progress report in seconds.
Default: 600 (10 minutes).

`-Ddse.nodesync.min_validation_interval_sec`
Set to the minimum number of seconds between validations of the same segment, mostly to avoid busy spinning on new/empty clusters.  
Default: \texttt{300} (5 minutes).

\texttt{-Ddse.nodesync.min\_warn\_interval\_sec} \nSet to the minimum number of seconds between logging warnings. \n\textbf{Tip:} Avoid logging warnings too often.  
Default: \texttt{36000} (10 hours).

\texttt{-Ddse.nodesync.rate\_checker\_interval\_sec} \nSet the frequency in seconds of comparing the current configured rate to tables and their deadline. Log a warning if rate considered too low.  
Default: \texttt{1800} (30 minutes).

\texttt{-Ddse.nodesync.segment\_lock\_timeout\_sec} \nSet the Time-to-live (TTL) on locks inserted in the status table in seconds.  
Default: \texttt{600} (10 minutes).

\texttt{-Ddse.nodesync.segment\_size\_target\_bytes} \nSet to the targeted maximum size for segments in bytes.  
Default: \texttt{26214400} (200 MB).

\texttt{-Ddse.nodesync.size\_checker\_interval\_sec} \nSet the frequency to check if the depth used for a table should be updated due to data size changes in seconds.  
Default: \texttt{7200} (2 hours).

\section*{Choosing a compaction strategy}

To implement the chosen compaction strategy:

1. To understand how compaction and compaction strategies work, read \textit{How is data maintained}?

2. Review your application's requirements use this information to answer the questions below.

3. Configure the table (page 167) to use the most appropriate strategy.

4. Test the compaction strategies (page 167) against your data.

Which compaction strategy is best?

The following questions are based on the experiences of developers and users with the strategies.

\textbf{Does your table process time series data?} 
If so, your best choice is \textit{Compaction strategiesTWCS}. If not, the following questions introduce other considerations to guide your choice.

\textbf{Does your table handle more reads than writes, or more writes than reads?} 
\textit{LCS} is a good choice if your table processes twice as many reads as writes or more — especially randomized reads. If the proportion of reads to writes is closer, the
performance hit exacted by LCS may not be worth the benefit. Be aware that LCS can be quickly overwhelmed by a high volume of writes.

**Does the data in your table change often?**

One advantage of LCS is that it keeps related data in a small set of SSTables. If your data is *immutable* or not subject to frequent *upserts*, STCS accomplishes the same type of grouping without the LCS performance hit.

**Do you require predictable levels of read and write activity?**

LCS keeps the SSTables within predictable sizes and numbers. For example, if your table’s read/write ratio is small, and it is expected to conform to a Service Level Agreements (SLAs) for reads, it may be worth taking the write performance penalty of LCS in order to keep read rates and latency at predictable levels. And you may be able to overcome this write penalty through horizontal scaling (adding more nodes).

**Will your table be populated by a batch process?**

On both batch reads and batch writes, STCS performs better than LCS. The batch process causes little or no fragmentation, so the benefits of LCS are not realized; batch processes can overwhelm LCS-configured tables.

**Does your system have limited disk space?**

LCS handles disk space more efficiently than STCS: it requires about 10% *headroom* in addition to the space occupied by the data is handles. STCS and DTCS generally require, in some cases, as much as 50% more than the data space. (DateTieredStorageStrategy (DTCS) is deprecated.)

**Is your system reaching its limits for I/O?**

LCS is significantly more I/O intensive than DTCS or STCS. Switching to LCS may introduce extra I/O load that offsets the advantages.

Configuring and running compaction

Set the compaction strategy for a table in the parameters for the `CREATE TABLE` or `ALTER TABLE` command. For details, see *table_options*.

You can start compaction manually using the `nodetool compact` (*page 915*) command.

Testing compaction strategies

Suggestions for determining which compaction strategy is best for your system:

- Create a three-node cluster using one of the compaction strategies, stress test the cluster using `cassandra-stress` (*page 1378*) and measure the results.
- Set up a node on your existing cluster and use the `write survey mode` to sample live data.

**NodeSync service**

**About NodeSync**

NodeSync is an easy to use continuous background repair that has low overhead and provides consistent performance and virtually eliminates manual efforts to run repair operations in a DataStax cluster.

- Continuously validates that data is in sync on all replica.
• Always running but low impact on cluster performance
• Fully automatic, no manual intervention needed
• Completely replace anti-entropy repairs

NodeSync service

By default, each node runs the NodeSync service. The service is idle unless it has something to validate. NodeSync is enabled on a per table basis. The service continuously validates local data ranges for NodeSync-enabled tables and repairs any inconsistency found. The local data ranges are split into small segments, which act as validation save points. Segments are prioritized in order to try to meet the per-table deadline target.

Segments

A segment is a small local token range of a table. NodeSync recursively splits local ranges in half a certain number of times (depth) to create segments. The depth is calculated using the total table size, assuming equal distribution of data. Typically segments cover no more than 200 MB. The token ranges can be no smaller than a single partition, so very large partitions can result in segments larger than the configured size.

Validation process and status

After a segment is selected for validation, NodeSync reads the entirety of the data it covers from all replica (using paging), checks for inconsistencies, and repairs if needed. When a node validates a segment, it “locks” it in a system table to avoid work duplication by other nodes. It is not a race-free lock; there is a possibility of duplicated work which saves the complexity and cost of true distributed locking.

Segment validation is saved on completion in the system_distributed.nodesync_status table, which is used internally for resuming on failure, prioritization, segment locking, and by tools. It is not meant to be read directly.

• Validation status is:
  # successful: All replicas responded and all inconsistencies (if any) were properly repaired.
  # full_in_sync: All replica were already in sync.
  # full_repaired: Some replica were repaired.
  # unsuccessful: Either some replicas did not respond or repairs on inconsistent replicas failed.
    # partial_in_sync: Not all replica responded, but all respondents were in sync.
    # partial_repaired: Not all replica responded, some that responded were repaired.
    # uncompleted: At most 1 node was available/responded; no validation happened.
    # failed: Some unexpected errors occurred. (Check the node logs.)
Note: If validation of a large segment is interrupted, increase the amount of redundant work.

Limitations

- For debugging/tuning, understanding of traditional repair will be mostly unhelpful, since NodeSync depends on the read repair path
- No special optimizations for remote DC - may perform poorly on particularly bad WAN links
- In aggregate, CPU consumption of NodeSync might exceed traditional repair
- NodeSync only makes internal adjustments to try to hit the configured rate - operators must ensure this configured throughput is sufficient to meet the gc_grace_seconds commitment and can be achieved by the hardware

Important: Tables with NodeSync enabled will be skipped for repair operations run against all or specific keyspaces. For individual tables, running the repair command will be rejected when NodeSync is enabled.

Starting and stopping the NodeSync service

The NodeSync service automatically starts with the dse cassandra (page 1198) command. You can manually start and stop the service on each node.

1. Verify the status of the NodeSync service:

   ```
   $ nodetool nodesyncservice status
   ```

   The output should indicate running.

   ```
   The NodeSync service is running
   ```

2. Disable the NodeSync service:

   ```
   $ nodetool nodesyncservice disable
   ```

   Note: On the next restart of DataStax Enterprise (DSE), the NodeSync service will start up.

3. Verify the status of the NodeSync service:

   ```
   $ nodetool nodesyncservice status
   ```

   The output should indicate not running.
Enabling NodeSync validation

By default, NodeSync is disabled when a table is created. It is also disabled on tables that were migrated from earlier versions. To continuously verify data consistency in the background without the need for anti-entropy repairs, enable NodeSync on one or more tables.

**Note:** Data only needs to be validated if the table is in more than one datacenter or is in a datacenter where the keyspace has a replication factor of 2 or more.

- Enable on an existing table:
  
  # Change the NodeSync setting on a single table using CQL syntax:
  ```
  ALTER TABLE table_name WITH
  nodesync={'enabled': 'true'};
  ```

  # All tables in a keyspace using `nodesync enable` *(page 1247)*:
  ```
  $ nodesync enable -v -k keyspace_name "*"
  ```

  # A list of tables using `nodesync enable` *(page 1247)*:
  ```
  $ nodesync enable keyspace_name.table_name keyspace_name.table_name
  ```

- Create a table with `nodesync` enabled:
  ```
  CREATE TABLE table_name ( column_list ) WITH
  nodesync={'enabled': 'true'};
  ```

Tuning validations

NodeSync tries to validate all tables within their respective deadlines, while respecting the configured rate limit. If a table is 10GB and has a `deadline_target_sec=10` and the `rate_in_kb` *(page 79)* is set to 1MB/sec, validation will not happen quickly enough. Configure the rate and deadlines realistically, take data sizes into account and adapt with data growth.

NodeSync records warnings to the system.log, if it detects any of the following conditions:

- `rate_in_kb` *(page 79)* is too low to validate all tables within their deadline, even under ideal circumstances.
• **rate\_in\_kb** *(page 79)* cannot be sustained by the node (too high for the node load/hardware).

### Setting the rate

**Estimating rate setting impact**

The *rate\_in\_kb* *(page 79)* sets the per node rate of the local NodeSync service. It controls the maximum number of bytes per second used to validate data. There is a fundamental tradeoff between how fast NodeSync validates data and how many resources it consumes. The rate is a limit on the amount of resources used and a target that NodeSync tries to achieve by auto-tuning internals. The set rate might not be achieved in practice, because validation can complete at a slower rate on new or small cluster or the node might temporarily or permanently lack available resources.

**Initial rate setting**

There is no strong requirement to keep all nodes validating at the same rate. Some nodes will simply validate more data than others. When setting the rate, use the simplest method first by using the defaults.

1. **Check the `rate\_in\_kb`** *(page 79)* setting within the `nodesync` section in the `cassandra.yaml` file.

2. **Try increasing or decreasing the value at run time:**
   ```
   $ nodetool nodesyncservice setrate value\_in\_kb\_sec
   ```

3. **Check the configured rate.**
   ```
   $ nodetool nodesyncservice getrate
   ```

   **Tip:** The configured rate is different from the effective rate, which can be found in the NodeSync Service metrics.

**Simulating rates**

When adjusting rates, use the NodeSync rate simulator *(page 1022)* to help determine the configuration settings by computing the rate necessary to validate all tables within their allowed deadlines.

Unfortunately, no perfect value exists because NodeSync also deals with many unknown or difficult to predict factors, such as:

- **Failures** - When a node fails, it does not participate in NodeSync validation while it is offline.
- **Temporary overloads** - During periods of overload, such as an unexpected events, nodes can not achieve the configured rate.
- **Data size variation** - The rate required to repair all tables within a fixed amount of time directly depends on the size of the data to validate, which is typically a moving target.
All these factors can impact the overall NodeSync rate. Therefore build safety margins within the configured rate. The NodeSyncServiceRate simulator helps to set the rate.

**Setting the deadline**

Each table with NodeSync enabled has a `deadline_target_sec` property. This is the target for the maximum time between 2 validations of the same data. As long as the deadline is met, all parts of the ring (for the table) are validated at least that often.

The deadline (`deadline_target_sec`) relates to grace period (`gc_grace_seconds`). The deadline should always be less than or equal to the grace period. As long as the deadline is met, no data is resurrected due to tombstone purging.

The deadline defaults to which ever is longer, the grace period or four days. Typically an acceptable default, unless the table has a grace period of zero. For testing, the deadline value can be less than the grace period. Verify for a few weeks if a lower `gc_grace` value is realistic without taking risk before changing it.

NodeSync prioritize segments in order to try to meet the deadline. The next segment to validate at any given time is the one the closest to missing its deadline. For example, if table 1 has half the deadline of table 2, table 1 validates approximately twice as often as table 2.

Use OpsCenter to get a graphical representation of the NodeSync validation status. See Viewing NodeSync Status.

The syntax to change the per-table `nodesync` property:

```
ALTER TABLE table_name
WITH nodesync = { 'enabled': 'true',
    'deadline_target_sec': value }
```

**Manually starting validation**

Force NodeSync to repair specific segments. After a user validation is submitted, it takes precedence over normal NodeSync work. Normal work resumes automatically after the validation finishes.

This is an advanced tool. Usually, it is better to let NodeSync prioritize segments on its own.

- Submitting user validations:
  
  ```
  $ nodesync validation submit keyspace_name.table_name
  ```

- Listing user validations:
  
  ```
  $ nodesync validation list
  ```

- Canceling user validations:
  
  ```
  $ nodesync validation cancel validation_id
  ```
Configuring Virtual Nodes

Virtual node (vnode) configuration

Virtual nodes simplify many tasks in DSE, such as eliminating the need to determine the partition range (calculate and assign tokens), rebalancing the cluster when adding or removing nodes, and replacing dead nodes. For a complete description of virtual nodes and how they work, see Virtual nodes.

DSE requires the same token architecture on all nodes in a datacenter. The nodes must all be vnode-enabled or single-token architecture. Across the entire cluster, datacenter architecture can vary. For example, a single cluster with:

- A transaction-only datacenter running OLTP.
- A single-token architecture search datacenter (no vnodes).
- An analytics datacenter with vnodes.

Guidelines for using virtual nodes

- DSE requires the same token architecture on all nodes in a datacenter.

  The nodes must all be vnode-enabled or single-token architecture. Across the entire cluster, datacenter architecture can vary.

  For example, a single cluster with:

  

  # A transaction-only datacenter running OLTP.
  # A single-token architecture search datacenter (no vnodes).
  # An analytics datacenter with vnodes.

- DataStax recommends using 8 vnodes (tokens).

  **Restriction:** DataStax recommends not using vnodes with DSE Search. However, if you decide to use vnodes with DSE Search, do not use more than 8 vnodes and ensure that `allocate_tokens_for_local_replication_factor` (page 81) option in `cassandra.yaml` is correctly configured for your environment.

  Using 8 vnodes distributes the workload between systems with a ~10% variance and has minimal impact on performance.

- Ensure correct vnode configuration with `cassandra.yaml` settings:

  

  # When adding a vnode to an existing cluster or setting up nodes in a new datacenter, set the target replication factor (RF) of keyspaces in the datacenter with the `allocate_tokens_for_local_replication_factor` (page 81) option.
  # The allocation algorithm distributes the token ranges proportionately using the `num_tokens` (page 80) settings.
Configuration

All systems in the datacenter should have the same `num_token` settings unless the systems performance varies between systems. To distribute more of the workload to the higher performance hardware, increase the number of tokens for those systems.

The allocation algorithm efficiently balances the workload using fewer tokens; when systems are added to a datacenter, the algorithm maintains the balance. Using a higher number of tokens more evenly distributes the workload, but also significantly increases token management overhead.

Set the number of vnode tokens based on the workload distribution requirements of the datacenter:

Table 5: Allocation algorithm workload distribution variance

<table>
<thead>
<tr>
<th>Replication factor</th>
<th>4 vnode (tokens)</th>
<th>8 vnode (tokens)</th>
<th>64 vnode (tokens)</th>
<th>128 vnode (tokens)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>~17.5%</td>
<td>~12.5%</td>
<td>~3%</td>
<td>~1%</td>
</tr>
<tr>
<td>3</td>
<td>~14%</td>
<td>~10%</td>
<td>~2%</td>
<td>~1%</td>
</tr>
<tr>
<td>5</td>
<td>~11%</td>
<td>~7%</td>
<td>~1%</td>
<td>~1%</td>
</tr>
</tbody>
</table>

- Add nodes to the cluster one at a time.

Caution: When adding multiple nodes to the cluster using the allocation algorithm, ensure that nodes are added one at a time. If nodes are added concurrently, the algorithm assigns the same tokens to different nodes.

Enabling vnodes

In the cassandra.yaml file:

1. Uncomment `num_tokens (page 80)` and set the required number of tokens.

2. (Recommended) To use the allocation algorithm uncomment `allocate_tokens_for_local_replication_factor (page 81)` and set it to the target replication factor for the keyspaces in the datacenter. If the replication varies, alternate between the replication factor (RF) settings.

3. Comment out the `initial_token (page 80)` or leave unset.

To upgrade existing clusters to vnodes, see Enabling virtual nodes on an existing production cluster.

Disabling vnodes

Important: If you do not use vnodes, you must make sure that each node is responsible for roughly an equal amount of data. To ensure that each node is responsible for an equal amount of data, assign each node an `initial-token (page`
value and calculate the tokens for each datacenter as described in Generating tokens.

1. In the cassandra.yaml file:
   a. Comment out the num_tokens (page 80) and allocate_tokens_for_local_replication_factor (page 81).
   b. Uncomment the initial_token (page 80) and set it to 1 or to the value of a generated token for a multi-node cluster.
Using DataStax Enterprise advanced functionality

Information on using DSE Analytics, DSE Search, DSE Graph, DSEFS (DataStax Enterprise file system), and DSE Advanced Replication.

DSE Analytics

DataStax Enterprise (DSE) integrates real-time and batch operational analytics capabilities with an enhanced version of Apache Spark™. With DSE Analytics you can easily generate ad-hoc reports, target customers with personalization, and process real-time streams of data. The analytics toolset lets you write code once and then use it for both real-time and batch workloads.

About DSE Analytics

DataStax Enterprise (DSE) integrates real-time and batch operational analytics capabilities with an enhanced version of Apache Spark™. With DSE Analytics you can easily generate ad-hoc reports, target customers with personalization, and process real-time streams of data. The analytics toolset lets you write code once and then use it for both real-time and batch workloads.

DSE Analytics jobs can use the DataStax Enterprise File System (DSEFS) to handle the large data sets typical of analytic processing. DSEFS replaces CFS (Cassandra File System).

DSE Analytics features

No single point of failure
DSE Analytics supports a peer-to-peer, distributed cluster for running Spark jobs. Being peers, any node in the cluster can load data files, and any analytics node can assume the responsibilities of Spark Master.

Spark Master management
DSE Analytics provides automatic Spark Master management.

Analytics without ETL
Using DSE Analytics, you run Spark jobs directly against data in the database. You can perform real-time and analytics workloads at the same time without one workload affecting the performance of the other. Starting some cluster nodes as Analytics nodes and others as pure transactional real-time nodes automatically replicates data between nodes.

DataStax Enterprise file system (DSEFS (page 285))
DSEFS (DataStax Enterprise file system) is a fault-tolerant, general-purpose, distributed file system within DataStax Enterprise. It is designed for use cases that need to leverage a distributed file system for data ingestion, data staging, and state management for Spark Streaming applications (such as checkpointing or write-ahead logging). DSEFS is similar to HDFS, but avoids the deployment complexity
and single point of failure typical of HDFS. DSEFS is HDFS-compatible and is designed to work in place of HDFS in Spark and other systems.

**DSE Analytics Solo**

DSE Analytics Solo *(page 180)* datacenters are devoted entirely to DSE Analytics processing, for deployments that require separation of analytics jobs from transactional data.

**Integrated security**

DSE Analytics uses the advanced security features of DSE, simplifying configuration and deployment.

**AlwaysOn SQL**

*AlwaysOn SQL*(page 257) is a highly-available service that provides JDBC and ODBC interfaces to applications accessing DSE Analytics data.

Enabling DSE Analytics

To enable Analytics, follow the architecture guidelines for choosing a workload type for the datacenters in the cluster.

### Setting the replication factor for analytics keyspaces

Keyspaces and tables are automatically created when DSE Analytics nodes are started for the first time. The replication factor must be adjusted for these keyspaces in order for the analytics features to work properly and to avoid data loss.

The keyspaces used by DSE Analytics are the following:

- `dse_analytics`
- `dse_leases`
- `dsefs`
- "HiveMetaStore"

All analytics keyspaces are initially created with the SimpleStrategy replication strategy and a replication factor (RF) of 1. Each of these must be updated in production environments to avoid data loss. After starting the cluster, alter the keyspace to use the NetworkTopologyStrategy replication strategy with an appropriate settings for the replication factor and datacenters. For most environments using DSE Analytics, a suitable replication factor will be either 3 or the cluster size, whichever is smaller.

For example, use a CQL statement to configure the `dse_leases` keyspace for a replication factor of 3 in both DC1 and DC2 datacenters using NetworkTopologyStrategy:

```cql
ALTER KEYSPACE dse_leases
WITH REPLICAICATION = {
    'class': 'NetworkTopologyStrategy',
    'DC1': '3',
    'DC2': '3'
};
```

The datacenter name used is case-sensitive. If needed, use the `dsetool status` command to confirm the exact datacenter spelling.
After adjusting the replication factor, `nodetool repair` must be run on each node in the affected datacenters. For example to repair the altered keyspace `dse_leases`:

```
$ nodetool repair -full dse_leases
```

Repeat the above steps for each of the analytics keyspaces listed above. For more information see Changing keyspace replication strategy.

### DSE Analytics and Search integration

An integrated DSE SearchAnalytics cluster allows analytics jobs to be performed using CQL queries. This integration allows finer-grained control over the types of queries that are used in analytics workloads, and improves performance by reducing the amount of data that is processed. However, a DSE SearchAnalytics cluster does not provide workload isolation and there are no detailed guidelines for provisioning and performance in production environments.

Nodes that are started in SearchAnalytics mode allow you to create analytics queries that use DSE Search indexes. These queries return RDDs that are used by Spark jobs to analyze the returned data.

The following code shows how to use a DSE Search query from the DSE Spark console.

```scala
val table = sc.cassandraTable("music","solr")
val result = table.select("id","artist_name")
  .where("solr_query='artist_name:Miles*'")
  .take(10)
```

You can use Spark Spark Datasets/DataFrames instead of RDDs.

```scala
val table = spark.read.format("org.apache.spark.sql.cassandra")
  .options(Map("keyspace"->"music", "table" -> "solr"))
  .load()
val result =
  table.select("id","artist_name").where("solr_query='artist_name:Miles*'")
  .show(10)
```

You may alternately use a Spark SQL query.

```scala
val result = spark.sql("SELECT id, artist_name FROM music.solr WHERE solr_query = 'artist_name:Miles*' LIMIT 10")
```

For a detailed example, see Running the Wikipedia demo with SearchAnalytics (page 279).

### Configuring a DSE SearchAnalytics cluster

1. Create DSE SearchAnalytics nodes in a mixed-workload cluster, as described in Initializing a single datacenter per workload type.
The name of the datacenter is set to SearchAnalytics when using the DseSimpleSnitch. Do not modify existing search or analytics nodes that use DseSimpleSnitch to be SearchAnalytics nodes. If you use another snitch like GossipingPropertyFileSnitch you can have a mixed workload within a datacenter.

2. Perform load testing to ensure your hardware has enough CPU and memory for the additional resource overhead that is required by Spark and Solr.

   **Note:** SearchAnalytics nodes always use driver paging settings. See [Using pagination (cursors) with CQL Solr queries](#).

SearchAnalytics nodes might consume more resources than search or analytics nodes. Resource requirements of the nodes greatly depend on the type of query patterns you are using.

Considerations for DSE SearchAnalytics clusters

Care should be taken when enabling both Search and Analytics on a DSE node. Since both workloads will be enabled, ensure proper resources are provisioned for these simultaneous workloads. This includes sufficient memory and compute resources to accommodate the specific indexing, query, and processing appropriate to the use case.

SearchAnalytics clusters are appropriate for production environments, provided these environments provide sufficient resources for the specific workload, as is the case for all DSE clusters.

All of the fields that are queried on DSE SearchAnalytics clusters must be defined in the search index schema definition (page 324). Fields that are not defined in the search index schema columns are excluded from the results returned from Spark queries.

**Using predicate push down on search indexes in Spark SQL**

Search predicate push down allows queries in SearchAnalytics datacenters to use Solr-indexed columns in Spark SQL queries. To enable Search predicate push down, set the `spark.sql.dse.search.enableOptimization` property to `on` or `auto`. By default, `spark.sql.dse.search.enableOptimization` is set to `auto`.

When in auto mode the predicate push down will do a COUNT operation against the Search indices both with and without the predicate filters applied. If the number of records with the predicate filter is less than the result of the following formula:

\[
\text{spark.sql.dse.search.autoRatio} \times \text{the total number of records}
\]

the optimization occurs automatically.

The property `spark.sql.dse.search.autoRatio` is user configurable. The default value is 0.03.
Using DataStax Enterprise advanced functionality

The performance of DSE Search is directly related to the number of records returned in a query. Requests which require a large portion of the dataset are likely better served by a full table scan without using predicate push downs.

To enable Solr predicate push down on a Scala dataset:

```scala
val solrEnabledDataSet = spark.read
  .format("org.apache.spark.sql.cassandra")
  .options(Map(
    "keyspace" -> "ks",
    "table" -> "tab",
    "spark.sql.dse.search.enableOptimization" -> "on")
  ).load()
```

To create a temporary table in Spark SQL with Solr predicate push down enabled:

```sql
CREATE TEMPORARY TABLE temp USING org.apache.spark.sql.cassandra OPTIONS (
  table "tab",
  keyspace "ks",
  spark.sql.dse.search.enableOptimization "on");
```

Set the `spark.sql.dse.search.enableOptimization` property globally by adding it to the server configuration file (page 224).

The optimizer works on the push down level so only predicates which are being pushed to the source can be optimized. Use the `explain` command to see exactly what predicates are being pushed to the `CassandraSourceRelation`.

```scala
val query = spark.sql("query")
query.explain
```

Logging optimization plans

The optimization plans for a query using predicate push downs are logged by setting the `org.apache.spark.sql.SolrPredicateRules` logger to `DEBUG` in the Spark logging configuration files (page 229).

```xml
<logger name="org.apache.spark.sql.SolrPredicateRules" level="DEBUG"/>
```

About DSE Analytics Solo

DSE Analytics Solo datacenters provide analytics processing with Spark and distributed storage using DSEFS without storing transactional database data.

DataStax Enterprise is flexible when deploying analytic processing in concert with transactional workloads. There are two main ways to deploy DSE Analytics: collocated with the database processing nodes, and on segregated machines in their own datacenter.
Traditional DSE Analytics deployments have both the DataStax database process and the Spark process running on the same machine. This allows for simple deployment of analytic processing when the analysis is not as intensive, or the database is not as heavily used.
Using DataStax Enterprise advanced functionality

DSE Analytics Solo allows customers to deploy DSE Analytics processing on segregated hardware configurations in a different datacenter from the transactional DSE nodes. This ensures consistent behavior of both engines in a configuration that does not compete for computer resources. This configuration is good for processing-intensive analytic workloads.

DSE Analytics Solo allows the flexibility to have more nodes dedicated to data processing than are used for database transactions. This is particularly good for situations where the processing needs far exceed the transactional resource needs. For example, suppose you have a Spark Streaming job that will analyze and filter 99.9% of the incoming data, storing only a few records after analysis. The resources required by the transactional datacenter are much smaller than the resources required to analyze the data.

DSE Analytics Solo is more elastic in terms of scaling up, or down, the analytic processing in the cluster. This is particularly useful when you need extra analytics processing, such as end of the day or end of the quarter surges in analytics jobs. Since a DSE Analytics Solo node does not store database data, when new nodes are added to a cluster there is very little data moved across the network to the new nodes. In an analytics and transactional collocated environment, adding a node means moving transactional data between the existing nodes and the new nodes.

For information on creating a DSE Analytics Solo datacenter, see Creating a DSE Analytics Solo datacenter (page 238).

Analyzing data using Spark

Spark is the default mode when you start an analytics node in a packaged installation.

About Spark

Apache Spark is a framework for analyzing large data sets across a cluster, and is enabled when you start an Analytics node. Spark runs locally on each node and executes in memory when possible. Spark uses multiple threads instead of multiple processes to achieve parallelism on a single node, avoiding the memory overhead of several JVMs.

Apache Spark integration with DataStax Enterprise includes:

- Spark Cassandra Connector (page 211) for accessing data stores in DSE
- DSE Resource Manager for managing (page 215) Spark components in a DSE cluster
- Spark Job Server (page 269)
- Spark SQL (page 244) support
- AlwaysOn SQL (page 257)
- Spark SQL Thrift Server (page 254)
- Spark streaming (page 241)
- DataFrames (page 253) API to manipulate data within Spark
- SparkR integration (page 256)
Spark architecture

The software components for a single DataStax Enterprise analytics node are:

- Spark Worker
- DataStax Enterprise File System (DSEFS)
- The database

A Spark Master acts purely as a resource manager for Spark applications. Spark Workers launch executors that are responsible for executing part of the job that is submitted to the Spark Master. Each application has its own set of executors. Spark architecture is described in the Apache documentation.

DSE Spark nodes use a different resource manager than standalone Spark nodes. The DSE Resource Manager simplifies integration between Spark and DSE. In a DSE Spark cluster, client applications use the CQL protocol to connect to any DSE node, and that node redirects the request to the Spark Master.

The communication between the Spark client application (or driver) and the Spark Master is secured the same way as connections to DSE, which means that plain password authentication as well as Kerberos authentication is supported, with or without SSL encryption. Encryption and authentication can be configured per application, rather than per cluster. Authentication and encryption between the Spark Master and Worker nodes can be enabled or disabled regardless of the application settings.

Spark supports multiple applications. A single application can spawn multiple jobs and the jobs run in parallel. An application reserves some resources on every node and these resources are not freed until the application finishes. For example, every session of Spark shell is an application that reserves resources. By default, the scheduler tries allocate the application to the highest number of different nodes. For example, if the application declares that it needs four cores and there are ten servers, each offering two cores, the application most likely gets four executors, each on a different node, each consuming a single core. However, the application can get also two executors on two different nodes, each consuming two cores. You can configure the application scheduler. Spark Workers and Spark Master are part of the main DSE process. Workers spawn executor JVM processes which do the actual work for a Spark application (or driver). Spark executors use native integration to access data in local transactional nodes through the Open Source Spark-Cassandra Connector. The memory settings for the executor JVMs are set by the user submitting the driver to DSE.

In deployment for each Analytics datacenter one node runs the Spark Master, and Spark Workers run on each of the nodes. The Spark Master comes with automatic high availability (page 184).
As you run Spark, you can access data in the Hadoop Distributed File System (HDFS), or the DataStax Enterprise File System (DSEFS) by using the URL for the respective file system.

**Highly available Spark Master**

The Spark Master High Availability mechanism uses a special table in the `dse_analytics` keyspace to store information required to recover Spark workers and the application. Unlike the high availability mechanism mentioned in Spark documentation, DataStax Enterprise does not use ZooKeeper.
If the original Spark Master fails, the reserved one automatically takes over. To find the current Spark Master, run:

```
$ dse client-tool spark leader-address
```

DataStax Enterprise provides Automatic Spark Master management *(page 225)*.

### Unsupported features

The following Spark features and APIs are not supported:

- Writing to blob columns from Spark
  
  Reading columns of all types is supported; however, you must convert collections of blobs to byte arrays before serializing.

### Using Spark with DataStax Enterprise

DataStax Enterprise integrates with Apache Spark to allow distributed analytic applications to run using database data.

### Starting Spark

Before you start Spark, configure Authorizing remote procedure calls (RPC) for the `DseClientTool` object.

**Note:** RPC permission for the `DseClientTool` object is required to run Spark because the `DseClientTool` object is called implicitly by the Spark launcher.

**Note:** By default DSEFS is required to execute Spark applications. DSEFS should not be disabled when Spark is enabled on a DSE node. If there is a strong reason not to use DSEFS as the default file system, reconfigure Spark to use a different file system. For example to use a local file system set the following properties in `spark-daemon-defaults.conf`:

```
spark.hadoop.fs.defaultFS=file:///  
spark.hadoop.hive.metastore.warehouse.dir=file:///tmp/warehouse
```

How you start Spark depends on the installation and if you want to run in Spark mode or SearchAnalytics mode:

**Package installations:**

To start the Spark trackers on a cluster of analytics nodes, edit the `/etc/default/dse` file to set `SPARK_ENABLED` to 1.

When you start DataStax Enterprise as a service *(page 1437)*, the node is launched as a Spark node. You can enable additional components.
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Mode</th>
<th>Option in /etc/default/dse</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td>SPARK_ENABLED=1</td>
<td>Start the node in Spark mode.</td>
</tr>
<tr>
<td>SearchAnalytics mode</td>
<td>SPARK_ENABLED=1 SEARCH_ENABLED=1</td>
<td>SearchAnalytics mode requires testing in your environment before it is used in production clusters. In dse.yaml, <code>cql_solr_query_paging: driver (page 119)</code> is required.</td>
</tr>
</tbody>
</table>

**Tarball installations:**

To start the Spark trackers on a cluster of analytics nodes, use the `-k` option:

```
$ installation_location/bin/dse cassandra -k
```

**Note:** Nodes started with `-k` are automatically assigned to the default Analytics datacenter if you do not configure a datacenter in the snitch property file.

You can enable additional components:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td><code>-k</code></td>
<td>Start the node in Spark mode.</td>
</tr>
<tr>
<td>SearchAnalytics mode</td>
<td><code>-k -s</code></td>
<td>In dse.yaml, <code>cql_solr_query_paging: driver (page 119)</code> is required.</td>
</tr>
</tbody>
</table>

For example:

To start a node in SearchAnalytics mode, use the `-k` and `-s` options.

```
$ installation_location/bin/dse cassandra -k -s
```

Starting the node with the Spark option starts a node that is designated as the master, as shown by the Analytics(SM) workload in the output of the `dsetool ring` command:

```
$ dsetool ring

Address            DC       Graph Status State Load Owns Workload Workload Workload
                  DC       Graph Status State Load       Token Health [0,1]
10.200.175.149     Analytics no     Up Normal 185 KiB ?       0.90
-9223372036854775808
10.200.175.148     Analytics no     Up Normal 194.5 KiB ? 0.90
                            0.90

Note: you must specify a keyspace to get ownership information.
```
If you use `sudo` to start DataStax Enterprise, remove the `~/.spark` directory before you restart the cluster:

```bash
$ sudo rm -r ~/.spark
```

Launching Spark

After starting a Spark node, use `dse` commands to launch Spark.

Usage:

Package installations: `dse spark`

Tarball installations: `installation_location/bin/dse spark`

You can use Cassandra specific properties (page 235) to start Spark. Spark binds to the `listen_address` that is specified in cassandra.yaml.

DataStax Enterprise supports these commands for launching Spark on the DataStax Enterprise command line:

**dse spark**

Enters interactive Spark shell, offers basic auto-completion.

- Package installations: `dse spark`
- Tarball installations: `installation_location/bin/dse spark`

**dse spark-submit**

Launches applications on a cluster like `spark-submit`. Using this interface you can use Spark cluster managers without the need for separate configurations for each application. The syntax for package installations is:

```bash
$ dse spark-submit --class class_name jar_file other_options
```

For example, if you write a class that defines an option named d, enter the command as follows:

```bash
$ dse spark-submit --class com.datastax.HttpSparkStream target/HttpSparkStream.jar -d $NUM_SPARK_NODES
```

Note: The JAR file can be located in a DSEFS directory. If the DSEFS cluster is secured, provide authentication credentials as described in DSEFS authentication (page 298).

The `dse spark-submit` command supports the same options as Apache Spark's `spark-submit`. For example, to submit an application using cluster mode using the `supervise` option to restart in case of failure:
Using DataStax Enterprise advanced functionality

$ dse spark-submit --deploy-mode cluster --supervise --class com.datastax.HttpSparkStream target/HttpSparkStream.jar -d $NUM_SPARK_NODES

**Note:** The directory in which you run the dse Spark commands must be writable by the current user.

Internal authentication is supported.

Use the optional environment variables **DSE_USERNAME** and **DSE_PASSWORD** to increase security and prevent the user name and passwords from appearing in the Spark log files or in the process list on the Spark Web UI. To specify a user name and password using environment variables, add the following to your Bash `.profile` or `.bash_profile`:

```bash
export DSE_USERNAME=user
export DSE_PASSWORD=secret
```

These environment variables are supported for all Spark and dse client-tool *(page 1225)* commands.

**Note:** DataStax recommends using the environment variables instead of passing user credentials on the command line.

You can provide authentication credentials in several ways, see [Credentials for authentication](#).

**Specifying Spark URLs**

You do not need to specify the Spark Master address when starting Spark jobs with DSE. If you connect to any Spark node in a datacenter, DSE will automatically discover the Master address and connect the client to the Master.

Specify the URL for any Spark node using the following format:

```
dse://[Spark node address[(:port number)]][?parameter name=parameter value;]...
```

By default the URL is `dse://?`, which is equivalent to `dse://localhost:9042`. Any parameters you set in the URL will override the configuration read from DSE's Spark configuration settings.

You can specify the work pool in which the application will be run by adding the `workpool=work pool name` as a URL parameter. For example, `dse://1.1.1.1:123?workpool=workpool2`.

**Valid parameters are** `CassandraConnectorConf` settings with the `spark.cassandra.` prefix stripped. For example, you can set the `spark.cassandra.connection.local_dc` option to `dc2` by specifying `dse://?connection.local_dc=dc2`.  

---

---
Or to specify multiple spark.cassandra.connection.host addresses for high-availability if the specified connection point is down: dse://1.1.1.1:123?
connection.host=1.1.2.2,1.1.3.3.

If the connection.host parameter is specified, the host provided in the standard URL is prepended to the list of hosts set in connection.host. If the port is specified in the standard URL, it overrides the port number set in the connection.port parameter.

Connection options when using dse spark-submit are retrieved in the following order: from the Master URL, then the Spark Cassandra Connector options, then the DSE configuration files.

Detecting Spark application failures

DSE has a failure detector for Spark applications, which detects whether a running Spark application is dead or alive. If the application has failed, the application will be removed from the DSE Spark Resource Manager.

The failure detector works by keeping an open TCP connection from a DSE Spark node to the Spark Driver in the application. No data is exchanged, but regular TCP connection keep-alive control messages are sent and received. When the connection is interrupted, the failure detector will attempt to reacquire the connection every 1 second for the duration of the appReconnectionTimeoutSeconds timeout value (5 seconds by default). If it fails to reacquire the connection during that time, the application is removed.

A custom timeout value is specified by adding appReconnectionTimeoutSeconds=value in the master URI when submitting the application. For example to set the timeout value to 10 seconds:

$ dse spark --master dse://?appReconnectionTimeoutSeconds=10

Running Spark commands against a remote cluster

To run Spark commands against a remote cluster, you must export the DSE configuration from one of the remote nodes to the local client machine.

To run a driver application remotely, there must be full public network communication between the remote nodes and the client machine.

1. Export the DataStax Enterprise client configuration from the remote node to the client node:

   a. On the remote node:

   $ dse client-tool configuration export dse-config.jar

   b. Copy the exported JAR to the client nodes.
Using DataStax Enterprise advanced functionality

$ scp dse-config.jar user@clientnode1.example.com:

c. On the client node:

$ dse client-tool configuration import dse-config.jar

2. Run the Spark command against the remote node.

$ dse spark-submit submit options myApplication.jar

To set the driver host to a publicly accessible IP address, pass in the spark.driver.host option.

$ dse spark-submit --conf spark.driver.host=IP address myApplication.jar

Accessing database data from Spark

DataStax Enterprise integrates Spark with DataStax Enterprise database. Database tables are fully usable from Spark.

Accessing the database from a Spark application
To access the database from a Spark application, follow instructions in the Spark example Portfolio Manager demo using Spark (page 271).

Accessing database data from the Spark shell
DataStax Enterprise uses the Spark Cassandra Connector to provide database integration for Spark. By running the Spark shell in DataStax Enterprise, you have access to enriched Spark context objects for accessing transactional nodes directly. See the Spark Cassandra Connector Java Doc on GitHub.

To access database data from the Spark Shell, just run the dse spark command and follow instructions in subsequent sections.

$ dse spark

Creating a new Spark Session
Spark context Web UI available at http://10.0.0.1:4041
Spark Context available as 'sc' (master = dse://?, app id = app-20180406174903-0008).
Spark Session available as 'spark'.
Spark SqlContext (Deprecated use Spark Session instead) available as 'sqlContext'
Welcome to

/__/ __ __ __/__/
The Spark Shell creates a default Spark session named `spark`, an instance of `org.apache.spark.sql.SparkSession`.


Note:

In previous versions of DSE, the default `HiveContext` instance was named `hc`. If your application uses `hc` instead of `sqlContext`, you can work around this change by adding a line:

```scala
val hc = sqlContext
```

Previous versions also created a `CassandraSqlContext` instance named `csc`. Starting in DSE 5.0, this is no longer the case. Use the `sqlContext` object instead.

Using the Spark session

A Spark session is encapsulated in an instance of `org.apache.spark.sql.SparkSession`. The session object has information about the Spark Master, the Spark application, and the configuration options.

The DSE Spark shell automatically configures and creates a Spark session object named `spark`. Use this object to begin querying database tables in DataStax Enterprise.

```scala
spark.sql("SELECT * FROM keyspace.table_name")
```

Note:

In Spark 1.6 and earlier, there were separate `HiveContext` and `SQLContext` objects. Starting in Spark 2.0, the `SparkSession` encapsulates both.

Spark applications can use multiple sessions to use different underlying data catalogs. You can use an existing Spark session to create a new session by calling the `newSession` method.
Using DataStax Enterprise advanced functionality

```scala
val newSpark = spark.newSession()
```

**Building a Spark session using the Builder API**

The Builder API allows you to create a Spark session manually.

```scala
import org.apache.spark.sql.SparkSession
val sparkSession = SparkSession.builder
  .master("dse://localhost?")
  .appName("my-spark-app")
  .enableHiveSupport()
  .config("spark.executor.logs.rolling.maxRetainedFiles", "3")
  .config("spark.executor.logs.rolling.strategy", "size")
  .config("spark.executor.logs.rolling.maxSize", "50000")
  .getOrCreate
```

**Stopping a Spark session**

Use the `stop` method to end the Spark session.

```scala
spark.stop
```

**Getting and setting configuration options**

Use the `spark.conf.get` and `spark.conf.set` methods to retrieve or set Spark configuration options for the session.

```scala
spark.conf.set("spark.executor.logs.rolling.maxRetainedFiles", "3")
spark.conf.get("spark.executor.logs.rolling.maxSize")
```

**Using the Spark context**

**Note:** Starting in DSE 5.1, the entry point for Spark applications is the `SparkSession` object ([page 191](#)). Using the Spark context directly is deprecated and may be removed in future releases.

Access the deprecated context object, call `spark.sparkContext`.

```scala
val sc = spark.sparkContext
```

To get a Spark RDD that represents a database table, load data from a table into Spark using the sc-dot (sc.) syntax to call the `cassandraTable` method on the Spark context, where `sc` represents the Spark API `SparkContext` class.

```scala
sc.cassandraTable ( "keyspace", "table name" )
```

By default, the DSE Spark shell creates an `sc` object. The Spark context can be manually retrieved from the Spark `session` object in the Spark shell by calling `spark.sparkContext`.  

---

**Page 192**

DSE 6.7 Developer Guide (Latest version)**
val sc = spark.sparkContext()

Data is mapped into Scala objects and DataStax Enterprise returns a `CassandraRDD[CassandraRow]`. To use the Spark API for creating an application that runs outside DataStax Enterprise, import `com.datastax.spark.connector.SparkContextCassandraFunctions`.

The following example shows how to load a table into Spark and read the table from Spark.

1. Create this keyspace and table in using `cqlsh`. Use the Analytics datacenter to create the keyspace.

   ```
   CREATE KEYSPACE test WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'Analytics' : 1};
   CREATE TABLE test.words (word text PRIMARY KEY, count int);
   ```

   This example assumes you start a single-node cluster in Spark mode (page 185).

2. Load data into the `words` table.

   ```
   INSERT INTO test.words (word, count) VALUES ('foo', 10);
   INSERT INTO test.words (word, count) VALUES ('bar', 20);
   ```

3. Assuming you started the node in Spark mode, start the Spark shell. Do not use `sudo` to start the shell.

   ```
   $ bin/dse spark
   ```

   The Welcome to Spark output and prompt appears.

4. Use the `showSchema` command to view the user keyspaces and tables.

   ```
   :showSchema
   ```

   Information about all user keyspaces appears.

   ```
   ==============================  
   Keyspace: HiveMetaStore  
   ==============================  
   Table: MetaStore  
   ==============================  
   - key : String (partition key column)  
   - entity : String (clustering column)  
   - value : java.nio.ByteBuffer
   ```
5. Get information about only the **test** keyspace.

:showSchema test

```
Keyspace: test
------------------------
Table: words
------------------------
- word : String (partition key column)
- count : Int
```

6. Get information about the **words** table.

:showSchema test words

```
Keyspace: test
------------------------
Table: words
------------------------
- word : String (partition key column)
- count : Int
```
7. Define a base RDD to point to the data in the `test.words` table.

```scala
val rdd = sc.cassandraTable("test", "words")
```

The RDD is returned in the `rdd` value. To read the table, use this command.

```scala
rdd.toArray.foreach(println)
```

```scala
CassandraRow{word: bar, count: 20}
CassandraRow{word: foo, count: 10}
```

Now, you can use methods on the returned RDD to query the `test.words` table.

**Python support for loading cassandraTables**

Python supports loading `cassandraTables` from a Spark streaming context and saving a `DStream` to the database.

**Reading column values**

You can read columns in a table using the `get` methods of the `CassandraRow` object. The `get` methods access individual column values by column name or column index. Type conversions are applied on the fly. Use `getOption` variants when you expect to receive null values.

Continuing with the previous example, follow these steps to access individual column values.

1. Store the first item of the RDD in the `firstRow` value.

```scala
val firstRow = rdd.first
```

```scala
firstRow: com.datastax.spark.connector.CassandraRow = CassandraRow{word: foo, count: 10}
```

2. Get the column names.

```scala
rdd.columnNames
```

```scala
res3: com.datastax.spark.connector.ColumnSelector = AllColumns
```

3. Use a generic get to query the table by passing the return type directly.
Using DataStax Enterprise advanced functionality

```scala
firstRow.get[Int]("count")
res4: Int = 10

firstRow.get[Long]("count")
res5: Long = 10

firstRow.get[BigInt]("count")
res6: BigInt = 10

firstRow.get[java.math.BigInteger]("count")
res7: java.math.BigInteger = 10

firstRow.get[Option[Int]]("count")
res8: Option[Int] = Some(10)

firstRow.get[Option[BigInt]]("count")
res9: Option[BigInt] = Some(10)
```

Reading collections

You can read collection columns in a table using the get methods of the `CassandraRow` object. The get methods access the collection column and returns a corresponding Scala collection.

Assuming you set up the test keyspace earlier, follow these steps to access a collection.

1. In the test keyspace, set up a collection set using `cqlsh`.
   ```sql
   CREATE TABLE test.users (
     username text PRIMARY KEY, 
     emails SET text
   );
   INSERT INTO test.users (username, emails)
   VALUES ('someone', {'someone@email.com', 's@email.com'});
   ```

2. If Spark is not running, start the Spark shell. Do not use `sudo` to start the shell.
   ```bash
   $ bin/dse spark
   ```
   The Welcome to Spark output and prompt appears.

3. Define a `CassandraRDD[CassandraRow]` to access the collection set.
Using DataStax Enterprise advanced functionality

```scala
val row = sc.cassandraTable("test", "users").toArray.apply(0)

row: com.datastax.spark.connector.CassandraRow =
  CassandraRow{username: someone,
    emails: {s@email.com,someone@email.com}}

4. Query the collection set from Spark.

row.getList[String]("emails")
res2: Vector[String] = Vector(s@email.com, someone@email.com)

row.getList[String]("emails")
res3: List[String] = List(s@email.com, someone@email.com)

row.getList[String]("emails")
res4: Seq[String] = List(s@email.com, someone@email.com)

row.getList[String]("emails")
res5: IndexedSeq[String] = Vector(s@email.com, someone@email.com)

row.getList[String]("emails")
res6: Set[String] = Set(s@email.com, someone@email.com)

row.getList[String]("emails")
res7: String = {s@email.com,someone@email.com}

Restricting the number of fetched columns

For performance reasons, you should not fetch columns you don't need. You can achieve this with the `select` method:

To restrict the number of fetched columns:

```scala
val row = sc.cassandraTable("test", "users").select("username").toArray

row: Array[com.datastax.spark.connector.CassandraRow] =
  Array(CassandraRow{username: someone})
```
Mapping rows to tuples and case classes

Instead of mapping your rows to objects of the `CassandraRow` class, you can directly unwrap column values into tuples of the desired type.

To map rows to tuples:

```scala
sc.cassandraTable[(String, Int)]("test", "words").select("word", "count").toArray
res9: Array[(String, Int)] = Array((bar,20), (foo,10))

sc.cassandraTable[(Int, String)]("test", "words").select("count", "word").toArray
res10: Array[(Int, String)] = Array((20,bar), (10,foo))
```

Define a `case` class with properties of the same name as the columns. For multi-word column identifiers, separate each word using an underscore when creating the columns, and use `camel case` abbreviation on the Scala side.

To map rows to `case` classes:

```scala
case class WordCount(word: String, count: Int)
defined class WordCount

sc.cassandraTable[WordCount]("test", "words").toArray
```

You can name columns using these conventions:

- Use the underscore convention and lowercase letters. (Recommended)
- Use the camel case convention, exactly the same as properties in Scala.

The following examples show valid column names.

<table>
<thead>
<tr>
<th>Database column name</th>
<th>Scala property name</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>count</td>
</tr>
<tr>
<td>column_1</td>
<td>column1</td>
</tr>
<tr>
<td>user_name</td>
<td>userName</td>
</tr>
<tr>
<td>user_address</td>
<td>UserAddress</td>
</tr>
</tbody>
</table>
Table 7: Alternative naming convention

<table>
<thead>
<tr>
<th>Database column name</th>
<th>Scala property name</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>count</td>
</tr>
<tr>
<td>column1</td>
<td>column1</td>
</tr>
<tr>
<td>userName</td>
<td>userName</td>
</tr>
<tr>
<td>UserAddress</td>
<td>UserAddress</td>
</tr>
</tbody>
</table>

Mapping rows to objects with a user-defined function

Invoke `as` on the `CassandraRDD` to map every row to an object of a different type. Contrary to `map`, `as` expects a function having the same number of arguments as the number of columns to be fetched. Invoking `as` in this way performs type conversions. Using `as` to directly create objects of a particular type eliminates the need to create `CassandraRow` objects and also decreases garbage collection pressure.

To map columns using a user-defined function:

```scala
val table = sc.cassandraTable("test", "words")

val total = table.select("count").as((c: Int) => c).sum

val frequencies = table.select("word", "count").as((w: String, c: Int) => (w, c / total)).toArray

val frequencies: Array[(String, Double)] = Array((bar,0.6666666666666666), (foo,0.3333333333333333))
```

Filtering rows on the server

To filter rows, you can use the filter transformation provided by Spark. Filter transformation fetches all rows from the database first and then filters them in Spark. Some CPU cycles are wasted serializing and de-serializing objects excluded from the result. To avoid this overhead, `CassandraRDD` has a method that passes an arbitrary CQL condition to filter the row set on the server.

This example shows how to use Spark to filter rows on the server.

1. Download and unzip the CQL commands for this example. The commands in this file perform the following tasks:
   - Create a `cars` table in the `test` keyspace.
Using DataStax Enterprise advanced functionality

- Index the color column.
- Insert some data into the table

2. Run the test_cars.cql file using cqlsh or DevCenter. For example using cqlsh:

```bash
$ cqlsh -f test_cars.cql
```

3. Filter the rows using Spark:

```scala
cassandraTable("test", "cars").select("id", "model").where("color = ?", "black").toArray.foreach(println)"n
```

```
CassandraRow{id: AS-8888, model: Aston Martin DB9 Volante}
CassandraRow{id: KF-334L, model: Ford Mondeo}
CassandraRow{id: MT-8787, model: Hyundai x35}
CassandraRow{id: MZ-1038, model: Mazda CX-9}
CassandraRow{id: DG-2222, model: Dodge Avenger}
CassandraRow{id: DG-8897, model: Dodge Charger}
CassandraRow{id: BT-3920, model: Bentley Continental GT}
CassandraRow{id: IN-9964, model: Infinity FX}
```

```
cassandraTable("test", "cars").select("id", "model").where("color = ?", "silver").toArray.foreach(println)
```

```
CassandraRow{id: FR-8877, model: Ferrari FF}
CassandraRow{id: HD-1828, model: Honda Accord}
CassandraRow{id: WX-2234, model: Toyota Yaris}
```

### Controlling automatic direct join optimizations in queries

DSE can optimize join queries to directly lookup data in the database without performing a Spark shuffle, which uses a full table scan.

By default, this optimization is turned on. Direct joins are used when:

```
(table size * directJoinSizeRatio) > size of keys
```

The value of `directJoinSizeRatio` should be between 0 and 1. By default, this value is 0.9. The `directJoinSizeRatio` setting can be set when creating the reference to the database table or in the Spark Session.

```
spark.conf.set("directJoinSizeRatio", 0.2)
```

You can permanently enable or disable this optimization by setting the `directJoinSetting` option. Valid settings for `directJoinSetting` are:

- **on** to permanently enable the optimization
- **off** to permanently disable the optimization
• **auto** (the default value) to let DSE determine when to enable it according to the criteria from the `directJoinSizeRatio` setting

You can programmatically enable or disable `directJoinSetting` by calling the `directJoin` function.

```java
import org.apache.spark.sql.cassandra.CassandraSourceRelation._
import org.apache.spark.sql.cassandra._
val table = spark.read.cassandraFormat("tab", "ks").load
spark
  .range(1L, 100000L)
  .withColumn("id", concat(lit("Store "), 'id))
  .join(table.directJoin(AlwaysOff), 'id === 'store)
```

The `directJoin` function can be set to **AlwaysOn** to permanently enable the optimization, **AlwaysOff** to permanently disable the optimization, or **Automatic** to let DSE determine when to use the optimization according to the formula for `directJoinSizeRatio` described earlier.

Most users should not change the `directJoinSetting` option. In most cases the direct join should be faster than a full table scan. If the calculation is producing less than optimal results adjust the threshold for automatic joins, or turn the optimization off.

**Accessing the Spark session and context for applications running outside of DSE Analytics**

You can optionally create session and context objects for applications that are run outside of the DSE Analytics environment. This is for advanced use cases where applications do not use `dse spark-submit` for handling the classpath and configuration settings.

All classpath and JAR distribution must be handled by the application. The application classpath must include the output of the `dse spark-classpath` command.

```bash
$ dse spark-classpath
```

**Using the Builder API to create a DSE Spark session**

To create a DSE Spark session outside of the DSE Analytics application environment, use the `DseConfiguration` class and the `enableDseSupport` method when creating a Spark session.

```java
import org.apache.spark.sql.SparkSession
import com.datastax.spark.connector.DseConfiguration._
val spark = SparkSession.builder
  .appName("Datastax Scala example")
  .master("dse://127.0.0.1?")
  .config("spark.jars", "target/scala-2.11/writeread_2.11-0.1.jar")
  .enableHiveSupport()
  .enableDseSupport()
```
Using DataStax Enterprise advanced functionality

.getOrCreate()

Creating a Spark Context
When creating a Spark Context object, use the DseConfiguration class and call the
enableDseSupport method when creating the SparkConfiguration instance. In Scala:
import com.datastax.spark.connector.DseConfiguration._
new SparkConf().enableDseSupport()

In Java:
SparkConf rawConf = new SparkConf();
SparkConf conf = DseConfiguration.enableDseSupport(rawConf);

Saving RDD data to DSE
With DataStax Enterprise, you can save almost any RDD to the database. Unless
you do not provide a custom mapping, the object class of the RDD must be a tuple or
have property names corresponding to table column names. To save the RDD, call
the saveToCassandra method with a keyspace name, table name, and optionally, a
list of columns. Before attempting to use the RDD in a standalone application, import
com.datastax.spark.connector.
You can also use the DataFrames API (page 253) to manipulate data within Spark.

Saving a collection of tuples
The following example shows how to save a collection of tuples to the database.
val collection = sc.parallelize(Seq(("cat", 30), ("fox", 40)))
collection: org.apache.spark.rdd.RDD[(String, Int)] =
ParallelCollectionRDD[6] at parallelize at console:22
collection.saveToCassandra("test", "words", SomeColumns("word",
"count"))

At the last Scala prompt in this example, no output means that the data was saved to the
database.
In cqlsh, query the words table to select all the contents.
SELECT * FROM test.words;

word | count
------+------bar |
20
foo |
10
cat |
30

Page 202

DSE 6.7 Developer Guide (Latest version)


Using DataStax Enterprise advanced functionality

Saving a collection of case class objects to the database

The following example shows how to save a collection of case class objects.

```scala
case class WordCount(word: String, count: Long)
val collection = sc.parallelize(Seq(WordCount("dog", 50),
   WordCount("cow", 60)))
collection.saveToCassandra("test", "words", SomeColumns("word",
   "count"))
```

In cqlsh, query the words table to select all the contents.

```sql
SELECT * FROM test.words;
```

<table>
<thead>
<tr>
<th>word</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>bar</td>
<td>20</td>
</tr>
<tr>
<td>foo</td>
<td>10</td>
</tr>
<tr>
<td>cat</td>
<td>30</td>
</tr>
<tr>
<td>fox</td>
<td>40</td>
</tr>
<tr>
<td>dog</td>
<td>50</td>
</tr>
<tr>
<td>cow</td>
<td>60</td>
</tr>
</tbody>
</table>

Using non-default property-name to column-name mappings

Mapping rows to tuples and case classes work out-of-the-box, but in some cases, you might need more control over database-Scala mapping. For example, Java classes are likely to use the JavaBeans naming convention, where accessors are named with `get`, `is` or `set` prefixes. To customize column-property mappings, put an appropriate `ColumnMapper[YourClass]` implicit object in scope. Define such an object in a companion object of the class being mapped. The `ColumnMapper` affects both loading and saving data. DataStax Enterprise includes a few `ColumnMapper` implementations.

**Working with JavaBeans**

To work with Java classes, use `JavaBeanColumnMapper`. Make sure objects are serializable; otherwise Spark cannot send them over the network. The following example shows how to use the `JavaBeanColumnMapper`.

To use JavaBeans style accessors:

```scala
:paste
// Entering paste mode (ctrl-D to finish)
```

Paste this import command and class definition:
Using DataStax Enterprise advanced functionality

```scala
import com.datastax.spark.connector.mapper.JavaBeanColumnMapper
class WordCount extends Serializable {
  private var _word: String = 
  private var _count: Int = 0
  def setWord(word: String) { _word = word }
  def setCount(count: Int) { _count = count }
  override def toString = _word + ":" + 
  object WordCount {
    implicit object Mapper extends JavaBeanColumnMapper[WordCount]
  }
}
```

Enter CTRL D to exit paste mode. The output is:

```scala
// Exiting paste mode, now interpreting.

import com.datastax.spark.connector.mapper.JavaBeanColumnMapper
defined class WordCount
defined module WordCount
```

Query the `WordCount` object.

```scala
sc.cassandraTable[WordCount]("test", "words").toArray
```

To save the data, you need to define getters.

Manually specifying a property-name to column-name relationship

If for some reason you want to associate a property with a column of a different name, pass a column translation map to the `DefaultColumnMapper` or `JavaBeanColumnMapper`.

To change column names:

```scala
// Entering paste mode (ctrl-D to finish)

import com.datastax.spark.connector.mapper.DefaultColumnMapper
case class WordCount(w: String, c: Int)
object WordCount { implicit object Mapper extends DefaultColumnMapper[WordCount](Map("w" -> "word", "c" -> "count")) }
```

Enter CTRL D.

```scala
// Exiting paste mode, now interpreting.

import com.datastax.spark.connector.mapper.DefaultColumnMapper
defined class WordCount
defined module WordCount
```
Continue entering these commands:

```scala
cassandraTable[WordCount]("test", "words")
sc.parallelize(Seq(WordCount("bar", 20), WordCount("foo", 40))).saveToCassandra("test", "words", SomeColumns("word", "count"))
```

Writing custom ColumnMappers

To define column mappings for your classes, create an appropriate implicit object implementing `ColumnMapper[YourClass]` trait.

**Spark supported types**

This table maps CQL types to Scala types. All CQL types are supported by the DataStax Enterprise Spark integration. Other type conversions might work, but cause loss of precision or not work for all values. Most types are convertible to strings. You can convert strings that conform to the CQL standard to numbers, dates, addresses or UUIDs. You can convert maps to or from sequences of key-value tuples.

**Table 8: Supported types**

<table>
<thead>
<tr>
<th>CQL Type</th>
<th>Scala Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii</td>
<td>String</td>
</tr>
<tr>
<td>bigint</td>
<td>Long</td>
</tr>
<tr>
<td>blob</td>
<td>ByteBuffer, Array</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>counter</td>
<td>Long</td>
</tr>
<tr>
<td>decimal</td>
<td>BigDecimal, java.math.BigDecimal</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>inet</td>
<td>java.net.InetAddress</td>
</tr>
<tr>
<td>int</td>
<td>Int</td>
</tr>
<tr>
<td>list</td>
<td>Vector, List, Iterable, Seq, IndexedSeq, java.util.List</td>
</tr>
<tr>
<td>map</td>
<td>Map, TreeMap, java.util.HashMap</td>
</tr>
<tr>
<td>set</td>
<td>Set, TreeSet, java.util.HashSet</td>
</tr>
<tr>
<td>text, varchar</td>
<td>String</td>
</tr>
<tr>
<td>timestamp</td>
<td>Long, java.util.Date, java.sql.Date, org.joda.time.DateTime</td>
</tr>
<tr>
<td>timeuuid</td>
<td>java.util.UUID</td>
</tr>
<tr>
<td>uuid</td>
<td>java.util.UUID</td>
</tr>
</tbody>
</table>
### Loading external HDFS data into the database using Spark

This task demonstrates how to access Hadoop data and save it to the database using Spark on DSE Analytics nodes.

To simplify accessing the Hadoop data, it uses WebHDFS, a REST-based server for interacting with a Hadoop cluster. WebHDFS handles redirect requests to the data nodes, so every DSE Analytics node needs to be able to route to every HDFS node using the Hadoop node's hostname.

These instructions use example weather data, but the principles can be applied to any kind of Hadoop data that can be stored in the database.

### Prerequisites:

You will need:

- A working Hadoop installation with HDFS and WebHDFS enabled and running. You will need the hostname of the machine on which Hadoop is running, and the cluster must be accessible from the DSE Analytics nodes in your DataStax Enterprise cluster.
- A running DataStax Enterprise cluster with DSE Analytics nodes enabled.
- Git installed on a DSE Analytics node.

1. Clone the GitHub repository containing the test data.

   ```
   $ git clone https://github.com/brianmhess/DSE-Spark-HDFS.git
   ```

2. Load the maximum temperature test data into the Hadoop cluster using WebHDFS.

   In this example, the Hadoop node has a hostname of `hadoopNode.example.com`. Replace it with the hostname of a node in your Hadoop cluster.

   ```
   $ hadoop fs -mkdir webhdfs://hadoopNode.example.com:50070/user/guest/data &&
   hadoop fs -copyFromLocal data/sftmax.csv webhdfs://hadoopNode:50070/user/guest/data/sftmax.csv
   ```

3. Create the keyspace and table and load the minimum temperature test data using `cqlsh`.

   ```
   CREATE KEYSPACE IF NOT EXISTS spark_ex2 WITH REPLICATION =
   {'class':'SimpleStrategy', 'replication_factor':1}
   DROP TABLE IF EXISTS spark_ex2.sftmin
   ```
Using DataStax Enterprise advanced functionality

```
CREATE TABLE IF NOT EXISTS spark_ex2.sftmin(location TEXT, year INT, month INT, day INT, tmin DOUBLE, datestring TEXT, PRIMARY KEY ((location), year, month, day)) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC)
COPY spark_ex2.sftmin(location, year, month, day, tmin, datestring) FROM 'data/sftmin.csv'
```

4. Ensure that we can access the HDFS data by interacting with the data using `hadoop fs`.

   The following command counts the number of lines of HDFS data.

   ```bash
   $ hadoop fs -cat webhdfs://hadoopNode.example.com:50070/user/guest/data/sftmax.csv | wc -l
   
   You should see output similar to the following:
   ```
   16/05/10 11:21:51 INFO snitch.Workload: Setting my workload to Cassandra 3606
   ```

5. Start the Spark console and connect to the DataStax Enterprise cluster.

   ```bash
   $ dse spark
   
   Import the Spark Cassandra connector and create the session.
   ```
   ```scala
   import com.datastax.spark.connector.cql.CassandraConnector
   val connector = CassandraConnector(csc.conf)
   val session = connector.openSession()
   ```

6. Create the table to store the maximum temperature data.

   ```scala
   session.execute(s"DROP TABLE IF EXISTS spark_ex2.sftmax")
   session.execute(s"CREATE TABLE IF NOT EXISTS spark_ex2.sftmax(location TEXT, year INT, month INT, day INT, tmax DOUBLE, datestring TEXT, PRIMARY KEY ((location), year, month, day)) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC)"")
   ```

7. Create a Spark RDD from the HDFS maximum temperature data and save it to the table.

   First create a case class representing the maximum temperature sensor data:

   ```scala
   case class Tmax(location: String, year: Int, month: Int, day: Int, tmax: Double, datestring: String)
   ```

   Read the data into an RDD.
Using DataStax Enterprise advanced functionality

val tmax_raw = sc.textFile("webhdfs://sandbox.hortonworks.com:50070/user/guest/data/sftmax.csv")

Transform the data so each record in the RDD is an instance of the \texttt{Tmax} case class.

val tmax_c10 = tmax_raw.map(x=>x.split(\"\")).map(x => Tmax(x(0), x(1).toInt, x(2).toInt, x(3).toInt, x(4).toDouble, x(5)))

Count the case class instances to make sure it matches the number of records.

tmax_c10.count
res11: Long = 3606

Save the case class instances to the database.

tmax_c10.saveToCassandra("spark_ex2", "sftmax")

8. Verify the records match by counting the rows using CQL.

session.execute("SELECT COUNT(*) FROM spark_ex2.sftmax").all.get(0).getLong(0)
res23: Long = 3606

9. Join the maximum and minimum data into a new table.

Create a \texttt{Tmin} case class to store the minimum temperature sensor data.

case class Tmin(location: String, year: Int, month: Int, day: Int, tmin: Double, datestring: String)
val tmin_raw = sc.cassandraTable("spark_ex2", "sftmin")
val tmin_c10 = tmin_raw.map(x => Tmin(x.getString(\"location\"), x.getInt(\"year\"), x.getInt(\"month\"), x.getInt(\"day\"), x.getDouble(\"tmin\"), x.getString(\"datestring\")))

In order to join RDDs, they need to be \texttt{PairRDDs}, with the first element in the pair being the join key.

val tmin_pair = tmin_c10.map(x=>(x.datestring,x))
val tmax_pair = tmax_c10.map(x=>(x.datestring,x))

Create a \texttt{THiLoDelta} case class to store the difference between the maximum and minimum temperatures.

case class THiLoDelta(location: String, year: Int, month: Int, day: Int, hi: Double, low: Double, delta: Double, datestring: String)
Join the data using the `join` operation on the PairRDDS. Convert the joined data to the THiLoDelta case class.

```scala
val tdelta_join1 = tmax_pair1.join(tmin_pair1)
val tdelta_c10 = tdelta_join1.map(x => THiLoDelta(x._2._1._1, x._2._1._2, x._2._1._3, x._2._1._4, x._2._1._5, x._2._2._5, x._2._1._5 - x._2._2._5, x._1))
```

Create a new table within Spark using CQL to store the temperature difference data.

```sql
session.execute(s"DROP TABLE IF EXISTS spark_ex2.sftdelta")
session.execute(s"CREATE TABLE IF NOT EXISTS
spark_ex2.sftdelta(location TEXT, year INT, month INT, day INT, hi DOUBLE, low DOUBLE, delta DOUBLE, datestring TEXT, PRIMARY KEY ((location), year, month, day)) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC)")
```

Save the temperature difference data to the table.

```scala
tdelta_c10.saveToCassandra("spark_ex2", "sftdelta")
```

### Monitoring Spark with the web interface

A web interface, bundled with DataStax Enterprise, facilitates monitoring, debugging, and managing Spark.

**Using the Spark web interface**

To use the Spark web interface enter the *listen IP address (page 87)* of any Spark node in a browser followed by port number 7080 (configured in the *spark-env.sh* configuration file (page 221)). Starting in DSE 5.1, all Spark nodes within an Analytics datacenter will redirect to the current Spark Master.

If the Spark Master is not available, the UI will keep polling for the Spark Master every 10 seconds until the Master is available.

The Spark web interface can be secured using SSL. SSL encryption of the web interface is enabled by default when client encryption is enabled.

If authentication is enabled, and plain authentication is available, you will be prompted for authentication credentials when accessing the web UI. We recommend using SSL with authentication.

**Note:** Kerberos authentication is not supported in the Spark web UI. If authentication is enabled and either LDAP or Internal authentication is not available, the Spark web UI will not be accessible. If this occurs, disable authentication for the Spark web UI only by removing the *spark.ui.filters* setting in spark-daemon-defaults.conf located in the Spark configuration directory.
Using DataStax Enterprise advanced functionality

DSE SSL encryption and authentication only apply to the Spark Master and Worker Uls, not the Spark Driver UI. To use encryption and authentication with the Driver UI, refer to the Spark security documentation.

The UI includes information on the number of cores and amount of memory available to Spark in total and in each work pool, and similar information for each Spark worker. The applications list the associated work pool.

See the Spark documentation for information on using the Spark web UI.

Authorization in the Spark web UI

When authorization is enabled and an authenticated user accesses the web UI, what they can see and do is controlled by their permissions. This allows administrators to control who has permission to view specific application logs, view the executors for the application, kill the application, and list all applications. Viewing and modifying applications can be configured per datacenter, work pool, or application.

See Using authorization with Spark (page 222) for details on granting permissions.

Displaying fully qualified domain names in the web UI

To display fully qualified domain names (FQDNs) in the Spark web UI, set the SPARK_PUBLIC_DNS variable in spark-env.sh on each Analytics node.

Set SPARK_PUBLIC_DNS to the FQDN of the node if you have SSL enabled for the web UI.
Redirecting to the fully qualified domain name of the master

Set the SPARK_LOCAL_IP or SPARK_LOCAL_HOSTNAME in the spark-env.sh file on each node to the fully qualified domain name (FQDN) of the node to force any redirects to the web UI using the FQDN of the Spark master. This is useful when enabling SSL in the web UI.

```bash
export SPARK_LOCAL_HOSTNAME=FQDN of the node
```

Filtering properties in the Spark Driver UI

The Spark Driver UI has an Environment tab that lists the Spark configuration and system properties used by Spark. This can include sensitive information like passwords and security tokens. DSE Spark filters these properties and mask their values with sequences of asterisks. The `spark.redaction.regex` filter is configured as a regular expression that by default includes all properties that contain the string "secret", "token", or "password" as well as all system properties. To modify the filter, edit the `spark.redaction.regex` property in `spark-defaults.conf` in the Spark configuration directory.

Getting started with the Spark Cassandra Connector

The Spark Cassandra Connector allows you to create Java applications that use Spark to analyze database data. See the Spark Cassandra Connector Java Doc on GitHub. See the component versions (page 23) for the latest version of the Spark Cassandra Connector used by DataStax Enterprise.

Using the Java API in SBT build files

Add the following library dependency to the `build.sbt` or other SBT build file.

```scala
val dseVersion = "6.0.0"
// Please make sure that following DSE version matches your DSE cluster version.
// SBT 0.13.13 or greater required because of a dependency resolution bug
libraryDependencies += "com.datastax.dse" % "dse-spark-dependencies" % dseVersion % "provided"
```

For example project templates, see https://github.com/datastax/SparkBuildExamples.

Using the Java API in Maven build files

Add the following dependencies to the `pom.xml` file:

```xml
<dependency>
  <groupId>com.datastax.dse</groupId>
  <artifactId>dse-spark-dependencies</artifactId>
  <version>${dse.version}</version>
  <scope>provided</scope>
</dependency>
```
Using DataStax Enterprise advanced functionality

Then add the DataStax repository:

```xml
<repository>
  <id>DataStax-Repo</id>
</repository>
```

For example project templates, see [https://github.com/datastax/SparkBuildExamples](https://github.com/datastax/SparkBuildExamples)

Accessing database data in Scala applications

To perform Spark actions on table data, you first obtain a **RDD** object. To create the **RDD** object, create a Spark configuration object, which is then used to create a Spark context object.

```scala
import com.datastax.spark.connector._
val conf = new SparkConf(true)
  .set("spark.cassandra.connection.host", "127.0.0.1")
val sc = new SparkContext("dse://127.0.0.1:7077", "test", conf)
val rdd = sc.cassandraTable("my_keyspace", "my_table")
```

To save data to the database in Scala applications, use the **saveToCassandra** method, passing in the keyspace, table, and mapping information.

```scala
val collection = sc.parallelize(Seq(("key3", 3), ("key4", 4)))
collection.saveToCassandra("my_keyspace", "my_table",
  SomeColumns("key", "value"))
```

To perform DSE Graph queries in a Scala application, you can cast a **CassandraConnector** session to a **com.datastax.driver.dse.DseSession** and then run graph statements using the **executeGraph** method.

```scala
val session = CassandraConnector(sc.getConf).withSessionDo(session =>
  session.asInstanceOf[DseSession])
session.executeGraph(graph statement)
```

Accessing database data in Java applications

To perform Spark actions on table data, you first obtain a **CassandraJavaRDD** object, a subclass of the **JavaRDD** class. The **CassandraJavaRDD** is the Java language equivalent of the **CassandraRDD** object used in Scala applications.

To create the **CassandraJavaRDD** object, create a Spark configuration object, which is then used to create a Spark context object.

```java
SparkConf conf = new SparkConf()
  .setAppName( "My application");
SparkContext sc = new SparkContext(conf);
```
Use the static methods of the
com.datastax.spark.connector.japi.CassandraJavaUtil class to get and manipulate
CassandraJavaRDD instances. To get a new CassandraJavaRDD instance, call one of the
javaFunctions methods in CassandraJavaUtil, pass in a context object, and then call
the cassandraTable method and pass in the keyspace, table name, and mapping class.

JavaRDD<
    string cassandraRdd = CassandraJavaUtil.javaFunctions(sc)
    .cassandraTable("my_keyspace",
    "my_table", .mapColumnTo(String.class))
    .select("my_column");

Mapping column data to Java types

You can specify the Java type of a single column from a table row by specifying the type in
when creating the CassandraJavaRDD<T> instance and calling the mapColumnTo method
and passing in the type. Then call the select method to set the column name.

JavaRDD<Integer> cassandraRdd = CassandraJavaUtil.javaFunctions(sc)
    .cassandraTable("my_keyspace",
    "my_table", .mapColumnTo(Integer.class))
    .select("column1");

JavaBeans classes can be mapped using the mapRowTo method. The JavaBeans property
names should correspond to the column names following the default mapping rules. For
example, the firstName property will map by default to the first_name column name.

JavaRDD<Person> personRDD = CassandraJavaUtil.javaFunctions(sc)
    .cassandraTable("my_keyspace", "my_table",
    mapRowTo(Person.class));

CassandraJavaPairRDD<T, T> instances are extensions of the JavaPairRDD class, and
have mapping readers for rows and columns similar to the previous examples. These pair
RDDs typically are used for key/value pairs, where the first type is the key and the second
type is the value.

When mapping a single column for both the key and the value, call mapColumnTo and
specify the key and value types, then the select method and pass in the key and value
column names.

CassandraJavaPairRDD<Integer, String> pairRDD =
CassandraJavaUtil.javaFunctions(sc)
    .cassandraTable("my_keyspace", "my_table",
    mapColumnTo(Integer.class), mapColumnTo(String.class))
    .select("id", "first_name");

Use the mapRowTo method to map row data to a Java type. For example, to create a pair
RDD instance with the primary key and then a JavaBeans object:

CassandraJavaPairRDD<Integer, Person> idPersonRdd =
CassandraJavaUtil.javaFunctions(sc)
Using DataStax Enterprise advanced functionality

```java
.cassandraTable("my_keyspace", "my_table",
mapColumnTo(Integer.class), mapRowTo(Person.class))
.select("id", "first_name", "last_name", "birthdate", "email");
```

Saving data to the database in Java applications

To save data from an RDD to the database call the `writerBuilder` method on the `CassandraJavaRDD` instance, passing in the keyspace, table name, and optionally type mapping information for the column or row.

```java
CassandraJavaUtil.javaFunctions(personRdd)
    .writerBuilder("my_keyspace", "my_table",
                 mapToRow(Person.class)).saveToCassandra();
```

Using DSE Spark with third party tools and integrations

The `dse exec (page 1202)` command sets the required environment variables required to run third-party tools that integrate with Spark.

```bash
$ dse exec command
```

**Jupyter integration**

Download and install Jupyter notebook on a DSE node.

To launch Jupyter notebook:

```bash
$ dse exec jupyter notebook
```

A Jupyter notebook starts with the correct Python path. You must create a context to work with DSE. In contrast to Livy and Zeppelin integrations, the Jupyter integration does not start an interpreter that creates a context.

**Livy integration**

Download and install Livy on a DSE node. By default Livy runs Spark in local mode. Before starting Livy create a configuration file by copying the `conf/livy.conf.template` to `conf/livy.conf`, then uncomment or add the following two properties:

```ini
livy.spark.master = dse:///n
livy.repl.enable-hive-context = true
```

To launch Livy:

```bash
$ dse exec livy-server
```

**RStudio integration**

Download and install R (page 256) on all DSE Analytics nodes, install RStudio desktop on one of the nodes, then run RStudio:
$ dse exec rstudio

In the RStudio session start a Spark session:

```r
library(SparkR, lib.loc = c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib")))
sparkR.session()
```

**Note:** These instructions are for RStudio desktop, not RStudio Server. In multiuser environments, we recommend using **AlwaysOn SQL (page 257)** and **JDBC (page 264)** connections rather than SparkR.

Zeppelin integration

Download and install Zeppelin on a DSE node. To launch Zeppelin server:

```
$ dse exec zeppelin.sh
```

By default Zeppelin runs Spark in local mode. Update the master property to `dse:///` in the Spark session in the Interpreters configuration page. No configuration file changes are required to run Zeppelin.

**Configuring Spark**

Configuring Spark for DataStax Enterprise includes:

**Configuring Spark nodes**

Modify the settings for Spark nodes security, performance, and logging.

To manage Spark performance and operations:

- Set environment variables (page 215)
- Protect Spark directories (page 216)
- Grant access to default Spark directories (page 217)
- Secure Spark nodes (page 217)
- Configure Spark memory and cores (page 218)
- Configure Spark logging options (page 229)

Set environment variables

DataStax recommends using the default values of Spark environment variables unless you need to increase the memory settings due to an `OutOfMemoryError` condition or garbage collection taking too long. Use the **Spark memory** (page 130) configuration options in the `dse.yaml` and `spark-env.sh` files.

You can set a user-specific `SPARK_HOME` directory if you also set `ALLOW_SPARK_HOME=true` in your environment before starting DSE.
Using DataStax Enterprise advanced functionality

For example, on Debian or Ubuntu using a package installation:

```bash
$ export SPARK_HOME=$HOME/spark &&
export ALLOW_SPARK_HOME=true &&
sudo service dse start
```

The temporary directory for shuffle data, RDDs, and other ephemeral Spark data can be configured for both the locally running driver and for the Spark server processes managed by DSE (Spark Master, Workers, shuffle service, executor and driver running in cluster mode).

For the locally running Spark driver, the `SPARK_LOCAL_DIRS` environment variable can be customized in the user environment or in `spark-env.sh`. By default, it is set to the system temporary directory. For example, on Ubuntu it is `/tmp/`. If there’s no system temporary directory, then `SPARK_LOCAL_DIRS` is set to a `.spark` directory in the user’s home directory.

For all other Spark server processes, the `SPARK_EXECUTOR_DIRS` environment variable can be customized in the user environment or in `spark-env.sh`. By default it is set to `/var/lib/spark/rdd`.

**Note:** The default `SPARK_LOCAL_DIRS` and `SPARK_EXECUTOR_DIRS` environment variable values differ from non-DSE Spark.

To configure worker cleanup, modify the `SPARK_WORKER_OPTS` environment variable and add the `cleanup` properties. The `SPARK_WORKER_OPTS` environment variable can be set in the user environment or in `spark-env.sh`. For example, the following enables worker cleanup, sets the cleanup interval to 30 minutes (i.e. 1800 seconds), and sets the length of time application worker directories will be retained to 7 days (i.e. 604800 seconds).

```bash
$ export SPARK_WORKER_OPTS="$SPARK_WORKER_OPTS \
-Dspar...appDataTtl=604800"
```

Protect Spark directories

After you start up a Spark cluster, DataStax Enterprise creates a Spark work directory for each Spark Worker on worker nodes. A worker node can have more than one worker, configured by the `SPARK_WORKER_INSTANCES` option in `spark-env.sh`. If `SPARK_WORKER_INSTANCES` is undefined, a single worker is started. The work directory contains the standard output and standard error of executors and other application specific data stored by Spark Worker and executors; the directory is writable only by the DSE user.

By default, the Spark parent work directory is located in `/var/lib/spark/work`, with each worker in a subdirectory named `worker-number`, where the number starts at 0. To change the parent worker directory, configure `SPARK_WORKER_DIR` in the `spark-env.sh` file.
The Spark RDD directory is the directory where RDDs are placed when executors decide to spill them to disk. This directory might contain the data from the database or the results of running Spark applications. If the data in the directory is confidential, prevent access by unauthorized users. The RDD directory might contain a significant amount of data, so configure its location on a fast disk. The directory is writable only by the cassandra user. The default location of the Spark RDD directory is /var/lib/spark/rdd. The directory should be located on a fast disk. To change the RDD directory, configure SPARK_EXECUTOR_DIRS in the spark-env.sh file.

Grant access to default Spark directories

Before starting up nodes on a tarball installation, you need permission to access the default Spark directory locations: /var/lib/spark and /var/log/spark. Change ownership of these directories as follows:

```
sudo mkdir -p /var/lib/spark/rdd; sudo chmod a+w /var/lib/spark/rdd;
sudo chown -R $USER:$GROUP /var/lib/spark/rdd &&
sudo mkdir -p /var/log/spark; sudo chown -R $USER:$GROUP /var/log/spark
```

In multiple datacenter clusters, use a virtual datacenter to isolate Spark jobs. Running Spark jobs consume resources that can affect latency and throughput.

DataStax Enterprise supports the use of virtual nodes (vnodes) with Spark.

Secure Spark nodes

**Client-to-node SSL**

Ensure that the truststore entries in cassandra.yaml are present as described in Client-to-node encryption, even when client authentication is not enabled.

**Enabling security and authentication**

Security is enabled using the spark_security_enabled option in dse.yaml. Setting it to enabled turns on authentication between the Spark Master and Worker nodes, and allows you to enable encryption. To encrypt Spark connections for all components except the web UI, enable spark_security_encryption_enabled. The length of the shared secret used to secure Spark components is set using the spark_shared_secret_bit_length option, with a default value of 256 bits. These options are described in DSE Analytics options (page 130). For production clusters, enable these authentication and encryption. Doing so does not significantly affect performance.

**Authentication and Spark applications**

If authentication is enabled, users need to be authenticated in order to submit an application.

**Authorization and Spark applications**

If DSE authorization is enabled, users needs permission to submit an application. Additionally, the user submitting the application automatically receives permission to manage the application, which can optionally be extended to other users.

**Database credentials for the Spark SQL Thrift server**
Using DataStax Enterprise advanced functionality

In the `hive-site.xml` file, configure authentication credentials for the Spark SQL Thrift server. Ensure that you use the `hive-site.xml` file in the Spark directory:

- Package installations: `/etc/dse/spark/hive-site.xml`
- Tarball installations: `installation_location/resources/spark/conf/hive-site.xml`

**Kerberos with Spark**

With Kerberos authentication, the Spark launcher connects to DSE with Kerberos credentials and requests DSE to generate a delegation token. The Spark driver and executors use the delegation token to connect to the cluster. For valid authentication, the delegation token must be renewed periodically. For security reasons, the user who is authenticated with the token should not be able to renew it. Therefore, delegation tokens have two associated users: token owner and token renewer.

The token renewer is none so that only a DSE internal process can renew it. When the application is submitted, DSE automatically renews delegation tokens that are associated with Spark application. When the application is unregistered (finished), the delegation token renewal is stopped and the token is cancelled.

Set Kerberos options, see Defining a Kerberos scheme.

**Configure Spark memory and cores**

Spark memory options affect different components of the Spark ecosystem:

**Spark History server and the Spark Thrift server memory**

The `SPARK_DAEMON_MEMORY` option configures the memory that is used by the Spark SQL Thrift server and history-server. Add or change this setting in the `spark-env.sh` file on nodes that run these server applications.

**Spark Worker memory**

The `memory_total` option (page 132) in the `resource_manager_options.worker_options` section of `dse.yaml` configures the total system memory that you can assign to all executors that are run by the work pools on the particular node. The default work pool will use all of this memory if no other work pools are defined. If you define additional work pools, you can set the total amount of memory by setting the `memory` option (page 132) in the work pool definition.

**Application executor memory**

You can configure the amount of memory that each executor can consume for the application. Spark uses a 512MB default. Use either the `spark.executor.memory` option, described in "Spark Available Properties", or the `--executor-memory mem` argument to the `dse spark` (page 1211) command.

**Application memory**

You can configure additional Java options that are applied by the worker when spawning an executor for the application. Use the `spark.executor.extraJavaOptions` property, described in Spark 1.6.2 Available Properties. For example:
spark.executor.extraJavaOptions -XX:+PrintGCDetails -Dkey=value -Dnumbers="one two three"

Core management

You can manage the number of cores by configuring these options.

- **Spark Worker cores**

  The `cores_total` option (page 131) in the `resource_manager_options.worker_options` section of `dse.yaml` configures the total number of system cores available to Spark Workers for executors. If no work pools are defined in the `resource_manager_options.workpools` section (page 132) of `dse.yaml` the default work pool will use all the cores defined by `cores_total`. If additional work pools are defined, the default work pool will use the cores available after allocating the cores defined by the work pools.

  A single executor can borrow more than one core from the worker. The number of cores used by the executor relates to the number of parallel tasks the executor might perform. The number of cores offered by the cluster is the sum of cores offered by all the workers in the cluster.

- **Application cores**

  In the Spark configuration object of your application, you configure the number of application cores that the application requests from the cluster using either the `spark.cores.max` configuration property or the `--total-executor-cores cores` argument to the `dse spark` (page 121) command.

See the [Spark documentation](#) for details about memory and core allocation.

DataStax Enterprise can control the memory and cores offered by particular Spark Workers in semi-automatic fashion. The `resource_manager_options.worker_options` section in the `dse.yaml` file has options to configure the proportion of system resources that are made available to Spark Workers and any defined work pools, or explicit resource settings. When specifying decimal values of system resources the available resources are calculated in the following way:

- Spark Worker memory = `memory_total` * (total system memory - memory assigned to DSE)
- Spark Worker cores = `cores_total` * total system cores

This calculation is used for any decimal values. If the setting is not specified, the default value 0.7 is used. If the value does not contain a decimal place, the setting is the explicit number of cores or amount of memory reserved by DSE for Spark.

**Note:** Setting `cores_total` or a workpool's `cores` to 1.0 is a decimal value, meaning 100% of the available cores will be reserved. Setting `cores_total` or `cores` to 1 (no decimal point) is an explicit value, and one core will be reserved.
Using DataStax Enterprise advanced functionality

The lowest values you can assign to a named work pool's memory and cores are 64 MB and 1 core, respectively. If the results are lower, no exception is thrown and the values are automatically limited.

The following example shows a work pool named `workpool1` with 1 core and 512 MB of RAM assigned to it. The remaining resources calculated from the values in `worker_options` are assigned to the default work pool.

```yaml
resource_manager_options:
  worker_options:
    cores_total: 0.7
    memory_total: 0.7

  workpools:
    - name: workpool1
      cores: 1
      memory: 512M
```

Running Spark clusters in cloud environments

If you are using a cloud infrastructure provider like Amazon EC2, you must explicitly open the ports for publicly routable IP addresses in your cluster. If you do not, the Spark workers will not be able to find the Spark Master.

One work-around is to set the `prefer_local` setting in your `cassandra-rackdc.properties` snitch setup file to true:

```bash
# Uncomment the following line to make this snitch prefer the internal
# ip when possible, as the Ec2MultiRegionSnitch does.
prefer_local=true
```

This tells the cluster to communicate only on private IP addresses within the datacenter rather than the public routable IP addresses.

Configuring the number of retries to retrieve Spark configuration

When Spark fetches configuration settings from DSE, it will not fail immediately if it cannot retrieve the configuration data, but will retry 5 times by default, with increasing delay between retries. The number of retries can be set in the Spark configuration, by modifying the `spark.dse.configuration.fetch.retries` configuration property when calling the `dse spark` command, or in `spark-defaults.conf`.

Disabling continuous paging

Continuous paging streams bulk amounts of records from DSE to the DataStax Java Driver used by DSE Spark. By default, continuous paging in queries is enabled. To disable it, set the `spark.dse.continuous_paging_enabled` setting to false when starting the Spark SQL shell or in `spark-defaults.conf`. For example:

```bash
$ dse spark-sql --conf spark.dse.continuous_paging_enabled=false
```
**Note:** Using continuous paging can potentially improve performance up to 3 times, though the improvement will depend on the data and the queries. Some factors that impact the performance improvement are the number of executor JVMs per node and the number of columns included in the query. Greater performance gains were observed with fewer executor JVMs per node and more columns selected.

Configuring the Spark web interface ports

By default the Spark web UI runs on port 7080. To change the port number, do the following:

1. Open the spark-env.sh file in a text editor.

2. Set the `SPARK_MASTER_WEBUI_PORT` variable to the new port number. For example, to set it to port 7082:

   ```bash
   export SPARK_MASTER_WEBUI_PORT=7082
   ```

3. Repeat these steps for each Analytics node in your cluster.

4. Restart the nodes in the cluster.

Enabling Graphite Metrics in DSE Spark

Users can add third party JARs to Spark nodes by adding them to the Spark lib directory on each node and restart the cluster. Add the Graphite Metrics JARs to this directory to enable metrics in DSE Spark.

The default location of the Spark lib directory depends on the type of installation:

- **Package installations:** `/usr/share/dse/spark/lib`
- **Tarball installations:** `/var/lib/spark`

To add the Graphite JARs to Spark in a package installation, copy them to the Spark lib directory:

```bash
$ cp metrics-graphite-3.1.2.jar /usr/share/dse/spark/lib/ &&
cp metrics-json-3.1.2.jar /usr/share/dse/spark/lib/
```

Setting Spark properties for the driver and executor

Additional Spark properties for the Spark driver and executors are set in `spark-defaults.conf`. For example, to enable Spark's `commons-crypto` encryption library:
Using DataStax Enterprise advanced functionality

spark.network.crypto.enabled true

Using authorization with Spark

Set permissions on roles to allow Spark applications to be started, stopped, managed, and viewed. To configure the permissions for a particular role, modify the WORKPOOL and SUBMISSION database objects by issuing CQL commands.

There are two kinds of authorization permissions which apply to Spark. Work pool permissions control the ability to submit or view a Spark application to DSE. Submission permissions control the ability to view or manage a particular application. If authentication and authorization are enabled for the Spark web UI (page 209), these permissions control what the authenticated user is allowed to view and modify.

All the following instructions assume you are issuing the CQL commands as a database superuser. In order to issue the following CQL commands as a regular database user, the user needs to have permission to use the DSE resource manager RPC:

    GRANT ALL ON REMOTE OBJECT DseResourceManager TO role;

Each DSE Analytics user needs to have permission to use the client tools RPC:

    GRANT ALL ON REMOTE OBJECT DseClientTool TO role;

Authorizing roles to start Spark applications

The CREATE permission allows roles to start Spark applications on a work pool.

The following CQL command grants permission to submit a Spark application to any Analytics datacenter.

    GRANT CREATE ON ANY WORKPOOL TO role;

The following CQL command grants permission to submit a Spark application to a particular work pool in an Analytics datacenter.

    GRANT CREATE ON WORKPOOL datacenter_name.workpool_name TO role;

You can use a wildcard for workpool_name so it applies to all work pools in the datacenter:

    GRANT CREATE ON WORKPOOL datacenter_name.* TO role;

Note: You must specify a work pool name or wildcard when specifying a datacenter. In DSE versions prior to 6.0, you could specify the datacenter name only, but omitting the work pool name or wildcard will result in a syntax error.
There are similar revoke commands:

```plaintext
REVOKE CREATE ON ANY WORKPOOL FROM role

REVOKE CREATE ON WORKPOOL datacenter_name.workpool_name FROM role
```

When an application is submitted, the user who submits that application is automatically granted permission to manage and remove the application. You may also grant the ability to manage the application to another user or role.

Use the `REVOKE` command to remove permissions:

```plaintext
REVOKE CREATE ON ANY WORKPOOL FROM role;
```

Authorizing roles to stop or manage Spark applications

Setting the `MODIFY` permission on the `SUBMISSION` object controls the ability to modify or stop a Spark application.

The following CQL command grants permission to manage any submission in any work pool to the specified role.

```plaintext
GRANT MODIFY ON ANY SUBMISSION TO role;
```

The following CQL command grants permission to manage any submission in a specified datacenter.

```plaintext
GRANT MODIFY ON ANY SUBMISSION
IN WORKPOOL datacenter_name
TO role;
```

The following CQL command grant permission to manage a submission identified by the provided id in a given data center’s work pool.

```plaintext
GRANT MODIFY ON SUBMISSION id
IN WORKPOOL datacenter_name.workpool_name
TO role;
```

The ID is a string that is either the Spark application ID or the ID of the Spark driver running in cluster mode.

Use the `REVOKE` command to remove permissions:

```plaintext
REVOKE MODIFY ON SUBMISSION id
IN WORKPOOL datacenter_name.workpool_name
FROM role;
```
Using DataStax Enterprise advanced functionality

Authorizing roles to browse Spark application information in the Spark web UI

The `DESCRIBE` permission allows roles to browse applications in the Spark web UI. The permissions can be set at the work pool or application level.

The following CQL command allows a role to view all applications in any Analytics datacenter.

```cql
GRANT DESCRIBE ON ANY WORKPOOL TO role;
```

The following CQL command limits the role’s ability to view applications to a specific work pool in an Analytics datacenter.

```cql
GRANT DESCRIBE ON WORKPOOL datacenter_name.workpool_name TO role;
```

The following CQL command allows a role to view all submissions, including executors, in any work pool to the specified role.

```cql
GRANT DESCRIBE ON ANY SUBMISSION TO role;
```

You can limit viewing to a specific datacenter:

```cql
GRANT DESCRIBE ON ANY SUBMISSION IN WORKPOOL datacenter_name.workpool_name TO role;
```

You can further limit a role to only viewing the executors for a single application in a datacenter:

```cql
GRANT DESCRIBE ON SUBMISSION id IN WORKPOOL datacenter_name.datacenter_name TO role;
```

Use the `REVOKE` command to remove permissions:

```cql
REVOKE DESCRIBE ON ANY SUBMISSION IN WORKPOOL datacenter_name.workpool_name FROM role
```

**Spark server configuration**

The `spark-daemon-defaults.conf` file configures DSE Spark Masters and Workers.
### Table 9: Spark server configuration properties

<table>
<thead>
<tr>
<th>Option</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dse.spark.application.timeout</td>
<td>30</td>
<td>The duration in seconds after which the application will be considered dead if no heartbeat is received.</td>
</tr>
<tr>
<td>spark.dseShuffle.sasl.port</td>
<td>7447</td>
<td>The port number on which a shuffle service for SASL secured applications is started. Bound to the listen_address in cassandra.yaml.</td>
</tr>
<tr>
<td>spark.dseShuffle.noSasl.port</td>
<td>7437</td>
<td>The port number on which a shuffle service for unsecured applications is started. Bound to the listen_address in cassandra.yaml.</td>
</tr>
</tbody>
</table>

By default Spark executor logs, which log the majority of your Spark Application output, are redirected to standard output. The output is managed by Spark Workers. Configure logging by adding `spark.executor.logs.rolling.*` properties to `spark-daemon-defaults.conf` file.

```
spark.executor.logs.rolling.maxRetainedFiles 3
spark.executor.logs.rolling.strategy size
spark.executor.logs.rolling.maxSize 50000
```

Additional Spark properties that affect the master and driver can be added to `spark-daemon-defaults.conf`. For example, to enable Spark's commons-crypto encryption library:

```
spark.network.crypto.enabled true
```

### Automatic Spark Master election

Spark Master elections are automatically managed, and do not require any manual configuration.

DSE Analytics datacenters communicate with each other to elect one of the nodes as the Spark Master and another as the reserve Master. The Master keeps track of each Spark Worker and application, storing the information in a system table. If the Spark Master node fails, the reserve Master takes over and a new reserve Master is elected from the remaining Analytics nodes.

Each Analytics datacenter elects its own master.

For dsetool commands and options, see dsetool (page 1309).
Determining the Spark Master address

You do not need to specify the Master address when configuring or using Spark with DSE Analytics. Configuring applications with a valid URL (page 188) is sufficient for DSE to connect to the Master node and run the application. The following commands give information about the Spark configuration of DSE:

- To view the URL used to configure Spark applications:
  ```
  $ dse client-tool spark master-address
  dse://10.200.181.62:9042?
  connection.local_dc=Analytics;connection.host=10.200.181.63;
  ```

- To view the current address of the Spark Master in this datacenter:
  ```
  $ dse client-tool spark leader-address
  10.200.181.62
  ```

- Workloads for Spark Master (page 182) are flagged as Workload: Analytics(SM).
  ```
  $ dsetool ring
  Address          DC                   Rack         Workload: Analytics(SM)
  Token                                        Health [0,1]
  0
  10.200.181.62    Analytics            rack1        Analytics(SM)
  no     Up      Normal   111.91 KiB       ?
  -9223372036854775808                         0.10
  ```

- Query the dse_leases.leases table to list all the masters from each data center with Analytics nodes:
  ```
  select * from dse_leases.leases ;
  ```

<table>
<thead>
<tr>
<th>name</th>
<th>dc</th>
<th>duration_ms</th>
<th>epoch</th>
<th>holder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leader/master/6.0</td>
<td>Analytics</td>
<td>30000</td>
<td>805254</td>
<td></td>
</tr>
<tr>
<td>10.200.176.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leader/master/6.0</td>
<td>SearchGraphAnalytics</td>
<td>30000</td>
<td>1300800</td>
<td></td>
</tr>
<tr>
<td>10.200.176.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leader/master/6.0</td>
<td>SearchAnalytics</td>
<td>30000</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>10.200.176.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ensure that the replication factor is configured correctly for the dse_leases keyspace

If the dse_leases keyspace is not properly replicated, the Spark Master might not be elected.

**Important:** Every time you add a new datacenter, you **must** manually increase the replication factor of the dse_leases keyspace for the new DSE Analytics datacenter. If DataStax Enterprise or Spark security options are enabled on the cluster, you must also increase the replication factor for the dse_security keyspace across all logical datacenters.

The initial node in a multi datacenter has a replication factor of 1 for the dse_leases keyspace. For new datacenters, the first node is created with the dse_leases keyspace with a replication factor of 1 for that datacenter. However, any datacenters that you add have a replication factor of 0 and require configuration before you start DSE Analytics nodes. You must change the replication factor of the dse_leases keyspace for multiple analytics datacenters. See [Setting the replication factor for analytics keyspaces](page 177).

Monitoring the lease subsystem

All changes to lease holders are recorded in the dse_leases.logs table. Most of the time, you do not want to enable logging.

1. To turn on logging, ensure that the lease_metrics_options (page 118) is enabled in the dse.yaml file:

```yaml
lease_metrics_options:
  enabled: true
  ttl_seconds: 604800
```

2. Look at the dse_leases.logs table:

```sql
select * from dse_leases.logs ;
```

<table>
<thead>
<tr>
<th>name</th>
<th>dc</th>
<th>monitor</th>
<th>at</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>new_holder</td>
<td>old_holder</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
<td>---------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Leader/master/6.0</td>
<td>dc1</td>
<td>10.200.180.44</td>
<td>2018-05-17 00:45:02.971000+0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.200.180.49</td>
<td>2018-05-17 02:37:07.381000+0000</td>
</tr>
<tr>
<td>Leader/master/6.0</td>
<td>dc1</td>
<td>10.200.180.49</td>
<td>2018-05-17 00:45:02.971000+0000</td>
</tr>
</tbody>
</table>

3. When the lease_metrics_option is enabled, you can examine the acquire, renew, resolve, and disable operations. Most of the time, these operations should complete in 100 ms or less:
Using DataStax Enterprise advanced functionality

```
select * from dse_perf.leases;
```

<table>
<thead>
<tr>
<th>name</th>
<th>dc</th>
<th>monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>acquire_average_latency_ms</td>
<td>acq</td>
<td>acquire_latency99ms</td>
</tr>
<tr>
<td>acquire_max_latency_ms</td>
<td>acq</td>
<td>acquire_rate15</td>
</tr>
<tr>
<td>disable_average_latency_ms</td>
<td>dis</td>
<td>disable_latency99ms</td>
</tr>
<tr>
<td>disable_max_latency_ms</td>
<td>dis</td>
<td>disable_rate15</td>
</tr>
<tr>
<td>renew_average_latency_ms</td>
<td>ren</td>
<td>renew_latency99ms</td>
</tr>
<tr>
<td>renew_max_latency_ms</td>
<td>ren</td>
<td>renew_rate15</td>
</tr>
<tr>
<td>resolve_average_latency_ms</td>
<td>res</td>
<td>resolve_latency99ms</td>
</tr>
<tr>
<td>resolve_max_latency_ms</td>
<td>res</td>
<td>resolve_rate15</td>
</tr>
<tr>
<td>renew_average_latency_ms</td>
<td></td>
<td>renew_latency99ms</td>
</tr>
<tr>
<td>renew_max_latency_ms</td>
<td></td>
<td>renew_rate15</td>
</tr>
<tr>
<td>resolve_average_latency_ms</td>
<td></td>
<td>resolve_latency99ms</td>
</tr>
<tr>
<td>resolve_max_latency_ms</td>
<td></td>
<td>resolve_rate15</td>
</tr>
<tr>
<td>up</td>
<td>up</td>
<td>up_or_down_since</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------</td>
<td>----------------</td>
</tr>
<tr>
<td>Leader/master/6.0</td>
<td>dc1</td>
<td>10.200.180.44</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>True</td>
<td>2018-05-03 19:30:38.395000+0000</td>
<td></td>
</tr>
<tr>
<td>Leader/master/6.0</td>
<td>dc1</td>
<td>10.200.180.49</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>10</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>True</td>
<td>2018-05-03 19:30:55.656000+0000</td>
<td></td>
</tr>
</tbody>
</table>

4. If the log warnings and errors do not contain relevant information, edit the logback.xml file and add:

```
<logger name="com.datastax.bdp.leasemanager" level="DEBUG"/>
```

5. Restart the node for the debugging settings to take effect.

**Troubleshooting**

Perform these various lease holder troubleshooting activities before you contact DataStax Support.

**Verify the workload status**

Run the `dsetool ring` command:

```
$ dsetool ring
```
If the replication factor is inadequate or if the replicas are down, the output of the `dsetool ring` command contains a warning:

<table>
<thead>
<tr>
<th>Address</th>
<th>DC</th>
<th>Rack</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Graph Status</td>
<td>State</td>
<td>Load</td>
</tr>
<tr>
<td>Token</td>
<td></td>
<td></td>
<td>Owns</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Health [0,1]</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.200.178.232</td>
<td>SearchGraphAnalytics rack1</td>
<td>SearchAnalytics</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.200.178.230</td>
<td>SearchGraphAnalytics rack1</td>
<td>SearchAnalytics(SM)</td>
<td>yes</td>
</tr>
<tr>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If the automatic Job Tracker or Spark Master election fails, verify that an appropriate replication factor is set for the `dse_leases` keyspace (page 177).

Use `cqlsh` commands to verify the replication factor of the analytics keyspaces:

1. Describe the `dse_leases` keyspace:
   ```sql
   DESCRIBE KEYSPACE dse_leases;
   CREATE KEYSPACE dse_leases WITH replication =
   {'class': 'NetworkTopologyStrategy', 'Analytics1': '1'}
   AND durable_writes = true;
   
2. Increase the replication factor of the `dse_leases` keyspace:
   ```sql
   ALTER KEYSPACE dse_leases WITH replication =
   {'class': 'NetworkTopologyStrategy', 'Analytics1': '3',
   'Analytics2':'3'}
   ;
   
3. Run `nodetool repair` (page 1054).

**Configuring Spark logging options**

You can configure Spark logging options for the Spark logs.

**Log directories**

The Spark logging directory is the directory where the Spark components store individual log files. DataStax Enterprise places logs in the following locations:

**Executor logs**

- `SPARK_WORKER_DIR/worker-n/application_id/executor_id/stderr`
• `SPARK_WORKER_DIR/worker-n/application_id/executor_id/stdout`

**Spark Master/Worker logs**
- Spark Master: the global `system.log`
- Spark Worker: `SPARK_WORKER_LOG_DIR/worker-n/worker.log`

The default `SPARK_WORKER_LOG_DIR` location is `/var/log/spark/worker`.

**Default log directory for Spark SQL Thrift server**
The default log directory for starting the Spark SQL Thrift server is `$HOME/spark-thrift-server`.

**Spark Shell and application logs**
- Spark Shell and application logs are output to the console.

**SparkR shell log**
The default location for the SparkR shell is `$HOME/.sparkR.log`

**Log configuration file**
Log configuration files are located in the same directory (page 215) as `spark-env.sh`.

To configure Spark logging options:

1. Configure logging options, such as log levels, in the following files:

<table>
<thead>
<tr>
<th>Executors</th>
<th>logback-spark-executor.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark Master</td>
<td>logback.xml</td>
</tr>
<tr>
<td>Spark Worker</td>
<td>logback-spark-server.xml</td>
</tr>
<tr>
<td>Spark Driver</td>
<td>logback-spark.xml</td>
</tr>
<tr>
<td>SparkR</td>
<td>logback-sparkR.xml</td>
</tr>
</tbody>
</table>

2. If you want to enable rolling logging for Spark executors, add the following options to `spark-daemon-defaults.conf`.

   Enable rolling logging with 3 log files retained before deletion. The log files are broken up by size with a maximum size of 50,000 bytes.

   ```
   spark.executor.logs.rolling.maxRetainedFiles 3
   spark.executor.logs.rolling.strategy size
   spark.executor.logs.rolling.maxSize 50000
   ```

   The default location of the Spark configuration files depends on the type of installation:
   - Package installations: `/etc/dse/spark/`
   - Tarball installations: `installation_location/resources/spark/conf`
3. Configure a safe communication channel to access the Spark user interface.

**Note:** When user credentials are specified in plain text on the dse command line, like `dse -u username -p password`, the credentials are present in the logs of Spark workers when the driver is run in cluster mode.

The Spark Master, Spark Worker, executor, and driver logs might include sensitive information. Sensitive information includes passwords and digest authentication tokens for Kerberos guidelines mode that are passed in the command line or Spark configuration. DataStax recommends using only safe communication channels like VPN and SSH to access the Spark user interface.

**Tip:** You can provide authentication credentials in several ways, see Credentials for authentication.

---

### Running Spark processes as separate users

Spark processes can be configured to run as separate operating system users.

By default, processes started by DSE are run as the same OS user who started the DSE server process. This is called the DSE service user. One consequence of this is that all applications that are run on the cluster can access DSE data and configuration files, and access files of other applications.

You can delegate running Spark applications to runner processes and users by changing options in `dse.yaml`.

#### Overview of the run_as process runner

The run_as process runner allows you to run Spark applications as a different OS user than the DSE service user. When this feature is enabled and configured:

- All simultaneously running applications deployed by a single DSE service user will be run as a single OS user.
- Applications deployed by different DSE service users will be run by different OS users.
- All applications will be run as a different OS user than the DSE service user.

This allows you to prevent an application from accessing DSE server private files, and prevent one application from accessing the private files of another application.

#### How the run_as process runner works

DSE uses `sudo` to run Spark applications components (drivers and executors) as specific OS users. DSE doesn't link a DSE service user with a particular OS user. Instead, a configurable number of spare user accounts or *slots* are used. When a request to run an executor or a driver is received, DSE finds an unused slot, and locks it for that application. Until the application is finished, all of that application's processes run as that slot user.
When the application completes, the slot user will be released and will be available to other applications.

Since the number of slots is limited, a single slot is shared among all the simultaneously running applications run by the same DSE service user. Such a slot is released once all the applications of that user are removed. When there is not enough slots to run an application, an error is logged and DSE will try to run the executor or driver on a different node. DSE does not limit the number of slots you can configure. If you need to run more applications simultaneously, create more slot users.

Slots assignment is done on a per node basis. Executors of a single application may run as different slot users on different DSE nodes. When DSE is run on a fat node, different DSE instances running within the same OS should be configured with different sets of slot users. If they use the same slot users, a single OS user may run the applications of two different DSE service users.

When a slot is released, all directories which are normally managed by Spark for the application are removed. If the application doesn't finish, but all executors are done on a node, and a slot user is about to be released, all the application files are modified so that their ownership is changed to the DSE service user with owner-only permission. When a new executor for this application is run on this node, the application files are reassigned back to the slot user assigned to that application.

Configuring the run_as process runner

The administrator needs to prepare slot users in the OS before configuring DSE. The run_as process runner requires:

- Each slot user has its own primary group, which name is the same as the name of slot user. This is typically the default behaviour of the OS. For example, the slot1 user's primary group is slot1.
- The DSE service user is a member of each slot's primary group. For example, if the DSE service user is cassandra, the cassandra user is a member of the slot1 group.
- The DSE service user is a member of a group with the same name as the service user. For example, if the DSE service user is cassandra, the cassandra user is a member of the cassandra group.
- sudo is configured so that the DSE service user can execute any command as any slot user without providing a password.

Override the umask setting to 007 for slot users so that files created by sub-processes will not be accessible by anyone else by default, and DSE configuration files are not visible to slot users.

You may further secure the DSE server environment by modifying the OS's limits.conf file to set exact disk space quotas for each slot user.

After adding the slot users and groups and configuring the OS, modify the dse.yaml file. In the spark_process_runner section enable the run_as process runner and set the list of slot users on each node.

```yaml
spark_process_runner:
```
Using DataStax Enterprise advanced functionality

Example configuration for run_as process runner

In this example, two slot users, `slot1` and `slot2` will be created and configured with DSE. The default DSE service user of `cassandra` is used.

1. Create the slot users.

   ```sh
   $ sudo useradd -r -s /bin/false slot1 &&
   sudo useradd -r -s /bin/false slot2
   ```

2. Add the slot users to the DSE service user's group.

   ```sh
   $ sudo usermod -a -G slot1,slot2 cassandra
   ```

3. Make sure the DSE service user is a member of a group with the same name as the service user. For example, if the DSE service user is `cassandra`:

   ```sh
   $ groups cassandra
   cassandra : cassandra
   ```

4. Log out and back in again to make the group changes take effect.

5. Modify the `sudoers` file with the slot users.

   ```sh
   RunAs_Alias     SLOTS = slot1, slot2
   Defaults>SLOTS  umask=007
   Defaults>SLOTS  umask_override
   cassandra       ALL=(SLOTS) NOPASSWD: ALL
   ```

6. Modify `dse.yaml` to enable the `run_as` process runner and add the new runners.

   ```yaml
   # Configure the way how the driver and executor processes are created and managed.
   spark_process_runner:
     # Allowed options are: default, run_as
     runner_type: run_as

     # RunAs runner uses sudo to start Spark drivers and executors. A set of predefined fake users, called slots, is used
     # for this purpose. All drivers and executors owned by some DSE user are run as some slot user x. At the same time
   ```
# drivers and executors of any other DSE user use different slots.
run_as_runner_options:
user_slots:
  - slot1
  - slot2

## Configuring the Spark history server

The Spark history server provides a way to load the event logs from Spark jobs that were run with event logging enabled. The Spark history server works only when files were not flushed before the Spark Master attempted to build a history user interface.

To enable the Spark history server:

1. Create a directory for event logs in the DSEFS file system:

   ```bash
   $ dse fs 'mkdir -p /spark/events'
   ```

2. On each node in the cluster, edit the `spark-defaults.conf` file to enable event logging and specify the directory for event logs:

   ```
   #Turns on logging for applications submitted from this machine
   spark.eventLog.dir dsefs:///spark/events
   spark.eventLog.enabled true
   #Sets the logging directory for the history server
   spark.history.fs.logDirectory dsefs:///spark/events
   # Optional property that changes permissions set to event log files
   # spark.eventLog.permissions=777
   ```

3. Start the Spark history server on one of the nodes in the cluster:

   The Spark history server is a front-end application that displays logging data from all nodes in the Spark cluster. It can be started from any node in the cluster.

   If you’ve enabled authentication set the authentication method and credentials in a properties file and pass it to the `dse` command. For example, for basic authentication:

   ```
   spark.hadoop.com.datastax.bdp.fs.client.authentication.basic.username=role
   name
   spark.hadoop.com.datastax.bdp.fs.client.authentication.basic.password=password
   ```

   If you set the event log location in `spark-defaults.xml`, set the `spark.history.fs.logDirectory` property in your properties file.

   ```
   spark.history.fs.logDirectory=dsefs:///spark/events
   ```
$ dse spark-history-server start

With a properties file:

dse spark-history-server start --properties-file properties file

The history server is started and can be viewed by opening a browser to

**Note:** The Spark Master web UI does not show the historical logs. To work around this known issue, access the history from port 18080.

4. When event logging is enabled, the default behavior is for all logs to be saved, which causes the storage to grow over time. To enable automated cleanup edit `spark-defaults.conf` and edit the following options:

```
spark.history.fs.cleaner.enabled true
spark.history.fs.cleaner.interval 1d
spark.history.fs.cleaner.maxAge 7d
```

For these settings, automated cleanup is enabled, the cleanup is performed daily, and logs older than seven days are deleted.

### Setting Spark Cassandra Connector-specific properties

Spark integration uses the Spark Cassandra Connector under the hood. You can use the configuration options defined in that project to configure DataStax Enterprise Spark. Spark recognizes system properties that have the `spark.` prefix and adds the properties to the configuration object implicitly upon creation. You can avoid adding system properties to the configuration object by passing `false` for the `loadDefaults` parameter in the `SparkConf` constructor.

The full list of parameters is included in the Spark Cassandra Connector documentation.

You pass settings for Spark, Spark Shell, and other DataStax Enterprise Spark built-in applications using the intermediate application `spark-submit`, described in Spark documentation.

### Configuring the Spark shell

Pass Spark configuration arguments using the following syntax:

```
$ dse spark [submission_arguments] [application_arguments]
```

where **submission_arguments** are:

```
[--help] [--verbose]
[--conf name=spark.value|sparkproperties.conf]
[--executor-memory memory]
```
Using DataStax Enterprise advanced functionality

```
[--jars additional-jars]
[--master dse://?appReconnectionTimeoutSeconds=secs]
[--properties-file path_to_properties_file]
[--total-executor-cores cores]

--conf name=spark.value|sparkproperties.conf
An arbitrary Spark option to the Spark configuration prefixed by spark.
  • name-spark.value
  • sparkproperties.conf - a configuration

--executor-memory mem
The amount of memory that each executor can consume for the application.
Spark uses a 512 MB default. Specify the memory argument in JVM format using
the k, m, or g suffix.

--help
Shows a help message that displays all options except DataStax Enterprise
Spark shell options.

--jars path_to_additional_jars
A comma-separated list of paths to additional JAR files.

--properties-file path_to_properties_file
The location of the properties file that has the configuration settings. By default,
Spark loads the settings from spark-defaults.conf.

--total-executor-cores cores
The total number of cores the application uses.

--verbose
Displays which arguments are recognized as Spark configuration options and
which arguments are forwarded to the Spark shell.

Spark shell application arguments:

-i app_script_file
Spark shell application argument that runs a script from the specified file.

Configuring Spark applications

You pass the Spark submission arguments using the following syntax:

```
$ dse spark-submit [submission_arguments] application_file
[application_arguments]
```

All submission_arguments (page 235) and these additional spark-submit
submission_arguments:

--class class_name
The full name of the application main class.

--name appname
The application name as displayed in the Spark web application.

--py-files files
A comma-separated list of the .zip, .egg, or .py files that are set on
PYTHONPATH for Python applications.

--files files
A comma-separated list of files that are distributed among the executors and available for the application.

In general, Spark submission arguments are translated into system properties – Dname=value and other VM parameters like classpath. The application arguments are passed directly to the application.

Property list

When you run dse spark-submit on a node in your Analytics cluster, all the following properties are set automatically, and the Spark Master is automatically detected. Only set the following properties if you need to override the automatically managed properties.

**spark.cassandra.connection.native.port**
Default = 9042. Port for native client protocol connections.

**spark.cassandra.connection.rpc.port**
Default = 9160. Port for thrift connections.

**spark.cassandra.connection.host**
The host name or IP address to which the Thrift RPC service and native transport is bound. The `native_transport_address` property in the `cassandra.yaml`, which is localhost by default, determines the default value of this property.

You can explicitly set the Spark Master address (page 188) using the `--master master address` parameter to dse spark-submit.

```
$ dse spark-submit --master master address application JAR file
```

For example, if the Spark node is at 10.0.0.2:

```
$ dse spark-submit --master dse://10.0.0.2? myApplication.jar
```

The following properties can be overridden for performance or availability:

**Read properties**

**spark.cassandra.input.split.size**
Default = 100000. Approximate number of rows in a single Spark partition. The higher the value, the fewer Spark tasks are created. Increasing the value too much may limit the parallelism level.

**spark.cassandra.input.fetch.size_in_rows**
Default = 1000. Number of rows being fetched per round-trip to the database. Increasing this value increases memory consumption. Decreasing the value increases the number of round-trips. In earlier releases, this property was `spark.cassandra.input.page.row.size`.

**spark.cassandra.input.consistency.level**
Default = LOCAL_QUORUM. Consistency level to use when reading.

**spark.cassandra.input.throughputMBPerSec**
Default = Unlimited. Threshold in MB per second to set a read throttle per task. This threshold helps manage resources when multiple jobs are running in parallel.
Using DataStax Enterprise advanced functionality

Write properties

You can set the following properties in SparkConf to fine tune the saving process.

**spark.cassandra.output.batch.size.bytes**
- Default = auto. Number of bytes per single batch. The default, auto, means the connector adjusts the number of bytes based on the amount of data.

**spark.cassandra.output.consistency.level**
- Default = LOCAL_ONE. Consistency level to use when writing.

**spark.cassandra.output.concurrent.writes**
- Default = 100. Maximum number of batches executed in parallel by a single Spark task.

**spark.cassandra.output.batch.size.rows**
- Default = 64K. The maximum total size of the batch in bytes.

See the Spark Cassandra Connector documentation for details on additional, low-level properties.

Creating a DSE Analytics Solo datacenter

DSE Analytics Solo datacenters do not store any database or search data, but are strictly used for analytics processing. They are used in conjunction with one or more datacenters that contain database data.

Creating a DSE Analytics Solo datacenter within an existing DSE cluster

In this example scenario, there is an existing datacenter, DC1 which has existing database data. Create a new DSE Analytics Solo datacenter, DC2, which does not store any data but will perform analytics jobs using the database data from DC1.

- Make sure all keyspaces in the DC1 datacenter use NetworkTopologyStrategy. If necessary, alter the keyspace.

  ```
  ALTER KEYSPACE mykeyspace
  WITH REPLICAION = { 'class' = 'NetworkTopologyStrategy', 'DC1' : 3 };
  ```

- Add nodes to a new datacenter named DC2, then enable Analytics on those nodes (page 1437).

- Configure the dse_leases and dse_analytics keyspaces to replicate to both DC1 and DC2. For example:

  ```
  ALTER KEYSPACE dse_leases
  WITH REPLICAION = { 'class' = 'NetworkTopologyStrategy', 'DC1' : 3, 'DC2' : 3 };
  ```

- When submitting Spark applications specify the --master URL with the name or IP address of a node in the DC2 datacenter, and set the spark.cassandra.connection.local_dc configuration option to DC1.
Using DataStax Enterprise advanced functionality

dse spark-submit --master "dse://?connection.local_dc=DC2"
        --class com.datastax.dse.demo.loss.Spark10DayLoss --conf
"spark.cassandra.connection.local_dc=DC1" portfolio.jar

The Spark workers read the data from the DC1.

Accessing an external DSE transactional cluster from a DSE Analytics Solo cluster

To access an external DSE transactional cluster, explicitly set the connection to the transactional cluster when creating RDDs or Datasets within the application.

In the following examples, the external DSE transactional cluster has a node running on 10.10.0.2.

To create an RDD from the transactional cluster's data:

```scala
import com.datastax.spark.connector._
import com.datastax.spark.connector.cql._
import org.apache.spark.SparkContext

def analyticsSoloExternalDataExample ( sc: SparkContext) = {
  val connectorToTransactionalCluster =
    CassandraConnector(sc.getConf.set("spark.cassandra.connection.host", "10.10.0.2"))

  val rddFromTransactionalCluster = {
    // Sets connectorToTransactionalCluster as default connection for everything in this code block
    implicit val c = connectorToTransactionalCluster
    // get the data from the test.words table
    sc.cassandraTable("test","words")
  }
}
```

Creating a Dataset from the transactional:

```scala
import org.apache.spark.sql.cassandra._
import com.datastax.spark.connector.cql.CassandraConnectorConf

// set params for the particular cluster
spark.setCassandraConf("TransactionalCluster",
    CassandraConnectorConf.ConnectionHostParam.option("10.10.0.2"))

val df = spark
  .read
  .format("org.apache.spark.sql.cassandra")
  .options(Map( "table" -> "words", "keyspace" -> "test"))
  .load()
```
When you submit the application to the DSE Analytics Solo cluster, it will retrieve the data from the external DSE transactional cluster.

**Spark JVMs and memory management**

Spark jobs running on DataStax Enterprise are divided among several different JVM processes, each with different memory requirements.

**DataStax Enterprise and Spark Master JVMs**

The Spark Master runs in the same process as DataStax Enterprise, but its memory usage is negligible. The only way Spark could cause an `OutOfMemoryError` in DataStax Enterprise is indirectly by executing queries that fill the client request queue. For example, if it ran a query with a high limit and paging was disabled or it used a very large batch to update or insert data in a table. This is controlled by `MAX_HEAP_SIZE` in `cassandra-env.sh`. If you see an `OutOfMemoryError` in `system.log`, you should treat it as a standard `OutOfMemoryError` and follow the usual troubleshooting steps.

**Spark executor JVMs**

The Spark executor is where Spark performs transformations and actions on the RDDs and is usually where a Spark-related `OutOfMemoryError` would occur. An `OutOfMemoryError` in an executor will show up in the `stderr` log for the currently executing application (usually in `/var/lib/spark`). There are several configuration settings that control executor memory and they interact in complicated ways.

- The `memory_total` option in the `resource_manager_options.worker_options` section of `dse.yaml` defines the maximum fraction of system memory to give all executors for all applications running on a particular node. It uses the following formula:

  \[
  \text{memory\_total} \times (\text{total system memory} - \text{memory assigned to DataStax Enterprise})
  \]

- `spark.executor.memory` is a system property that controls how much executor memory a specific application gets. It must be less than or equal to the calculated value of `memory_total`. It can be specified in the constructor for the `SparkContext` in the driver application, or via `--conf spark.executor.memory` or `--executor-memory` command line options when submitting the job using `spark-submit`.

**The client driver JVM**

The driver is the client program for the Spark job. Normally it shouldn’t need very large amounts of memory because most of the data should be processed within the executor. If it does need more than a few gigabytes, your application may be using an anti-pattern like pulling all of the data in an RDD into a local data structure by using `collect` or `take`. Generally you should never use `collect` in production code and if you use `take`, you should be only taking a few records. If the driver runs out of memory, you will see the `OutOfMemoryError` in the driver `stderr` or wherever it's been configured to log. This is controlled one of two places:
Using DataStax Enterprise advanced functionality

- **SPARK_DRIVER_MEMORY** in spark-env.sh
- spark.driver.memory system property which can be specified via --conf spark.driver.memory or --driver-memory command line options when submitting the job using spark-submit. This cannot be specified in the SparkContext constructor because by that point, the driver has already started.

Spark worker JVMs

The worker is a watchdog process that spawns the executor, and should never need its heap size increased. The worker's heap size is controlled by **SPARK_DAEMON_MEMORY** in spark-env.sh. **SPARK_DAEMON_MEMORY** also affects the heap size of the Spark SQL thrift server.

Using Spark modules with DataStax Enterprise

Getting started with Spark Streaming

**Spark Streaming** allows you to consume live data streams from sources, including Akka, Kafka, and Twitter. This data can then be analyzed by Spark applications, and the data can be stored in the database.

You use Spark Streaming by creating an instance based on your Spark configuration. You then create a DStream instance, or a *discretionized stream*, an object that represents an input stream. DStream objects are created by calling one of the methods of StreamingContext, or using a utility class from external libraries to connect to other sources like Twitter.

The data you consume and analyze is saved to the database by calling one of the saveToCassandra methods on the stream object, passing in the keyspace name, the table name, and optionally the column names and batch size.

**Note:** Spark Streaming applications require synchronized clocks to operate correctly. See Synchronize clocks (page 58).

The following Scala example demonstrates how to connect to a text input stream at a particular IP address and port, count the words in the stream, and save the results to the database.

1. Import the streaming context objects.

```scala
import org.apache.spark.streaming._
```

2. Create a new `StreamingContext` object based on an existing `SparkConf` configuration object, specifying the interval in which streaming data will be divided into batches by passing in a batch duration.

```scala
val sparkConf = ....
```
Using DataStax Enterprise advanced functionality

val ssc = new StreamingContext(sc, Seconds(1)) // Uses the context automatically created by the spark shell

Spark allows you to specify the batch duration in milliseconds, seconds, and minutes.

3. Import the database-specific functions for StreamingContext, DStream, and RDD objects.

import com.datastax.spark.connector.streaming._

4. Create the DStream object that will connect to the IP and port of the service providing the data stream.

val lines = ssc.socketTextStream(server IP address, server port number)

5. Count the words in each batch and save the data to the table.

val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)
    .saveToCassandra("streaming_test", "words_table", SomeColumns("word", "count"))

6. Start the computation.

ssc.start()
ssc.awaitTermination()

In the following example, you start a service using the nc utility that repeats strings, then consume the output of that service using Spark Streaming.

Using cqlsh, start by creating a target keyspace and table for streaming to write into.

CREATE KEYSPACE IF NOT EXISTS streaming_test
WITH REPLICATION = {'class': 'SimpleStrategy',
  'replication_factor': 1};

CREATE TABLE IF NOT EXISTS streaming_test.words_table
(word TEXT PRIMARY KEY, count COUNTER);

In a terminal window, enter the following command to start the service:

$ nc -lk 9999
one two two three three three four four four four someword
In a different terminal start a Spark shell.

$ dse spark

In the Spark shell enter the following:

```scala
import org.apache.spark.streaming._
import com.datastax.spark.connector.streaming._

val ssc = new StreamingContext(sc, Seconds(1))
val lines = ssc.socketTextStream( "localhost", 9999)
val words = lines.flatMap(_.split( " "))
val pairs = words.map(word => (word, 1))

val wordCounts = pairs.reduceByKey(_ + _)
wordCounts.saveToCassandra( "streaming_test", "words_table",
  SomeColumns( "word", "count")
wordCounts.print()
ssc.start()
ssc.awaitTermination()
exit()
```

Using `cqlsh` connect to the `streaming_test` keyspace and run a query to show the results.

$ cqlsh -k streaming_test

```sql
select * from words_table;
```

<table>
<thead>
<tr>
<th>word</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>three</td>
<td>3</td>
</tr>
<tr>
<td>one</td>
<td>1</td>
</tr>
<tr>
<td>two</td>
<td>2</td>
</tr>
<tr>
<td>four</td>
<td>4</td>
</tr>
<tr>
<td>someword</td>
<td>1</td>
</tr>
</tbody>
</table>

What's next:

Run the [http_receiver demo (page 282)](https://www.datastax.com). See the Spark Streaming Programming Guide for more information, API documentation, and examples on Spark Streaming.

**Creating a Spark Structured Streaming sink using DSE**

Spark Structured Streaming is a high-level API for streaming applications. DSE supports Structured Streaming for storing data into DSE.

The following Scala example shows how to store data from a streaming source to DSE using the `cassandraFormat` method.
Using DataStax Enterprise advanced functionality

```scala
val query = source.writeStream
  .option("checkpointLocation", checkpointDir.toString)
  .cassandraFormat("table name", "keyspace name")
  .outputMode(OutputMode.Update)
  .start()
```

This example sets the `OutputMode` to `Update`, described in the Spark API documentation.

The `cassandraFormat` method is equivalent to calling the `format` method and in `org.apache.spark.sql.cassandra`.

```scala
val query = source.writeStream
  .option("checkpointLocation", checkpointDir.toString)
  .format("org.apache.spark.sql.cassandra")
  .option("keyspace", ks)
  .option("table", "kv")
  .outputMode(OutputMode.Update)
  .start()
```

Using Spark SQL to query data

Spark SQL allows you to execute Spark queries using a variation of the SQL language. Spark SQL includes APIs for returning Spark `Dataset` in Scala and Java, and interactively using a SQL shell.

Spark SQL basics

In DSE, Spark SQL allows you to perform relational queries over data stored in DSE clusters, and executed using Spark. Spark SQL is a unified relational query language for traversing over distributed collections of data, and supports a variation of the SQL language used in relational databases. Spark SQL is intended as a replacement for Shark and Hive, including the ability to run SQL queries over Spark data sets. You can use traditional Spark applications in conjunction with Spark SQL queries to analyze large data sets.

The `SparkSession` class and its subclasses are the entry point for running relational queries in Spark.

`DataFrame`s are Spark `Dataset`s organized into named columns, and are similar to tables in a traditional relational database. You can create `DataFrame` instances from any Spark data source, like CSV files, Spark RDDs, or, for DSE, tables in the database. In DSE, when you access a Spark SQL table from the data in DSE transactional cluster, it registers that table to the Hive metastore so SQL queries can be run against it.

**Note:** Any tables you create or destroy, and any table data you delete, in a Spark SQL session will not be reflected in the underlying DSE database, but only in that session’s metastore.
Starting the Spark SQL shell

The Spark SQL shell allows you to interactively perform Spark SQL queries. To start the shell, run `dse spark-sql`:

```
$ dse spark-sql
```

The Spark SQL shell in DSE automatically creates a Spark session and connects to the Spark SQL Thrift server (page 254) to handle the underlying JDBC connections.

If the schema changes in the underlying database table during a Spark SQL session (for example, a column was added using CQL), drop the table and then refresh the metastore to continue querying the table with the correct schema.

```
DROP TABLE tablename;
SHOW TABLES;
```

Queries to a table whose schema has been modified cause a runtime exception.

Spark SQL limitations

- You cannot load data from one file system to a table in a different file system.

```
CREATE TABLE IF NOT EXISTS test (id INT, color STRING) PARTITIONED BY (ds STRING);
LOAD DATA INPATH 'hdfs2://localhost/colors.txt' OVERWRITE INTO TABLE test PARTITION (ds = '2008-08-15');
```

The first line creates a table on the default file system. The second line attempts to load data into that table from a path on a different file system, and will fail.

Querying database data using Spark SQL in Scala

When you start Spark, DataStax Enterprise creates a Spark session instance to allow you to run Spark SQL queries against database tables. The session object is named `spark` and is an instance of `org.apache.spark.sql.SparkSession`. Use the `sql` method to execute the query.

1. Start the Spark shell.

```
$ dse spark
```

2. Use the `sql` method to pass in the query, storing the result in a variable.

```
val results = spark.sql("SELECT * from my_keyspace_name.my_table")
```

3. Use the returned data.

```
results.show()
```
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>id</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>de2d0de1-4d70-11e...</td>
<td>thing</td>
</tr>
<tr>
<td>db7e4191-4d70-11e...</td>
<td>another</td>
</tr>
<tr>
<td>d576ad50-4d70-11e...</td>
<td>yet another</td>
</tr>
</tbody>
</table>

Querying database data using Spark SQL in Java

Java applications that query table data using Spark SQL first need an instance of `org.apache.spark.sql.SparkSession`.

The Spark session object is used to connect to DataStax Enterprise.

Create the Spark session instance using the **builder interface**:

```java
SparkSession spark = SparkSession
    .builder()
    .appName("My application name")
    .config("option name", "option value")
    .master("dse://1.1.1.1?connection.host=1.1.2.2,1.1.3.3")
    .getOrCreate();
```

After the Spark session instance is created, you can use it to create a `DataFrame` instance from the query. Queries are executed by calling the `SparkSession.sql` method.

```java
DataFrame employees = spark.sql("SELECT * FROM company.employees");
employees.registerTempTable("employees");
DataFrame managers = spark.sql("SELECT name FROM employees WHERE role = 'Manager'");
```

The returned `DataFrame` object supports the standard Spark operations.

```java
employees.collect();
```

Querying DSE Graph vertices and edges with Spark SQL

Spark SQL can query DSE Graph vertex and edge tables. The `dse_graph` database holds the vertex and edge tables for each graph. The naming format for the tables is `graph name_vertices` and `graph name_edges`. For example, if you have a graph named `gods`, the vertices and edges are accessible in Spark SQL in the `dse_graph.gods_vertices` and `dse_graph.gods_edges` tables.

```sql
select * from dse_graph.gods_vertices;
```

If you have properties that are spelled the same but with different capitalizations (for example, `id` and `Id`), start Spark SQL with the `--conf spark.sql.caseSensitive=true` option.

Prerequisites:
Start your cluster with both Graph and Spark enabled (page 1437).

1. Start the Spark SQL shell.
   
   ```
   $ dse spark-sql
   ```

2. Query the vertices and edges using `SELECT` statements.

   ```
   USE dse_graph;
   SELECT * FROM gods_vertices where name = 'Zeus';
   ```

3. Join the vertices and edges in a query.

   Vertices are identified by `id` columns. Edge tables have `src` and `dst` columns that identify the from and to vertices, respectively. A join can be used to traverse the graph. For example to find all vertex ids that are reached by the out edges:

   ```
   SELECT gods_edges.dst FROM gods_vertices JOIN gods_edges ON gods_vertices.id = gods_edges.src;
   ```

What's next: The same steps work from the Spark shell using `spark.sql()` to run the query statements, or using the JDBC (page 264)/ODBC (page 265) driver and the Spark SQL Thrift Server (page 254).

Using Spark predicate push down in Spark SQL queries

Spark predicate push down to database allows for better optimized Spark queries. A predicate is a condition on a query that returns true or false, typically located in the `WHERE` clause. A predicate push down filters the data in the database query, reducing the number of entries retrieved from the database and improving query performance. By default the Spark Dataset API will automatically push down valid `WHERE` clauses to the database.

You can also use predicate push down on DSE Search indices (page 179) within SearchAnalytics data centers.

Restrictions on column filters

Partition key columns can be pushed down as long as:

- All partition key columns are included in the filter.
- No more than one equivalence predicate per column.

Use an `IN` clause to specify multiple restrictions for a particular column:

```
val primaryColors = List("red", "yellow", "blue")
val df = spark.read.cassandraFormat("cars", "inventory").load
```
df.filter(df("car_color").isin(primaryColors: _*))

Clustering key columns can be pushed down with the following rules:

- Only the last predicate in the filter can be a non equivalence predicate.
- If there is more than one predicate for a column, the predicates cannot be equivalence predicates.

When predicate push down occurs

When a Dataset has no push down filters, all requests on the Dataset do a full unfiltered table scan. Adding predicate filters on the Dataset for eligible database columns modifies the underlying query to narrow its scope.

Determining if predicate push down is being used in queries

By using the `explain` method on a Dataset (or `EXPLAIN` in Spark SQL), queries can be analyzed to see if the predicates need to be cast to the correct data type. For example, create the following CQL table:

```cql
CREATE KEYSPACE test WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
USE test;
CREATE table words(
    user TEXT,
    word TEXT,
    count INT,
    PRIMARY KEY (user, word));
```

```cql
INSERT INTO words (user, word, count ) VALUES ( 'Russ', 'dino', 10 );
INSERT INTO words (user, word, count ) VALUES ( 'Russ', 'fad', 5 );
INSERT INTO words (user, word, count ) VALUES ( 'Sam', 'alpha', 3 );
INSERT INTO words (user, word, count ) VALUES ( 'Zebra', 'zed', 100 );
```

Then create a Spark Dataset in the Spark console using that table and look for PushedFilters in the output after issuing the `EXPLAIN` command:

```scala
val df = spark.read.cassandraFormat("words", "test").load
df.explain
```

```text
== Physical Plan ==
*Scan org.apache.spark.sql.cassandra.CassandraSourceRelation
  [user#0,word#1,count#2] ReadSchema:
    struct<user:string,word:string,count:int>
```

Because this query doesn't filter on columns capable of being pushed down, there are no PushedFilters in the physical plan.

Adding a filter, however, does change the physical plan to include PushedFilters:

```scala
val dfWithPushdown = df.filter(df("word") > "ham")
```
dfWithPushdown.explain

== Physical Plan ==
*Scan org.apache.spark.sql.cassandra.CassandraSourceRelation
  [user#0,word#1,count#2] PushedFilters: [GreaterThan(word,ham)],
  ReadSchema: struct<user:string,word:string,count:int>

The PushedFilters section of the physical plan includes the GreaterThan push down filter. The asterisk indicates that push down filter will be handled only at the datasource level.

Troubleshooting predicate push down

When creating Spark SQL queries that use comparison operators, making sure the predicates are pushed down to the database correctly is critical to retrieving the correct data with the best performance.

For example, given a CQL table with the following schema:

```sql
CREATE TABLE test.common (
    year int,
    birthday timestamp,
    userid uuid,
    likes text,
    name text,
    PRIMARY KEY (year, birthday, userid)
)
```

Suppose you want to write a query that selects all entries where the birthday is earlier than a given date:

```
SELECT * FROM test.common WHERE birthday < '2001-1-1';
```

Use the `EXPLAIN` command to see the query plan:

```
EXPLAIN SELECT * FROM test.common WHERE birthday < '2001-1-1';
```

== Physical Plan ==
*Filter (cast(birthday#1 as string) < 2001-1-1)
+- *Scan org.apache.spark.sql.cassandra.CassandraSourceRelation
  [year#0,birthday#1,userid#2,likes#3,name#4] ReadSchema:
    struct<year:int,birthday:timestamp,userid:string,likes:string,name:string>
  Time taken: 0.72 seconds, Fetched 1 row(s)

Note that the Filter directive is treating the birthday column, a CQL TIMESTAMP, as a string. The query optimizer looks at this comparison and needs to make the types match before generating a predicate. In this case the optimizer decides to cast the birthday column as a string to match the string '2001-1-1', but cast functions cannot be pushed down. The predicate isn't pushed down, and it doesn't appear in PushedFilters. A full
Using DataStax Enterprise advanced functionality

Table scan will be performed at the database layer, with the results returned to Spark for further processing.

To push down the correct predicate for this query, use the `cast` method to specify that the predicate is comparing the `birthday` column to a `TIMESTAMP`, so the types match and the optimizer can generate the correct predicate.

```
EXPLAIN SELECT * FROM test.common WHERE birthday < cast('2001-1-1' as TIMESTAMP);
```

```
== Physical Plan ==
*Scan org.apache.spark.sql.cassandra.CassandraSourceRelation
  [year#0,birthday#1,userid#2,likes#3,name#4]
PushedFilters: [*LessThan(birthday,2001-01-01 00:00:00.0)],
ReadSchema:
  struct<year:int,birthday:timestamp,userid:string,likes:string,name:string>
Time taken: 0.034 seconds, Fetched 1 row(s)
```

Note the `PushedFilters` indicating that the `LessThan` predicate will be pushed down for the column data in `birthday`. This should speed up the query as a full table scan will be avoided.

**Supported syntax of Spark SQL**

**Syntax:**

The following syntax defines a `SELECT` query.

```
SELECT [DISTINCT] [column names] | [wildcard] 
FROM [keyspace name.]table name
[JOIN clause table name ON join condition] 
[WHERE condition]
[GROUP BY column name]
[HAVING conditions]
[ORDER BY column names [ASC | DSC]]
```

A `SELECT` query using joins has the following syntax.

```
SELECT statement 
FROM statement 
[JOIN | INNER JOIN | LEFT JOIN | LEFT SEMI JOIN | LEFT OUTER JOIN | 
RIGHT JOIN | RIGHT OUTER JOIN | FULL JOIN | FULL OUTER JOIN] 
on join condition
```

Several select clauses can be combined in a `UNION`, `INTERSECT`, or `EXCEPT` query.

```
SELECT statement 1
[UNION | UNION ALL | UNION DISTINCT | INTERSECT | EXCEPT] 
SELECT statement 2
```
**Note:** Select queries run on new columns return ' ', or empty results, instead of None.

Syntax:
The following syntax defines an **INSERT** query.

```
INSERT [OVERWRITE] INTO [keyspace name.]table name VALUES values
```

Syntax:
The following syntax defines a **CACHE TABLE** query.

```
CACHE TABLE table name [AS table alias]
```

You can remove a table from the cache using a **UNCACHE TABLE** query.

```
UNCACHE TABLE table name
```

Keywords in Spark SQL
The following keywords are reserved in Spark SQL.

- ALL
- AND
- AS
- ASC
- APPROXIMATE
- AVG
- BETWEEN
- BY
- CACHE
- CAST
- COUNT
- DESC
- DISTINCT
- FALSE
- FIRST
- LAST
- FROM
- FULL
- GROUP
- HAVING
- IF
- IN
- INNER
Inserting data into tables with static columns using Spark SQL

Static columns are mapped to different columns in Spark SQL and require special handling. Spark SQL Thrift servers use Hive. When you when run an insert query, you must pass data to those columns.

To work around the different columns, set `cql3.output.query` in the insertion Hive table properties to limit the columns that are being inserted. In Spark SQL, alter the external table to configure the prepared statement as the value of the Hive CQL output query. For example, this prepared statement takes values that are inserted into columns a and b in
Using DataStax Enterprise advanced functionality

mytable and maps these values to columns b and a, respectively, for insertion into the
new row.

spark-sql> ALTER TABLE mytable SET TBLPROPERTIES ('cql3.output.query' = 'update
    mykeyspace.mytable set b = ? where a = ?');
spark-sql> ALTER TABLE mytable SET SERDEPROPERTIES ('cql3.update.columns' = 'b,a');

Running HiveQL queries using Spark SQL

Spark SQL supports queries written using HiveQL, a SQL-like language that produces
queries that are converted to Spark jobs. HiveQL is more mature and supports more
complex queries than Spark SQL. To construct a HiveQL query, first create a new
HiveContext instance, and then submit the queries by calling the sql method on the
HiveContext instance.

See the Hive Language Manual for the full syntax of HiveQL.

Note: Creating indexes with DEFERRED REBUILD is not supported in Spark SQL.

1. Start the Spark shell.

   $ bin/dse spark

2. Use the provided HiveContext instance sqlContext to create a new query in
   HiveQL by calling the sql method on the sqlContext object.

   val results = sqlContext.sql("SELECT * FROM
    my_keyspace.my_table")

Using the DataFrames API

The Spark DataFrames API encapsulates data sources, including DataStax Enterprise
data, organized into named columns.

The Spark Cassandra Connector provides an integrated DataSource to simplify
creating DataFrames. For more technical details, see the Spark Cassandra Connector
documentation that is maintained by DataStax and the Cassandra and PySpark
DataFrames post.

Examples of using the DataFrames API

This Python example shows using the DataFrames API to read from the table ks.kv and
insert into a different table ks.othertable.

$ dse pyspark
Using DataStax Enterprise advanced functionality

```scala
table1 = spark.read.format("org.apache.spark.sql.cassandra")
  .options(table="kv", keyspace="ks")
  .load()

table1.write.format("org.apache.spark.sql.cassandra")
  .options(table="othertable", keyspace = "ks")
  .save(mode ="append")
```

Using the DSE Spark console, the following Scala example shows how to create a DataFrame object from one table and save it to another.

```
$ dse spark

val table1 = spark.read.format("org.apache.spark.sql.cassandra")
  .options(Map( "table" -> "words", "keyspace" -> "test"))
  .load()

table1.createCassandraTable("test", "otherwords", partitionKeyColumns = Some(Seq("word")), clusteringKeyColumns = Some(Seq("count")))
table1.write.cassandraFormat("otherwords", "test").save()
```

The write operation uses one of the helper methods, `cassandraFormat`, included in the Spark Cassandra Connector. This is a simplified way of setting the format and options for a standard DataFrame operation. The following command is equivalent to write operation using `cassandraFormat`:

```
val table1 = spark.read.format("org.apache.spark.sql.cassandra")
  .options(Map("table" -> "othertable", "keyspace" -> "test"))
  .load()

table1.write.format("org.apache.spark.sql.cassandra")
  .options(Map("table" -> "othertable", "keyspace" -> "test"))
  .save()
```

**Using the Spark SQL Thriftserver**

The Spark SQL Thriftserver uses JDBC and ODBC interfaces for client connections to the database.

The [AlwaysOn SQL (page 257)](https://example.com) service is a high-availability service built on top of the Spark SQL Thriftserver. The Spark SQL Thriftserver is started manually on a single node in an Analytics datacenter, and will not failover to another node. Both AlwaysOn SQL and the Spark SQL Thriftserver provide JDBC and ODBC interfaces to DSE, and share many configuration settings.

1. If you are using Kerberos authentication, in the `hive-site.xml` file, configure your authentication credentials for the Spark SQL Thrift server.

   ```xml
   <property>
   <name>hive.server2.authentication.kerberos.principal</name>
   <value>thriftserver/\_HOST@EXAMPLE.COM</value>
   </property>
   <property>
   <name>hive.server2.authentication.kerberos.keytab</name>
   <value>/etc/dse/dse.keytab</value>
   ```
Ensure that you use the `hive-site.xml` file in the Spark directory:

- **Package installations:** `/etc/dse/spark/hive-site.xml`
- **Tarball installations:** `installation_location/resources/spark/conf/hive-site.xml`

2. Start DataStax Enterprise with Spark enabled as a service (page 1437) or in a standalone (page 1440) installation.

3. Start the server by entering the `dse spark-sql-thriftserver start` command as a user with permissions to write to the Spark directories.

   To override the default settings for the server, pass in the configuration property using the `--hiveconf` option. See the HiveServer2 documentation for a complete list of configuration properties.

   
   ```
   $ dse spark-sql-thriftserver start
   ```

   By default, the server listens on port 10000 on the localhost interface on the node from which it was started. You can specify the server to start on a specific port. For example, to start the server on port 10001, use the `--hiveconf hive.server2.thrift.port=10001` option.

   ```
   $ dse spark-sql-thriftserver start --hiveconf hive.server2.thrift.port=10001
   ```

   You can configure the port and bind address permanently in `resources/spark/conf/spark-env.sh`:

   ```
   $ export HIVE_SERVER2_THRIFT_PORT=10001
   export HIVE_SERVER2_THRIFT_BIND_HOST=1.1.1.1
   ```

   You can specify general Spark configuration settings by using the `--conf` option.

   ```
   $ dse spark-sql-thriftserver start --conf spark.cores.max=4
   ```

4. Use DataFrames to read and write large volumes of data. For example, to create the `table_a_cass_df` table that uses a DataFrame while referencing `table_a`:

   ```
   CREATE TABLE table_a_cass_df using org.apache.spark.sql.cassandra OPTIONS (table "table_a", keyspace "ks")
   ```

   **Note:** With DataFrames, compatibility issues exist with UUID and Inet types when inserting data with the JDBC driver.
5. Use the Spark Cassandra Connector tuning parameters to optimize reads and writes.

6. To stop the server, enter the `dse spark-sql-thriftserver stop` command.

```
$ dse spark-sql-thriftserver stop
```

What's next:

You can now connect your application by using the Simba JDBC driver (page 264) to the server at the URI: `jdbc:hive2://hostname:port number`, using the Simba ODBC driver (page 265) or use `dse beeline` (page 265).

**Using SparkR with DataStax Enterprise**

Apache SparkR is a front-end for the R programming language for creating analytics applications. DataStax Enterprise integrates SparkR to support creating data frames from DSE data.

SparkR support in DSE requires you to first install R on the client machines on which you will be using SparkR. To use R user defined functions and distributed functions the same version of R should be installed on all the nodes in the Analytics cluster. DSE SparkR is built against R version 3.1.1. Many Linux distributions by default install older versions of R.

For example, on Debian and Ubuntu clients:

```
$ sudo sh -c 'echo "deb http://cran.rstudio.com/bin/linux/ubuntu trusty/" >> /etc/apt/sources.list' &&
gpg --keyserver keyserver.ubuntu.com --recv-key E084DAB9 &&
gpg -a --export E084DAB9 | sudo apt-key add - &&
sudo apt-get update &&
sudo apt-get install r-base
```

On RedHat and CentOS clients:

```
$ sudo yum install R
```

**Starting SparkR**

Start the SparkR shell using the `dse command` (page 1147) to automatically set the Spark session within R.

1. Start the R shell using the `dse` command.
Using AlwaysOn SQL service

AlwaysOn SQL is a high availability service that responds to SQL queries from JDBC and ODBC applications. By default, AlwaysOn SQL is disabled. It is built on top of the Spark SQL Thriftserver (page 254), but provides failover and caching between instances so there is no single point of failure. AlwaysOn SQL provides enhanced security, leveraging the same user management as the rest of DSE, executing queries to the underlying database as the user authenticated to AlwaysOn SQL.

In order to run AlwaysOn SQL, you must have:

- A running datacenter with DSE Analytics nodes enabled.
- Enabled AlwaysOn SQL (page 257) on every Analytics node in the datacenter.
- Modified the replication factor for all Analytics nodes (page 177), if necessary.
- Set the native_transport_address in cassandra.yaml to an IP address accessible by the AlwaysOn SQL clients. This address depends on your network topology and deployment scenario.
- Configured AlwaysOn SQL for security (page 261), if authentication is enabled.

Lifecycle Manager allows you to enable and configure AlwaysOn SQL in managed clusters.

When AlwaysOn SQL is enabled within an Analytics datacenter, all nodes within the datacenter must have AlwaysOn SQL enabled. Use dsetool ring (page 1361) to find which nodes in the datacenter are Analytics nodes.

Note: AlwaysOn SQL is not supported when using DSE Multi-Instance or other deployments with multiple DSE instances on the same server.

The dse client-tool alwayson-sql command controls the server. The command works on the local datacenter unless you specify the datacenter with the --dc option:

```
$ dse client-tool alwayson-sql --dc datacenter_name command
```

Enabling AlwaysOn SQL

Set enabled to true and uncomment the AlwaysOn SQL options (page 134) in dse.yaml.

Configuring AlwaysOn SQL

The alwayson_sql_options section in dse.yaml, described in detail at AlwaysOn SQL options (page 134), has options for setting the ports, timeout values, log location, and other Spark or Hive configuration settings. Additional configuration options are located in spark-alwayson-sql.conf.

AlwaysOn SQL binds to the native_transport_address in cassandra.yaml.
Using DataStax Enterprise advanced functionality

If you have changed some configuration settings in dse.yaml while AlwaysOn SQL is running, you can have the server pick up the new configuration by entering:

```bash
dse client-tool alwayson-sql reconfig
```

The following settings can be changed using `reconfig`:

- `reserve_port_wait_time_ms`
- `alwayson_sql_status_check_wait_time_ms`
- `log_dsefs_dir`
- `runner_max_errors`

Changing other options requires a restart, except for the `enabled` option. Enabling or disabling AlwaysOn SQL requires restarting DSE (page 1437).

The `spark-alwayson-sql.conf` file contains Spark and Hive settings as properties. When AlwaysOn SQL is started, `spark-alwayson-sql.conf` is scanned for Spark properties, similar to other Spark applications started with `dse spark-submit`. Properties that begin with `spark.hive` are submitted as properties using `--hiveconf`, removing the `spark.` prefix.

For example, if `spark-alwayson-sql.conf` has the following setting:

```text
spark.hive.server2.table.type.mapping CLASSIC
```

That setting will be converted to `--hiveconf hive.server2.table.type.mapping=CLASSIC` when AlwaysOn SQL is started.

Configuring AlwaysOnSQL in a DSE Analytics Solo datacenter

If AlwaysOn SQL is used in a DSE Analytics Solo datacenter, modify `spark-alwayson-sql.conf` to configure Spark with the DSE Analytics Solo datacenters. In the following example, the transactional datacenter name is `dc0` and the DSE Analytics Solo datacenter is `dc1`.

Under `spark.master` set the Spark URI to the connect to the DSE Analytics Solo datacenter.

```bash
spark.master=dse://?connection.local_dc=dc1
```

Add the `spark.cassandra.connection.local_dc` property to `spark-alwayson-sql.conf` and set it to the name of the transactional datacenter.

```bash
spark.cassandra.connection.local_dc=dc0
```

Starting and stopping AlwaysOn SQL

If you have enabled AlwaysOn SQL, it will start when the cluster is started. If AlwaysOn SQL is enabled and DSE is restarted, AlwaysOn SQL will be started regardless of the
previous state of AlwaysOn SQL. You only need to explicitly start the server if it has been stopped, for example for a configuration change.

To start AlwaysOn SQL service:

```bash
$ dse client-tool alwayson-sql start
```

To start the server on a specific datacenter, specify the datacenter name with the `--dc` option:

```bash
$ dse client-tool alwayson-sql --dc dc-west start
```

To completely stop AlwaysOn SQL service:

```bash
$ dse client-tool alwayson-sql stop
```

The server must be manually started after issuing a `stop` command.

To restart a running server:

```bash
$ dse client-tool alwayson-sql restart
```

Checking the status of AlwaysOn SQL

To find the status of AlwaysOn SQL issue a `status` command using `dse-client-tool`.

```bash
$ dse client-tool alwayson-sql status
```

You can also view the status in a web browser by going to `http://node name or IP address:AlwaysOn SQL web UI port`. By default, the port is 9077. For example, if 10.10.10.1 is the IP address of an Analytics node with AlwaysOn SQL enabled, navigate to `http://10.10.10.1:9077`.

The returned status is one of:

- **RUNNING**: the server is running and ready to accept client requests.
- **STOPPED_AUTO_RESTART**: the server is being started but is not yet ready to accept client requests.
- **STOPPED_MANUAL_RESTART**: the server was stopped with either a `stop` or `restart` command. If the server was issued a `restart` command, the status will be changed to `STOPPED_AUTO_RESTART` as the server starts again.
- **STARTING**: the server is actively starting up but is not yet ready to accept client requests.

Caching tables within Spark SQL queries

To increase performance, you can specify tables to be cached into RAM using the `CACHE TABLE` directive. Permanent cached tables will be recached on server restart.
You can cache an existing table by issuing a `CACHE TABLE` Spark SQL command through a client:

```
CACHE TABLE keyspace_name.table_name;
CACHE TABLE keyspace_name.table_name AS select statement;
```

The temporary cache table is only valid for the session in which it was created, and will not be recreated on server restart.

Create a permanent cache table using the `CREATE CACHE TABLE` directive and a `SELECT` query:

```
CREATE CACHE TABLE keyspace_name.table_name AS select_statement;
```

Cached tables can be destroyed using the `UNCACHE TABLE` and `CLEAR CACHE` directives.

```
UNCACHE TABLE keyspace_name.table_name;
```

The `CLEAR CACHE` directive removes all cached tables.

```
CLEAR CACHE;
```

**Enabling SSL for AlwaysOn SQL**

Communication between the driver and AlwaysOn SQL can be encrypted using SSL.

The following instructions give an example of how to set up SSL with a self-signed keystore and truststore.

1. Ensure client-to-node encryption is enabled and configured correctly.

2. If the SSL keystore and truststore used for AlwaysOn SQL differ from the keystore and truststore configured in `cassandra.yaml`, add the required settings to enable SSL to the `hive-site.xml` configuration file.

   **Note:** By default the SSL settings in `cassandra.yaml` will be used with AlwaysOn SQL.

```
<property>
    <name>hive.server2.thrift.bind.host</name>
    <value>hostname</value>
</property>
<property>
    <name>hive.server2.use.SSL</name>
    <value>true</value>
</property>
<property>
    <name>hive.server2.keystore.path</name>
    <value>path to keystore/keystore.jks</value>
</property>
```
3. Start or restart the AlwaysOn SQL service.

   **Note:** Changes in the `hive-site.xml` configuration file only require a restart of AlwaysOn SQL service, not DSE.

   ```
   $ dse client-tool alwayson-sql start
   ```

4. Test the connection with Beeline.

   ```
   $ dse beeline
   beeline> !connect jdbc:hive2://hostname:10000/
   default;ssl=true;sslTrustStore=path to truststore/
   truststore.jks;trustStorePassword=truststore password
   ```

   **Note:** The JDBC URL for the Simba JDBC Driver is:

   ```
   jdbc:spark://hostname:10000/default;SSL=1;SSLTrustStore=path to truststore/truststore.jks;SSLTrustStorePwd=truststore password
   ```

**Using authentication with AlwaysOn SQL**

AlwaysOn SQL can be configured to use DSE authentication.

When **DSE authentication is enabled**, modify the `hive-site.xml` configuration file to enable JDBC authentication. DSE supports configurations for password authentication and Kerberos authentication. The `hive-site.xml` file has sections with preconfigured settings to use no authentication (the default), password authentication, or Kerberos authentication. Uncomment the preferred authentication mechanism, then restart AlwaysOn SQL.

   **Note:** DSE supports multiple authentication mechanisms, but AlwaysOn SQL only supports one mechanism per datacenter.

AlwaysOn SQL supports **DSE proxy authentication**. The user who executes the queries is the user who authenticated using JDBC. If AlwaysOn SQL was started by user Amy, and then Bob begins a JDBC session, the queries are executed by Amy on behalf of Bob. Amy must have permissions to execute these queries on behalf of Bob.
To enable authentication in AlwaysOn SQL `alwayson_sql_options (page 135)`, follow these steps.

1. Create the `auth_user (page 135)` role specified in AlwaysOn SQL options (page 134) and grant the following permissions to the role.

```sql
CREATE ROLE alwayson_sql WITH LOGIN=true; // role name matches auth_user (page 135)

// Required if scheme_permissions (page 105) true
GRANT EXECUTE ON ALL AUTHENTICATION SCHEMES TO alwayson_sql;

// Spark RPC settings
GRANT ALL PERMISSIONS ON REMOTE OBJECT DseResourceManager TO alwayson_sql;
GRANT ALL PERMISSIONS ON REMOTE OBJECT DseClientTool TO alwayson_sql;
GRANT ALL PERMISSIONS ON REMOTE OBJECT AlwaysOnSqlRoutingRPC to alwayson_sql;

// Spark and DSE required table access
GRANT SELECT ON system.size_estimates TO alwayson_sql;
GRANT SELECT, MODIFY ON "HiveMetaStore".sparkmetastore TO alwayson_sql;
GRANT SELECT, MODIFY ON dse_analytics.alwayson_sql_cache_table TO alwayson_sql;
GRANT SELECT, MODIFY ON dse_analytics.alwayson_sql_info TO alwayson_sql;

// Permissions to create and change applications
GRANT CREATE, DESCRIBE ON ANY WORKPOOL TO alwayson_sql;
GRANT MODIFY, DESCRIBE ON ANY SUBMISSION TO alwayson_sql;
```

See Setting up DSE Spark application permissions for more details.

2. Create the user role.

For internal authentication:

```sql
CREATE ROLE 'user_name'
WITH LOGIN = true;
```

If you use Kerberos, set up a role that matches the full Kerberos principal name for each user.

```sql
CREATE ROLE 'user_name@example.com@EXAMPLE.COM'
WITH LOGIN = true;
```

3. Grant permissions to access keyspaces and tables to the user role.

```sql

dse_analytics.alwayson_sql_cache_table TO alwayson_sql;
GRANT SELECT, MODIFY ON dse_analytics.alwayson_sql_info TO alwayson_sql;

// Permissions to create and change applications
GRANT CREATE, DESCRIBE ON ANY WORKPOOL TO alwayson_sql;
GRANT MODIFY, DESCRIBE ON ANY SUBMISSION TO alwayson_sql;
```

See Setting up DSE Spark application permissions for more details.

2. Create the user role.

For internal authentication:

```sql
CREATE ROLE 'user_name'
WITH LOGIN = true;
```

If you use Kerberos, set up a role that matches the full Kerberos principal name for each user.

```sql
CREATE ROLE 'user_name@example.com@EXAMPLE.COM'
WITH LOGIN = true;
```

3. Grant permissions to access keyspaces and tables to the user role.
For internal roles:

```sql
GRANT SELECT ON KEYSPACE keyspace_name
TO 'user_name';
```

For Kerberos roles:

```sql
GRANT SELECT ON KEYSPACE keyspace_name
TO 'user_name@example.com@EXAMPLE.COM';
```

4. Allow the AlwaysOn SQL role (auth_user) to execute commands with the user role.

For internal roles:

```sql
GRANT PROXY.EXECUTE
ON ROLE 'user_name'
TO alwayson_sql;
```

For Kerberos roles:

```sql
GRANT PROXY.EXECUTE
ON ROLE 'user_name@example.com@EXAMPLE.COM'
TO alwayson_sql;
```

5. Open the `hive-site.xml` configuration file in an editor.

6. Uncomment and modify the authentication mechanism used in `hive-site.xml`.

   - If password authentication is used, enable password authentication in DSE.
   - If Kerberos authentication is to be used, Kerberos does not need to be enabled in DSE. AlwaysOn SQL must have its own service principal and keytab.
   - The user must have login permissions in DSE in order to login through JDBC to AlwaysOn SQL.

This example shows how to enable Kerberos authentication. Modify the Kerberos domain and path to the keytab file.

```xml
<!-- Start of: configuration for authenticating JDBC users with Kerberos -->
<property>
    <name>hive.server2.enable.doAs</name>
    <value>true</value>
</property>

<property>
    <name>hive.server2.authentication</name>
    <value>KERBEROS</value>
</property>
</config>
```
Using DataStax Enterprise advanced functionality

```
<property>
   <name>hive.server2.authentication.kerberos.principal</name>
   <value>hiveserver2/_HOST@KERBEROS_DOMAIN</value>
</property>

<property>
   <name>hive.server2.authentication.kerberos.keytab</name>
   <value>path to hiveserver2.keytab</value>
</property>

<!-- End of: configuration for authenticating JDBC users with Kerberos -->
```

7. Restart AlwaysOn SQL.

```
$ dse client-tool alwayson-sql restart
```

**Simba JDBC Driver for Apache Spark**

The Simba JDBC Driver for Spark provides a standard JDBC interface to the information stored in DataStax Enterprise with AlwaysOn SQL running.

Your DSE license includes a license to use the Simba drivers.

**Prerequisites:**

You must have a running **DSE Analytics cluster with Spark enabled** *(page 1437)*, and have **enabled AlwaysOn SQL** *(page 257)*.

1. Download the Simba JDBC Driver for Apache Spark from the **DataStax Drivers Download page**.

2. Expand the ZIP file containing the driver.

3. In your JDBC application, configure the following details:

   a. Add **SparkJDBC41.jar** and the rest of the JAR files included in the ZIP file in your classpath.

   b. The JDBC **driver class** is **com.simba.spark.jdbc41.Driver** and the JDBC **data source** is **com.simba.spark.jdbc41.DataSource**.

   c. Set the connection **URL** to **jdbc:spark://<hostname>:<port>** where 
      **<hostname>** is the hostname of the node on which AlwaysOn SQL is running, 
      and **<port>** is the port number on which AlwaysOn SQL is listening.

      ```
jdbc:spark://node1.example.com:10000
```
4. For more details, see the Simba JDBC configuration documentation.

**Simba ODBC Driver for Apache Spark**

The Simba ODBC Driver for Spark provides users access to DataStax Enterprise (DSE) clusters with a running AlwaysOn SQL. The driver is compliant with the latest ODBC 3.52 specification and automatically translates any SQL-92 query into Spark SQL.

Your DSE license includes a license to use the Simba drivers.

To use the Simba ODBC Driver for Spark you must have:

- A running DSE Analytics cluster with Spark enabled (page 1437), and enabled AlwaysOn SQL (page 257).

**Linux**

1. Download the Simba ODBC Driver for Apache Spark (Linux 32- or 64-bit) from the DataStax Drivers Download page.

2. Expand the downloaded file:

   ```bash
   $ mkdir simba-odbc &&
   cd simba-odbc &&
   tar xvf version.tar.gz
   ```

3. For more details, see the Simba ODBC configuration documentation.

**Windows**

1. Download the Simba ODBC Driver for Apache Spark (Windows 32- or 64-bit) from the DataStax Drivers Download page.

2. Double-click the downloaded installer and follow the installation wizard.

3. For more details, see the Simba ODBC configuration documentation.

**Connecting to AlwaysOn SQL server using Beeline**

You can use Shark Beeline to test AlwaysOn SQL (page 257).

1. Start AlwaysOn SQL (page 258).

2. Start the Beeline shell.

   ```bash
   $ dse beeline
   ```

3. Connect to the server using the JDBC URI for your server.
Using DataStax Enterprise advanced functionality

4. Connect to a keyspace and run a query from the Beehive shell.

```
0: jdbc:hive2://localhost:10000> use test;
0: jdbc:hive2://localhost:10000> select * from test;
```

### Accessing DataStax Enterprise data from external Spark clusters

DataStax Enterprise works with external Spark clusters in a bring-your-own-Spark (BYOS) model.

#### Overview of BYOS support in DataStax Enterprise

BYOS support in DataStax Enterprise consists of a JAR file and a generated configuration file that provides all the necessary classes and configuration settings for connecting to a particular DataStax Enterprise cluster from an external Spark cluster. To specify a different classpath to accommodate applications originally written for open source Apache Spark, specify the `-framework` option with dse spark commands.

All DSE resources, including DSEFS file locations, can be accessed from the external Spark cluster.

BYOS is tested against the version of Spark integrated into DSE (described in the DataStax Enterprise 6.7 release notes (page 23)) and the following Spark distributions:

- Hortonworks Data Platform (HDP) 2.5
- Cloudera CDH 5.10

#### Generating the BYOS configuration file

The `byos.properties` file is used to connect to a DataStax Enterprise cluster from a Spark cluster. The configuration file contains connection information about the DataStax Enterprise cluster. This file must be generated on a node in the DataStax Enterprise cluster. You can specify an arbitrary name for the generated configuration file. The `byos.properties` name is used throughout the documentation to refer to this configuration file.

1. Connect to a node in your DataStax Enterprise cluster.

2. Generate the `byos.properties` file using the `dse client-tool` command.

```
$ dse client-tool configuration byos-export ~/byos.properties
```

This will generate the `byos.properties` file in your home directory. See `dse client-tool` (page 1225) for more information on the options for `dse client-tool`.

What's next:
The `byos.properties` file can be copied to a node in the external Spark cluster and used with the Spark shell, as described in Connecting to DataStax Enterprise using the Spark shell on an external Spark cluster (page 267).

**Connecting to DataStax Enterprise using the Spark shell on an external Spark cluster**

Use the generated `byos.properties` configuration file (page 266) and the `byos-version.jar` from a DataStax Enterprise node to connect to the DataStax Enterprise cluster from the Spark shell on an external Spark cluster.

**Prerequisites:**

You must generate the `byos.properties` on a node in your DataStax Enterprise cluster.

1. Copy the `byos.properties` file you previously generated from the DataStax Enterprise node to the local Spark node.

   ```bash
   $ scp user@dsenode1.example.com:~/byos.properties .
   ```

   If you are using Kerberos authentication, specify the `--generate-token` and `--token-renewer <username>` options when generating `byos.properties`, as described in dse client-tool configuration byos-export (page 1231).

2. Copy the `byos-version.jar` file from the clients directory from a node in your DataStax Enterprise cluster to the local Spark node.

   The `byos-version.jar` file location depends on the type of installation.

   ```bash
   $ scp user@dsenode1.example.com:/usr/share/dse/clients/dse-byos_2.11-6.0.2.jar byos-6.0.jar
   ```

3. Merge external Spark properties into `byos.properties`.

   ```bash
   $ cat ${SPARK_HOME}/conf/spark-defaults.conf >> byos.properties
   ```

4. If you are using Kerberos authentication, set up a CRON job or other task scheduler to periodically call `dse client-tool cassandra renew-token <token>` where `<token>` is the encoded token string in `byos.properties`.

5. Start the Spark shell using the `byos.properties` and `byos-version.jar` file.
Using DataStax Enterprise advanced functionality

Generating Spark SQL schema files

Spark SQL can import schema files generated by DataStax Enterprise.

1. Export the schema file using `dse client-tool`.

   ```
   $ dse client-tool --use-server-config spark sql-schema --all > output.sql
   ```

2. Copy the schema to an external Spark node.

   ```
   $ scp output.sql user@sparknode1.example.com:
   ```

3. On a Spark node, import the schema using Spark.

   ```
   $ spark-sql --jars byos-5.1.jar --properties-file byos.properties -f output.sql
   ```

Starting Spark SQL Thrift Server with Kerberos

Spark SQL Thrift Server is a long running service and must be configured to start with a keytab file if Kerberos is enabled. The user principal must be added to DSE, and Spark SQL Thrift Server restarted with the generated BYOS configuration file and `byos-version.jar`.

Prerequisites:

These instructions are for the Spark SQL Thrift Server included in HortonWorks 2.4. The Hadoop Spark SQL Thrift Server principal is `hive/_HOST@REALM`.

1. Create the principal on the DSE node using `cqlsh`.

   ```
   create user hive/spark_sql_thrift_server_host@REALM;
   ```

2. Login as the `hive` user on the Spark SQL Thrift Server host.

3. Create a `~/.java.login.config` file with a JAAS Kerberos configuration.

4. Merge the existing Spark SQL Thrift Server configuration properties with the generated BYOS configuration file into a new file.

   ```
   $ cat /usr/hdp/current/spark-thriftserver/conf/spark-thriftserver.conf byos.properties > custom-sparkconf.conf
   ```
5. Start Spark SQL Thrift Server with the custom configuration file and `byos-version.jar`.

   ```bash
   $ /usr/hdp/2.4.2.0-258/spark/sbin/start-thriftserver.sh --jars byos-version.jar --properties-file custom-sparkconf.conf
   ```


   ```bash
   $ beeline -u 'jdbc:hive2://hostname:port/default;principal=hive/_HOST@REALM'
   ```

**What's next:**

Generated SQL schema *(page 268)* files can be passed to `beeline` with the `-f` option to generate a mapping for DSE tables so both Hadoop and DataStax Enterprise tables will be available through the service for queries.

**Using the Spark Jobserver**

DataStax Enterprise includes a bundled copy of the open-source Spark Jobserver, an optional component for submitting and managing Spark jobs, Spark contexts, and JARs on DSE Analytics clusters. Refer to the [Components (page 23)](index.html) in the release notes to find the version of the Spark Jobserver included in this version of DSE.

Valid `spark-submit` options *(page 235)* are supported and can be applied to the Spark Jobserver. To use the Jobserver:

- **Start the job server:**
  
  ```bash
  $ dse spark-jobserver start [any_spark_submit_options]
  ```

- **Stop the job server:**

  ```bash
  $ dse spark-jobserver stop
  ```

The default location of the Spark Jobserver depends on the type of installation:

- **Package installations**: `/usr/share/dse/spark/spark-jobserver`
- **Tarball installations**: `installation_location/resources/spark/spark-jobserver`

All the uploaded JARs, temporary files, and log files are created in the user's `$HOME/.spark-jobserver` directory, first created when starting Spark Jobserver.

Beneficial use cases for the Spark Jobserver include sharing cached data, repeated queries of cached data, and faster job starts.

**Note:**
Running multiple SparkContext instances in a single JVM is not recommended. Therefore it is not recommended to create a new SparkContext for each submitted job in a single Spark Jobserver instance. We recommend one of the two following Spark Jobserver usages.

- **Persistent Context Mode**: a single pre-created SparkContext shared by all jobs.
- **Context per JVM**: each job has its own SparkContext in a separate JVM.

By default, the H2 database is used for storing Spark Jobserver related metadata. In this setup, using Context per JVM requires additional configuration. See the Spark Jobserver docs for details.

**Note**: In Context per JVM mode, job results must not contain instances of classes that are not present in the Spark Jobserver classpath. Problems with returning unknown (to server) types can be recognized by following log line:

```
Association with remote system [akka.tcp://JobServer@127.0.0.1:45153]
has failed, address is now gated for [5000] ms.
Reason: [unknown type name is placed here]
```

Please consult Spark Jobserver docs to see configuration details.

For an example of how to create and submit an application through the Spark Jobserver, see the spark-jobserver demo included with DSE.

The default location of the demos directory depends on the type of installation:

- **Package installations**: /usr/share/dse/demos
- **Tarball installations**: installation_location/demos

**Enabling SSL communication with Jobserver**

To enable SSL encryption when connecting to Jobserver, you must have a server certificate, and a truststore containing the certificate. Add the following configuration section to the dse.conf file in the Spark Jobserver directory.

```
spray.can.server {
    ssl-encryption = on
    keystore = "path to keystore"
    keystorePW = "keystore password"
}
```

The default location of the Spark Jobserver depends on the type of installation:

- **Package installations**: /usr/share/dse/spark/spark-jobserver
- **Tarball installations**: installation_location/resources/spark/spark-jobserver
Restart the Jobserver after saving the configuration changes.

**Spark examples**

DataStax Enterprise includes Spark example applications that demonstrate different Spark features.

**Portfolio Manager demo using Spark**

The Portfolio Manager demo runs an application that is based on a financial use case. You run scripts that create a portfolio of stocks. On the OLTP (online transaction processing) side, each portfolio contains a list of stocks, the number of shares purchased, and the purchase price. The demo's pricer utility simulates real-time stock data. Each portfolio gets updated based on its overall value and the percentage of gain or loss compared to the purchase price. The utility also generates 100 days of historical market data (the end-of-day price) for each stock. On the DSE OLAP (online analytical processing) side, a Spark job calculates the greatest historical 10 day loss period for each portfolio, which is an indicator of the risk associated with a portfolio. This information is then fed back into the real-time application to allow customers to better gauge their potential losses.

To run the demo:

**Note:** DataStax Demos do not work with LDAP or internal authorization (username/password) enabled.

1. **Install a node**
   
   **Note:** If using a tarball installation, the Portfolio Manager demo is installed as part of the normal installation. If using a package install, you must include the command for installing the demos.
   
   - Default Interface: localhost (127.0.0.1) You must use this IP for the demo.

2. **Start DataStax Enterprise as DSE Analytics node:**
   
   - For package *(page 1438)* installations:
     
     a. In /etc/default/dse, set:

     ```
     SPARK_ENABLED=1
     ```

     b. Start the node:

     ```
     $ sudo service dse start
     ```

   - For tarball *(page 1441)* installations:
Using DataStax Enterprise advanced functionality

$ installation_location/bin/dse cassandra -k ## Starts node in analytics mode

3. Go to the Portfolio Manager demo directory.

   The default location of the Portfolio Manager demo depends on the type of installation:
   - Package installations: /usr/share/dse/demos/portfolio_manager
   - Tarball installations: installation_location/demos/portfolio_manager

4. Run the bin/pricer utility to generate stock data for the application:
   - To see all of the available options for this utility:
     $ bin/pricer --help
   - Start the pricer utility:
     $ bin/pricer -o INSERT_PRICES &&
     bin/pricer -o UPDATE_PORTFOLIOS &&
     bin/pricer -o INSERT_HISTORICAL_PRICES -n 100

   The pricer utility takes several minutes to run.

5. Start the web service:

   cd website &&
   sudo ./start


   The real-time Portfolio Manager demo application is displayed.
7. Open another terminal.

8. Run the Spark SQL job in the `10-day-loss.q` file.

```
$ dse spark-sql -f 10-day-loss.q
```

9. Run the equivalent Spark Scala job in the `10-day-loss.sh` script.

```bash
$ ./10-day-loss.sh
```

10. Run the equivalent Spark Java job in the `10-day-loss-java.sh` script.

```bash
$ ./10-day-loss-java.sh
```

11. After the job completes, refresh the Portfolio Manager web page.

The results of the Largest Historical 10 day Loss for each portfolio are displayed.
Using DataStax Enterprise advanced functionality

What's next:

The Scala and Java source code for the demo are in the `src` directory.

Running the Weather Sensor demo

Using the Weather Sensor demo, you can compare how long it takes to run Spark SQL queries against aggregated data for a number of weather sensors in various cities. For example, you can view reports using different metrics, such as temperature or humidity, and get a daily roll up.
You run customize Spark SQL queries using different metrics and different dates. In addition to querying CQL tables, you time Spark SQL queries against data in DataStax Enterprise File System (DSEFS).

**Note:** DataStax Demos do not work with LDAP or internal authorization (username/password) enabled.

**Prerequisites**

Before running the demo, install the following source code and tools if you do not already have them:

- **Python 2.7:**
  - # Debian and Ubuntu
    
    ```bash
    $ sudo apt-get install python2.7-dev
    ```
  - # RedHat or CentOS
    
    ```bash
    $ sudo yum install python27
    ```
  - # Mac OS X already has Python 2.7 installed.

- **pip installer tool:**
  - # Debian and Ubuntu
    
    ```bash
    $ sudo apt-get install python-pip
    ```
  - # RedHat or CentOS
    
    ```bash
    $ sudo yum install python-pip
    ```
  - # Mac OS X
    
    ```bash
    $ sudo easy_install pip
    ```

- **The libsasl2-dev package:**
  - # Debian and Ubuntu
    
    ```bash
    $ sudo apt-get install libsasl2-dev
    ```
  - # RedHat or CentOS
    
    ```bash
    $ sudo yum install cyrus-sasl-lib
    ```

- **The required Python packages:**
  - # All platforms
Using DataStax Enterprise advanced functionality

sudo pip install pyhs2 six flask cassandra-driver

If you installed DataStax Enterprise using a tarball, set the `PATH` environment variable to the DataStax Enterprise installation `/bin` directory.

```bash
export PATH=$PATH:$installation_location/bin
```

Start DataStax Enterprise and import data

You start DataStax Enterprise in Spark mode, and then run a script that creates the schema for weather sensor data model. The script also imports aggregated data from CSV files into DSE tables. The script uses the `hadoop fs` command to put the CSV files into the DSEFS.

1. Start DataStax Enterprise in **Spark mode (page 185)**.

2. Run the create-and-load CQL script in the `demos/weather_sensors/resources` directory. On Linux, for example:

   ```bash
   cd  $installation_location/demos/weather_sensors && bin/create-and-load
   ```

   The default location of the `demos` directory depends on the type of installation:
   - **Package installations**: `/usr/share/dse/demos`
   - **Tarball installations**: `$installation_location/demos`

   The output confirms that the script imported the data into CQL and copied files to DSEFS.

   If an error occurs, set the `PATH` as described in **Prerequisites (page 275)**, and retry.

Starting the Spark SQL Thrift server

You start the Spark SQL Thrift server on a specific port to avoid conflicts. Start using your local user account. Do not use `sudo`.

1. Start the Spark SQL Thrift server on port 5588. On Linux, for example:

   ```bash
   cd  $installation_location &&
   dse spark-sql-thriftserver start --hiveconf
   hive.server2.thrift.port=5588
   ```

Start the web app and query the data

1. Open another terminal and start the Python service that controls the web interface:

   ```bash
   cd  $installation_location/demos/weather_sensors &&
   ```
2. Open a browser and go to the following URL: http://localhost:8983/

The weather sensors app appears. Select **Near Real-Time Reports** on the horizontal menu. A drop-down listing weather stations appears:

3. Select a weather station from the drop-down, view the graph, and select different metrics from the vertical menu on the left side of the page.

4. On the horizontal menu, click **Sample Live Queries**, then select a sample script. Click the **Spark SQL** button, then click Submit.

   The time spent loading results using Spark appears.
Using DataStax Enterprise advanced functionality

**Note:** If you are running the demo on a SearchAnalytics datacenter, port 8983 conflicts with the Search web UI. Change the port in the `demos/weather_sensors/web/weather.py` to a free port.

```python
app.run(host='0.0.0.0', port=8984, threaded=True, debug=True)
```

5. From the horizontal menu, click Custom Live Queries. Click a Week Day, and then a metric, such as Wind Direction. Click Recalculate Query. The query reflects the selections you made.

6. From the horizontal menu, click DSEFS Live Queries. Click Submit query. The time spent loading results from DSEFS using Spark SQL appears.
Clean up

To remove all generated data, run the following commands:

```
cd installation_location/demos/weather_sensors && bin/cleanup
```

To remove the keyspace from the cluster, run the following command:

```
$ cqlsh -e "DROP KEYSPACE weathercql;"
```

Running the Wikipedia demo with SearchAnalytics

The following instructions describe how to use search queries in the Spark console on SearchAnalytics nodes using the Wikipedia demo.

Prerequisites:
You must have created a new SearchAnalytics datacenter as described in the single datacenter deployment scenario.

1. Start the node or nodes in SearchAnalytics mode.
   - Package installations: See Starting DataStax Enterprise as a service (page 1437).
   - Package installations: See Starting DataStax Enterprise as a stand-alone process (page 1440).

2. Ensure that the cluster is running correctly by running dsetool ring. The node type should be SearchAnalytics.
   - Package installations: dsetool ring
   - Tarball installations: installation_location/bin/dsetool ring

3. In a terminal, go to the Wikipedia demo directory.
   - The default wikipedia demo location depends on the type of installation:
     - Package installations: /usr/share/dse/demos/wikipedia
     - Tarball installations: installation_location/demos/wikipedia

   $ cd /usr/share/dse/demos/wikipedia

4. Add the schema by running the 1-add-schema.sh script.

   $ ./1-add-schema.sh

5. Create the search indexes.

   $ ./2-index.sh

6. Start the Spark console.

   $ dse spark

7. Create an RDD based on the wiki.solr table.

   scala> val table = sc.cassandraTable("wiki","solr")

     table: com.datastax.spark.connector.rdd.CassandraTableScanRDD[com.datastax.spark.connector.CassandraRow] = CassandraTableScanRDD[0] at RDD at CassandraRDD.scala:15

8. Run a query using the title Solr index and collect the results.
Using DataStax Enterprise advanced functionality

```scala>
val result = 
  table.select("id","title").where("solr_query='title:Boroph*'").collect
```

Equivalent JSON query:

```json
where("solr_query='{"q": "title:Boroph*"}'")
```

```java
result:
  Array[com.datastax.spark.connector.CassandraRow] = Array(
    CassandraRow{id: 23729958, title: Borophagus parvus},
    CassandraRow{id: 23730195, title: Borophagus dudleyi},
    CassandraRow{id: 23730528, title: Borophagus hilli},
    CassandraRow{id: 23730810, title: Borophagus diversidens},
    CassandraRow{id: 23730974, title: Borophagus littoralis},
    CassandraRow{id: 23731282, title: Borophagus orc},
    CassandraRow{id: 23731616, title: Borophagus pugnator},
    CassandraRow{id: 23732450, title: Borophagus secundus})
```

**What's next:**

For details on using search query syntax in CQL, see [CQL queries](https://www.datastax.com/docs/enterprise/security/query-language).

**Running the Spark MLLib demo application**

The Spark MLLib demo application demonstrates how to run machine-learning analytic jobs using Spark and DataStax Enterprise. The demo solves the classic iris flower classification problem, using the iris flower data set. The application will use the iris flower data set to build a Naive Bayes classifier that will recognize a flower based on four feature measurements.

**Prerequisites:**

We strongly recommend that you install the BLAS library on your machines before running Spark MLLib jobs. For instructions on installing the BLAS library on your platform, see [https://github.com/fommil/netlib-java/blob/master/README.md#machine-optimised-system-libraries](https://github.com/fommil/netlib-java/blob/master/README.md#machine-optimised-system-libraries).

The BLAS library is not distributed with DSE due to licensing restrictions, but improves MLLib performance significantly.

You must have the Gradle build tool installed to build the demo. See [https://gradle.org/](https://gradle.org/) for details on installing Gradle on your OS.

1. Start the nodes in Analytics mode.
   - Package installations: See [Starting DataStax Enterprise as a service](https://www.datastax.com/docs/enterprise/security/query-language) (page 1437).
Using DataStax Enterprise advanced functionality

- Tarball installations: See Starting DataStax Enterprise as a stand-alone process (page 1440).

2. In a terminal, go to the spark-mlib directory located in the Spark demo directory.
   
   The default location of the Spark demo depends on the type of installation:
   
   - Package installations: /usr/share/dse/demos/portfolio_manager
   - Tarball installations: installation_location/demos/portfolio_manager

3. Build the application using the gradle build tool.

   $ gradle

4. Use spark-submit to submit the application JAR.

   The Spark MLlib demo application reads the Spark demo directory/spark-mllib/iris.csv file on each node. This file must be accessible in the same location on each node. If some nodes do not have the same local file path, set up a shared network location accessible to all the nodes in the cluster.

   To run the application where each node has access to the same local location of iris.csv.

   $ dse spark-submit NaiveBayesDemo.jar

   To specify a shared location of iris.csv:

   $ dse spark-submit NaiveBayesDemo.jar /mnt/shared/iris.csv

   **Running the http_receiver demo**

   The http_receiver demo uses Spark Streaming to save data to DSE. It is located in the http-receivers directory in the demos directory.

   The default location of the demos directory depends on the type of installation:

   - Package installations: /usr/share/dse/demos
   - Tarball installations: installation_location/demos

   See the README.txt file for instructions on running http_receivers.

   **Importing a text file into a table**

   This example shows how to use Spark to import a local or DSEFS based text file into an existing table. You use the saveToCassandra method present in the Spark RDDs to save an arbitrary RDD to the database.
1. Create a keyspace and a table in the database. For example, use cqlsh.

```cql
CREATE KEYSPACE int ks WITH replication =
   {'class': 'NetworkTopologyStrategy', 'Analytics':1};
USE int ks;
CREATE TABLE int_compound ( pkey int, ckey1 int, data1 int ,
   PRIMARY KEY (pkey,ckey1));
```

2. Insert data into the table

```cql
INSERT INTO int_compound ( pkey, ckey1, data1 ) VALUES ( 1, 2, 3 );
INSERT INTO int_compound ( pkey, ckey1, data1 ) VALUES ( 2, 3, 4 );
INSERT INTO int_compound ( pkey, ckey1, data1 ) VALUES ( 3, 4, 5 );
INSERT INTO int_compound ( pkey, ckey1, data1 ) VALUES ( 4, 5, 1 );
INSERT INTO int_compound ( pkey, ckey1, data1 ) VALUES ( 5, 1, 2 );
```

3. Create a text file named `normalfill.csv` that contains this data.

```csv
6,7,8
7,8,6
8,6,7
```

4. Put the CSV file into DSEFS. For example:

```
$ dse fs "put file://mypath/normalfill.csv dsefs:"
```

5. Start the Spark shell.

6. Verify that Spark can access the `int ks` keyspace:

```
:showSchema int ks
```

```
========================================
Keyspace: int ks
Table: int_compound
----------------------------------------
- pkey : Int (partition key column)
- ckey1 : Int (clustering column)
- data1 : Int
```

`int ks` appears in the list of keyspaces.

7. Read in the file, splitting it on the comma delimiter. Transform each element into an Integer.
Using DataStax Enterprise advanced functionality

```scala
val normalfill = sc.textFile("/normalfill.csv").map(line =>
  line.split(",").map(_.toInt));

map at console:22

Alternatively, read in the file from the local file system.

val file = sc.textFile("file:///local-path/normalfill.csv")

at console:22

8. Check that Spark can find and read the CSV file.

normalfill.take(1);

res2: Array[Array[Int]] = Array(Array(6, 7, 8))

9. Save the new data to the database.

normalfill.map(line => (line(0), line(1),
  line(2))).saveToCassandra(  
  "int_ks", "int_compound", Seq("pkey", "ckey1", "data1"))

The step produces no output.

10. Check that the data was saved using cqlsh.

SELECT * FROM int_ks.int_compound;

<table>
<thead>
<tr>
<th>pkey</th>
<th>ckey1</th>
<th>data1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

(8 rows)

Running spark-submit job with internal authentication

This example shows how to run a spark-submit job with internal authentication.
When you use `dse spark-submit` to submit a Spark job, the Spark Master URL and the Spark database connection URL are set automatically. Use the Spark session builder API to set the application name. For example:

```java
SparkSession spark = SparkSession
    .builder()
    .appName("Datastax Java example")
    .getOrCreate();
```

1. Clone the example source files from github.

   ```bash
   $ git clone https://github.com/datastax/SparkBuildExamples.git
   ```

2. Select you preferred language and build system. For example for Java and Maven:

   ```bash
   $ cd SparkBuildExamples/java/maven/dse
   ```

3. Build the package with Maven:

   ```bash
   $ mvn package
   ```

4. Create your authentication credentials.

   Authentication credentials can be provided in several ways, see Providing credentials from DSE tools.

5. Use `spark-submit` to run the application. The following example assumes you've set your authentication credentials in an environment variable or config file.

   ```bash
   $ dse spark-submit --class com.datastax.spark.example.WriteRead ./target/writeRead-0.1.jar
   ```

---

**DSEFS (DataStax Enterprise file system)**

DSEFS is the default distributed file system on DSE Analytics nodes.

**About DSEFS**

DSEFS (DataStax Enterprise file system) is a fault-tolerant, general-purpose, distributed file system within DataStax Enterprise. It is designed for use cases that need to leverage a distributed file system for data ingestion, data staging, and state management for Spark Streaming applications (such as checkpointing or write-ahead logging). DSEFS is similar to HDFS, but avoids the deployment complexity and single point of failure typical of HDFS. DSEFS is HDFS-compatible and is designed to work in place of HDFS in Spark and other systems.
DSEFS is the default distributed file system in DataStax Enterprise, and is automatically enabled on all analytics nodes.

DSEFS stores file metadata (such as filepath, ownership, permissions) and file contents separately:

- Metadata is stored in the database.
- File data blocks are stored locally on each node and are replicated onto multiples nodes.

The redundancy factor is set at the DSEFS directory or file level, which is more granular than the replication factor that is set at the keyspace level in the database.

For performance on production clusters, store the DSEFS data on physical devices that are separate from the database. For development and testing you may store DSEFS data on the same physical device as the database.

**Deployment overview**

- The DSEFS server runs in the same JVM as DataStax Enterprise. Similar to the database, there is no master node. All nodes running DSEFS are equal.
- A single DSEFS cannot span multiple datacenters. To deploy DSEFS in multiple datacenters, you can create a separate instance of DSEFS for each datacenter.
- You can use different keyspaces to configure multiple DSEFS file systems (page 291) in a single datacenter.
- For optimal performance, locate the local DSEFS data on a different physical drive than the database.
- Encryption is not supported. Use operating system access controls to protect the local DSEFS data directories. Other limitations (page 290) apply.
- DSEFS uses the LOCAL_QUORUM consistency level to store file metadata. DSEFS will always try to write each data block to replicated node locations, and even if a write fails, it will retry to another node before acknowledging the write. DSEFS writes are very similar to the ALL consistency level, but with additional failover to provide high-availability. DSEFS reads are similar to the ONE consistency level.

**Enabling DSEFS**

DSEFS is automatically enabled on analytics nodes, and disabled on non-analytics nodes. You can enable the DSEFS service on any node in a DataStax Enterprise cluster. Nodes within the same datacenter with DSEFS enabled will join together to behave as a DSEFS cluster.

On each node:

1. In the dse.yaml file, uncomment and set the properties for the DSEFS options:

```yaml
# dsefs_options:
#   enabled:
#   keyspace_name: dsefs
#   work_dir: /var/lib/dsefs
```
a. To enable DSEFS:

```bash
enabled: true
```

If enabled is blank or commented out, DSEFS starts only if the node is configured to run analytics workloads.

b. Define the keyspace for storing the DSEFS metadata:

```bash
keyspace_name: dsefs
```

You can optionally configure multiple DSEFS file systems (page 291) in a single datacenter.

c. Define the work directory for storing the DSEFS metadata for the local node. The work directory should not be shared with other DSEFS nodes:

```bash
work_dir: /var/lib/dsefs
```

d. Define the public port on which DSEFS listens for clients:

```bash
public_port: 5598
```

**Note:** DataStax recommends that all nodes in the cluster have the same value. Firewalls must open this port to trusted clients. The service on this port is bound to the `native_transport_address (page 87)`.

e. Define the private port for DSEFS inter-node communication:

```bash
private_port: 5599
```

**Caution:** Do not open this port to firewalls; this private port must be not visible from outside of the cluster.

f. Set the data directories where the file data blocks are stored locally on each node.

```bash
data_directories:
  - dir: /var/lib/dsefs/data
```

If you use the default `/var/lib/dsefs/data` data directory, verify that the directory exists and that you have root access. Otherwise, you can define your own directory location, change the ownership of the directory, or both:

```
$ sudo mkdir -p /var/lib/dsefs/data; sudo chown -R $USER:$GROUP /var/lib/dsefs/data
```

Ensure that the data directory is writeable by the DataStax Enterprise user. Put the data directories on different physical devices than the database. Using multiple data directories on JBOD improves performance and capacity.

g. For each data directory, set the weighting factor to specify how much data to place in this directory, relative to other directories in the cluster. This soft constraint determines how DSEFS distributes the data. For example, a directory with a value of 3.0 receives about three times more data than a directory with a value of 1.0.

```
data_directories:
  - dir: /var/lib/dsefs/data
    storage_weight: 1.0
```

h. For each data directory, define the reserved space, in bytes, to not use for storing file data blocks. See `min_free_space` (page 137).

```
data_directories:
  - dir: /var/lib/dsefs/data
    storage_weight: 1.0
    min_free_space: 5368709120
```

2. Restart the node.

3. Repeat steps for the remaining nodes.

4. With guidance from DataStax Support, you can tune advanced DSEFS properties:

```
# service_startup_timeout_ms: 30000
# service_close_timeout_ms: 600000
# server_close_timeout_ms: 2147483647 # Integer.MAX_VALUE
# compression_frame_max_size: 1048576
# query_cache_size: 2048
# query_cache_expire_after_ms: 2000
# gossip_options:
#   round_delay_ms: 2000
#   startup_delay_ms: 5000
#   shutdown_delay_ms: 10000
# rest_options:
#   request_timeout_ms: 330000
#   connection_open_timeout_ms: 55000
#   client_close_timeout_ms: 60000
```
Using DataStax Enterprise advanced functionality

5. Continue with using DSEFS (page 290).

Disabling DSEFS

To disable DSEFS and remove metadata and data:

1. Remove all directories and files from the DSEFS file system:

   $ dse fs rm -r filepath

2. Wait a while for all nodes to perform the delete operations.

3. Verify that all DSEFS data directories where the file data blocks are stored locally on each node are empty.

   These data directories are configured in dse.yaml. Your directories are probably different from this default data_directories value:

   ```yaml
   data_directories:
       - dir: /var/lib/dsefs/data
   ```

4. Disable the DSEFS entries in all dse.yaml files on all nodes.

5. Restart DataStax Enterprise.

6. Truncate all of the tables in the dsefs keyspace.

   Do not remove the dsefs keyspace. If you inadvertently removed the dsefs keyspace, you must specify a different keyspace name in dse.yaml or create an
Using DataStax Enterprise advanced functionality

empty dsefs keyspace (this empty dsefs keyspace will be populated with tables during DSEFS start up).

Using DSEFS

You must configure data replication. You can optionally configure (page 286) multiple DSEFS file systems in a datacenter, and perform other functions, including setting the Kafka log retention.

DSEFS does not span datacenters. Create a separate DSEFS instance in each datacenter, as described in the steps below.

DSEFS limitations

Know these limitations when you configure and tune DSEFS. The following functionality and features are not supported:

• Encryption.
  Use operating system access controls to protect the local DSEFS data directories.

• File system consistency checks (fsck) and file repair have only limited support. Running fsck will re-replicate blocks that were under-replicated because a node was taken out of a cluster.

• File repair.

• Forced rebalancing, although the cluster will eventually reach balance.

• Checksum.

• Automatic backups.

• Multi-datacenter replication.

• Symbolic links (soft links, symlinks) and hardlinks.

• Snapshots.

1. Configure replication for the metadata and the data blocks.

   You must set the replication factor appropriately to prevent data loss in the case of node failure. Replication factors must be set for both the metadata and the data blocks. The replication factor of 3 for data blocks is suitable for most use-cases.

   a. Globally: set replication for the metadata in the dsefs keyspace that is stored in the database.

   For example, use a CQL statement to configure a replication factor of 3 on the Analytics datacenter using NetworkTopologyStrategy:

   ```sql
   ALTER KEYSPACE dsefs
   WITH REPLICATION = {
     'class': 'NetworkTopologyStrategy',
     'Analytics': '3'};
   ```
Using DataStax Enterprise advanced functionality

**Note:** Datacenter names are case-sensitive. Verify the case of the using utility, such as `dsetool status (page 1367).

b. Run `nodetool repair` on the DSEFS keyspace.

```bash
$ nodetool repair dsefs
```

c. Locally: set the replication factor on a specific DSEFS file or directory where the data blocks are stored.

For example, use the command line:

```bash
$ dse fs mkdir -n 4 newdirectory
```

When a replication factor (RF) is not specified, the RF is inherited from the parent directory.

2. If you have multiple Analytics datacenters, you must configure each DSEFS file system to replicate within its own datacenter:

a. In the `dse.yaml` file, specify a separate DSEFS keyspace for each logical datacenter.

For example, on a cluster with logical datacenters DC1 and DC2.

On each node in DC1:

```yaml
dsefs_options:
  ...
  keyspace_name: dsefs1
```

On each node in DC2:

```yaml
dsefs_options:
  ...
  keyspace_name: dsefs2
```

b. Restart the nodes.

c. Alter the keyspace replication to exist only on the specific datacenters.

On DC1:

```sql
ALTER KEYSSPACE dsefs1
WITH REPLICATION = {
  'class': 'NetworkTopologyStrategy',
  'DC1': '3'};
```

On DC2:
Using DataStax Enterprise advanced functionality

```sql
ALTER KEYSPACE dsefs2
WITH REPLICATION = {
    'class': 'NetworkTopologyStrategy',
    'DC2': '3'};
```

d. Run `nodetool repair` on the DSEFS keyspace.

$ nodetool repair dsefs

For example, in a cluster with multiple datacenters, the keyspace names `dsefs1` and `dsefs2` define separate file systems in each datacenter.

3. When bouncing a streaming application, verify the Kafka log configuration (especially `log.retention.check.interval.ms` and `policies.log.retention.bytes`). Ensure the Kafka log retention policy is robust enough to handle the length of time expected to bring the application and consumers back up.

   For example, if the log retention policy is too conservative and deletes or rolls are logged very frequently to save disk space, the users are likely to encounter issues when attempting to recover from a checkpoint that references offsets that are no longer maintained by the Kafka logs.

DSEFS command line tool

The DSEFS functionality supports operations including uploading, downloading, moving, and deleting files, creating directories, and verifying the DSEFS status.

DSEFS commands are available only in the logical datacenter. DSEFS works with secured and unsecured clusters, see DSEFS authentication (page 298).

You can interact with the DSEFS file system in several modes:

- Interactive command line shell (page 1269).
- As part of `dse` commands (page 1147).
- With a REST API.

Configuring DSEFS shell logging

The default location of the DSEFS shell log file `.dsefs-shell.log` is the user home directory. The default log level is INFO. To configure DSEFS shell logging, edit the `installation_location/resources/dse/conf/logback-dsefs-shell.xml` file.
Using with the dse command line

Precede the DSEFS command with `dse fs`:

```
$ dse [dse_auth_credentials] fs dsefs_command [options]
```

For example, to list the file system status and disk space usage in human-readable format:

```
$ dse -u user1 -p mypassword fs "df -h"
```

Optional command arguments are enclosed in square brackets. For example, `[dse_auth_credentials]` and `[-R]`

Variable values are italicized. For example, `directory` and `[subcommand]`.

Working with the local file system in the DSEFS shell

You can refer to files in the local file system by prefixing paths with `file:`. For example the following command will list files in the system root directory:

```
dsefs dsefs://127.0.0.1:5598/ > ls file:/
```

If you need to perform many subsequent operations on the local file system, first change the current working directory to `file:` or any local file system path:

```
dsefs dsefs://127.0.0.1:5598/ > cd file:
dsefs file:/home/user1/path/to/local/files > ls
```

DSEFS shell remembers the last working directory of each file system separately. To go back to the previous DSEFS directory, enter:

```
dsefs file:/home/user1/path/to/local/files > cd dsefs:
dsefs dsefs://127.0.0.1:5598/ >
```

To go back again to the previous local directory:

```
dsefs dsefs://127.0.0.1:5598/ > cd file:
dsefs file:/home/user1/path/to/local/files >
```

To refer to a path relative to the last working directory of the file system, prefix a relative path with either `dsefs:` or `file:`. The following session will create a directory `new_directory` in the `directory` `/home/user1`:

```
dsefs dsefs://127.0.0.1:5598/ > cd file:/home/user1
```
Using DataStax Enterprise advanced functionality

dsefs file:/home/user1 > cd dsefs:
dsefs dsefs://127.0.0.1:5598/ > mkdir file:new_directory
dsefs dsefs://127.0.0.1:5598/ > realpath file:new_directory
file:/home/user1/new_directory
dsefs dsefs://127.0.0.1:5598/ > stat file:new_directory:
  Owner           user1
  Group           user1
  Permission      rwrx-r-x
  Created         2017-01-15 13:10:06+0200
  Modified        2017-01-15 13:10:06+0200
  Accessed        2017-01-15 13:10:06+0200
  Size            4096

To copy a file between two different file systems, you can also use the cp command with explicit file system prefixes in the paths:

dsefs file:/home/user1/test > cp dsefs:archive.tgz another-archive-copy.tgz
dsefs file:/home/user1/test > ls
 another-archive-copy.tgz archive-copy.tgz archive.tgz

Authentication

For dse dse_auth_credentials you can provide user credentials in several ways, see Providing credentials from DSE tools. For authentication with DSEFS, see DSEFS authentication (page 298).

Wildcard support

Some DSEFS commands support wildcard pattern expansion in the path argument. Path arguments containing wildcards are expanded before method invocation into a set of paths matching the wildcard pattern, then the given method is invoked for each expanded path.

For example in the following directory tree:

```
dirA
|--dirB
|--file1
|--file2
```

Giving the stat dirA/* command would be transparently translated into three invocations:
stat dirA/dirB, stat dirA/file1, and stat dirA/file2.

DSEFS supports the following wildcard patterns:

- * matches any files system entry (file or directory) name, as in the example of stat dirA/*.
- ? matches any single character in the file system entry name. For example stat dirA/dir? matches dirA/dirB.
- [] matches any characters enclosed within the brackets. For example stat dirA/file[0123] matches dirA/file1 and dirA/file2.
• `{} matches any sequence of characters enclosed within the brackets and separated with `, `. For example `stat dirA/{dirB, file2}` matches `dirA/dirB` and `dirA/file2`.

There are no limitations on the number of wildcard patterns in a single path.

For authentication with DSEFS, see DSEFS authentication (page 298).

Executing multiple commands

DSEFS can execute multiple commands on one line. Use quotes around the commands and arguments. Each command will be executed separately by DSEFS.

```
$ dse fs 'cat file1 file2 file3 file4' 'ls dir1'
```

Forcing synchronization

Before confirming writing a file, DSEFS by default forces all blocks of the file to be written to the storage devices. This behavior can be controlled with `--no-force-sync` and `--force-fsync` flags when creating files or directories in the DSEFS shell with `mkdir`, `put`, and `cp` commands. The force/no-force behavior is inherited from the parent directory, if not specified. For example, if a directory is created with `--no-force-sync`, then all files are created with `--no-force-sync` unless `--force-fsync` is explicitly set during file creation.

Turning off forced synchronization improves latency and performance at a cost of durability. For example, if a power loss occurs before writing the data to the storage device, you may lose data. Turn off forced synchronization only if you have a reliable backup power supply in your datacenter and failure of all replicas is unlikely, or if you can afford losing file data.

The Hadoop `SYNC_BLOCK` flag has the same effect as `--force-sync` in DSEFS. The Hadoop `LAZY_PERSIST` flag has the same effect as `--no-force-sync` in DSEFS.

Removing a DSEFS node

When removing a node running DSEFS from a DSE cluster, additional steps are needed to ensure proper correctness within the DSEFS data set.

1. From a node in the same datacenter as the node to be removed, start the DSEFS shell.

```
$ dse fs
```

2. Show the current DSEFS nodes with the `df` command.

```
dsefs > df
```

<table>
<thead>
<tr>
<th>Location</th>
<th>Host</th>
<th>Address</th>
<th>Status</th>
<th>DC</th>
<th>Rack</th>
<th>Free</th>
<th>Reserved</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>144e587c-11b1-4d74-80f7-dc5e0c744aca</td>
<td>up</td>
<td>GraphAnalytics</td>
<td>rack1</td>
<td>Free</td>
<td>10.200.179.38</td>
<td>5598</td>
<td>0</td>
</tr>
</tbody>
</table>
3. Find the node to be removed in the list and note the UUID value for it under the Location column.

4. If the node is up, unmount it from DSEFS with the command `umount UUID`.

5. If the node is not up (for example, after a hardware failure), force unmount it from DSEFS with the command `umount -f UUID`.

6. Continue with the normal steps for removing a node.

Examples

Using the DSEFS shell, these commands put the local `bluefile` to the remote DSEFS `greenfile`:

```
dsefs / > ls -l
```
```
dsefs / > put file:/bluefile greenfile
```

To view the new file in the DSEFS directory:

```
dsefs / > ls -l
```
```
<table>
<thead>
<tr>
<th>Type</th>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
<th>Length</th>
<th>Modified</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>rwxrwxrwx</td>
<td>none</td>
<td>none</td>
<td>17</td>
<td>2016-05-11 09:34:26+0000</td>
<td>greenfile</td>
</tr>
</tbody>
</table>
```

Using the dse command, these commands create the `test2` directory and upload the local `README.md` file to the new DSEFS directory.

```
$ dse fs "mkdir /test2" &&
dse fs "put README.md /test2/README.md"
```

To view the new directory listing:

```
$ dse fs "ls -l /test2"
```
```
<table>
<thead>
<tr>
<th>Type</th>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
<th>Length</th>
<th>Modified</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>rwxrwxrwx</td>
<td>none</td>
<td>none</td>
<td>3382</td>
<td>2016-03-07 23:20:34+0000</td>
<td>README.md</td>
</tr>
</tbody>
</table>
```

You can use two or more dse commands in a single command line. This is faster because the JVM is launched and connected/disconnected with DSEFS only once. For example:

```
$ dse fs "mkdir /test2" "put README.md /test/README.md"
```

The following example shows how to use the `--no-force-sync` flag on a directory, and how to check the state of the `--force-sync` flag using `stat`. These commands are run from within the DSEFS shell.
DSEFS is able to compress files to save storage space and bandwidth. Compression is performed by DSE during upload upon a user’s explicit request. Decompression is transparent. Data is always uncompressed by the server before it is returned to the client.

Compression is performed within block boundaries. The unit of compression—the chunk of data that gets compressed individually—is called a frame and its size can be specified during file upload.

Encoders

DSEFS is shipped with the lz4 encoder which works out of the box.

Compression

To compress files use the `-c` or `--compression-encoder` parameter for `put` or `cp` (page 292) command. The parameter specifies the compression encoder to use for the file that is about to get uploaded.

```bash
dsefs / > put -c lz4 file /path/to/file
```

The frame size can optionally be set with the `-f`, `--compression-frame-size` option.

The maximum frame size in bytes is set in the `compression_frame_max_size` option in `dse.yaml`. If a user sets the frame size to a value greater than `compression_frame_max_size` when using `put -f` an error will be thrown and the command will fail. Modify the `compression_frame_max_size` setting based on the available memory of the node.

Files that are compressed can be appended in the same way as uncompressed files. If the file is compressed the appended data gets transparently compressed with the file’s encoder specified for the initial `put` operation.
Directories can have a default compression encoder specified during directory creation with the `mkdir` command. Newly added files with the `put` command inherit the default compression encoder from containing directory. You can override the default compression encoder with the `c` parameter during `put` operations.

```
dsefs / > mkdir -c lz4 /some/path
```

### Decompression

Decompression is performed automatically for all commands that transport data to the client. There is no need for additional configuration to retrieve the original, decompressed file content.

### Storage space

Enabling compression creates a distinction between the logical and physical file size.

The logical size is the size of a file before uploading it to DSEFS, where it is then compressed. The logical size is shown by the `stat` command under `Size`.

```
dsefs dsefs://10.0.0.1:5598/ > stat /tmp/wikipedia-sample.bz2
FILE dsefs://10.0.0.1:5598/tmp/wikipedia-sample.bz2:
Owner           none
Group           none
Permission      rwxrwxrwx
Created         2017-04-06 20:06:21+0000
Modified        2017-04-06 20:06:21+0000
Accessed        2017-04-06 20:06:21+0000
Size            7723180
Block size      67108864
Redundancy      3
Compressed      true
Encrypted       false
Comment
```

The physical size is the actual size of a data stored on the storage device. The physical size is shown by the `df` command and by the `stat` command for each block separately, under the Compressed length column.

### Limitations

Truncating compressed files is not possible.

**DSEFS authentication**

DSEFS works with secured DataStax Enterprise clusters.
DSEFS authentication with secured clusters

Authentication is required only when it is enabled in the cluster. DSEFS on secured clusters requires the DseAuthenticator, see Configuring DSE Unified Authentication. Authentication is off by default.

DSEFS supports authentication using DSE Unified Authentication, and supports all authentication schemes supported by DSE Authenticator, including Kerberos.

DSEFS authentication can secure client to server communication.

Spark applications

For Spark applications, provide authentication credentials in one of these ways:

- **Set with the** `dse spark-submit` **command using one of the credential options described in Providing credentials on command line.**

- **Programmatically set the user credentials in the Spark configuration object before the SparkContext is created:**

```java
conf.set("spark.hadoop.com.datastax.bdp.fs.client.authentication.basic.username", <user>)
conf.set("spark.hadoop.com.datastax.bdp.fs.client.authentication.basic.password", <pass>)
```

If a Kerberos authentication token is in use, you do not need to set any properties in the context object. If you need to explicitly set the token, set the `spark.hadoop.cassandra.auth.token` property.

- **When running the Spark Shell, where the SparkContext is created at startup, set the properties in the Hadoop configuration object:**

```java
sc.hadoopConfiguration.set("com.datastax.bdp.fs.client.authentication.basic.username", <user>)
sc.hadoopConfiguration.set("com.datastax.bdp.fs.client.authentication.basic.password", <pass>)
```

**Note the absence of the spark.hadoop prefix.**

- **When running a Spark application or the Spark Shell, provide properties in the spark- defaults.conf configuration file:**

```xml
<property>
  <name>com.datastax.bdp.fs.client.authentication.basic.username</name>
  <value>username</value>
</property>
<property>
  <name>com.datastax.bdp.fs.client.authentication.basic.password</name>
  <value>password</value>
</property>
```
Optional: If you want to use this method, but do not have privileges to write to `core-default.xml`, copy this file to any location `path` and set the environment variable to point to the file with:

```bash
export HADOOP2_CONF_DIR=path
```

**DSEFS shell**

Providing authentication credentials while using the DSEFS shell is as easy as in other DSE tools. The DSEFS shell supports different authentication methods listed below in priority order. When more than one method can be used, the one with higher priority is chosen. For example when the `DSE_TOKEN` environment variable is set and the DSEFS shell is started with a username and password set as environment variables in the `$HOME/.dserc` file, the provided username and password is used for authentication as it has higher priority.

1. **Specifying a username and password.**
   - Providing credentials on command line
   - Providing credentials in a file
   - Providing credentials using environment variable

   ```bash
   $ export DSE_USERNAME=username &&
   export DSE_PASSWORD=password
   ```

   ```bash
   $ dse fs 'mkdir /dir1'
   ```

2. **Using a Kerberos delegation token.** See `dse client-tool cassandra` (page 1228) for further information.

   ```bash
   $ export DSE_TOKEN=`dse -u token_user -p password client-tool cassandra generate-token`
   ```

   ```bash
   $ dse fs 'mkdir /dir1'
   ```

3. **Using a cached Kerberos ticket** after authenticating using a tool like `kinit`.

   ```bash
   $ kinit username
   ```

   ```bash
   $ dse fs 'mkdir /dir1'
   ```

4. **Using a Kerberos keytab file and a login configuration file.**

   If the configuration file is in a non-default location, specify the location using the `java.security.auth.login.config` property in the `DSEFS_SHELL_OPTS` variable:
DSEFS REST interface

The DSEFS REST interface (page 305) supports Kerberos authentication using SPNEGO and Kerberos delegation token authentication.

To automate API calls to the DSEFS REST endpoint that is secured by SPNEGO, use curl 7.38.0 or later:

```
$ kinit ...
$ curl -v --negotiate -u "http://localhost:5598/webhdfs/v1/?op=LISTSTATUS"
```

To use Kerberos delegation token authentication:

1. Obtain a delegation token using one of these methods:
   - dse client-tool (page 1228)
     For example, to generate a delegation token with the current user as the token renewer
     ```
     $ dse client-tool cassandra --generate-token
     ```
   - curl
     ```
     $ curl -v --negotiate -u "http://10.200.177.136:5598/webhdfs/v1/?op=GETDELEGATIONTOKEN" # uses Spnego to obtain the token
     ```

2. Use the delegation token:

```
$ curl -v "http://localhost:5598/webhdfs/v1/?op=LISTSTATUS&delegation=delegation_token"
```

**DSEFS authorization**

DSEFS authorization verifies user and group permissions on files and directories stored in DSEFS.

DSEFS authorization is disabled by default. It requires no configuration, it is automatically enabled along with DSE authorization (page 107).
Owners, groups, and permissions

In unsecured clusters with DSEFS authentication disabled all newly created files and directories are created with the owner set to `none`, group set to `none`. In unsecured clusters every DSEFS user has full access to every file and directory.

```
dsefs dsefs://127.0.0.1:5598/ > ls -l
Type  Permission  Owner      Group      Length   Modified
Name
 dir   rwxrwxrwx   none       none            -   2016-12-01
  15:50:49+0100  some_dir
```

In secured clusters with DSEFS authentication enabled all newly created files and directories are created with owner set the authenticated user's username and group set to authenticated user primary role. See the CQL roles documentation for detailed information on user roles. File and directory permissions can be specified during creation as a parameter for the `put` and `mkdir` commands. Please use `help put` or `help mkdir` for details.

```
dsefs dsefs://127.0.0.1:5598/ > ls -l
Type  Permission  Owner      Group      Length   Modified
Name
 dir   rwxr-x---   john       admin           -   2016-12-02
  15:52:54+0100  other_dir
```

To change the owner or group of an existing file or directory use `chown` or `chgrp` commands. Please use `help chown` or `help chgrp` for details.

DSEFS by default creates directories with `rwxr-xr-x` (octal 755) permissions and files with `rw-r-r-` (octal 644). To change the permissions of an existing file or directory use the `chmod` command. Please use `help chmod` for details.

DSEFS superusers

A DSEFS user is a superuser if and only if the user is a database superuser. Superusers are allowed to read and write every file and directory stored in DSEFS. Only superusers are allowed to execute DSEFS maintenance operations like `fsck` and `umount`.

DSEFS users

User access is verified against:

- Owner permissions if the file or directory owner name is equal to the authenticated user's username.
- Group permissions if the file or directory group belongs to the authenticated user's groups. Groups are mapped from the database's user role names.
- Other permissions if the above conditions are false.

Each DSEFS command (page 292) requires it’s own set of permissions. For a given path `a/b/c`, `c` is a leaf and `a/b` is a parent path. The following table shows what permissions
must be present for the given operation to succeed. R indicates read, W indicates write, and X indicates execute privileges.

Table 10: Affect of permissions on files by DSEFS command

<table>
<thead>
<tr>
<th>Command</th>
<th>Path checked for permissions</th>
<th>Parent path permissions</th>
<th>Leaf permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>append a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td>W</td>
</tr>
<tr>
<td>cat a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td>R</td>
</tr>
<tr>
<td>cd a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>chgrp</td>
<td>same as in chown for group</td>
<td></td>
<td></td>
</tr>
<tr>
<td>chmod a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td>The user must be the owner.</td>
</tr>
<tr>
<td>chown a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td>Only superusers can change the owner. To change the group the user needs to be a member of the target group or be a superuser.</td>
</tr>
<tr>
<td>cp</td>
<td>same as in get and than put</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expand a/?/c</td>
<td>a/?/c</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>get a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td>R</td>
</tr>
<tr>
<td>ls a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td>RX if c is a directory.</td>
</tr>
<tr>
<td>mkdir a/b/c</td>
<td>a/b</td>
<td>X</td>
<td>WX</td>
</tr>
<tr>
<td>mv a/b/c d/e/f</td>
<td>a/b and d/e</td>
<td>X</td>
<td>WX</td>
</tr>
<tr>
<td>put a/b/c</td>
<td>a/b</td>
<td>X</td>
<td>WX</td>
</tr>
<tr>
<td>realpath a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>rename a/b/c d</td>
<td>a/b</td>
<td>X</td>
<td>WX</td>
</tr>
<tr>
<td>rm a/b/c</td>
<td>a/b</td>
<td>X</td>
<td>WX</td>
</tr>
<tr>
<td>rmdir a/b/c</td>
<td>a/b</td>
<td>X</td>
<td>WX</td>
</tr>
<tr>
<td>stat a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Command</th>
<th>Path checked for permissions</th>
<th>Parent path permissions</th>
<th>Leaf permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>truncate a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td>W</td>
</tr>
</tbody>
</table>

Authorization transitional mode

DSEFS authorization supports transitional mode *(page 107)* provided by DSEAuthorizer. Legacy authorizers, like TransitionalAuthorizer, are not supported. DSE will not start if unsupported authorizer is configured and error is reported in log messages.

**Enabling SSL encryption for DSEFS**

There are two parts to enabling SSL encryption for DSEFS: node-to-node encryption, and client-to-node encryption. Enabling node-to-node encryption in DSE automatically enables encrypted communication between DSEFS nodes. DSE nodes with client-to-node encryption enabled allow SSL connections from the DSEFS shell.

Configuring the DSEFS shell to use SSL encryption

In most cases, you don’t need to add any DSEFS shell settings to connect using SSL. If a `~/.dse/dsefs-shell.yaml` configuration file cannot be found, DSEFS shell will attempt to load server-side configuration and SSL settings from DSE configuration files.

To manually configure SSL, create and edit the DSEFS shell configuration file. The DSEFS shell is configured in the `~/.dse/dsefs-shell.yaml` configuration file. Add the following settings to enable SSL encryption:

```yaml
encryption_options:
  # set to true to enable secure client-server connection
  enabled: true
  # if optional is true, and enabled is true, ssl will be used if possible,
  # but will failover to non ssl
  optional: true
  # path to truststore
  truststore_path: path
  # optional truststore type; default value: JKS
  truststore_type:
  # optional, will be prompted for if doesn't exist
  truststore_password:
  # path to keystore
  keystore_path: path
  # optional truststore type; default value: JKS
  keystore_type:
  # optional, will be prompted for if doesn't exist
  keystore_password:
  # optional protocol name; default value: TLSv1.2
  protocol:
  # optional keymanager and trustmanager algorithm; default value: SunX509
  algorithm:
```
Using DataStax Enterprise advanced functionality

# optional list of ciphers
cipher_suites:
# set to true to enable checking if the certificate matches endpoint address
require_endpoint_verification: false

The same settings can be given as dse fs command-line options, except keystore_password, truststore_password, and cipher_suites. If passwords are not given in the configuration file, they will be prompted for at the DSEFS shell startup. The command line options override settings read from the configuration file.

**Note:** If a non-optional secure connection is established, a [secure] flag will appear in the prompt of the DSEFS shell.

**Using the DSEFS REST interface**

DSEFS provides a REST interface that implements the commands from WebHDFS.

The REST interface is enabled on all DSE nodes running DSEFS. It is available at the following base URI: `http://node_hostname_or_IP_address:5598/webhdfs/v1`

For example from a terminal using the curl command:

```
$ curl -L -X PUT 'localhost:5598/webhdfs/v1/fs/a/b/c/d/e?op=MKDIRS' &&
curl -L -X PUT -T logfile.txt '127.0.0.1:5598/webhdfs/v1/fs/log?op=CREATE&overwrite=true&blocksize=50000&rf=1' &&
curl -L -X POST logfile.txt 'localhost:5598/webhdfs/v1/fs/log?op=APPEND'
```

Or from the DSE Spark shell:

```
val rdd1 = sc.textFile("webhdfs://localhost:5598/webhdfs/v1/fs/log")
```

**Programmatic access to DSEFS**

DSEFS can be accessed programmatically from an application by obtaining DSEFS's implementation of Hadoop's `FileSystem` interface.

DSE includes a demo project with simple applications that demonstrate how to acquire, configure, and use this implementation. The demo project demonstrates reading, writing and connecting to a secured DSEFS using the API. The demo is located in the `dsefs` directory under the `demos` directory.

The default location of the `demos` directory depends on the type of installation:

- **Package installations:** `/usr/share/dse/demos`
- **Tarball installations:** `installation_location/demos`
The README.md has instructions on building and running the demo applications.

**Hadoop FileSystem interface implemented by DseFileSystem**

The DseFileSystem class has partial support of the Hadoop FileSystem interface. The following table outlines which methods have been implemented.

### Table 11: Methods of Hadoop FileSystem interface implemented by DseFileSystem

<table>
<thead>
<tr>
<th>Method</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>getScheme()</td>
<td>#</td>
<td>since 5.0.12, 5.1.6</td>
</tr>
<tr>
<td>getURI()</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>getName()</td>
<td>#</td>
<td>default, deprecated</td>
</tr>
<tr>
<td>getDefaultPort()</td>
<td>#</td>
<td>since 5.0.12, 5.1.6</td>
</tr>
<tr>
<td>makeQualified(Path)</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>getDelegationToken(String)</td>
<td>#</td>
<td>returns null</td>
</tr>
<tr>
<td>addDelegationTokens(String, Credentials)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>collectDelegationTokens(...)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>getChildFileSystems()</td>
<td>#</td>
<td>default, returns null</td>
</tr>
<tr>
<td>getFileBlockLocations(FileStatus, long, long)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>getFileBlockLocations(Path, long, long)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>getServerDefaults()</td>
<td>#</td>
<td>default, deprecated</td>
</tr>
<tr>
<td>getServerDefaults(Path)</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>resolvePath(Path)</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>open</td>
<td>#</td>
<td>all variants, buffer size not supported</td>
</tr>
<tr>
<td>create</td>
<td>#</td>
<td>all variants, checksum options, progress reporting and APPEND, NEW_BLOCK flags not supported</td>
</tr>
<tr>
<td>createNonRecursive</td>
<td>#</td>
<td>all variants</td>
</tr>
<tr>
<td>createNewFile</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>Method</td>
<td>Status</td>
<td>Comment</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>append</td>
<td>#</td>
<td>all variants, progress reporting not supported</td>
</tr>
<tr>
<td>concat</td>
<td>#</td>
<td>since 5.0.12, 5.1.6</td>
</tr>
<tr>
<td>getReplication(Path)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>setReplication(Path, short)</td>
<td>#</td>
<td>does nothing</td>
</tr>
<tr>
<td>rename</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>truncate(Path, long)</td>
<td>#</td>
<td>since 5.0.12, 5.1.6</td>
</tr>
<tr>
<td>delete(Path)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>delete(Path, boolean)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>deleteOnExit(Path)</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>cancelDeleteOnExit(Path)</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>exists(Path)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>isDirectory(Path)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>isFile(Path)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>getLength(Path)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>getContentSummary(Path)</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>listStatus</td>
<td>#</td>
<td>all variants</td>
</tr>
<tr>
<td>listCorruptFileBlocks(Path)</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>globStatus</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>listLocatedStatus</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>listStatusIterator</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>listFiles</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>getHomeDirectory()</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>getWorkingDirectory()</td>
<td>#</td>
<td></td>
</tr>
</tbody>
</table>
### Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Method</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>setWorkingDirectory()</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>getInitialWorkingDirectory()</td>
<td>#</td>
<td>default, returns null</td>
</tr>
<tr>
<td>mkdirs</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>copyFromLocalFile</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>moveFromLocalFile</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>copyToLocalFile</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>moveToLocalFile</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>startLocalOutput</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>close</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>getUsed</td>
<td>#</td>
<td>default, slow</td>
</tr>
<tr>
<td>getBlockSize</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>getDefaultBlockSize()</td>
<td>#</td>
<td>since 5.0.12, 5.1.6</td>
</tr>
<tr>
<td>getDefaultBlockSize(Path)</td>
<td>#</td>
<td>since 5.0.12, 5.1.6</td>
</tr>
<tr>
<td>getDefaultReplication()</td>
<td>#</td>
<td>since 5.0.12, 5.1.6</td>
</tr>
<tr>
<td>getDefaultReplication(Path)</td>
<td>#</td>
<td>since 5.0.12, 5.1.6</td>
</tr>
<tr>
<td>getFileStatus(Path)</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>access(Path, FsAction)</td>
<td>#</td>
<td>default</td>
</tr>
<tr>
<td>createSymLink</td>
<td></td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>getFileLinkStatus</td>
<td>#</td>
<td>default, same as getFileStatus</td>
</tr>
<tr>
<td>supportsSymLinks</td>
<td>#</td>
<td>returns false</td>
</tr>
<tr>
<td>getLinkTarget</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>resolveLink</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>getFileChecksum</td>
<td>#</td>
<td>returns null</td>
</tr>
<tr>
<td>setVerifyChecksum</td>
<td>#</td>
<td>does nothing</td>
</tr>
<tr>
<td>Method</td>
<td>Status</td>
<td>Comment</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>setWriteChecksum</td>
<td>#</td>
<td>does nothing</td>
</tr>
<tr>
<td>getStatus</td>
<td>#</td>
<td>default, returns incorrect default data</td>
</tr>
<tr>
<td>setPermission</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>setOwner</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>setTimes</td>
<td>#</td>
<td>does nothing</td>
</tr>
<tr>
<td>createSnapshot</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>renameSnapshot</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>deleteSnapshot</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>modifyAclEntries</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>removeAclEntries</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>removeDefaultAcl</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>removeAcl</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>setAcl</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>getAclStatus</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>setXAttr</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>getXAttr</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>getXAttrs</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
<tr>
<td>listXAttrs</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Method</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>removeXAttr</td>
<td>#</td>
<td>throws UnsupportedOperationException</td>
</tr>
</tbody>
</table>

Using JMX to read DSEFS metrics

DSEFS reports status and performance metrics through JMX in the domain `com.datastax.bdp:type=dsefs`. This page describes the classes exposed in JMX.

Location

Location metrics provide information about each DSEFS location status. There is one set of Location metrics for each DSEFS location. Every DSE node knows about all locations, so connect to any node to get the full status of the cluster. The following gauges are defined:

- **directory**: Path to the directory where DSEFS data is stored. This is a constant value configured in dse.yaml.
- **estFreeSpace**: Estimated amount of free space on the device where the storage directory is located, in bytes. This value is refreshed periodically, so if you need an up-to-date value, read the `BlockStore.freeSpace` metric.
- **estUsedSpace**: Estimated amount of space used by the contents of the storage directory, in bytes. This value is refreshed periodically, so if you need an up-to-date value, read the `BlockStore.usedSpace` metric.
- **minFreeSpace**: Amount of reserved space in bytes. Configured statically in dse.yaml.
- **privateAddress**: IP and port of the endpoint for DSEFS internode communication.
- **publicAddress**: IP and port of the endpoint for DSEFS clients.
- **readOnly**: Returns true if the location is in read-only mode.
- **status**: One of the following values: `up`, `down`, `unavailable`. If the location is `up`, the location is fully operational and this node will attempt to read or write from it. If the location is `down`, the location is on a node that has been gracefully shut down by the administrator and no reads or writes will be attempted. If the location is `unavailable`, this node has problems with communicating with that location, and the real status is unknown. This node will check the status periodically.
- **storageWeight**: How much data relative to other locations will be stored in this location. This is a static value configured in dse.yaml.
BlockStore

BlockStore metrics report how fast and how much data is being read/written by the data layer of the DSEFS node. They are reported only for the locations managed by the node to which you connect with JMX. In order to get metrics information for all the locations in the cluster, you need to individually connect to all nodes with DSEFS.

- **blocksDeleted**
  - How many blocks are deleted, in blocks per second.

- **blocksRead**
  - Read accesses in blocks per second.

- **blocksWritten**
  - Writes in blocks per second.

- **bytesDeleted**
  - How fast data is removed, in bytes per second.

- **bytesRead**
  - How fast data is being read, in bytes per second.

- **bytesWritten**
  - How fast data is written, in bytes per second.

- **readErrors**
  - The total count and rate of read errors (rate in errors per second).

- **writeErrors**
  - The total count and rate of write errors (rate in errors per second).

- **directory**
  - The path to the storage directory of this location.

- **freeSpace**
  - How much space is left on the device in bytes.

- **usedSpace**
  - Estimated amount of space used by this location in bytes.

RestServer

RestServer reports metrics related to the communication layer of DSEFS, separately for internode traffic and clients. Each set of these metrics is identified by a scope of the form: `listen address:listen port`. By default port 5598 is used for clients, and port 5599 is for internode communication.

- **connectionCount**
  - The current number of open inbound connections.

- **connectionRate**
  - The total rate and count of connections since the server was started.

- **requestRate**

- **deleteRate**

- **getRate**

- **postRate**

- **putRate**
  - The total rate and number of requests, respectively: all, DELETE, GET, POST, and PUT requests.

- **downloadBytesRate**
Using DataStax Enterprise advanced functionality

Throughput in bytes per second of the transfer from server to client. **uploadBytesRate**
Throughput in bytes per second of the transfer from client to server. **responseTime**
The time that elapses from receiving the full request body to the moment the server starts sending out the response. **uploadTime**
The time it takes to read the request body from the client. **downloadTime**
The time that it takes to send the response body to the client. **errors**
A counter which is increased every time the service handling the request throws an unexpected error. **errors** is not increased by errors handled by the service logic. For example, file not found errors do not increment **errors**.

CassandraClient

**CassandraClient** reports metrics related to the communication layer between DSEFS and the database. **responseTime**
Tracks the response times of database queries. **errors**
A counter increased by query execution errors (for example, timeout errors).

DSE Search

DSE Search allows you to quickly find data and provide a modern search experience for your users, helping you create features like product catalogs, document repositories, ad-hoc reporting engines, and more.

Because DataStax Enterprise is a cohesive data management platform so other workloads such as **DSE Graph** (page 373), **DSE Analytics and Search integration** (page 178), and **DSE Analytics** (page 176) can take full advantage of the indexing and query capabilities of DSE Search.

About DSE Search

DSE Search is an integral part of the always-on DataStax Enterprise (DSE) data platform. DSE Search simplifies using search applications for data stored in a database. DSE Search allows you to quickly find data and provide a modern search experience for your users, helping you create features like product catalogs, document repositories, ad-hoc reporting engines, and more. See **DSE Search architecture**.

Because DataStax Enterprise is a cohesive data management platform so other workloads such as **DSE Graph** (page 373), **DSE Analytics and Search integration** (page 178), and **DSE Analytics** (page 176) can take full advantage of the indexing and query capabilities of DSE Search.
DSE Search manages search indexes with a persistent store.

The benefits of running enterprise search functions through DataStax Enterprise and DSE Search include:

- DSE Search is backed by a scalable database.
- A persistent store for search indexes.
- A fully fault-tolerant, no-single-point-of-failure search architecture across multiple datacenters.
- Add search capacity just like you add capacity in the DSE database.
- Set up replication for DSE Search nodes the same way as other nodes by creating a keyspace or changing the replication factor of a keyspace to optimize performance.
- DSE Search has two indexing modes: Near-real-time (NRT) and live indexing, also called real-time (RT) indexing. Configure and tune DSE Search for maximum indexing throughput.
- Near real-time query capabilities.
- TDE encryption of DSE Search data, including search indexes and commit logs. See Encrypting Search indexes.
- CQL index management commands simplify search index management.
- Local node (optional) management of search indexing resources with dsetool (page 327) commands.
- Read/write to any DSE Search node and automatically index stored data.
- Examine and aggregate real-time data using CQL.
- Fault-tolerant queries, efficient deep paging, and advanced search node resiliency.
- Virtual nodes (vnodes) (page 173) support.
- Set the location of the search index.
- Native CQL queries leverage search indexes for a wide array of CQL query functionality and indexing support.
- Using CQL, DSE Search supports partial document updates that enable you to modify existing information while maintaining a lower transaction cost.
- Supports indexing and querying of advanced data types, including tuples and user-defined types (UDT).
- DSE Search is built with a production-certified version of Apache Solr™. DSE Search uses some Solr tools and APIs, the implementation does not guarantee that Solr tools and APIs work as expected. Be sure to review all unsupported features for DSE Search (page 314).

See the DataStax blog post What’s New for Search in DSE 6.

**DSE Search versus Open Source Apache Solr™**

By virtue of its integration into DataStax Enterprise, differences exist between DSE Search and Open Source Solr (OSS).
## Major differences

<table>
<thead>
<tr>
<th>Capability</th>
<th>DSE Search</th>
<th>OS Solr</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Includes a database</td>
<td>yes</td>
<td>no</td>
<td>For OSS, create an interface to add a database.</td>
</tr>
<tr>
<td>Indexes real-time data</td>
<td>yes</td>
<td>no</td>
<td>Ingests real-time data and automatically indexes the data.</td>
</tr>
<tr>
<td>Provides an intuitive way to update data</td>
<td>yes</td>
<td>no</td>
<td>CQL for loading and updating data.</td>
</tr>
<tr>
<td>Supports data distribution</td>
<td>yes</td>
<td>yes [1]</td>
<td>Transparently distributes real-time, analytics, and search data to multiple nodes in a cluster.</td>
</tr>
<tr>
<td>Balances loads on nodes/shards</td>
<td>yes</td>
<td>no</td>
<td>Unlike Solr and Solr Cloud, DSE Search loads can be efficiently rebalanced.</td>
</tr>
<tr>
<td>Spans indexes over multiple datacenters</td>
<td>yes</td>
<td>no</td>
<td>A DSE cluster can have more than one datacenter for different types of nodes.</td>
</tr>
<tr>
<td>Makes durable updates to data</td>
<td>yes</td>
<td>no</td>
<td>Updates are durable and written to the commit log for all updates.</td>
</tr>
<tr>
<td>Automatically reindexes search data</td>
<td>yes</td>
<td>no</td>
<td>OSS requires the client to reingest everything to reindex data in Solr.</td>
</tr>
<tr>
<td>Upgrades of Apache Lucene® preserve data</td>
<td>yes</td>
<td>no</td>
<td>DataStax integrates Lucene upgrades periodically and data is preserved when you upgrade DSE.</td>
</tr>
<tr>
<td>Supports timeAllowed queries with deep paging.</td>
<td>yes</td>
<td>no</td>
<td>OSS Solr does not support using timeAllowed queries with deep paging.</td>
</tr>
</tbody>
</table>


### Feature differences

DSE Search supports limiting queries by time by using the `solr timeAllowed` parameter.

DSE Search differs from native Solr:

- If the timeAllowed is exceeded, an exception is thrown.
- If the timeAllowed is exceeded, and the additional shards.tolerant parameter is set to true, the application returns the partial results collected so far.

When partial results are returned, the CQL custom payload contains the `DSESearch.isPartialResults` key.

### Unsupported features for DSE Search

Unsupported features include Apache Cassandra™ and Apache Solr™ features.
Apache Solr™ and Apache Lucene® limitations

Apache Solr and Lucene limitations apply to DSE Search. For example:

- The 2 billion records per node limitation as described in Lucene limitations.
- The 1024 maxBoolean clause limit in SOLR-4586.

- Solr field name policy applies to the indexed field names:
  
  # Every field must have a name.
  # Field names must consist of alphanumeric or underscore characters only.
  # Fields cannot start with a digit.
  # Names with both leading and trailing underscores (for example, _version_) are reserved.

  Note: Non-compliant field names are not supported from all components. Backward compatibility is not guaranteed.

- Limitations and known Apache Solr issues apply to DSE Search queries. For example: incorrect SORT results for tokenized text fields.

Unsupported Apache Cassandra features

These limitations apply to DSE Search:

- Column aliases are not supported in solr_query queries.
- Continuous paging.
- Static columns
- Counter columns
- Super columns
- Thrift-compatible tables with column comparators other than UTF-8 or ASCII.
- PER PARTITION clause is not supported for DSE Search solr_query queries.
- Indexing frozen maps is not supported. However, indexing frozen sets and lists of native and user-defined (tuple/UDT) element types is supported.
- Using DSE Search with newly created COMPACT STORAGE tables is deprecated.

Unsupported Apache Solr™ features

These limitations apply to DSE Search:

- DSE Search does not support Solr Managed Resources.
- Solr schema fields that are both dynamic and multiValued only for CQL-based search indexes.
  
  The deprecated replaceFields request parameters on document updates for CQL-based search indexes. Instead, use the suggested procedure for inserting/updating data.
  
  Block joins based on the Lucene BlockJoinQuery in search indexes and CQL tables.
  
  Schemless mode.
  
  Partial schema updates through the REST API after search indexes are changed.
For example, to update individual fields of a schema using the REST API to add a new field to a schema, you must change the `schema.xml` file, upload it again, and reload the core (same for copy fields).

- `org.apache.solr.spelling.IndexBasedSpellChecker` and `org.apache.solr.spelling.FileBasedSpellChecker`  
  Instead use `org.apache.solr.spelling.DirectSolrSpellChecker` for spell checking.

- The commitWithin parameter.
- The `SolrCloud CloudSolrServer` feature of SolrJ for endpoint discovery and round-robin load balancing.
- The DSE Search configurable `SolrFilterCache` does not support auto-warming.
- DSE Search does not support the duration Cassandra data type.
- SELECT statements with DISTINCT are not supported with solr_query.
- RealTime Get.
- GetReplicationHandler: Store & Restore.
- `useDocValuesAsStored` in schema fields and as a query request parameter.
- Solr Graph queries.
- Solr SQLStreaming aggregations.
- Data import handler.
- Tuple/UDT subfield sorting and faceting.
- The `dataDir` parameter in `solrconfig.xml`.

**Deprecated Solr and Lucene features**

- The Tika functionality that is bundled with Apache Solr is deprecated. Instead, use the stand-alone Apache Tika project.
- Highlighting.
- MoreLikeThis search component.
- SpellCheck search component.
- Suggester (suggest search component).
- ClassicSimilarityFactory class.

**Other deprecated features**

The following features that were previously available for use with DSE Search are deprecated and no longer supported.

- The DSE custom URP implementation is deprecated. Use the `field input/output` (page 364) (FIT) transformer API instead.

**Other unsupported features**

- HTTP delete-by-query, HTTP delete-by-id, and other Solr HTTP updates.
- JBOD mode.
- The Solr updatelog is not supported in DSE Search.
The commit log replaces the Solr updatelog. Consequently, features that require the updateLog are not supported. Instead of using atomic updates, partial document updates are available by running the update with CQL.

- CQL Solr queries do not support native functions or column aliases as selectors.
- RamDirectoryFactory or other non-persistent DirectoryFactory implementations.
- Tuple and UDT limitations apply.

## Configuring DSE Search

### DSE Search reference

Reference information for DSE Search.

### Search index config

Reference information to change query behavior for search indexes:

- DataStax recommends CQL `CREATE SEARCH INDEX` and `ALTER SEARCH INDEX CONFIG` commands.
- `dsetool (page 327)` commands can also be used to manage search indexes.

### Changing search index config

To create and make changes to the search index config, follow these basic steps:

1. Create a search index. For example:

   ```
   CREATE SEARCH INDEX ON demo.health_data;
   ```

2. Alter the search index. For example:

   ```
   ALTER SEARCH INDEX CONFIG ON demo.health_data SET autoCommitTime = 30000;
   ```

3. Optionally view the XML of the pending search index. For example:

   ```
   DESCRIBE PENDING SEARCH INDEX CONFIG on demo.health_data;
   ```

4. Make the pending changes active. For example:

   ```
   RELOAD SEARCH INDEX ON demo.health_data;
   ```

### Sample search index config

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<config>
```
For CQL index management, use configuration element shortcuts with CQL commands.

Configuration elements are listed alphabetically by shortcut. The XML element is shown with the element start tag. An ellipsis indicates that other elements or attributes are not shown.

**autoCommitTime**

Defines the time interval between updates to the search index with the most recent data after an INSERT, UPDATE, or DELETE. By default, changes are automatically committed every 10000 milliseconds. To change the time interval between updates:

1. Set auto commit time on the pending search index:

   ```sql
   ALTER SEARCH INDEX CONFIG ON wiki.solr SET autoCommitTime = 30000;
   ```

2. You can view the pending search config:

   ```sql
   DESCRIBE PENDING SEARCH INDEX CONFIG on wiki.solr;
   ```

   The resulting XML shows the maximum time between updates is 30000 milliseconds:

   ```xml
   <updateHandler class="solr.DirectUpdateHandler2">
   <autoSoftCommit>
   <maxTime>30000</maxTime>
   ```
Using DataStax Enterprise advanced functionality

3. To make the pending changes active, reload the search index:

    RELOAD SEARCH INDEX ON wiki.solr;

See Configuring and tuning indexing performance.

defaultQueryField
Name of the default field to query. Default not set. To set the field to use when no field is specified by the query, see Setting up default query field.

directoryFactory
The directory factory to use for search indexes. Encryption is enabled per search index. To enable encryption for a search index, change the class for directoryFactory to EncryptedFSDirectoryFactory.

1. Enable encryption on the pending search index:

    ALTER SEARCH INDEX CONFIG ON wiki.solr SET directoryFactory = EncryptedFSDirectoryFactory;

2. You can view the pending search config:

    DESCRIBE PENDING SEARCH INDEX CONFIG on wiki.solr;

    The resulting XML shows that encryption is enabled:

    <directoryFactory class="solr.EncryptedFSDirectoryFactory" name="DirectoryFactory"/>

3. To make the pending changes active, reload the search index:

    RELOAD SEARCH INDEX ON wiki.solr;

Even though additional properties are available to tune encryption, DataStax recommends using the default settings.

filterCacheLowWaterMark
Default is 1024 MB. See below.

filterCacheHighWaterMark
Default is 2048 MB. The DSE Search configurable filter cache reliably bounds the filter cache memory usage for a search index. This implementation contrasts with the default Solr implementation which defines bounds for filter cache usage per segment. SolrFilterCache bounding works by evicting cache entries after the configured per search index (per core) high watermark is reached, and stopping after the configured lower watermark is reached.
Note:
- The filter cache is cleared when the search index is reloaded.
- SolrFilterCache does not support auto-warming.

SolrFilterCache defaults to offheap. In general, the larger the index is, then the larger the filter cache should be. A good default is 1 to 2 GB. If the index is 1 billion docs per node, then set to 4 to 5 GB.

1. To change cache eviction for a large index, set the low and high values one at a time:

```
ALTER SEARCH INDEX CONFIG ON solr.wiki SET
  filterCacheHighWaterMark = 5000;

ALTER SEARCH INDEX CONFIG ON solr.wiki SET
  filterCacheLowWaterMark = 2000;
```

2. View the pending search index config:

```
<query>
  ...
  <filterCache class="solr.SolrFilterCache"
    highWaterMarkMB="5000" lowWaterMarkMB="2000"/>
  ...
</query>
```

3. To make the pending changes active, reload the search index:

```
RELOAD SEARCH INDEX ON wiki.solr;
```

**mergeFactor**

When a new segment causes the number of lowest-level segments to exceed the merge factor value, then those segments are merged together to form a single large segment. When the merge factor is 10, each merge results in the creation of a single segment that is about ten times larger than each of its ten constituents. When there are 10 of these larger segments, then they in turn are merged into an even larger single segment. Default is 10.

1. To change the number of segments to merge at one time:

```
ALTER SEARCH INDEX CONFIG ON solr.wiki SET mergeFactor = 5;
```

2. View the pending search index config:

```
<indexConfig>
  ...
  <mergeFactor>10</mergeFactor>
```
... 
</indexConfig>

3. To make the pending changes active, reload the search index:

```
RELOAD SEARCH INDEX ON wiki.solr;
```

**mergeMaxThreadCount**
Must configure with mergeMaxMergeCount. The number of concurrent merges that Lucene can perform for the search index. The default mergeScheduler settings are set automatically. Do not adjust this setting.
Default: \(\frac{1}{2}\) the number of tpc_cores

**mergeMaxMergeCount**
Must configure with mergeMaxThreadCount. The number of pending merges (active and in the backlog) that can accumulate before segment merging starts to block/throttle incoming writes. The default mergeScheduler settings are set automatically. Do not adjust this setting.
Default: \(2x\) the mergeMaxThreadCount

**ramBufferSize**
The index RAM buffer size in megabytes (MB). The RAM buffer holds uncommitted documents. A larger RAM buffer reduces flushes. Segments are also larger when flushed. Fewer flushes reduces I/O pressure which is ideal for higher write workload scenarios.

For example, adjust the ramBufferSize when you configure live indexing:

```
ALTER SEARCH INDEX CONFIG ON wiki.solr SET autoCommitTime = 100;
ALTER SEARCH INDEX CONFIG ON wiki.solr SET realtime = true;
ALTER SEARCH INDEX CONFIG ON wiki.solr SET ramBufferSize = 2048;
RELOAD SEARCH INDEX ON wiki.solr;
```

Default: 512

**realtime**
Enables live indexing to increase indexing throughput. Enable live indexing on only one node per cluster. Live indexing, also called real-time (RT) indexing, supports searching directly against the Lucene RAM buffer and more frequent, cheaper soft-commits, which provide earlier visibility to newly indexed data.
Live indexing requires a larger RAM buffer and more memory usage than an otherwise equivalent NRT setup. See Tuning RT indexing.

Configuration elements without shortcuts

To specify configuration elements that do not have shortcuts, you can specify the XML path to the setting and separate child elements using a period.

**deleteApplicationStrategy**
Controls how to retrieve deleted documents when deletes are being applied. Seek exact is the safe default most people should choose, but for a little extra performance you can try seekceiling.
Valid case-insensitive values are:

- **seekexact**
  Uses bloom filters to avoid reading from most segments. Use when memory is limited and the unique key field data does not fit into memory.

- **seekceiling**
  More performant when documents are deleted/inserted into the database with sequential keys, because this strategy can stop reading from segments when it is known that terms can no longer appear.

Default: seekexact

**mergePolicyFactory**
The AutoExpungeDeletesTieredMergePolicy custom merge policy is based on TieredMergePolicy. This policy cleans up the large segments by merging them when deletes reach the percentage threshold. A single auto expunge merge occurs at a time. Use for large indexes that are not merging the largest segments due to deletes. To determine whether this merge setting is appropriate for your workflow, view the segments on the Solr Segment Info screen.

When set, the XML is described as:

```xml
<indexConfig>
  <mergePolicyFactory
class="org.apache.solr.index.AutoExpungeDeletesTieredMergePolicyFactory">
    <int name="maxMergedSegmentMB">1005</int>
    <int name="forceMergeDeletesPctAllowed">25</int>
    <bool name="mergeSingleSegments">true</bool>
  </mergePolicyFactory>
</indexConfig>
```

To extend TieredMergePolicy to support automatic removal of deletes:

1. To enable automatic removal of deletes, set the custom policy:

```sql
ALTER SEARCH INDEX CONFIG ON wiki.solr SET
indexConfig.mergePolicyFactory[@class='org.apache.solr.index.AutoExpungeDeletesTieredMergePolicyFactory'].bool[@name='mergeSingleSegments'] = true;
```

2. Set the maximum segment size in MB:

```sql
ALTER SEARCH INDEX CONFIG ON wiki.solr SET
indexConfig.mergePolicyFactory[@class='org.apache.solr.index.AutoExpungeDeletesTieredMergePolicyFactory'].int[@name='maxMergedSegmentMB'] = 1005;
```

3. Set the percentage threshold for deleting from the large segments:
Using DataStax Enterprise advanced functionality

```
ALTER SEARCH INDEX CONFIG ON wiki.solr SET
  indexConfig.mergePolicyFactory[@class='org.apache.solr.index.AutoExpungeDeletesTieredMergePolicyFactory'].int[@name='forceMergeDeletesPctAllowed'] = 25;
```

If `mergeFactor` is in the existing index config, you must drop it from the search index before you alter the table to support automatic removal of deletes:

```
ALTER SEARCH INDEX CONFIG ON wiki.solr DROP
  indexConfig.mergePolicyFactory;
```

**parallelDeleteTasks**

Regulates how many tasks are created to apply deletes during soft/hard commit in parallel. Supported for RT and NRT indexing. Specify a positive number greater than 0.

Leave `parallelDeleteTasks` at the default value, except when issues occur with write load when running a mixed read/write workload. If writes occasionally spike in utilization and negatively impact your read performance, then set this value lower.

Default: the number of available processors

### Search index schema

Search index schema reference information to use for creating and altering a search index schema:

- DataStax recommends CQL `CREATE SEARCH INDEX` and `ALTER SEARCH INDEX SCHEMA` commands.
- `dsetool (page 327)` commands can also be used to manage search indexes.

The schema defines the relationship between data in a table and a search index. See [Creating a search index with default values](/) and [Quick Start for CQL index management](/) for details and examples.

A sample search index schema XML:

**Sample XML**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
  <types>
    <fieldType class="org.apache.solr.schema.TextField" name="TextField">
      <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <filter class="solr.LowerCaseFilterFactory"/>
      </analyzer>
    </fieldType>
    <fieldType class="org.apache.solr.schema.TrieIntField" name="TrieIntField"/>
  </types>
</schema>
```
<fields>
  <field indexed="true" multiValued="false" name="grade_completed" type="TextField"/>
  <field indexed="true" multiValued="false" name="diagnosed_thyroid_disease" type="TextField"/>
  <field indexed="true" multiValued="false" name="pets" type="TextField"/>
  <field indexed="true" multiValued="false" name="secondary_smoke" type="TextField"/>
  <field indexed="true" multiValued="false" name="diagnosed_lupus" type="TextField"/>
  <field indexed="true" multiValued="false" name="gender" type="TextField"/>
  <field indexed="true" multiValued="false" name="birthplace" type="TextField"/>
  <field indexed="true" multiValued="false" name="income_group" type="TrieIntField"/>
  <field indexed="true" multiValued="false" name="marital_status" type="TextField"/>
  <field indexed="true" multiValued="false" name="age_months" type="TrieIntField"/>
  <field indexed="true" multiValued="false" name="bird" type="TextField"/>
  <field indexed="true" multiValued="false" name="hay_fever" type="TextField"/>
  <field indexed="true" multiValued="false" name="diagnosed_hay_fever" type="TextField"/>
  <field indexed="true" multiValued="false" name="routine_medical_coverage" type="TextField"/>
  <field indexed="true" multiValued="false" name="annual_income_20000" type="TextField"/>
  <field indexed="true" multiValued="false" name="exam_status" type="TextField"/>
  <field indexed="true" multiValued="false" name="other_pet" type="TextField"/>
  <field indexed="true" multiValued="false" name="diagnosed_stroke" type="TextField"/>
  <field indexed="true" multiValued="false" name="employer_paid_plan" type="TextField"/>
  <field indexed="true" multiValued="false" name="family_sequence" type="TrieIntField"/>
  <field indexed="true" multiValued="false" name="diagnosed_cataracts" type="TextField"/>
  <field indexed="true" multiValued="false" name="major_medical_coverage" type="TextField"/>
  <field indexed="true" multiValued="false" name="diagnosed_gout" type="TextField"/>
  <field indexed="true" multiValued="false" name="age_unit" type="TextField"/>
  <field indexed="true" multiValued="false" name="goiter" type="TextField"/>
  <field indexed="true" multiValued="false" name="chronic_bronchitis" type="TextField"/>
</fields>
Using DataStax Enterprise advanced functionality

```
<fieldindexed="true" multiValued="false" name="county" type="TextField"/>
<fielddocValues="true" indexed="true" multiValued="false" name="num_smokers" type="TrieIntField"/>
<fieldindexed="true" multiValued="false" name="screening_month" type="TextField"/>
<fieldindexed="true" multiValued="false" name="diagnosed_emphysema" type="TextField"/>
<fieldindexed="true" multiValued="false" name="diagnosed_other_cancer" type="TextField"/>
<fielddocValues="true" indexed="true" multiValued="false" name="id" type="TrieIntField"/>
<fieldindexed="true" multiValued="false" name="dental_coverage" type="TextField"/>
<fieldindexed="true" multiValued="false" name="health_status" type="TextField"/>
<fielddocValues="true" indexed="true" multiValued="false" name="monthly_income_total" type="TrieIntField"/>
<fieldindexed="true" multiValued="false" name="fish" type="TextField"/>
<fieldindexed="true" multiValued="false" name="dog" type="TextField"/>
<fieldindexed="true" multiValued="false" name="asthma" type="TextField"/>
<fieldindexed="true" multiValued="false" name="ethnicity" type="TextField"/>
<fielddocValues="true" indexed="true" multiValued="false" name="age" type="TrieIntField"/>
<fieldindexed="true" multiValued="false" name="diagnosed_ashtha" type="TextField"/>
<fieldindexed="true" multiValued="false" name="race_ethnicity" type="TextField"/>
<fieldindexed="true" multiValued="false" name="diagnosed_congestive_heart_failure" type="TextField"/>
<fielddocValues="true" indexed="true" multiValued="false" name="family_size" type="TrieIntField"/>
<fieldindexed="true" multiValued="false" name="race" type="TextField"/>
<fieldindexed="true" multiValued="false" name="thyroid_disease" type="TextField"/>
<fieldindexed="true" multiValued="false" name="bronchitis" type="TextField"/>
<fielddocValues="true" indexed="true" multiValued="false" name="household_size" type="TrieIntField"/>
<fieldindexed="true" multiValued="false" name="cat" type="TextField"/>
<fieldindexed="true" multiValued="false" name="diagnosed_goiter" type="TextField"/>
<fieldindexed="true" multiValued="false" name="diagnosed_skin_cancer" type="TextField"/>
<fieldindexed="true" multiValued="false" name="fips" type="TextField"/>
</fields>
<uniqueKey>(id,age)</uniqueKey>
```
dsetool search index commands

dsetool commands for DSE Search

The dsetool commands for DSE Search provide search index management.

- dsetool create_core (page 1315)
- dsetool core_indexing_status (page 1313)
- dsetool get_core_config (page 1322)
- dsetool get_core_schema (page 1324)
- dsetool index_checks (experimental) (page 1327)
- dsetool infer_solr_schema (page 1330)
- dsetool list_index_files (page 1338)
- dsetool read_resource (page 1356)
- dsetool rebuild_indexes (page 1358)
- dsetool reload_core (page 1359)
- dsetool stop_core_reindex (page 1368)
- dsetool unload_core (page 1373)
- dsetool upgrade_index_files (page 1374)
- dsetool write_resource (page 1376)

DataStax recommends using CQL commands to manage search indexes.

Configuration properties

Reference information for DSE Search configuration properties.

- Data location in cassandra.yaml (page 327)
- Scheduler settings in dse.yaml (page 328)
- Indexing settings in dse.yaml (page 328)
- Safety thresholds in cassandra.yaml (page 329)
- Inter-node communication in dse.yaml (page 330)
- Query options in dse.yaml (page 330)
- Client connections in dse.yaml (page 331)
- Performance in cassandra.yaml (page 331)
- Performance in dse.yaml (page 331)

Data location in cassandra.yaml

See Set the location of search indexes.

data_file_directories

The directory where table data is stored on disk. The database distributes data evenly across the location, subject to the granularity of the configured compaction strategy. If not set, the directory is $DSE_HOME/data/data.
 Tip: For production, DataStax recommends RAID 0 and SSDs.
Default: - /var/lib/cassandra/data

Scheduler settings in dse.yaml

Configuration options to control the scheduling and execution of indexing checks.

**ttl_index_rebuild_options**
Section of options to control the schedulers in charge of querying for and removing expired records, and the execution of the checks.

**fix_rate_period**
Time interval to check for expired data in seconds.
Default: 300

**initial_delay**
The number of seconds to delay the first TTL check to speed up start-up time.
Default: 20

**max_docs_per_batch**
The maximum number of documents to check and delete per batch by the TTL rebuild thread. All documents determined to be expired are deleted from the index during each check, to avoid memory pressure, their unique keys are retrieved and deletes issued in batches.
Default: 4096

**thread_pool_size**
The maximum number of cores that can execute TTL cleanup concurrently. Set the thread_pool_size to manage system resource consumption and prevent many search cores from executing simultaneous TTL deletes.
Default: 1

Indexing settings in dse.yaml

**solr_resource_upload_limit_mb**
Option to disable or configure the maximum file size of the search index config or schema. Resource files can be uploaded, but the search index config and schema are stored internally in the database after upload.

- 0 - disable resource uploading
- upload size - The maximum upload size limit in megabytes (MB) for a DSE Search resource file (search index config or schema).

Default: 10

**flush_max_time_per_core**
The maximum time, in minutes, to wait for the flushing of asynchronous index updates that occurs at DSE Search commit time or at flush time. Expert level knowledge is required to change this value. Always set the value reasonably high to ensure flushing completes successfully to fully sync DSE Search indexes with the database data. If the configured value is exceeded, index updates are only partially committed and the commit log is not truncated which can undermine data durability.
Note: When a timeout occurs, it usually means this node is being overloaded and cannot flush in a timely manner. Live indexing increases the time to flush asynchronous index updates.

Default: commented out (5)

**load_max_time_per_core**
The maximum time, in minutes, to wait for each DSE Search index to load on startup or create/reload operations. This advanced option should be changed only if exceptions happen during search index loading. When not set, the default is 5 minutes.
Default: commented out (5)

**enable_index_disk_failure_policy**
Whether to apply the configured disk failure policy if IOExceptions occur during index update operations.
- true - apply the configured Cassandra disk failure policy to index write failures
- false - do not apply the disk failure policy

When not set, the default is false.
Default: commented out (false)

**solr_data_dir**
The directory to store index data. By default, each DSE Search index is saved in solrconfig_data_dir/keyspace_name.table_name, or as specified by the dse.solr.data.dir system property.

Tip: See Managing the location of DSE Search data.

Default: commented out (/MyDir)

**solr_field_cache_enabled**
The Apache Lucene® field cache is deprecated. Instead, for fields that are sorted, faceted, or grouped by, set docValues="true" on the field in the search index schema. Then reload the search index and reindex. When not set, the default is false.
Default: commented out (false)

**async_bootstrap_reindex**
For DSE Search, configure whether to asynchronously reindex bootstrapped data. Default: false

- If enabled, the node joins the ring immediately after bootstrap and reindexing occurs asynchronously. Do not wait for post-bootstrap reindexing so that the node is not marked down.
- If disabled, the node joins the ring after reindexing the bootstrapped data.

Safety thresholds
Configure safety thresholds and fault tolerance for DSE Search with options in dse.yaml and cassandra.yaml.

**Safety thresholds in cassandra.yaml**
Configuration options include:

**read_request_timeout_in_ms**
Using DataStax Enterprise advanced functionality

Default: 5000. How long the coordinator waits for read operations to complete before timing it out.

Security in dse.yaml
Security options for DSE Search. See DSE Search security checklist.

solr_encryption_options
Settings to tune encryption of search indexes.

decryption_cache_offheap_allocation
Whether to allocate shared DSE Search decryption cache off JVM heap.

- true - allocate shared DSE Search decryption cache off JVM heap
- false - do not allocate shared DSE Search decryption cache off JVM heap

When not set, the default is true.
Default: commented out (true)

decryption_cache_size_in_mb
The maximum size of shared DSE Search decryption cache in megabytes (MB).
Default: commented out (256)

http_principal
The http_principal is used by the Tomcat application container to run DSE Search. The Tomcat web server uses the GSSAPI mechanism (SPNEGO) to negotiate the GSSAPI security mechanism (Kerberos). Set REALM to the name of your Kerberos realm. In the Kerberos principal, REALM must be uppercase.

Inter-node communication in dse.yaml
Inter-node communication between DSE Search nodes.

shard_transport_options
Fault tolerance option for inter-node communication between DSE Search nodes.

netty_client_request_timeout
Timeout behavior during distributed queries. The internal timeout for all search queries to prevent long running queries. The client request timeout is the maximum cumulative time (in milliseconds) that a distributed search request will wait idly for shard responses.
Default: 60000 (1 minute)

Query options in dse.yaml
Options for CQL Solr queries.

cql_solr_query_paging

- driver - Respects driver paging settings. Specifies to use Solr pagination (cursors) only when the driver uses pagination. Enabled automatically for DSE SearchAnalytics workloads.
- off - Paging is off. Ignore driver paging settings for CQL queries and use normal Solr paging unless:

  # The current workload is an analytics workload, including SearchAnalytics. SearchAnalytics nodes always use driver paging settings.
  # The cqlsh query parameter paging is set to driver.
Using DataStax Enterprise advanced functionality

Even when `cql_solr_query_paging`: off, paging is dynamically enabled with the "paging":"driver" parameter in JSON queries.

When not set, the default is off.
Default: commented out (off)

**cql_solr_query_row_timeout**
The maximum time in milliseconds to wait for each row to be read from the database during CQL Solr queries.
Default: commented out (10000 10 seconds)

**Client connections in dse.yaml**
The default IP address that the HTTP and Solr Admin interface uses to access DSE Search. See Changing Tomcat web server settings.

**native_transport_address**
When left blank, uses the configured hostname of the node. Unlike the `listen_address`, this value can be set to 0.0.0.0, but you must set the `native_transport_broadcast_address` to a value other than 0.0.0.0.

   **Note:** Set `native_transport_address` OR `native_transport_interface`, not both.

Default: localhost

**Performance in cassandra.yaml**
Decreasing the memtable space to make room for Solr caches might improve performance. See Changing the stack size and memtable space.

**memtable_heap_space_in_mb (deprecated)**

   **Notice:** This setting is deprecated. Use `memtable_space_in_mb` (page 70) instead.

The amount of on-heap memory allocated for memtables. The database uses the total of this amount and the value of `memtable_offheap_space_in_mb` to set a threshold for automatic memtable flush.
Default: *calculated 1/4 of heap size* (2048)

**Performance in dse.yaml**
Node routing options.

**node_health_options**
Node health options are always enabled.

**refresh_rate_ms**
Default: 60000

**uptime_ramp_up_period_seconds**
The amount of continuous uptime required for the node's uptime score to advance the node health score from 0 to 1 (full health), assuming there are no recent dropped mutations. The health score is a composite score based on dropped mutations and uptime.

   **Tip:** If a node is repairing after a period of downtime, you might want to increase the uptime period to the expected repair time.

Default: commented out (10800 3 hours)
dropped_mutation_window_minutes
The historic time window over which the rate of dropped mutations affect the node health score.
Default: 30

Viewing search index schema and config

Search index schema and config are stored internally in the database. When you modify a search index schema or config, the changes are pending.

Use the RELOAD SEARCH INDEX command to apply the pending changes to the active (in use) search index.

DataStax recommends using CQL to view the pending or active (in use) schema or config.

CQL shell DESCRIBE command
Use the CQL shell command DESCRIBE SEARCH INDEX to view the active and pending schema and config.

Show the active index config for wiki.solr:

```
DESCRIBE ACTIVE SEARCH INDEX CONFIG ON demo.health_data;
```

The results are shown in XML:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<config>
    <abortOnConfigurationError>${solr.abortOnConfigurationError:true}</abortOnConfigurationError>
    <luceneMatchVersion>LUCENE_6_0_0</luceneMatchVersion>
    <dseTypeMappingVersion>2</dseTypeMappingVersion>
    <directoryFactory class="solr.StandardDirectoryFactory" name="DirectoryFactory"/>
    <indexConfig>
        <rt>false</rt>
        <rtOffheapPostings>true</rtOffheapPostings>
        <useCompoundFile>false</useCompoundFile>
        <reopenReaders>true</reopenReaders>
        <deletionPolicy class="solr.SolrDeletionPolicy">
            <str name="maxCommitsToKeep">1</str>
            <str name="maxOptimizedCommitsToKeep">0</str>
        </deletionPolicy>
        <infoStream file="INFOSTREAM.txt">false</infoStream>
    </indexConfig>
    <jmx/>
    <updateHandler class="solr.DirectUpdateHandler2">
        <autoSoftCommit>
            <maxTime>10000</maxTime>
        </autoSoftCommit>
    </updateHandler>
    <query>
        <maxBooleanClauses>1024</maxBooleanClauses>
    </query>
</config>
```
Using DataStax Enterprise advanced functionality

```xml
<filterCache class="solr.SolrFilterCache" highWaterMarkMB="2048"
    lowWaterMarkMB="1024"/>
<enableLazyFieldLoading>true</enableLazyFieldLoading>
<useColdSearcher>true</useColdSearcher>
<maxWarmingSearchers>16</maxWarmingSearchers>
</query>
<requestDispatcher handleSelect="true">
    <requestParsers enableRemoteStreaming="true"
        multipartUploadLimitInKB="2048000"/>
    <httpCaching never304="true"/>
</requestDispatcher>
<requestHandler class="solr.SearchHandler" default="true"
    name="search">
    <lst name="defaults">
        <int name="rows">10</int>
    </lst>
</requestHandler>
<requestHandler class="com.datastax.bdp.search.solr.handler.component.CqlSearchHandler"
    name="solr_query">
    <lst name="defaults">
        <int name="rows">10</int>
    </lst>
</requestHandler>
<requestHandler class="solr.UpdateRequestHandler" name="/update"/>
<requestHandler class="solr.UpdateRequestHandler" name="/update/csv"
    startup="lazy"/>
<requestHandler class="solr.UpdateRequestHandler" name="/update/json"
    startup="lazy"/>
<requestHandler class="solr.FieldAnalysisRequestHandler" name="/analysis/field"
    startup="lazy"/>
<requestHandler class="solr.DocumentAnalysisRequestHandler" name="/analysis/document"
    startup="lazy"/>
<requestHandler class="solr.admin.AdminHandlers" name="/admin/"/>
<requestHandler class="solr.PingRequestHandler" name="/admin/ping">
    <lst name="invariants">
        <str name="qt">search</str>
        <str name="q">solrpingquery</str>
    </lst>
    <lst name="defaults">
        <str name="echoParams">all</str>
    </lst>
</requestHandler>
<requestHandler class="solr.DumpRequestHandler" name="/debug/dump">
    <lst name="defaults">
        <str name="echoParams">explicit</str>
        <str name="echoHandler">true</str>
    </lst>
</requestHandler>
<admin>
    <defaultQuery>*</defaultQuery>
</admin>
```
Show the pending index config:

View the pending search index config or schema before it is active. For example, to view the pending index schema for demo.health_data:

```
DESCRIBE PENDING SEARCH INDEX SCHEMA ON demo.health_data;
```

The results are shown in XML:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
  <types>
    <fieldType class="org.apache.solr.schema.TextField"
      name="TextField">
      <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <filter class="solr.LowerCaseFilterFactory"/>
      </analyzer>
    </fieldType>
    <fieldType class="org.apache.solr.schema.TrieIntField"
      name="TrieIntField"/>
  </types>
  <fields>
    <field indexed="true" multiValued="false" name="grade_completed"
      type="TextField"/>
    <field indexed="true" multiValued="false"
      name="diagnosed_thyroid_disease" type="TextField"/>
    <field indexed="true" multiValued="false" name="pets"
      type="TextField"/>
    <field indexed="true" multiValued="false" name="secondary_smoke"
      type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_lupus"
      type="TextField"/>
    <field indexed="true" multiValued="false" name="gender"
      type="TextField"/>
    <field indexed="true" multiValued="false" name="birthplace"
      type="TextField"/>
    <field docValues="true" indexed="true" multiValued="false"
      name="income_group" type="TrieIntField"/>
    <field indexed="true" multiValued="false" name="marital_status"
      type="TextField"/>
    <field docValues="true" indexed="true" multiValued="false"
      name="age_months" type="TrieIntField"/>
    <field indexed="true" multiValued="false" name="bird"
      type="TextField"/>
    <field indexed="true" multiValued="false" name="hay_fever"
      type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_hay_fever"
      type="TextField"/>
    <field indexed="true" multiValued="false"
      name="routine_medical_coverage" type="TextField"/>
  </fields>
</schema>
```
Using DataStax Enterprise advanced functionality

[field indexed="true" multiValued="false" name="annual_income_20000" type="TextField"/>
[field indexed="true" multiValued="false" name="exam_status" type="TextField"/>
[field indexed="true" multiValued="false" name="other_pet" type="TextField"/>
[field indexed="true" multiValued="false" name="diagnosed_stroke" type="TextField"/>
[field indexed="true" multiValued="false" name="employer_paid_plan" type="TextField"/>
[field docValues="true" indexed="true" multiValued="false" name="family_sequence" type="TrieIntField"/>
[field indexed="true" multiValued="false" name="diagnosed_cataracts" type="TextField"/>
[field indexed="true" multiValued="false" name="major_medical_coverage" type="TextField"/>
[field indexed="true" multiValued="false" name="diagnosed_gout" type="TextField"/>
[field indexed="true" multiValued="false" name="age_unit" type="TextField"/>
[field indexed="true" multiValued="false" name="goiter" type="TextField"/>
[field indexed="true" multiValued="false" name="chronic_bronchitis" type="TextField"/>
[field indexed="true" multiValued="false" name="county" type="TextField"/>
[field docValues="true" indexed="true" multiValued="false" name="num_smokers" type="TrieIntField"/>
[field indexed="true" multiValued="false" name="screening_month" type="TextField"/>
[field indexed="true" multiValued="false" name="diagnosed_emphysema" type="TextField"/>
[field indexed="true" multiValued="false" name="diagnosed_other_cancer" type="TextField"/>
[field docValues="true" indexed="true" multiValued="false" name="id" type="TrieIntField"/>
[field indexed="true" multiValued="false" name="dental_coverage" type="TextField"/>
[field indexed="true" multiValued="false" name="health_status" type="TextField"/>
[field docValues="true" indexed="true" multiValued="false" name="monthly_income_total" type="TrieIntField"/>
[field indexed="true" multiValued="false" name="fish" type="TextField"/>
[field indexed="true" multiValued="false" name="dog" type="TextField"/>
[field indexed="true" multiValued="false" name="asthma" type="TextField"/>
[field indexed="true" multiValued="false" name="ethnicity" type="TextField"/>
[field docValues="true" indexed="true" multiValued="false" name="age" type="TrieIntField"/>
[field indexed="true" multiValued="false" name="diagnosed_asthma" type="TextField"/>

Using DataStax Enterprise advanced functionality

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
  <types>
    <fieldType class="org.apache.solr.schema.TextField" name="TextField">
      <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <filter class="solr.LowerCaseFilterFactory"/>
      </analyzer>
    </fieldType>
    <fieldType class="org.apache.solr.schema.TrieIntField" name="TrieIntField"/>
  </types>
  <fields>
    <field indexed="true" multiValued="false" name="grade_completed" type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_thyroid_disease" type="TextField"/>
    <field indexed="true" multiValued="false" name="pets" type="TextField"/>
    <field indexed="true" multiValued="false" name="secondary_smoke" type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_lupus" type="TextField"/>
    <field indexed="true" multiValued="false" name="gender" type="TextField"/>
    <field indexed="true" multiValued="false" name="race_ethnicity" type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_congestive_heart_failure" type="TextField"/>
    <field docValues="true" indexed="true" multiValued="false" name="family_size" type="TrieIntField"/>
    <field indexed="true" multiValued="false" name="race" type="TextField"/>
    <field indexed="true" multiValued="false" name="thyroid_disease" type="TextField"/>
    <field indexed="true" multiValued="false" name="bronchitis" type="TextField"/>
    <field docValues="true" indexed="true" multiValued="false" name="household_size" type="TrieIntField"/>
    <field indexed="true" multiValued="false" name="cat" type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_goiter" type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_skin_cancer" type="TextField"/>
    <field indexed="true" multiValued="false" name="fips" type="TextField"/>
  </fields>
  <uniqueKey>(id, age)</uniqueKey>
</schema>
```
Using DataStax Enterprise advanced functionality

```xml
<field indexed="true" multiValued="false" name="birthplace" type="TextField"/>
<field indexed="true" multiValued="false" name="income_group" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="marital_status" type="TextField"/>
<field indexed="true" multiValued="false" name="age_months" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="bird" type="TextField"/>
<field indexed="true" multiValued="false" name="hay_fever" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_hay_fever" type="TextField"/>
<field indexed="true" multiValued="false" name="routine_medical_coverage" type="TextField"/>
<field indexed="true" multiValued="false" name="annual_income_20000" type="TextField"/>
<field indexed="true" multiValued="false" name="exam_status" type="TextField"/>
<field indexed="true" multiValued="false" name="other_pet" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_stroke" type="TextField"/>
<field indexed="true" multiValued="false" name="employer_paid_plan" type="TextField"/>
<field indexed="true" multiValued="false" name="family_sequence" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="diagnosed_cataracts" type="TextField"/>
<field indexed="true" multiValued="false" name="major_medical_coverage" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_gout" type="TextField"/>
<field indexed="true" multiValued="false" name="age_unit" type="TextField"/>
<field indexed="true" multiValued="false" name="goiter" type="TextField"/>
<field indexed="true" multiValued="false" name="chronic_bronchitis" type="TextField"/>
<field indexed="true" multiValued="false" name="county" type="TextField"/>
<field indexed="true" multiValued="false" name="num_smokers" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="screening_month" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_emphysema" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_other_cancer" type="TextField"/>
<field indexed="true" multiValued="false" name="id" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="dental_coverage" type="TextField"/>
```
Alternate ways to view

Other ways to view the search index schema and config in XML:

- **dsetool**
  
  View the pending (uploaded) or active (in use) schema or config.
  
  # dsetool get_core_config (page 1322)
  
  # dsetool get_core_schema (page 1324)

- **Solr Admin**
Customizing the search index schema

A search schema defines the relationship between data in a table and a search index. The schema identifies the columns to index and maps column names to Apache Solr™ types.

Schema defaults

DSE Search automatically maps the CQL column type to the corresponding Solr field type, defines the field type analyzer and filtering classes, and sets the DocValue.

Tip: If required, modify the schema using the CQL-Solr type compatibility matrix.

Table and schema definition

Fields with \texttt{indexed="true"} are indexed and stored as secondary files in Lucene so that the fields are searchable. The indexed fields are stored in the database, not in Lucene, with the exception of copy fields. Copy field destinations are not stored in the database.

Sample schema

The following example from \textit{Querying CQL collections} uses a simple primary key. The schema version attribute is the Solr version number for the schema syntax and semantics. In this example, version="1.5".

```xml
<schema name="my_search_demo" version="1.5">
  <types>
    <fieldType class="solr.StrField" multiValued="true" name="StrCollectionField"/>
    <fieldType name="string" class="solr.StrField"/>
    <fieldType name="text" class="solr.TextField" />
    <fieldType class="solr.TextField" name="textcollection" multiValued="true">
      <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
      </analyzer>
    </fieldType>
  </types>
  <fields>
    <field name="id" type="string" indexed="true"/>
    <field name="quotes" type="textcollection" indexed="true"/>
    <field name="name" type="text" indexed="true"/>
    <field name="title" type="text" indexed="true"/>
  </fields>
  <defaultSearchField>quotes</defaultSearchField>
  <uniqueKey>id</uniqueKey>
</schema>
```

DSE Search indexes the id, quotes, name, and title fields.
Mapping CQL primary keys and Solr unique keys

DSE Search supports CQL tables using simple or compound primary keys.

If the field is a compound primary key or Defining a multi-column partition key column in the database, the unique key value is enclosed parentheses. The schema for this kind of table requires a different syntax than the simple primary key:

- List each compound primary key column that appears in the CQL table in the schema as a field, just like any other column.
- Declare the unique key using the key columns enclosed in parentheses.
- Order the keys in the uniqueKey element as the keys are ordered in the CQL table.
- When using composite partition keys, do not include the extra set of parentheses in the uniqueKey.

Changing auto-generated search index settings

Using dsetool, you can customize the default settings for auto-generated search indexes by providing a YAML-formatted file with these options:

- **auto_soft_commit_max_time:**ms
  The maximum auto soft commit time in milliseconds.

- **default_query_field:**field
  The query field to use when no field is specified in queries.

- **distributed:**true | false
  Whether to distribute and apply the operation to all nodes in the local datacenter.
  - True applies the operation to all nodes in the local datacenter.
  - False applies the operation only to the node it was sent to. False works only when recovery=true.

  Default: true

  Warning: Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

- **enable_tokenized_text_copy_fields:**( true | false )
  Whether to generate tokenized text.

  Default: false

- **exclude_columns:** col1, col2, col3, ...
  A comma-separated (CSV) list of columns to exclude.

- **generate_DocValues_for_fields:**( * | field1, field2, ... )
  The fields to automatically configure DocValues in the generated search index schema. Specify '*' to add all possible fields:

  - `generate_DocValues_for_fields: '*'`

  or specify a comma-separated list of fields, for example:

  - `generate_DocValues_for_fields: uuidfield, bigintfield`
Due to SOLR-7264, setting docValues to true on a boolean field in the Solr schema does not work. A workaround for boolean docValues is to use 0 and 1 with a TrieIntField.

**generateResources=true | false**
Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with schema= and solrconfig=.

Valid values:
- true - Automatically generate search index schema and configuration resources if resources do not already exist.
- false - Default. Do not automatically generate search index resources.

**include_columns=col1, col2, col3, ...**
A comma-separated (CSV) list of columns to include. Empty = includes all columns.

**index_merge_factor:segments**
How many segments of equal size to build before merging them into a single segment.

**lenient=( true | false )**
Ignore non-supported type columns and continue to generate resources, instead of erroring out when non-supported type columns are encountered. Default: false

**resource_generation_profiles**
To minimize index size, specify a CSV list of profiles to apply while generating resources.

### Table 12: Resource generation profiles

<table>
<thead>
<tr>
<th>Profile name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceSavingAll</td>
<td>Applies spaceSavingNoJoin and spaceSavingSlowTriePrecision profiles.</td>
</tr>
<tr>
<td>spaceSavingNoJoin</td>
<td>Do not index a hidden primary key field. Prevents joins across cores.</td>
</tr>
<tr>
<td>spaceSavingSlowTriePrecision</td>
<td>Set trie fields precisionStep to '0', allowing for greater space saving but slower querying.</td>
</tr>
</tbody>
</table>

**Note:** Using spaceSavings profiles disables auto generation of DocValues.

For example:

```
resource_generation_profiles: spaceSavingNoJoin, spaceSavingSlowTriePrecision
```

**rt=true**
Whether to enable live indexing to increase indexing throughput. Enable live indexing on only one search index per cluster.

```
rt=true
```
CQL index management command examples

For example:

```java
CREATE SEARCH INDEX CONFIG ON wiki.solr SET
defaultQueryField='last_name';
```

See About search commands.

Using dsetool

Customize the search index config with YAML-formatted files

Create a `config.yaml` file that lists the following options to customize the config and schema files:

```yaml
default_query_field: name
auto_soft_commit_max_time: 1000
generate_DocValues_for_fields: '*'
enable_string_copy_fields: false
```

Use the dsetool command to generate the search index with these options to customize the config and schema generation. Use `coreOptions` to specify the `config.yaml` file:

```
$ dsetool create_core demo.health_data coreOptions=config.yaml
```

Customize the search index with options inline

Use the dsetool command to generate the search index and customize the schema generation. Use `coreOptions` to turn on live indexing (also called RT):

```
$ dsetool create_core udt_ks.users generateResources=true reindex=true coreOptions=rt.yaml
```

You can verify that DSE Search created the solrconfig and schema by reading core resources using dsetool.

Enable encryption for a new search index

Specify the class for `directoryFactory` to `solr.EncryptedFSDirectoryFactory` with `coreOptionsInline`:

```bash
$ dsetool create_core keyspace_name.table_name generateResources=true coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```

**Configuring additional search components**

To configure additional search components, add the search component and define it in the handler.

For example, to add the Java spelling checking package JaSpell:

```bash
$ dsetool create_core keyspace_name.table_name generateResources=true coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```
Using DataStax Enterprise advanced functionality

Configure the parameters for the request handler:

DSE Search operations

You can run DSE Search on one or more nodes. Typical operations including configuration of nodes, policies, query routing, balancing loads, and communications.

DSE Search initial data migration

Best practices and guidelines for loading data into DSE Search.

When you initially load data into DataStax Enterprise (DSE) resource contention requires planning to ensure performance.

- DSE is performant when writing data.
- Apache Solr™ is resource intensive when creating a search index.

These two activities compete for resources, so proper resource allocation is critical to maximize efficiency for initial data load.

Recommendations

- For maximum throughput, store the search index data and DataStax Enterprise (Cassandra) data on separate physical disks.
If you are unable to use separate disks, DataStax recommends that SSDs have a minimum of 500 MB/s read/write speeds (bandwidth).

- Enable OpsCenter 6.1 repair service.

Also see memory recommendations in the planning guide.

Initial bulk loading

DataStax recommends following this high-level procedure:

1. Install DSE and configure nodes for search workloads.

2. Use the CQL CREATE SEARCH INDEX command to create search indexes.

3. Tune the index for maximum indexing throughput.

4. Load data into the database using best practices for data loading. For example, load data with the driver with the consistency level at LOCAL_ONE (CL.LOCAL_ONE) and a sufficiently high write timeout.

   After data loading is completed, there might be lag time because indexing is asynchronous.

5. Verify the indexing QueueSize (page 354) with the IndexPool MBean. After the index queue size has receded, run this CQL query to verify that the number of records is as expected:

   ```
   SELECT count(*) FROM ks.table WHERE solr_query = '::*';
   ```

New data is automatically indexed.

Troubleshooting

If the record count does not stabilize:

- If dropped mutations exist in the nodetool tpstats (page 1130) output for some nodes, and OpsCenter repair service is not enabled, run manual repair on those nodes.
- If dropped mutations do not exist, check the system.log and the Solr validation log for indexing errors.

**Verifying indexing status**

You can check the indexing status using dsetool, the Core Admin, or the logs.

**Examples**

These examples use the demo keyspace and health_data table.

To view the indexing status for the local node:
Using DataStax Enterprise advanced functionality

```bash
$ dsetool core_indexing_status demo.health_data
```

The results are displayed:

```
[demo.health_data]: INDEXING, 38% complete, ETA 452303 milliseconds (7 minutes 32 seconds)
```

To view the indexing status for a search index on a specified node:

```bash
$ dsetool -h 200.192.10.11 core_indexing_status demo.health_data
```

To view indexing status of all search indexes in the data center:

```bash
$ dsetool core_indexing_status demo.health_data --all
```

The results are displayed for 3 nodes in the data center:

<table>
<thead>
<tr>
<th>Address</th>
<th>Core Indexing Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>200.192.10.11</td>
<td>FINISHED</td>
</tr>
<tr>
<td>200.192.10.12</td>
<td>FINISHED</td>
</tr>
<tr>
<td>200.192.10.23</td>
<td>FINISHED</td>
</tr>
</tbody>
</table>

Checking the indexing status using the Core Admin

To check the indexing status, open the Solr Admin and click Core Admin.

![Solr Admin](image.png)
Using DataStax Enterprise advanced functionality

Checking the indexing status using the logs

You can also check the logs to get the indexing status. For example, you can check information about the plugin initializer:

```
INDEXING / REINDEXING -
INFO SolrSecondaryIndex plugin initializer. 2013-08-26 19:25:43,347
SolrSecondaryIndex.java (line 403) Reindexing 439171 keys for core wiki.solsr
```

Or you can check the `SecondaryIndexManager.java` information:

```
INFO Thread-38 2013-08-26 19:31:28,498 SecondaryIndexManager.java (line 136) Submitting index build of wiki.solr for data in SSTableReader(path='/mnt/cassandra/data/wiki/solr/wiki-solr-ic-5-Data.db'), SSTableReader(path='/mnt/cassandra/data/wiki/solr/wiki-solr-ic-6-Data.db')
FINISH INDEXING -
INFO Thread-38 2013-08-26 19:38:10,701 SecondaryIndexManager.java (line 156) Index build of wiki.solr complete
```

Restoring a search node from backup

Reload data and rebuild the indexes as we load data.

1. Use the DataStax Enterprise restore steps with indexing enabled and let the data write as data is coming in.

   Use the OpsCenter Backup Service.

2. Follow the steps in DSE Search initial data migration (page 343).

Monitoring DSE Search

DataStax Enterprise exposes a number of statistics and management operations via Java Management Extensions (JMX). JMX is a Java technology that supplies tools for managing and monitoring Java applications and services. Any statistic or operation that a Java application has exposed as an MBean can then be monitored or manipulated using JMX.

Metrics MBeans are used for troubleshooting, tuning performance and consistency issues.

Use the IndexPool JMX MBean (page 353) to view the progress of indexing tasks.

The following paths identify the MBeans:

```
type=search,index=search_index,name=CommitMetrics (page 346)
type=search,index=search_index,name=MergeMetrics (page 356)
type=search,index=search_index,name=QueryMetrics (page 357)
type=search,index=search_index,name=UpdateMetrics (page 359)
```
where search_index is the name of the search index that is referenced by the metrics.

For example, the following figure shows the com.datastax.bdp merge metrics MBean in JConsole. The wiki.solr search index under search is expanded.

**Using MBeans to evaluate performance**

To use the MBeans on Linux to obtain information about performance on a DSE Search node:

1. Start a single DSE Search node.

2. Find the process ID (PID) of the DSE Search node:

   ```
   $ pgrep -f dse
   368
   668
   45706
   ```

   **Tip:** To verify the PID:
Using DataStax Enterprise advanced functionality

```sh
$ pgrep -f cassandra
pgrep -f cassandra
45706
```

PID 45706 is the correct PID for dse and cassandra.

3. Start JConsole using the PID of the DSE Search node:

```sh
$ jconsole 45706
```

4. In JConsole, connect to a DSE Search node. For example, connect to the Local Process `com.datastax.bdp.DseModule`.

**MBeans search demo**

Use MBeans to evaluate performance for the search demo.

1. Complete the steps to use the MBeans (page 346) on Linux to obtain information about performance.

2. Change to the `demos` directory.

   The default location of the `demos` directory depends on the type of installation:
   - **Package installations:** `/usr/share/dse/demos`
   - **Tarball installations:** `installation_location/demos`


4. Execute this script to create the schema:

   ```bash
   pushd resources/schema &&
   ./create-schema.sh
   ```

   where the script options are:

   **CQL table creation options**
   - `--ssl` use SSL for table creation over cqlsh

   **Solr HTTP options**
   - `--e` `CA_CERT_FILE` use HTTPS with the provided CA certificate
   - `--E` `CLIENT_CERT_FILE` use the provided client certificate
   - `--h` `HOST` hostname or IP for Solr HTTP requests
   - `--a` enable Kerberos
   - `--u` `USERNAME` Kerberos username
   - `--p` `PASSWORD` Kerberos password

   The script creates the schema and posts the `solrconfig.xml` and `schema.xml` files to these locations:
Using DataStax Enterprise advanced functionality


The script then creates the search index by posting to the following location:

You can override the script defaults by specifying command line parameters:

```
$ -x schemafile.xml -t tableCreationFile.cql -r solrCofgFile.xml
 -k solrCore
```

5. Execute this script to run the benchmark:

```
```

where the script options are:

- **--clients**
  The number of client threads to create.
  Default: 1

- **--loops**
  The number of times the commands list gets executed if running sequentially or the number of commands to run if running randomly.
  Default: 1

- **--fetch**
  Fetch size for CQL pagination (disabled by default). Only the first page is retrieved.

- **--solrCore**
  Search index name to run the benchmark against.

- **--testData**
  Name of the file that contains the test data.

- **--seed**
  Value to set the random generator seed to.

- **--qps**
  Maximum number of queries per second allowed.

- **--stats**
  Specifies whether to gather statics during runtime and create a csv file with the recorded values.
  Default: false

- **--url**
  Default: http://localhost:8983

The demo creates a Search index named demo.solr and indexes 50,000 documents.
Using DataStax Enterprise advanced functionality

Example CQL commands:

```
$ ./run-benchmark.sh --url=http://localhost:8983 --
testData=resources/testCqlQuery.txt --solrCore=demo.solr
```

```
./run-benchmark.sh --url=http://localhost:8983 --
testData=resources/testCqlWrite.txt --solrCore=demo.solr
```

See /demos/solr_stress/README.txt for execution modes and sample script commands.

6. In JConsole, expand `com.datastax.bdp#search#demo.solr` to view the MBeans. The CommitMetrics and QueryMetrics MBeans are present.

7. In JConsole, in `Search#demo.solr#CommitMetrics#Operations#getLatencyPercentile`, type `EXECUTE` in the p0 text entry box and 0.95 in the p1 text entry box. Click the `getLatencyPercentile` button.

The Operation return value, 582 microseconds, appears:

![Java Monitoring & Management Console](image)

**Commit metrics MBean**

The commit metrics MBean used for troubleshooting index performance as well as data consistency issues caused by asynchronous commits between different index replicas. This MBean is also useful for fine-tuning indexing back pressure. The commit metrics...
MBean records the amount of time that is spent to execute two main phases (page 351) of a commit operation on the index.

Main operational phases

The main phases of a commit operation on the index are:

**FLUSH**
Comprising the time spent by flushing the async indexing queue.

**EXECUTE**
Comprising the time spent by actually executing the commit on the index.

Commit metrics MBean operations use the FLUSH and EXECUTE commit phase names.

Commit metrics MBean set operations
The commit metrics MBean measures latency in microseconds. You can set these commit metrics MBean operations.

- setEnabled(boolean enabled)
  Enables/disables metrics recording. Enabled by default.

- resetLatency(String phase)
  Resets latency metrics for the given commit phase.

- resetLatencies()
  Resets all latency metrics.

Commit metrics MBean get operations
The commit metrics MBean measures latency in microseconds. You can get these commit metrics MBean operations:

- isEnabled()
  Checks that metrics recording is enabled.

- getLatencyPercentile(String phase, double percentile)
  Gets a commit latency percentile by its phase.

- getRecordedLatencyCount(String phase)
  Gets the total count of recorded latency metrics by its commit phase.

- getUnrecordedLatencyCount()
Using DataStax Enterprise advanced functionality

Gets the total count of unrecorded latency values due to exceeding the maximum tracked latency, which is 10 minutes.

**EndpointStateTracker MBean**

The EndpointStateTracker MBean is identified by the following path:

```
com.datastax.bdp:name=EndpointStateTracker,type=core,name=EndpointStateTracker
```

This MBean has an attribute to blacklist a node and operations that include node health, workload, and status.

**Attributes**

**Blacklisted**

Boolean attribute to remove a node from the list of searchable nodes while it’s being diagnosed, repaired, reindexed, and verified as healthy.

Sets blacklisted status that is gossiped around the cluster and used during the replica selection phase of distributed search queries.

- **true** - forcibly rank this node below active nodes for distributed search queries
- **false** - make this node eligible for selection during distributed search queries.

**ServerID**

String that identifies the server ID of a local node.

**Operations**

The arguments for the operations are strings for the IP address, except where noted.

- **getNodeHealth**
  
  Gets the node health for a given IP address.

- **getWorkloads**
  
  Gets the workload type of a remote endpoint. Persists between restarts.

- **getDatacenter**
  
  Gets the datacenter for the given endpoint, basing on the information from the Gossiper or information saved in the Cassandra system table. Persists between restarts.

- **getActiveStatus**

  Gets active status for the given endpoint. A node is active if the server and required plugins are all started. Computed at runtime.

- **getServerId**
Using DataStax Enterprise advanced functionality

- **getCoreIndexingStatus**
  Gets the dynamic indexing status (INDEXING, FINISHED, or FAILED) of the search index of a given endpoint. Computed at runtime.

- **getRing**
  Takes a single argument, the keyspace. Returns information about every node in the cluster. Computed at runtime.

- **getIsGraphServer**
  Returns true if graph is enabled for the given endpoint. Computed at runtime.

- **vnodesEnabled**
  Returns true if vnodes are enabled. Computed at runtime.

- **getBlacklistedStatus**
  Is node removed from node from the list of searchable nodes. Persists between restarts.

  **Note:** The gossip state is persisted locally. Set the Blacklisted attribute to remove the blacklisting status.

  You can also use the nodetool sjk command to blacklist a node.

### IndexPool MBean

The IndexPool MBean exposes metrics around the progress of indexing tasks as they move through the pipeline, and provides a mechanism to tweak the flushing, concurrency, and back-pressure behavior of a core indexing thread pool.

The index pool MBean is useful for controlling task submission and flush with these properties:

**Configurable concurrency**
- The maximum number of concurrent workers is predefined at construction time. The actual concurrency can be dynamically configured between 1 (synchronous execution) and the given max concurrency.

**Flow control via back pressure**
- To reduce memory consumption in case of fast producers, back pressure throttles incoming tasks. Back-pressure is applied directly as a result of the size of the global RAM buffer.

**Path**

The index pool MBean is identified by the following path:
Using DataStax Enterprise advanced functionality

where:

- search is the Mbean type
- keyspace_name.table_name is the search index (core) that the metrics reference
- IndexPool is the MBean name

For example:

IndexPool MBean attributes that you can modify

The attributes are effective only until the node is restarted. To make the change permanent, you must change the corresponding option in dse.yaml.

**FlushMaxTime**

The maximum time, in milliseconds, to wait before flushing asynchronous index updates, which occurs at DSE Search commit time or at database flush time. In dse.yaml (in minutes): `flush_max_time_per_core` (page 121).

**Concurrency**

The maximum number of concurrent asynchronous indexing threads.

IndexPool MBean view-only attributes

You can get the following MBean operations:

**BackPressurePauseNanos**

Get the average back pressure pause.

**IncomingRate**

Get the 1-minute rate of ingested tasks per second.

**MaxConcurrency**

Get the predefined max concurrency level.

**OutgoingRate**

Get the 1-minute rate of processed tasks per second.

**ProcessedTasks**

Get the total number of processed tasks for all workers.

**QueueSize**

Get the current size of each processing queues.

**TaskProcessingTimeNanos**

Get the last processing time for all workers. Could be 0 in case the clock resolution is too coarse.

**Throughput**

The 1-minute rate of work throughput per second.

**TotalQueueSize**

Get the total size of all processing queues.

**Reindex metrics MBean**

The reindex metrics describe the current state and history of reindexing tasks. Requesting a core reindex with `dsetool reload_core` (page 1359) or using a CQL `REBUILD SEARCH`
**INDEX** command will no longer build up a queue of reindexing tasks on a node. Instead, a reindexing task is started that handles all reindex requests already submitted to that node.

Reindex metrics MBean is registered for each core with metricType=ReindexMetrics

**Path**

The reindexing MBean is identified by the following path:

```
com.datastax.bdp.search.keyspace_name.table_name.ReindexMetrics
```

where:

- search is the Mbean type
- keyspace_name.table_name is the search index (core) that the metrics reference
- ReindexMetrics is the MBean name

**Available metrics are:**

**Table 13:**

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Allowed values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progress Percentage</td>
<td>0-100</td>
<td>Percent progress, where a value of 100 indicates that reindexing is not running.</td>
</tr>
<tr>
<td>Reindexing</td>
<td>True/False</td>
<td>Indicates if a search core is being reindexed.</td>
</tr>
<tr>
<td>Current reindexing reason</td>
<td>NONE, BOOTSTRAP, NEW_SSTABLES, USER_REQUEST</td>
<td>Reason for current reindexing.</td>
</tr>
<tr>
<td>Reindexing Pending</td>
<td>True/False</td>
<td>Indicates if a user-requested reindex is awaiting execution.</td>
</tr>
<tr>
<td>Reindex ETA</td>
<td></td>
<td>ETA in milliseconds</td>
</tr>
<tr>
<td>Total Requests Submitted</td>
<td></td>
<td>Number of reindex requests submitted since node start-up; includes all reindex types.</td>
</tr>
<tr>
<td>Total User Requests Submitted</td>
<td></td>
<td>Number of reindex requests submitted by user since node start-up.</td>
</tr>
<tr>
<td>Total Reindexings Run</td>
<td></td>
<td>Total number of reindexing tasks executed since startup.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Allowed values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reindexings Run Due To User Request</td>
<td></td>
<td>Total number of reindexing tasks executed on behalf of a user since startup.</td>
</tr>
<tr>
<td>Reindexings Run With Index Delete</td>
<td></td>
<td>Number of reindexing tasks executed with deleteAll==true since startup.</td>
</tr>
<tr>
<td>Reindexing Time</td>
<td></td>
<td>Timer for reindexing; uses ExponentiallyDecayingReservoir for histogram.</td>
</tr>
</tbody>
</table>

**Merge metrics MBean**

The merge metrics MBean tracks the time that Apache Solr™/Apache Lucene® spend on merging segments that accumulate on disk. Segments are files that store new documents and are a self-contained index. When data is deleted, Lucene does not remove it, but instead marks documents as deleted. For example, during the merging process, Lucene copies the data from 100 segment files into a single new file. Documents that are marked deleted are not included in the new segment files. Next, Lucene removes the 100 old segment files, and the single new file holds the index on disk.

After segments are written to disk, they are immutable.

In a high throughput environment, a single segment file is rare. Typically, there are several files and Lucene runs the merge metric operation concurrently with inserts and updates of the data using a merge policy and merge schedule.

Merge operations are costly and can impact the performance of CQL queries. A huge merge operation can cause a sudden increase in query execution time.

**Main operational phases**

The main phases of a merge operation on the index are:

**INIT**

How long it takes to initialize the merge process.

**EXECUTE**

How long it takes to execute the merge process.

**WARM**

How long it takes to warm up segments to speed up cold queries.

WARM time is part of EXECUTE time: EXECUTE time = WARM time + other operations. For example, if the EXECUTE phase is 340 ms, and the WARM phase is 120 ms, then other operations account for the remaining 220 ms.

The merge metrics MBean operations are:

- `getRecordedLatencyCount`
Using DataStax Enterprise advanced functionality

- `getLatencyPercentile`
- `getAverageLatency`
- `resetLatency`
- `resetLatencies`

To get merge metrics, insert one of the phases of the merge operation and select a phase, such as EXECUTE:

**Query metrics MBean**

Use the query metrics MBean to troubleshoot query performance; tune DSE Search configuration, such as the search index schema and caches; and tune server resources, such as the JVM heap. The query metrics MBean records the amount of time spent to run several main phases (page 357) of a distributed query on the search index.

The query metrics MBean measures latency in microseconds.

To group by query, provide an additional `query.name` parameter. For example, for a search index named demo.solr with an indexed field named `type`, use this URL to provide the additional `query.name` parameter:

```
http://localhost:8983/solr/demo.solr/select/?
q=type:1&query.name=myquery
```

All metrics collected under a given query name are recorded and retrieved separately. If a query name is not provided, all metrics are recorded together.

**Main operational phases**

The main phases of a distributed query operation are:
ENQUEUE
Comprises the time spent by a query request waiting for a thread to execute.

EXECUTE
Comprises the time spent by a single shard to execute the actual index query. This value is computed on the local node executing the shard query.

RETRIEVE
Comprises the time spent by a node to retrieve a single row from the database. This value will be computed on the local node hosting the requested data.

COORDINATE
Comprises the total amount of time spent by the coordinator node to distribute the query and gather/process results from shards. This value is computed only on query coordinator nodes. Includes RETRIEVE and EXECUTE in the total.

TOTAL
Comprises the total server-side time for a search query. Includes COORDINATE and ENQUEUE in the total.

Query metrics MBean set operations

Operations are:

- setEnabled(boolean enabled)
  Enables/disables metrics recording. Enabled by default.

- isEnabled()
  Checks if metrics recording is enabled.

- getLatencyPercentile(String phase, String query, double percentile)
  Gets a query latency percentile by its query name, which is optional and can be null, and phase.

- getRecordedLatencyCount(String phase, String query)
  Gets the total count of recorded latency metrics by its query name, which is optional and can be null, and phase.

- getUnrecordedLatencyCount()
  Gets the total count of unrecorded latency values due to exceeding the maximum tracked latency, which is 10 minutes.

- resetLatency(String query)
  Resets latency metrics for the given query name, which is optional and can be null.

- resetLatencies()
  Resets all latency metrics.
Query metrics MBean attributes

Attributes are:

**Enabled**
Indicates whether metrics recording is enabled or disabled.

**EnqueuedRequestCount**
Indicates the number of client requests that are currently waiting for a query thread.

If the value of the EnqueuedRequestCount MBean increases, or stabilizes above zero for a prolonged period, then DSE Search has reached a point of maximum throughput, where additional load will only increases latencies.

To access this attribute, use the `-f` option with the `nodetool sjk` command, as shown in the following example.

```
nodetool sjk -p 7199 mx -mg -b com.datastax.bdp:type=search,index=demo.solr,name=QueryMetrics -f EnqueuedRequestCount
```

**UnrecordedLatencyCount**
The total count of unrecorded latency values due to exceeding the maximum tracked latency, which is 10 minutes.

**Update metrics MBean**

This MBean records the amount of time spent to execute an index update, split by the following main phases:

**QUEUE**
Updated for reindexing only. The time spent by the index update task into the index pool.

**PREPARE**
The time spent preparing the actual index update.

**EXECUTE**
The time spent to actually execute the index update on Apache Lucene®.

Use the update metrics MBean tune all factors that impact indexing performance, such as back pressure, indexing threads, RAM buffer size, and merge factor.

**MBean operations**
The following MBean operations are provided:

- `setEnabled(boolean enabled)`
  Enables/disables metrics recording (enabled by default).

- `isEnabled()`
Using DataStax Enterprise advanced functionality

Checks if metrics recording is enabled.

- `getLatencyPercentile(String phase, double percentile)`
  Gets a commit latency percentile by its phase.

- `getRecordedLatencyCount(String phase)`
  Gets the total count of recorded latency metrics by its phase.

- `getUnrecordedLatencyCount()`
  Gets the total count of unrecorded latency values, because exceeding the max tracked latency.

- `resetLatency(String phase)`
  Resets latency metrics for the given phase.

- `resetLatencies()`
  Resets all latency metrics.

The maximum tracked latency is 10 minutes. Latency values are in microseconds.

**Uploading the search index schema and config**

After generating or changing the search index schema (page 324) and configuration (page 317), use dsetool to upload to a DSE Search node to create a search index. You can also post additional resource files.

You can configure the maximum resource file size or disable resource upload with the DSE Search resource upload limit (page 120) option in dse.yaml.

**Note:** Using custom resources is not supported by the CQL CREATE SEARCH INDEX command.

Index resources are stored internally in the database, not in the file system. The schema and configuration resources are persisted in the solr_admin.solr_resources database table.

1. Write the schema:

   ```bash
   $ dsetool write_resource keyspace.table name=schema.xml
   file=schemaFile.xml
   ```

2. Post the configuration file:

   ```bash
   $ dsetool write_resource keyspace.table name=solrconfig.xml
   file=solrconfigFile.xml
   ```
3. Post any other resources that you might need.

   $ dsetool write_resource keyspace.table name=ResourceFile.xml file=schemaFile.xml

   You can specify a path for the resource file:

   $ dsetool write_resource keyspace.table name=ResourceFile.xml file=myPath1/myPath2/schemaFile.xml

4. To verify the resources after they are posted:

   For example:

   $ dsetool read_resource keyspace.table name=ResourceFile.xml file=myPath1/myPath2/schemaFile.xml

Solr interfaces

Accessing search indexes from Solr Admin UI (deprecated)

When DataStax Enterprise authorization is enabled, access to search indexes (cores) is restricted from the Solr Admin UI. You must grant permissions to roles of Solr Admin UI users for HTTP operations.

<table>
<thead>
<tr>
<th>Table</th>
<th>Required permissions</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>solr_admin.solr_resources</td>
<td>SELECT</td>
<td>Read a resource</td>
</tr>
<tr>
<td>solr_admin.solr_resources</td>
<td>MODIFY</td>
<td>Write a resource</td>
</tr>
<tr>
<td>core table</td>
<td>ALTER</td>
<td>Stop core reindex</td>
</tr>
<tr>
<td>core table</td>
<td>SELECT</td>
<td>Query core and all remaining admin query operations on core</td>
</tr>
<tr>
<td>core table</td>
<td>MODIFY</td>
<td>Commit and delete</td>
</tr>
</tbody>
</table>

Tip: Permissions are inherited. Granting permissions on a keyspace allows users with that role to access all tables in the keyspace.

Examples

To grant permission to read resources:

GRANT SELECT ON solr_admin.solr_resources
Using DataStax Enterprise advanced functionality

TO role_name;

## Configuring the Solr library path

The location for library files in DataStax Enterprise is not the same location as open source Apache Solr™. Contrary to the examples shown in the solrconfig.xml file that indicate support for relative paths, DSE Search does not support the relative path values that are set for the <lib> property and cannot find files in directories that are defined by the <lib> property. The workaround is to place custom code or Solr contrib modules in the Solr library directories.

The default Solr library path location depends on the type of installation:

- **Package installations:** /usr/share/dse/solr/lib
- **Tarball installations:** installation_location/resources/solr/lib

When the plugin JAR file is not in the directory that is defined by the <lib> property, attempts to deploy custom Solr libraries in DataStax Enterprise fail with java.lang.ClassNotFoundException and an error in the system.log like this:

```
ERROR [http-8983-exec-5] 2015-12-06 16:32:33,992 CoreContainer.java (line 956) Unable to create core: boogle.main
org.apache.solr.common.SolrException: Error loading class 'com.boogle.search.CustomQParserPlugin'
at org.apache.solr.core.SolrCore.(SolrCore.java:851)
at org.apache.solr.core.SolrCore.(SolrCore.java:640)
at com.datastax.bdp.search.solr.core.CassandraCoreContainer.doCreate(CassandraCoreContainer.java:675)
at com.datastax.bdp.search.solr.core.CassandraCoreContainer.create(CassandraCoreContainer.java:234)
at com.datastax.bdp.search.solr.core.SolrCoreResourceManager.createCore(SolrCoreResourceManager.java:256)
at com.datastax.bdp.search.solr.handler.admin.CassandraCoreAdminHandler.handleCreateAction(CassandraCoreAdminHandler.java:117)
...  
Caused by: org.apache.solr.common.SolrException: Error loading class 'com.boogle.search.CustomQParserPlugin'
at org.apache.solr.core.SolrResourceLoader.findClass(SolrResourceLoader.java:474)
at org.apache.solr.core.SolrResourceLoader.findClass(SolrResourceLoader.java:405)
at org.apache.solr.core.SolrCore.createInstance(SolrCore.java:541)
...  
Caused by: java.lang.ClassNotFoundException: com.boogle.search.CustomQParserPlugin
at java.net.URLClassLoader$1.run(Unknown Source)
at java.net.URLClassLoader$1.run(Unknown Source)
...  
```
Workaround

Using the class in this example with the JAR file name com.boogle.search.CustomQParserPlugin-1.0.jar, follow these steps to get the custom plugin working on all DSE Search nodes.

1. Define the parser in the search index config file:

   ```xml
   <queryParser name="myCustomQP"
     class="com.boogle.search.CustomQParserPlugin"/>
   ```

2. Place custom code or Solr contrib modules in the Solr library directories.

3. Deploy the JAR file on all DSE Search nodes in the cluster in the appropriate lib/directory.

   For example, package installations: /usr/share/dse/solr/lib/com.boogle.search.CustomQParserPlugin-1.0.jar

4. Reload the search index with the new configuration.

Using the Solr HTTP API

You can use the Solr HTTP API to query data indexed in DSE Search.

   **Note:** Solr restrictions (page 315) apply to queries.

HTTP search queries use local/internal reads and do not actuate read repair.

With only the HTTP API, define the default number of rows in the solrconfig.xml file:

```xml
<requestHandler name="search" class="solr.SearchHandler" default="true">
  <lst name="defaults">
    <int name="rows">10</int>
  </lst>
</requestHandler>
```

**Solr HTTP API example**

Assuming you performed the example of using a collection set to find the titles in the mykeyspace.mysolr table that begin with the letters succ in XML, use this URL:

```
```

The response is:

```xml
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">2</int>
  </lst>
</response>
```
Field transformer (FIT)

DataStax Enterprise (DSE) supports using a field input/output transformer (FIT) API.

A field input/output transformer, an alternative for handling update requests, is executed later than a URP at indexing time. See the DataStax Developer Blog post An Introduction to DSE Field Transformers.

Note: The DSE custom URP implementation is deprecated.

DSE custom URP provided similar functionality to the Solr URP chain, but appeared as a plugin to Solr. The classic URP is invoked when updating a document using HTTP and the custom URP is invoked when updating a table using DSE. If both classic and custom URPs are configured, the classic version is executed first. The custom URP chain and the FIT API work with CQL and HTTP updates.

Examples are provided for using the field input/output (page 364) transformer API and the deprecated custom URP (page 367).

Field input/output (FIT) transformer API

Use the field input/output transformer API as an option to the input/output transformer support in Apache Solr™. An Introduction to DSE Field Transformers provides details on the transformer classes.

DSE Search includes the released version of a plugin API for Solr updates and a plugin to the CassandraDocumentReader. The plugin API transforms data from the secondary indexing API before data is submitted. The plugin to the CassandraDocumentReader transforms the results data from the database to DSE Search.

Using the API, applications can tweak a Solr Document before it is mapped and indexed according to the schema.xml. The API is a counterpart to the input/output transformer support in Solr.

The field input transformer (FIT) requires:

- name="dse"
• A trailing Z for date field values

To use the API:

1. Define the plugin in the top level <config> element in the solrconfig.xml for a table (search core).

```xml
<config>
  ...
  <fieldInputTransformer name="dse" class="com.datastax.bdp.cassandra.index.solr.functional.BinaryFieldInputTransformer">
  </fieldInputTransformer>

  <fieldOutputTransformer name="dse" class="com.datastax.bdp.cassandra.index.solr.functional.BinaryFieldOutputTransformer">
    </fieldOutputTransformer>
  ...
</config>
```

2. Write a transformer class something like this reference implementation (page 365) to tweak the data in some way.

3. Export the class to a JAR file. You must place the JAR file in this location:
   - Tarball installations: install-location/resources/solr/lib
   - Package installations: /usr/share/dse/solr/lib

   The JAR is added to the CLASSPATH automatically.

4. Test your implementation using something like the reference implementation.

**FIT transformer class examples**

The DataStax Developer Blog provides an introduction to DSE Field Transformers.

Here are examples of field input and output transformer (FIT) classes.

**Input transformer example**

```java
package com.datastax.bdp.search.solr.functional;

import java.io.IOException;
import org.apache.commons.codec.binary.Hex;
import org.apache.commons.lang.StringUtils;
import org.apache.solr.core.SolrCore;
import com.datastax.bdp.search.solr.FieldOutputTransformer;
import org.apache.solr.schema.SchemaField;

import com.datastax.bdp.search.solr.FieldOutputTransformer;
import org.apache.solr.schema.IndexSchema;
```
public class BinaryFieldInputTransformer extends FieldInputTransformer {
    @Override
    public boolean evaluate(String field) {
        return field.equals("binary");
    }

    @Override
    public void addFieldToDocument(SolrCore core, IndexSchema schema, String key, Document doc, SchemaField fieldInfo, String fieldValue, DocumentHelper helper) throws IOException {
        try {
            byte[] raw = Hex.decodeHex(fieldValue.toCharArray());
            byte[] decomp = DSP1493Test.decompress(raw);
            String str = new String(decomp, "UTF-8");
            String[] arr = StringUtils.split(str, ",");
            String binary_name = arr[0];
            String binary_type = arr[1];
            String binary_title = arr[2];

            SchemaField binaryNameField = core.getSchema().getFieldOrNull("binary_name");
            SchemaField binaryTypeField = core.getSchema().getFieldOrNull("binary_type");
            SchemaField binaryTitleField = core.getSchema().getFieldOrNull("binary_title");

            helper.addFieldToDocument(core, core.getSchema(), key, doc, binaryNameField, binary_name);
            helper.addFieldToDocument(core, core.getSchema(), key, doc, binaryTypeField, binary_type);
            helper.addFieldToDocument(core, core.getSchema(), key, doc, binaryTitleField, binary_title);
        } catch (Exception ex) {
            throw new RuntimeException(ex);
        }
    }
}

Output transformer example

package com.datastax.bdp.search.solr.functional;
import java.io.IOException;
import org.apache.commons.lang.StringUtils;
import org.apache.lucene.index.FieldInfo;
import org.apache.lucene.index.StoredFieldVisitor;
import com.datastax.bdp.search.solr.FieldOutputTransformer;

public class BinaryFieldOutputTransformer extends FieldOutputTransformer {
    @Override
    public void binaryField(FieldInfo fieldInfo, byte[] value, StoredFieldVisitor visitor, DocumentHelper helper) throws IOException {
        byte[] bytes = DSP1493Test.decompress(value);
        String str = new String(bytes, "UTF-8");
        String[] arr = StringUtils.split(str, ",");
        String binary_name = arr[0];
        String binary_type = arr[1];
        String binary_title = arr[2];

        FieldInfo binary_name_fi = helper.getFieldInfo("binary_name");
        FieldInfo binary_type_fi = helper.getFieldInfo("binary_type");
        FieldInfo binary_title_fi = helper.getFieldInfo("binary_title");

        visitor.stringField(binary_name_fi, binary_name);
        visitor.stringField(binary_type_fi, binary_type);
        visitor.stringField(binary_title_fi, binary_title);
    }
}

Custom URP example (deprecated)

DSE Search includes the released version of a plugin API for Solr updates and a plugin to the CassandraDocumentReader. The plugin API transforms data from the secondary indexing API before data is submitted. The plugin to the CassandraDocumentReader transforms the results data from the database to DSE Search.

**Notice:** The DSE custom URP implementation is deprecated. A custom URP is almost always unnecessary. Instead, DataStax recommends using the field input/output (page 364) (FIT) transformer API instead.

Using the API, applications can tweak a search document before it is mapped and indexed according to the index schema.

The field input transformer (FIT) requires a trailing Z for date field values.

To use the API:

1. Configure the custom URP in the solrconfig.xml.
Using DataStax Enterprise advanced functionality

```xml
<dseUpdateRequestProcessorChain name="dse">
  <processor
class="com.datastax.bdp.search.solr.functional.DSEUpdateRequestProcessorFactoryExample"/>
</dseUpdateRequestProcessorChain>
```

2. Write a class to use the custom URP that extends the Solr `UpdateRequestProcessor`. For example:

```java
package com.datastax.bdp.search.solr.functional;

import java.io.IOException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.datastax.bdp.search.solr.handler.update.CassandraAddUpdateCommand;
import com.datastax.bdp.search.solr.handler.update.CassandraCommitUpdateCommand;
import com.datastax.bdp.search.solr.handler.update.AddUpdateCommand;
import com.datastax.bdp.search.solr.handler.update.CommitUpdateCommand;
import org.apache.solr.update.processor.UpdateRequestProcessor;

public class TestUpdateRequestProcessor extends UpdateRequestProcessor
{
    protected final Logger logger = LoggerFactory.getLogger(TestUpdateRequestProcessor.class);

    public TestUpdateRequestProcessor(UpdateRequestProcessor next)
    {
        super(next);
    }

    public void processAdd(AddUpdateCommand cmd) throws IOException
    {
        if (cmd instanceof CassandraAddUpdateCommand)
        {
            logger.info("Processing Cassandra-actuated document update.");
        }
        else
        {
            logger.info("Processing HTTP-based document update.");
        }
        super.processAdd(cmd);
    }

    public void processCommit(CommitUpdateCommand cmd) throws IOException
    {
    }
}
```
Using DataStax Enterprise advanced functionality

```java
{
    if (cmd instanceof CassandraCommitUpdateCommand)
    {
        logger.info("Processing DSE-actuated commit.");
    }
    else
    {
        logger.info("Processing client-actuated commit.");
    }
    super.processCommit(cmd);
}
```

3. Export the class to a JAR, and place the JAR in this location:
   - **Tarball installations:** `install-location/resources/solr/lib`
   - **Package installations:** `/usr/share/dse/solr/lib`

   The JAR is added to the CLASSPATH automatically.

4. Test your implementation. For example:

```java
package com.datastax.bdp.search.solr.functional;

import com.datastax.bdp.search.solr.handler.update.DSEUpdateProcessorFactory;
import org.apache.solr.core.SolrCore;
import org.apache.solr.update.processor.UpdateRequestProcessor;

public class DSEUpdateRequestProcessorFactoryExample extends DSEUpdateProcessorFactory
{
    SolrCore core;

    public DSEUpdateRequestProcessorFactoryExample(SolrCore core) {
        this.core = core;
    }

    public UpdateRequestProcessor getInstance(UpdateRequestProcessor next)
    {
        return new TestUpdateRequestProcessor(next);
    }
}
```

**Interface for custom field types**

DSE Search implements a CustomFieldType interface that marks Apache Solr™ custom field types and provides their actual stored field type. The custom field type stores an integer trie field as a string representing a comma separated list of integer values. When
indexed the string is split into its integer values, each one indexed as a trie integer field. This class effectively implements a multi-valued field based on its string representation.

A CustomFieldType can override this method to provide the FieldType for the binary response writer to look at when it determines whether to call the field's toObject(). This allows the binary response writer, for instance, to return java.util.Date in place of text for a CustomFieldType that extends TrieDateField.

To ensure that custom field types control their serialized value, use:

```java
public Class<? extends FieldType> getKnownType()
{
    return getClass();
}
```

See the example reference implementation.

To use the CustomFieldType interface:

1. Implement a custom field type class something like the following reference implementation.

2. Export the class to a JAR, and place the JAR in this location:
   - Package installations: `/usr/share/dse/solr/libusr/share/dse`
   - Tarball installations: `install-location/resources/solr/` `libinstallation_location/resources/dse/lib`

   The JAR is added to the CLASSPATH automatically.

Reference implementation

Here is an example of a custom field type class:

```java
package com.datastax.bdp.search.solr.functional;

import com.datastax.bdp.search.solr.CustomFieldType;
import java.util.ArrayList;
import java.util.List;
import org.apache.lucene.index.IndexableField;
import org.apache.solr.schema.FieldType;
import org.apache.solr.schema.SchemaField;
import org.apache.solr.schema.StrField;
import org.apache.solr.schema.TrieField;

public class CustomTestField extends TrieField implements CustomFieldType
{
    public CustomTestField()
    {
        this.type = TrieField.TrieTypes.INTEGER;
    }
}
```
Deleting by query

Delete by query no longer accepts wildcard queries, including queries that match all documents (for example, `<delete><query>*:*</query></delete>`). Instead, use the CQL TRUNCATE command.
Delete by query supports deleting data based on search criteria. After you issue a delete by query, documents start getting deleted immediately and deletions continue until all documents are removed. For example, you can delete the data that you inserted using this command:

```
$ curl http://localhost:8983/solr/mykeyspace.mysolr/update --data '
  <delete><query>color:red</query></delete>' -H 'Content-type:text/xml; charset=utf-8'
```

Using `&allowPartialDeletes` parameter set to false (default) prevents deletes if a node is down. Using `&allowPartialDeletes` set to true causes the delete to fail if a node is down and the delete does not meet a consistency level of quorum. Delete by queries using `*::*` are an exception to these rules. These queries issue a truncate, which requires all nodes to be up in order to succeed.

**Best practices**

DataStax recommends that queries for delete-by-query operations touch columns that are not updated. For example, a column that is not updated is one of the elements of a compound primary key.

**Delete by query problem example**

The following workflow demonstrates that not following this best practice is problematic:

- When a search coordinator receives a delete-by-query request, the request is forwarded to every node in the search datacenter.
- At each search node, the query is run locally to identify the candidates for deletion, and then the LOCAL_ONE consistency level deletes the queries for each of those candidates.
- When those database deletes are perceived at the appropriate nodes across the cluster, the records are deleted from the search index.

For example, in a certificates table, each certificate has a date of issue that is a timestamp. When a certificate is renewed, the new issue date is written to the row, and that write is propagated to all replicas. In this example, let's assume that one replica misses it. If you run a periodic delete-by-query that removes all of the certificates with issue dates older than a specified date, unintended consequences occur when the replica that just missed the write with the "certificate renewal" matches the delete query. The certificate is deleted across the entire cluster, on all datacenters making that delete unrecoverable.

**HTTP API SolrJ and other Solr clients**

Apache Solr™ clients work with DataStax Enterprise. If you have an existing Solr application, you can create a schema, then import your data and query using your existing Solr tools. The Wikipedia demo is built and queried using SolrJ. The query is done using pure Ajax. No DataStax Enterprise API is used for the demo.
DataStax has extended SolrJ to protect internal Solr communication and HTTP access using SSL. You can also use SolrJ to change the consistency level of the write in the database on the client side.

**DSE Graph**

DataStax Enterprise Graph is the first graph database fast enough to power customer facing applications. It is capable of scaling to massive datasets and executing both transactional and analytical workloads. DSE Graph incorporates all of the enterprise-class functionality found in DataStax Enterprise, including advanced security protection, built-in analytics (page 176) and enterprise search (page 312) functionality, and visual management, monitoring, and development tools including DataStax Studio.

**About DSE Graph**

DataStax Enterprise Graph is the first graph database fast enough to power customer facing applications. It is capable of scaling to massive datasets and executing both transactional and analytical workloads. DSE Graph incorporates all of the enterprise-class functionality found in DataStax Enterprise, including advanced security protection, built-in analytics (page 176) and enterprise search (page 312) functionality, and visual management, monitoring, and development tools including DataStax Studio.

What is a graph database?

A graph database is a database that uses graph structures to store data along with the data’s relationships. Graph databases use a data model that is as simple as a whiteboard drawing. Graph databases employ vertices, edges, and properties as described in Data modeling (page 434).

What is DSE Graph?

The built-for-scale architecture of the DSE database means that it is capable of handling petabytes of information and thousands of concurrent users and operations per second. DSE Graph is built on top of the DSE database, a component of DataStax Enterprise. DSE Graph provides the following benefits:

<table>
<thead>
<tr>
<th>Support for large graphs</th>
<th>Graphs stored in DSE Graph scale with the number of machines in the cluster because the DSE database provides the distributed storage layer. Graphs can contain hundreds of millions ($10^8$) of vertices and billions ($10^9$) of edges.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for very many concurrent transactions and operational graph processing (OLTP)</td>
<td>The transactional capacity of DSE Graph scales with the number of machines in the cluster and answers complex traversal queries on huge graphs in milliseconds.</td>
</tr>
</tbody>
</table>
### Support for global graph analytics and batch graph processing (OLAP)
- Support for global graph analytics and batch graph processing (OLAP) through the Spark framework.

### Integration with DSE Search
- Integrates with DSE Search for efficient indexing.

### Support for geographic, numeric range, and full text search
- Support for geographic, numeric range, and full text search for vertices and edges on large graphs.

### Native support for Apache TinkerPop
- Native support for the popular property graph data model exposed by Apache TinkerPop.

### Native support for the Gremlin query language
- Native support for the graph traversal language Gremlin.

### Integration of the Gremlin Server
- Integration with the Gremlin graph server.

### Performance tuning options
- Numerous graph-level configurations provide options for tuning performance.

### Vertex-centric indexes provide optimal querying
- Vertex-centric indexes provide vertex-level querying to alleviate issues with the super node problem.

### Optimized disk representation
- Provides an optimized disk representation to allow for efficient use of storage and speed of access.

---

### How does DSE Graph differ from Titan?

DSE Graph has higher performance than Titan for the following reasons:

- Specifically engineered for the DSE database. DSE Graph is designed to take advantage of the DSE database’s features.
- Optimized storage for graph data. DSE Graph partitions the adjacency list of high-degree vertices, storing and efficient querying of graph data with highly-skewed degree distributions.
• Dedicated index structures that make queries faster.
• Optimized distributed queries. DSE Graph intelligently routes queries to the cluster nodes most suitable for handling each query. This routing achieves higher degrees of data locality and requires moving less data around the cluster. In Titan, all query executions are local on the coordinator, which pull in all data from other cluster instances.

In addition, DSE Graph takes advantage of features of DSE:
• Certified for production environments
• Advanced security features
• Integrated with Enterprise Search and Analytics
• Visual management and monitoring with OpsCenter
• Visual development with DataStax Studio
• Graph support in certified DataStax drivers
• No ETL or synchronization

How is DSE Graph different from other graph databases?
DSE Graph utilizes the DSE database as a storage backend, so the graph database is distributed, always available, and has a scale-out architecture. The data in a DSE Graph is automatically partitioned across all the nodes in a cluster. Additionally, DSE Graph has built-in support for analytics for OLAP analysis and search on graph data. Finally, all DSE components use advanced security options, so DSE Graph can be secured for sensitive data.

What is Apache TinkerPop?
Apache TinkerPop is an open source project that provides an abstraction framework used to interact with DSE Graph as well as other graph databases.

What is Gremlin?
Gremlin is the primary interface into DSE Graph. Gremlin is a graph traversal language and virtual machine developed by Apache TinkerPop. Gremlin is a functional language that enables Gremlin to naturally support imperative and declarative querying.

How do I interact with DSE Graph?
The most basic way to interact with DSE Graph is using the Gremlin console dse gremlin-console. Using the Gremlin console (page 638), you can create graph database schemas, insert and query data, plus query the database for metadata using graph traversals. Complex traversals are simple to define with Gremlin compared to SQL. If you prefer a graphical tool, use DataStax Studio. For production, DataStax supplies a number of drivers in various programming languages, which pass Gremlin statements to DSE Graph: Java, Python, C#, C/C++, Node.js, and Ruby.

DSE OpsCenter provides monitoring capability.
How can I move data to and from DSE Graph?

Use a variety of methods to insert data:

- The **DSE Graph Loader (page 484)** provides a command line utility that loads data from CSV, JSON, text files, Gryo files, and queries from JDBC-compatible databases.
- Gremlin scripts and commands in **DataStax Studio (page 378)** and the **Gremlin console (page 638)**.
- **GraphSON (page 478)** files are JSON files that can exchange graph data and metadata.
- **GraphML (page 478)** is a standard for exchanging graph data. It can exchange vertex and edge information, but metadata is limited.
- **Gryo (page 478)** is a Kryo variation, enabling the exchange of binary data.

**Important:** Best practices start with data modeling before inserting data. The paradigm shift between relational and graph databases requires careful analysis of data and data modeling before importing and querying data in a graph database. **DSE Graph data modeling (page 434)** provides information and examples.

What tools come with DSE Graph?

DSE Graph comes bundled with a number of tools:

- **DataStax Studio**, a web-based notebook for running Gremlin commands and visualizing graphs
- Gremlin Console, a shell for exploring DSE Graph
- Gremlin Server to serve remote queries
- DSE OpsCenter, a monitoring and administrative tool
- Integration with **DataStax Enterprise (DSE) Search (page 312)** and **DSE Analytics (page 176)**

What kind of hardware or cloud environment do I need to run DSE Graph?

DSE Graph runs on commodity hardware with common specifications like other DataStax Enterprise offerings. See **Planning a cluster deployment**.

**DSE Graph Terminology**

This terminology is specific to DSE Graph.

**adjacency list**
A collection of unordered lists used to represent a finite graph. Each list describes the set of neighbors of a vertex in the graph.

**adjacent vertex**
A vertex directly attached to another vertex by an edge.

**directed graph**
A set of vertices and a set of arcs (ordered pairs of vertices). In DSE Graph, the terminology "arcs" is not used, and edges are directional.

**edge**
A connection between vertices. Edges can be unordered (no directional orientation) or ordered (directional). An edge can also be described as an object that has a vertex at its tail and head.

element
   An element is a vertex, edge, or property.

global index
   An index structure over the entire graph.

graph
   A collection of vertices and edges.

graph degree
   The largest vertex degree of the graph.

graph partitioning
   A process that consists of dividing a graph into components, such that the components are of about the same size and there are few connections between the components.

graph traversal
   An algorithmic walk across the elements of a graph according to the referential structure explicit within the graph data structure.

incident edge
   An edge incident to a particular vertex, meaning that the edge and vertex touch.

index
   An index is a data structure that allows for the fast retrieval of elements by a particular key-value pair.

meta-property
   A property that describes some attribute of another property.

order
   The magnitude of the number of edges to the number of vertices.

partitioned vertex
   Used for vertices that have a very large number of edges, a partitioned vertex consists of a portion of a vertex’s data that results from dividing the vertex into smaller components for graph database storage. Experimental

property
   A key-value pair that describes some attribute of either a vertex or an edge. Property key is used to describe the key in the key-value pair. All properties are global in DSE Graph, meaning that a property can be used for any vertices. For example, “name” can be used for all vertices in a graph.

traversal source
   A domain specific language (DSL) that specifies the traversal methods used by a traversal.

undirected graph
   A set of vertices and a set of edges (unordered pairs of vertices).

vertex-centric index
   A local index structure built per vertex.

vertex
   A vertex is the fundamental unit of which graphs are formed. A vertex can also be described as an object that has incoming and outgoing edges.

vertex degree
The number of edges incident to a vertex.

**DSE Graph QuickStart**

**QuickStart Introduction**

Graph databases are useful for discovering simple and complex relationships between objects. Relationships are fundamental to how objects interact with one another and their environment. Graph databases perfectly represent the relationships between objects.

Graph databases consist of three elements:

**vertex**
A vertex is an object, such as a person, location, automobile, recipe, or anything else you can think of as nouns.

**edge**
An edge defines the relationship between two vertices. A person can create software, or an author can write a book. Typically an edge is equivalent to a verb.

**property**
A key-value pair that describes some attribute of either a vertex or an edge. A property key is used to describe the key in the key-value pair. All properties are global in DSE Graph, meaning that a property can be used for any vertices. For example, “name” can be used for all vertices in a graph.

Vertices, edges, and properties can have properties; for this reason, DSE Graph is classified as a **property graph**. The properties for elements are an important element of storing and querying information in a property graph.

Property graphs are typically quite large, although the nature of querying the graph varies depending on whether the graph has large numbers of vertices, edges, or both vertices and edges. To get started with graph database concepts, a toy graph is used for simplicity. The example used here explores the world of food.
Elements are labeled to distinguish the type of vertices and edges in a graph database using vertex labels and edge labels. A vertex labeled person holds information about an author or reviewer or someone who ate a meal. An edge between a person and a book is labeled authored. Specifying appropriate labels is an important step in graph data modeling (page 434).

Vertices and edges generally have properties. For instance, a person vertex can have properties name and gender. Edges can also have properties. A created edge can have a createDate property that identifies when the adjoining recipe vertex was created.

Information in a graph database is retrieved using graph traversals. Graph traversals walk a graph with a single or series of traversal steps from a defined starting point and filter each step until returning a result.

To retrieve information using graph traversals, you must first insert data. The steps listed in this section allow you to gain a rudimentary understanding of DSE Graph with a minimum amount of configuration and schema creation.

QuickStart Installation

1. Install DataStax Enterprise.
2. Start DataStax Enterprise with DSE Graph enabled (page 1437).

3. Start either DataStax Studio (page 380) or Gremlin console (page 380):
   a. Install DataStax Studio and start Studio.
   b. Start the Gremlin Console.

   $ bin/dse gremlin-console

   \,\,\,/
   (o o)
   ----oOo-(3)-oOo-----plugin activated:
   tinkerpop.tinkergraph
   plugin activated: tinkerpop.server
   plugin activated: tinkerpop.utilities
   ==>Connected - localhost/127.0.0.1:8182-[4edf75f9-ed27-4add-a350-172abe37f701]
   ==>Set remote timeout to 2147483647ms
   ==>All scripts will now be sent to Gremlin Server
   - [localhost/127.0.0.1:8182]-[4edf75f9-ed27-4add-a350-172abe37f701] - type ':remote console' to return to
   local mode
   gremlin>

   Gremlin console sends all commands typed at the prompt to the Gremlin Server that will process the commands. DSE Graph runs a Gremlin Server tinkerpop.server on each DSE node. Gremlin console automatically connects to the Gremlin Server.

   The Gremlin console runs in remote mode automatically, processing commands on the Gremlin server. The Gremlin console by default opens a session to run commands on the remote server. The Gremlin console can be switched to run commands locally using:

   :remote console

   All commands will need to be submitted remotely once this command is run. Using the command again will switch the context back to the Gremlin server.

**QuickStart Configuration**

1. Create a Studio notebook and configure a graph for the QuickStart. If you are using Gremlin console, skip to this step (page 381).
   a. This tutorial exists as a Studio notebook, DSE Graph QuickStart, so that you do not have to create a notebook. However, in Studio, creating a notebook is
simple. If running Studio on a DSE node, the default connection of localhost works, otherwise create a connection for the DSE cluster desired. Each notebook is connected to a particular graph. Multiple notebooks can be connected to the same graph, or multiple notebooks can be created to connect to different graphs.

A connection in Studio defines the graph and assigns a graph traversal \( g \) for that graph. A graph traversal is the mechanism for visiting each vertex in a graph, based on the filters defined in the graph traversal. To query DSE Graph, the graph traversal \( g \) must be assigned to a particular graph; Studio manages this assignment with connections.

A blank notebook opens with a single cell. DSE Graph runs a Gremlin Server \( \text{tinkerpop.server} \) on each DataStax Enterprise node. Studio automatically connects to the Gremlin Server, and if it doesn’t exist, it creates a graph using the connection information. The graph is stored as one graph instance per DSE database keyspace. Once a graph exists, a graph traversal source \( g \) is configured that allows graph traversals to be executed to query the graph. A graph traversal is bound to a specific traversal source, which by default is the standard OLTP traversal engine. The graph commands can add vertices and edges to the database, or get other graph information. The \( g \) commands can query or add vertices and edges.

b. Set the schema mode to Development and allow full scans.

```java
schema.config().option('graph.schema_mode').set('Development')
schema.config().option('graph.allow_scan').set('true')
```

**Caution:** Development is a more lenient mode that allows schema to be created automatically when adding data, and also allows full scans that can inspect the data with broad graph traversals. Full scans over large graphs will have high read latency, and are not appropriate for production applications. For production, the schema mode should be set to Production to require schema prior to inserting data and disallow full scans.

2. Create a graph in Gremlin Console and configure the graph for the QuickStart.

a. Create a graph to hold the data. The system commands are used to run commands that affect graphs in DSE Graph.

```java
system.graph('test').create()
```

```text
=>null
```

Once a graph exists, a graph traversal \( g \) is configured that will allow graph traversals to be executed. Graph traversals are used to query the graph data...
Using DataStax Enterprise advanced functionality

and return results. A graph traversal is bound to a specific traversal source which is the standard OLTP traversal engine.

b. Configure a graph traversal \( g \) to use the default graph traversal setting, which is \( \text{test.g} \). This step will also create an implicit graph object.

\[
:\text{remote config alias g test.g}
\]

\[
=>g=\text{test.g}
\]

The \textit{graph} commands allow graphs to be written to file, add vertices, properties, or edges to the database, and set or get other graph configuration. The \textit{g} commands create queries to obtain results, and can also add vertices, properties, or edges to the database.

c. Set the schema mode to Development and allow full scans.

\[
\text{schema.config().option('graph.schema_mode').set('Development')}
\]

\[
\text{schema.config().option('graph.allow_scan').set('true')}
\]

\textbf{Caution:} Development is a more lenient mode that allows schema to be created automatically when adding data, and also allows full scans that can inspect the data with broad graph traversals. Full scans over large graphs will have high read latency, and are not appropriate for production applications. For production, the schema mode should be set to Production to require schema prior to inserting data and disallow full scans.

d. When creating a new graph, to check what graphs already exist, use:

\[
\text{system.graphs()}
\]

\[
=>\text{test}
\]

\[
=>\text{anotherTest}
\]

**QuickStart Vertex and edge counting**

There are different methods for accomplishing vertex and edge counts in DSE Graph. Examples here will show how to use the Gremlin count() command either as a transactional or analytical query, and Spark SQL for analytical queries.

A transactional Gremlin query can be used to check the number of vertices that exist in the graph, and are useful for exploring small graphs. However, such a query scans the full graph, traversing every vertex, and should not be run on large graphs! If multiple DSE
nodes are configured, this traversal step intensively walks all partitions on all nodes in the
cluster that have graph data. This method is not appropriate for Production operations.

An analytical Gremlin query can be used to check the number of vertices that exist in any
graph, large or small, and are much safer for Production operations. The queries will be
written like transactional Gremlin queries, but executed with the analytic Spark engine.

Spark SQL provides another query method for counting vertices in transactional graph
traversals. If the AlwaysOnSQL service (page 257) is turned on, Studio uses the
JDBC interface to pass queries to DSE Analytics. Two tables, graphName_vertices and
graphName_edges, are automatically generated in the Spark database dse_graph for
each graph, where graphName is replaced with the graph used for the Studio connection
assigned to a Studio notebook. These tables can be queried with common Spark SQL
commands directly in Studio, or can be explored with the dse spark-sql (page 245) shell.
To learn more about using Spark SQL to query, see the Using Spark SQL to query data
(page 244) documentation.

**Transactional Gremlin count()**

- Use the traversal step `count()`; the current count will be zero, because no data
  exists yet. A graph traversal `g` is chained with `V()` to retrieve all vertices and `count()` to
  compute the number of vertices. Chaining executes sequential traversal steps in the
  most efficient order.

```
g.V().count()
```

**Analytical Gremlin count()**

- To use Gremlin console, configure the traversal to run an analytical query:

  ```
  :remote config alias g test.a
  ```

  where `test.a` denotes that the graph will be used for analytic purposes.

- To use Studio, configure the Run option to "Execute using analytic engine (Spark)"
  before running the query.

- Use the traversal step `count()`; the current count will be zero, because no data
  exists yet. A graph traversal `g` is chained with `V()` to retrieve all vertices and `count()` to
  compute the number of vertices. Chaining executes sequential traversal steps in the
  most efficient order.
Using DataStax Enterprise advanced functionality

```vbnet
g.V().count()
```

Spark SQL count

- Enable AlwaysOn SQL or start a Spark SQL Thrift server instance.

To use Spark SQL in Studio, enable AlwaysOn SQL service in the dse.yaml file by setting the option to `true` and restart DSE:

```yaml
# AlwaysOn SQL options
alwayson_sql_options:
  # If it's true, the node is enabled for AlwaysOn SQL. Only Analytics node
  # can be enabled as a AlwaysOn SQL node
  enabled: true
```

In a Studio cell, select Spark SQL in the language menu in a cell and set the database to `dse_graph`.

To use the Spark SQL shell, start the shell:

```
$ dse spark-sql
```

and navigate to the correct database:

```
USE dse_graph;
```

- Then, in either Studio or the Spark SQL shell, execute the Spark SQL query for finding the vertex count:

```
SELECT count(*) FROM DSE_GRAPH_QUICKSTART_vertices;
```

Edge counts

- To do an edge count with Gremlin, replace `V()` with `E()`:

```
g.E().count()
```

- To do an edge count with Spark SQL, replace the word `vertices` in the table name with `edges`:
QuickStart Simple example

Let's start with a simple example from the recipe data model. The data is composed of two vertices, one person who is an author (Julia Child) and one book (The Art of French Cooking, Vol. I) with an edge between them to identify that Julia Child authored that book. Although we could make this graph without schema, and DSE Graph would make a best guess about the data types, we'll supply schema before inserting the graph data.

Next `graph.addVertex` is used to add data for a single vertex. Note the use of label to designate the vertex label. A `g.addV` statement could also be used, as shown in the alternate method.

Run the command and look at the results using the buttons to display the Raw JSON, Table, and Graph views

1. Schema is defined for properties `personId`, `name`, and `gender`. Properties should be created first, before vertex labels. A vertex label `person` identifies a partitionKey `personId` using an user-defined vertex id with a single partitionKey; `personId` is an integer for simplicity in this example. The schema to add the partitionKey and properties are executed with two statements, but could be executed as a single chained statement.

```java
schema.propertyKey('personId').Int().single().create()
schema.propertyKey('name').Text().single().create()
schema.propertyKey('gender').Text().single().create()
schema.vertexLabel('person').partitionKey('personId').create()
schema.vertexLabel('person').properties('name','gender').add()
```

The user-defined vertex id is used to partition the graph data amongst the cluster's nodes (more information) (page 575). User-defined vertex (UDV) ids are strongly recommended, although auto-generated vertex ids are also available, but deprecated in DSE 6.0, with warnings logged when using auto-generated vertex ids. [(add a link here)](link info)

As you will see in the schema for a `book` vertex label, a property key can be reused for different types of information. While properties are “global” in the sense that they can be used with multiple vertex labels, it is important to understand that when specifying a property in a graph traversal, it is always used in conjunction with a vertex label.

2. First, insert a vertex for Julia Child using a `graph.addVertex()` command. The vertex label is `person` and two property key-value pairs are created for `name` and `gender`. Note that a label designates the key for a key-value pair that sets the vertex label.

```java
juliaChild = graph.addVertex(label,'person', 'personId', 1, 'name','Julia Child', 'gender','F')
```
Using DataStax Enterprise advanced functionality

Note that there is an alternative method of inserting the vertex with a graph traversal

\[g.addV()\]

\[juliaChild = g.addV('person').property('personId', 1).property('name', 'Julia Child').property('gender', 'F')\]

Performance tests show that the \( g.addV() \) is faster, but the \( g.addV() \) can be used in applications using DSE Drivers.

The Studio result:

**Tip:** In Studio, the result can be displayed using different views: Raw JSON, Table, or Graph. Explore the options.

The Gremlin console result:

\[==>v[\{~label=person, personId=1\}]\]

3. Create the schema for a vertex label \( book \) that has an user-defined vertex id single partitionKey \( bookId \) and includes the properties \( name \), \( publishYear \), and \( ISBN \).

\[schema.propertyKey('bookId').Int().single().create()\]
\[schema.propertyKey('publishYear').Int().single().create()\]
\[schema.propertyKey('ISBN').Text().single().create()\]
\[schema.vertexLabel('book').partitionKey('bookId').create()\]
\[schema.vertexLabel('book').properties('name','publishYear','ISBN').add()\]

4. Insert a book into the graph:


or optionally, the traversal query:


The Studio result:
As with the author vertex, you can see all the information about the book vertex created. In Graph view, use the Settings button (the gear) to change the display label for author by entering Chef {{name}}. Change the book display label with {{label}}:{{name}}. Change the book display label with {{name}}. To set graph display names more generally, look for “Configure Graph Display Names” under the three bars in the upper lefthand corner of Studio.

The Gremlin console result:

```
==> v[~label=book, bookId=1001]
```

5. Add schema for the edge between the two vertices:

```java
schema.edgeLabel('authored').multiple().create()
schema.edgeLabel('authored').connection('person', 'book').add()
```
6. The first query uses a variable `juliaChild` to hold the person vertex information, while the second query uses the variable `artOfFrenchCookingVolOne` to hold the book vertex information. The third query uses a graph traversal `g.V(firstVertex).addE(edgeLabel).to(secondVertex)` to create the edge between the author and book vertices.

```golang
juliaChild = g.V().has('person', 'personId', 1).next()
artOfFrenchCookingVolOne = g.V().has('book', 'bookId', 1001).next()
g.V(juliaChild).addE('authored').to(artOfFrenchCookingVolOne)
```
or the graph alternative:

```golang
juliaChild.addEdge('authored', artOfFrenchCookingVolOne)
```

Use **Graph view** in Studio to see the relationship. Scroll over elements to display additional information.

The Gremlin console result:

```plaintext
===>e[{~label=authored, ~out_vertex={~label=person, personId=1}, ~in_vertex={~label=book, bookId=1001}, ~local_id=5deac140-0562-11e8-a4a1-4b3271ac7767}][{~label=person, personId=1}-authored->{~label=book, bookId=1001}]
```

7. Ensure that the data inserted for the author is correct by checking with a `has()` step using the vertex label `person` and the property `name = Julia Child`. This graph traversal is a basic starting point for more complex traversals, because it narrows the search of the graph with specific information.

```golang
g.V().has('person', 'name', 'Julia Child')
```

In Studio, use the **Table view** to look at the results, as it is much more readable than the **Raw JSON view**.
The vertex information is displayed for the person vertex for Julia Child. Note the id consists of the label and the user-defined vertex id personId.

The Gremlin console result:

```groovy
g.V().hasLabel('person').valueMap()
```

Caution: Using `valueMap()` without specifying properties can result in slow query latencies, if a large number of property keys exist for the queried vertex or edge. Specific properties can be specified, such as `valueMap('name')`.

Although Spark SQL is used more for analytical queries, simple queries similar to Gremlin can be made, such as querying information about vertices. A query can look for specific columns for a specific vertex label, in this case, a person with the name Julia Child. Notice the use of backticks to escape the tilde in the column name `~label` and `name`.

```sql
SELECT personid,name,gender FROM DSE_GRAPH_QUICKSTART_vertices WHERE `~label` = 'person' AND `name` = 'Julia Child';
```
Using DataStax Enterprise advanced functionality

QuickStart Key features

A vertex label `person` specifies the type of vertex, `personId` provides a user-defined vertex id to manage cluster storage of the vertex, and the property keys `name` and `gender` display the properties for a person. Creating vertex labels (page 462) explains the id components.

1. A useful traversal is `valueMap()` which prints the key-value listing of each property value for specified vertices.

   ```
   g.V().hasLabel('person').valueMap()
   ```

   ```
   {}
   - "gender": [ "F"
     ],
   - "name": [ "Julia Child"
     ],
   - "personId": [ 1
     ]
   ```

   **Caution:** Using `valueMap()` without specifying properties can result in slow query latencies, if a large number of property keys exist for the queried vertex or edge. Specific properties can be specified, such as `valueMap('name')`.

2. If only the value of a particular property key is desired, the `values()` traversal step can be used. To get the `name` of all vertices, use:

   ```
   g.V().values('name')
   ```
3. Edge information may also be retrieved. The next command filters all edges to find those with an edge label `authored`.

```java
g.E().hasLabel('authored')
```

The Raw JSON view of the edge information displays details about the incoming and outgoing vertices as well as edge parameters `id, label, and type`.

In Gremlin console:

```java
===>e[{'~label=authored, ~out_vertex={~label=person, personId=1}, ~in_vertex={~label=book, bookId=1001}, ~local_id=5deac140-0562-11e8-a4a1-4b3271ac7767}]->{~label=person, personId=1}-authored->>{~label=book, bookId=1001}
```

4. Spark SQL can also be used to find information about edges. Notice that the Spark-generated tables display different information than the Gremlin graph query. The traversal step `count()` is useful for counting both the number of vertices and the number of edges. To count edges, use `E()` rather than `V()`. You should have one edge. The same cautions apply about real-time transactional uses in Production - Spark SQL count or OLAP execution, both analytical actions, will be a better choice!

```sql
SELECT * FROM DSE_GRAPH_QUICKSTART_edges;
```
Using DataStax Enterprise advanced functionality

QuickStart Graph schema

Before adding more data to the graph, let's stop and talk about schema. Schema defines the possible properties and their data types for the graph. These properties are then used in the definitions of vertex labels and edge labels. The last critical step in schema creation is index creation. Indexes play an important role in making graph traversals efficient and fast. See creating schema (page 447) and creating indexes (page 469) for more information.

First, let's create schema for the property keys. In the next two cells, the first command clears the schema for the previously created vertices and edge. After the schema creation is completed, the next step is to enter data for those elements again in a longer script.

Note: DSE Graph has two schema modes, Production and Development. In Production mode, all schema must be identified before data is entered. In Development mode, schema can be created or modified after data is entered.

1. Clear the schema:

   ```sql
   schema.drop()
   ```

2. To keep the Spark SQL data synchronized with the graph, drop the Spark SQL tables. The tables will be automatically rebuilt, so that the data will align with the graph schema and data entered later.

   ```sql
   DROP TABLE DSE_GRAPH_QUICKSTART_vertices;
   DROP TABLE DSE_GRAPH_QUICKSTART_edges;
   ```

Property Key schema

3. Create the new property key schema:
// ********
// PROPERTY KEYS
// ********

// SYNTAX:
// propertyKey('name').
//    type().
//    [ single() | multiple() ].
//    [ ttl ].
//    [ properties(metadata_property) ].
//    [ ifNotExists() ].
//    [ create() | add() | describe() | exists() ]
// DEFAULT IS SINGLE CARDINALITY
// ********

// SINGLE CARDINALITY PROPERTY KEY
schema.propertyKey('personId').Int().single().create()
schema.propertyKey('mealId').Int().single().create()
schema.propertyKey('itemId').Int().single().create()
schema.propertyKey('recipeId').Int().single().create()
schema.propertyKey('bookId').Int().single().create()
schema.propertyKey('ingredId').Int().single().create()
schema.propertyKey('homeId').Int().single().create()
schema.propertyKey('storeId').Int().single().create()
schema.propertyKey('locId').Text().single().create()
schema.propertyKey('stateId').Int().single().create()
schema.propertyKey('cityId').Int().single().create()
schema.propertyKey('sensorId').Int().single().create()
schema.propertyKey('name').Text().single().create()
schema.propertyKey('gender').Text().single().create()
schema.propertyKey('since').Int().single().create() // meta-property
schema.propertyKey('badge').Text().multiple().properties('since').create()

// MULTIPLE CARDINALITY PROPERTY KEY
// MULTIPLE CARDINALITY PROPERTY KEY WITH SINGLE CARDINALITY META-PROPERTY
// MULTIPLE CARDINALITY PROPERTY KEY WITH MULTIPLE CARDINALITY META-PROPERTY
Using DataStax Enterprise advanced functionality

```java
schema.propertyKey('startYear').Int().multiple().create()  // meta-property
schema.propertyKey('endYear').Int().multiple().create()  // meta-property
schema.propertyKey('country').Text().multiple().properties('startYear','endYear').create()

// EDGE PROPERTIES
schema.propertyKey('numServ').Int().single().create()
schema.propertyKey('mealDate').Date().single().create()
schema.propertyKey('useDate').Date().single().create()
schema.propertyKey('createDate').Date().single().create()
schema.propertyKey('expireDate').Date().single().create()
schema.propertyKey('stars').Int().single().create()
schema.propertyKey('time').Time().single().create()
schema.propertyKey('year').Date().single().create()
```

Each property must be defined with a data type (page 754). DSE Graph data types are aligned with the DSE database data types. By default, properties have single cardinality, but can be defined with multiple cardinality. Multiple cardinality allows more than one value to be assigned to a property.

In addition, properties can have their own properties, or meta-properties. Meta-properties can only be nested one deep, and are useful for keying information to an individual property. Notice that property keys can be created with an additional method `ifNotExists()`. This method prevents overwriting a definition that can already exist.

**Vertex label schema**

4. After property keys are created, vertex labels can be defined.

```java
// ********
// VERTEX LABELS
// ********
// SYNTAX:
// schema.vertexLabel('vertexLabel').
//  [ partitionKey(propertyKey, [ partitionKey(propertyKey) ])].
//  [ clusteringKey(propertyKey) ].
//  [ ttl ].
//  [ properties(property, property) ].
//  [ index ].
//  [ partition() ].
//  [ cache() ].
//  [ ifNotExists() ].
//  [ create() | add() | describe() | exists() ]
// ********
```

```java
// SINGLE-KEY VERTEX ID
schema.vertexLabel('person').partitionKey('personId').create()
```

```java
schema.vertexLabel('person').properties('name','nickname','gender','calGoal','macroGoal'
```

```java
schema.vertexLabel('book').partitionKey('bookId').create()
```

```java
schema.vertexLabel('book').properties('name','publishYear','ISBN','bookDiscount').add()
```
The schema for vertex labels defines the label type, and optionally defines the properties associated with the vertex label. There are two different methods for defining the association of the properties with vertex labels, either during creation, or by adding them after vertex label addition. The `ifNotExists()` method can be used for any schema creation.

**Vertex ids** should be user-defined (UDV) ids *(page 462)*, as auto-generated vertex ids *(page 462)* are deprecated in DSE 6.0. UDV ids are explained in further detail in the documentation, but note that partition keys and clustering keys may be defined.

DSE Graph limits the number of vertex labels to 200 per graph.

β

**Edge label schema**

5. After property keys are created, edge labels can be defined.

```java
// ********
// EDGE LABELS
// ********
// SYNTAX:
//schema.edgeLabel('edgeLabel').
// [ single() | multiple() ].
// [ connection( outVertex, inVertex) ].
// [ ttl ].
// [ properties(property[, property]) ].
// [ ifNotExists() ].
// [ create() | add() | describe() | exists() ]
// DEFAULT IS MULTIPLE CARDINALITY
// ********
```
Using DataStax Enterprise advanced functionality

```java
schema.edgeLabel('ate').multiple().create()
schema.edgeLabel('ate').properties('mealDate').add()
schema.edgeLabel('ate').connection('person', 'meal').add()
schema.edgeLabel('knows').multiple().create()
schema.edgeLabel('knows').properties('since').add()
schema.edgeLabel('knows').connection('person', 'person').add()
schema.edgeLabel('includes').multiple().create()
schema.edgeLabel('includes').properties('numServ').add()
schema.edgeLabel('includes').connection('meal', 'meal_item').add()
schema.edgeLabel('includedIn').multiple().create()
schema.edgeLabel('includedIn').properties('amount').add()
schema.edgeLabel('includedIn').connection('recipe', 'meal').add()
schema.edgeLabel('includedIn').connection('meal', 'book').add()
schema.edgeLabel('includedIn').connection('recipe', 'book').add()
schema.edgeLabel('includedIn').connection('ingredient', 'recipe').add()
schema.edgeLabel('created').multiple().create()
schema.edgeLabel('created').properties('createDate').add()
schema.edgeLabel('created').connection('person', 'recipe').add()
schema.edgeLabel('reviewed').multiple().create()
schema.edgeLabel('reviewed').properties('time', 'year', 'stars', 'comment').add()
schema.edgeLabel('reviewed').connection('person', 'recipe').add()
schema.edgeLabel('authored').multiple().create()
schema.edgeLabel('authored').connection('person', 'book').add()
schema.edgeLabel('contains').multiple().ttl(60800).create()
schema.edgeLabel('contains').properties('expireDate').add()
schema.edgeLabel('contains').connection('fridgeSensor', 'ingredient').add()
schema.edgeLabel('isStockedWith').multiple().ttl(60800).create()
schema.edgeLabel('isStockedWith').properties('expireDate').add()
schema.edgeLabel('isStockedWith').connection('store', 'ingredient').add()
schema.edgeLabel('isLocatedAt').multiple().create()
schema.edgeLabel('isLocatedAt').connection('home', 'location').add()
schema.edgeLabel('isLocatedAt').connection('store', 'location').add()
schema.edgeLabel('isLocatedAt').connection('fridgeSensor', 'home').add()
```

The schema for edge labels defines the label type, and defines the two vertex labels that are connected by the edge label with `connection()`. The `reviewed` edge label defines edges between adjacent vertices with the outgoing vertex label `person` and the incoming vertex label `recipe`. By default, edges have multiple cardinality (page 466), but can be defined with single cardinality. Multiple cardinality allows more than one edge with differing property values but the same edge label to be assigned.

QuickStart Indexing

Indexing (page 469) is a complex and highly important subject. Here, several types of indexes are created. Briefly, secondary and materialized indexes are two types of indexes that use the DSE database built-in indexing. Search indexes use DSE Search which is Solr-based. Only one search index per vertex label is allowed, but multiple properties can be included. Property indexes allow meta-properties to be indexed. Edge indexes allow properties on edges to be indexed. Note that indexes are added with `add()` to previously created vertex labels.
1. Create the index schema:

    // ********
    // VERTEX INDEX
    // ********
    // SYNTAX:
    // index('index_name').
    //    [secondary() | materialized() | search()].
    //    by('propertykey_name').
    //    [ asText() | asString() ].
    //    add()
    // ********

    schema.vertexLabel('person').index('byName').materialized().by('name').add()
    schema.vertexLabel('meal_item').index('byName').materialized().by('name').add()
    schema.vertexLabel('ingredient').index('byName').materialized().by('name').add()
    schema.vertexLabel('recipe').index('byCuisine').materialized().by('cuisine').add()
    schema.vertexLabel('book').index('byName').materialized().by('name').add()
    schema.vertexLabel('meal').index('byType').secondary().by('name').add()
    schema.vertexLabel('recipe').index('search').search().
        by('instructions').by('name').by('cuisine').add()
    schema.vertexLabel('book').index('search').search().by('name').by('publishYear').add()
    schema.vertexLabel('location').index('search').search().
        by('geoPoint').withError(0.000009,0.0).add()
    schema.vertexLabel('store').index('search').search().by('name').add()
    schema.vertexLabel('home').index('search').search().by('name').add()
    schema.vertexLabel('fridgeSensor').index('search').search().
        by('cityId').by('sensorId').by('name').add()

    // ********
    // EDGE INDEX
    // ********
    // SYNTAX:
    // index('index_name').
    //    [outE('edgeLabel') | inE('edgeLabel') ].
    //    by('propertykey_name').
    //    add()
    // ********

    schema.vertexLabel('recipe').index('byStars').inE('reviewed').
        by('stars').ifNotExists().add()
    schema.vertexLabel('person').index('ratedByStars').outE('reviewed').
        by('stars').ifNotExists().add()
    schema.vertexLabel('person').index('ratedByDate').outE('reviewed').
        by('year').ifNotExists().add()
    schema.vertexLabel('person').index('ratedByComments').outE('reviewed').
        by('comment').ifNotExists().add()
    schema.vertexLabel('recipe').index('byPersonOrRecipe').bothE('created').
Using DataStax Enterprise advanced functionality

```java
by('createDate').ifNotExists().add()

// ********
// PROPERTY INDEX using meta-property 'livedIn'
// ********
// SYNTAX:
// index('index_name').
// property('propertykey_name').
// by('meta-propertykey_name').
// add()
// ********
```

```java
schema.vertexLabel('person').index('byStartYear').
    property('country').by('startYear').add()

schema.vertexLabel('person').index('byEndYear').
    property('country').by('endYear').add()
```

QuickStart Inspecting schema

The `schema.describe()` query displays schema you can use to recreate the schema entered. If you enter data without creating schema, you can use this command verify the data types set for each property.

1. Examine the schema:

   ```java
   schema.describe()
   ```

   In Studio, a portion of the output:

   ![Schema Describe Output](image)

   The `schema.describe()` query displays schema you can use to recreate the schema entered. If you enter data without creating schema, you can use this command verify the data types set for each property. While entering data without schema creation is handy while developing and learning, it is strongly recommended against for actual applications. As a reminder, Production mode disallows schema creation once data is loaded.
2. Some groovy steps are useful in the Gremlin query to find specific schema descriptions. For instance, to find only the schema for vertex labels and their indexes, use the following command:

```java
schema.describe().split('\n').grep(~/.*vertexLabel.*\/)```

In Studio:

![Image of Gremlin query results]

In Gremlin console:

```gremlin
==gt schema.vertexLabel("recipe").partitionKey("recipeId").properties("name", "cuisine", "instructions", "notes").create()
==gt schema.vertexLabel("recipe").index("search").search().by("instructions").by("name").add()
==gt schema.vertexLabel("recipe").index("byStars").inE("reviewed").by("stars").add()
==gt schema.vertexLabel("recipe").index("byPersonOrRecipe").bothE("created").by("createDate").add()
==gt schema.vertexLabel("recipe").index("byStars").add()
==gt schema.vertexLabel("store").partitionKey("storeId").properties("name", "address").create()
==gt schema.vertexLabel("store").index("search").search().by("name").add()
==gt schema.vertexLabel("meal_item").partitionKey("itemId").properties("name", "servAmt", "macro", "calories").create()
==gt schema.vertexLabel("meal_item").index("byName").materialized().by("name").add()
==gt schema.vertexLabel("fridgeSensor").partitionKey("stateId", "cityId").clusteringKey("sensorId").properties("name").create()
==gt schema.vertexLabel("fridgeSensor").index("search").search().by("cityId").by("sensorId").add()
==gt schema.vertexLabel("home").partitionKey("homeId").properties("name", "address").create()
==gt schema.vertexLabel("home").index("search").search().by("name").add()
==gt schema.vertexLabel("ingredient").partitionKey("ingredId").properties("name").create()
==gt schema.vertexLabel("ingredient").index("byName").materialized().by("name").add()
```
Using DataStax Enterprise advanced functionality

```java
===> schema.vertexLabel("person").partitionKey("personId").properties("name", "nickname", "gender", "calGoal", "macroGoal", "country").create()
===> schema.vertexLabel("person").index("byName").materialized().by("name").add()
===> schema.vertexLabel("person").index("ratedByStars").outE("reviewed").by("stars").add()
===> schema.vertexLabel("person").index("ratedByDate").outE("reviewed").by("year").add()
===> schema.vertexLabel("person").index("ratedByComments").outE("reviewed").by("comment").add()
===> schema.vertexLabel("person").index("byStartYear").property("country").by("startYear").add()
===> schema.vertexLabel("person").index("byEndYear").property("country").by("endYear").add()

===> schema.vertexLabel("book").index("search").search().by("name").by("publishYear").add()
===> schema.vertexLabel("location").partitionKey("locId").properties("name", "geoPoint").create()
===> schema.vertexLabel("location").index("search").search().by("geoPoint").withError(9.0E-6, 0.0).add()
===> schema.vertexLabel("meal").partitionKey("type", "mealId").create()
===> schema.vertexLabel("meal").index("byType").secondary().by("type").add()
```

Additional steps can split the output per newline and grep for a string as shown for `index`. The Gremlin variant used here is based on Apache Groovy, so any Groovy commands can be used to manipulate graph traversals. Apache Groovy is a language that smoothly integrates with Java to provide scripting capabilities.

**QuickStart Modifying schema**

Schema can be modified after creation, using `schema add()` to add additional properties, vertex labels, edge labels, or indexes, as shown in the schema creation above. The `drop()` step can also be used to remove any element; see `propertyKey (page 657)`, `vertexLabel (page 659)`, and `edgeLabel (page 652)`. The data type of a property, however, cannot be changed, without removing and recreating the property. While entering data without schema creation is useful when developing and learning, it is strongly recommended against for actual applications. As a reminder, Production mode disallows schema creation once data is loaded.

1. Create a property to drop in the next step:

    ```java
    schema.propertyKey('nonsenseToDrop').Int().single().create()
    schema.describe().split('
').grep(~/.*nonsenseToDrop.*/)  
    ```

    In Studio:
2. Drop the property:

```
schema.propertyKey('nonsenseToDrop').drop()
schema.describe().split('\n').grep(~/.*nonsenseToDrop.*\/)```

In Studio:

```
schema.propertyKey('nonsenseToDrop').drop()
schema.describe().split('\n').grep(~/.*nonsenseToDrop.*\/)```

QuickStart Add data

Now that schema is created, add more vertices and edges using the following script. To explore more connections in the recipe data model, more vertices and edges are input into the graph. A script, `generateRecipe.groovy`, is entered and then executed by the remote Gremlin server. Note the first command, `g.V().drop().iterate();` this command can be used to drop all vertex and edge data from the graph before reading in new data. In Studio, be sure to select the **Graph view** after running the script.

Adding more data

1. Run `generateRecipe.groovy` in either Studio or the Gremlin console:

   If running in Gremlin console, use the following command to load:

   ```
   :load /tmp/generateRecipe.groovy
   ```

   replacing "/tmp" with the directory where you write the script. In Studio, run the script within a cell.
// Generates all Recipe Toy Graph vertices and edges except Reviews
// Add all vertices and edges for Recipe
g.V().drop().iterate()

// author vertices
juliaChild = graph.addVertex(label, 'person', 'personId', 1, 'name', 'Julia Child', 'gender', 'F')
simoneBeck = graph.addVertex(label, 'person', 'personId', 2, 'name', 'Simone Beck', 'gender', 'F')
louissetteBertholie = graph.addVertex(label, 'person', 'personId', 3, 'name', 'Louissette Bertholie', 'gender', 'F')
patriciaSimon = graph.addVertex(label, 'person', 'personId', 4, 'name', 'Patricia Simon', 'gender', 'F')
aliceWaters = graph.addVertex(label, 'person', 'personId', 5, 'name', 'Alice Waters', 'gender', 'F')
patriciaCurtan = graph.addVertex(label, 'person', 'personId', 6, 'name', 'Patricia Curtan', 'gender', 'F')
kelsieKerr = graph.addVertex(label, 'person', 'personId', 7, 'name', 'Kelsie Kerr', 'gender', 'F')
fritzStreiff = graph.addVertex(label, 'person', 'personId', 8, 'name', 'Fritz Streiff', 'gender', 'M')
emerilLagasse = graph.addVertex(label, 'person', 'personId', 9, 'name', 'Emeril Lagasse', 'gender', 'M')
JamesBeard = graph.addVertex(label, 'person', 'personId', 10, 'name', 'James Beard', 'gender', 'M')

// book vertices

// recipe vertices
beefBourguignon = graph.addVertex(label, 'recipe', 'recipeId', 2001, 'name', 'Beef Bourguignon', 'instructions', 'Braise the beef. Saute the onions and carrots. Add wine and cook in a dutch oven at 425 degrees for 1 hour.', 'notes', 'Takes a long time to make.')
ratatouille = graph.addVertex(label, 'recipe', 'recipeId', 2002, 'name', 'Ratatouille', 'instructions', 'Peel and cut the eggplant. Make sure you cut eggplant into lengthwise slices that are about 1-inch wmyIde, 3-inches long, and 3/8-inch thick', 'notes', 'I've made this 13 times.')
saladeNicoise = graph.addVertex(label, 'recipe', 'recipeId', 2003, 'name', 'Salade Nicoise', 'instructions', 'Take a salad bowl or platter and line it with lettuce leaves, shortly before serving. Drizzle some olive oil on the leaves and dust them with salt.', 'notes', '')
wildMushroomStroganoff = graph.addVertex(label, 'recipe', 'recipeId', 2004, 'name', 'Wild Mushroom Stroganoff', 'instructions', 'Cook the egg noodles according to the package directions and keep warm. Heat 1 1/2 tablespoons of the olive oil in a large sauté pan over medium-high heat.', 'notes', 'Good for Jan and Bill.')
spicyMeatloaf = graph.addVertex(label, 'recipe', 'recipeId', 2005, 'name', 'Spicy Meatloaf', 'instructions', 'Preheat the oven to 375 degrees F. Cook bacon in a large skillet over medium heat until very crisp and fat has rendered, 8-10 minutes.', 'notes', '')
oystersRockefeller = graph.addVertex(label, 'recipe', 'recipeId', 2006, 'name', 'Oysters Rockefeller', 'instructions', 'Sauté the shallots, celery, herbs, and seasonings in 3 tablespoons of the butter for 3 minutes. Add the watercress and let it wilt.', 'notes', '')
carrotSoup = graph.addVertex(label, 'recipe', 'recipeId', 2007, 'name', 'Carrot Soup', 'instructions', 'In a heavy-bottomed pot, melt the butter. When it starts to foam, add the onions and thyme and cook over medium-low heat until tender, about 10 minutes.', 'notes', 'Quick and easy.')
roastPorkLoin = graph.addVertex(label, 'recipe', 'recipeId', 2008, 'name', 'Roast Pork Loin', 'instructions', 'The day before, separate the meat from the ribs, stopping about 1 inch before the end of the bones. Season the pork liberally inside and out with salt and pepper and refrigerate overnight.', 'notes', 'Love this one!')

// ingredients vertices
beef = graph.addVertex(label, 'ingredient', 'ingredId', 3001, 'name', 'beef')
onion = graph.addVertex(label, 'ingredient', 'ingredId', 3002, 'name', 'onion')
mashedGarlic = graph.addVertex(label, 'ingredient', 'ingredId', 3003, 'name', 'mashed garlic')
butter = graph.addVertex(label, 'ingredient', 'ingredId', 3004, 'name', 'butter')
tomatoPaste = graph.addVertex(label, 'ingredient', 'ingredId', 3005, 'name', 'tomato paste')
egggplant = graph.addVertex(label, 'ingredient', 'ingredId', 3006, 'name', 'eggplant')
zucchini = graph.addVertex(label, 'ingredient', 'ingredId', 3007, 'name', 'zucchini')
oliveOil = graph.addVertex(label, 'ingredient', 'ingredId', 3008, 'name', 'olive oil')
yellowOnion = graph.addVertex(label, 'ingredient', 'ingredId', 3009, 'name', 'yellow onion')
greenBean = graph.addVertex(label, 'ingredient', 'ingredId', 3010, 'name', 'green beans')
Using DataStax Enterprise advanced functionality

tuna = graph.addVertex(label, 'ingredient', 'ingredId', 3011, 'name', 'tuna')
tomato = graph.addVertex(label, 'ingredient', 'ingredId', 3012, 'name', 'tomato')
hardBoiledEgg = graph.addVertex(label, 'ingredient', 'ingredId', 3013, 'name', 'hard-boiled egg')
eggNoodles = graph.addVertex(label, 'ingredient', 'ingredId', 3014, 'name', 'egg noodles')
mushroom = graph.addVertex(label, 'ingredient', 'ingredId', 3015, 'name', 'mushrooms')
bacon = graph.addVertex(label, 'ingredient', 'ingredId', 3016, 'name', 'bacon')
celery = graph.addVertex(label, 'ingredient', 'ingredId', 3017, 'name', 'celery')
greenBellPepper = graph.addVertex(label, 'ingredient', 'ingredId', 3018, 'name', 'green bell pepper')
groundBeef = graph.addVertex(label, 'ingredient', 'ingredId', 3019, 'name', 'ground beef')
porkSausage = graph.addVertex(label, 'ingredient', 'ingredId', 3020, 'name', 'pork sausage')
shallot = graph.addVertex(label, 'ingredient', 'ingredId', 3021, 'name', 'shallots')
chervil = graph.addVertex(label, 'ingredient', 'ingredId', 3022, 'name', 'chervil')
fennel = graph.addVertex(label, 'ingredient', 'ingredId', 3023, 'name', 'fennel')
parsley = graph.addVertex(label, 'ingredient', 'ingredId', 3024, 'name', 'parsley')
oyster = graph.addVertex(label, 'ingredient', 'ingredId', 3025, 'name', 'oyster')
pernod = graph.addVertex(label, 'ingredient', 'ingredId', 3026, 'name', 'Pernod')
thyme = graph.addVertex(label, 'ingredient', 'ingredId', 3027, 'name', 'thyme')
carrot = graph.addVertex(label, 'ingredient', 'ingredId', 3028, 'name', 'carrots')
chickenBroth = graph.addVertex(label, 'ingredient', 'ingredId', 3029, 'name', 'chicken broth')
porkLoin = graph.addVertex(label, 'ingredient', 'ingredId', 3030, 'name', 'pork loin')
redWine = graph.addVertex(label, 'ingredient', 'ingredId', 3031, 'name', 'red wine')

// meal vertices
meal1 = graph.addVertex(label, 'meal', 'mealId', 4001, 'type', 'lunch')
meal2 = graph.addVertex(label, 'meal', 'mealId', 4002, 'type', 'lunch')
meal3 = graph.addVertex(label, 'meal', 'mealId', 4003, 'type', 'lunch')
meal4 = graph.addVertex(label, 'meal', 'mealId', 4004, 'type', 'lunch')
meal5 = graph.addVertex(label, 'meal', 'mealId', 4005, 'type', 'breakfast')
meal6 = graph.addVertex(label, 'meal', 'mealId', 4006, 'type', 'snack')
meal7 = graph.addVertex(label, 'meal', 'mealId', 4007, 'type', 'dinner')
meal8 = graph.addVertex(label, 'meal', 'mealId', 4008, 'type', 'dinner')

// author-book edges
juliaChild.addEdge('authored', artOfFrenchCookingVolOne)
simoneBeck.addEdge('authored', artOfFrenchCookingVolOne)
louisetteBertholie.addEdge('authored', artOfFrenchCookingVolOne)
simoneBeck.addEdge('authored', simcasCuisine)
patriciaSimon.addEdge('authored', simcasCuisine)
juliaChild.addEdge('authored', frenchChefCookbook)
aliceWaters.addEdge('authored', artOfSimpleFood)
patriciaCurtan.addEdge('authored', artOfSimpleFood)
kelsieKerr.addEdge('authored', artOfSimpleFood)
fritzStreiff.addEdge('authored', artOfSimpleFood)

// author - recipe edges
juliaChild.addEdge('created', beefBourguignon, 'createDate', 1961-01-01)
juliaChild.addEdge('created', ratatouille, 'createDate', 1965-02-02)
juliaChild.addEdge('created', saladeNicoise, 'createDate', 1962-03-03)
emerilLagasse.addEdge('created', wildMushroomStroganoff, 'createDate', 2003-04-04)
emerilLagasse.addEdge('created', spicyMeatloaf, 'createDate', 2000-05-05)
aliceWaters.addEdge('created', carrotSoup, 'createDate', 1995-06-06)
aliceWaters.addEdge('created', roastPorkLoin, 'createDate', 1996-07-07)
jamesBeard.addEdge('created', oystersRockefeller, 'createDate', 1970-01-01)

// recipe - ingredient edges
beefBourguignon.addEdge('includedIn', beef, 'amount', '2 lbs')
beefBourguignon.addEdge('includedIn', onion, 'amount', '1 sliced')
beefBourguignon.addEdge('includedIn', mashedGarlic, 'amount', '2 cloves')
beefBourguignon.addEdge('includedIn', butter, 'amount', '3.5 Tbsp')
beefBourguignon.addEdge('includedIn', tomatoPaste, 'amount', '1 Tbsp')
ratatouille.addEdge('includedIn', eggplant, 'amount', '1 lb')
ratatouille.addEdge('includedIn', zucchini, 'amount', '1 lb')
ratatouille.addEdge('includedIn', mashedGarlic, 'amount', '2 cloves')
ratatouille.addEdge('includedIn', oliveOil, 'amount', '4-6 Tbsp')
ratatouille.addEdge('includedIn', yellowOnion, 'amount', '1 1/2 cups or 1/2 lb thinly sliced')
saladeNicoise.addEdge('includedIn', oliveOil, 'amount', '2-3 Tbsp')
saladeNicoise.addEdge('includedIn', greenBean, 'amount', '1 1/2 lbs blanched, trimmed')
saladeNicoise.addEdge('includedIn', tuna, 'amount', '8-10 ozs oil-packed, drained and flaked')
saladeNicoise.addEdge('includedIn', tomato, 'amount', '3 or 4 red, peeled, quartered, cored, and seasoned')
saladeNicoise.addEdge('includedIn', hardBoiledEgg, 'amount', '8 halved lengthwise')
wildMushroomStroganoff.addEdge('includedIn', eggNoodles, 'amount', '16 ozs wmyIde')
wildMushroomStroganoff.addEdge('includedIn', mushroom, 'amount', '2 lbs wild or exotic, cleaned, stemmed, and sliced')
wildMushroomStroganoff.addEdge('includedIn', yellowOnion, 'amount', '1 cup thinly sliced')
spicyMeatloaf.addEdge('includedIn', bacon, 'amount', '3 ozs diced')
spicyMeatloaf.addEdge('includedIn', onion, 'amount', '2 cups finely chopped')
spicyMeatloaf.addEdge('includedIn', celery, 'amount', '2 cups finely chopped')
spicyMeatloaf.addEdge('includedIn', greenBellPepper, 'amount', '1/4 cup finely chopped')
spicyMeatloaf.addEdge('includedIn', porkSausage, 'amount', '3/4 lbs hot')
spicyMeatloaf.addEdge('includedIn', groundBeef, 'amount', '1 1/2 lbs chuck')
oystersRockefeller.addEdge('includedIn', shallot, 'amount', '1/4 cup chopped')
oystersRockefeller.addEdge('includedIn', celery, 'amount', '1/4 cup chopped')
oystersRockefeller.addEdge('includedIn', chervil, 'amount', '1 tsp')
oystersRockefeller.addEdge('includedIn', fennel, 'amount', '1/3 cup chopped')
oystersRockefeller.addEdge('includedIn', parsley, 'amount', '1/3 cup chopped')
oystersRockefeller.addEdge('includedIn', oyster, 'amount', '2 dozen on the half shell')
oystersRockefeller.addEdge('includedIn', pernod, 'amount', '1/3 cup')
carrotSoup.addEdge('includedIn', butter, 'amount', '4 Tbsp')
carrotSoup.addEdge('includedIn', onion, 'amount', '2 medium sliced')
carrotSoup.addEdge('includedIn', thyme, 'amount', '1 sprig')
carrotSoup.addEdge('includedIn', carrot, 'amount', '2 1/2 lbs, peeled and sliced')
carrotSoup.addEdge('includedIn', chickenBroth, 'amount', '6 cups')
roastPorkLoin.addEdge('includedIn', chickenBroth, 'amount', '1 bone-in, 4-rib')
roastPorkLoin.addEdge('includedIn', redWine, 'amount', '1/2 cup')
roastPorkLoin.addEdge('includedIn', chickenBroth, 'amount', '1 cup')

// book - recipe edges
beefBourguignon.addEdge('includedIn', artOfFrenchCookingVolOne)
saladeNicoise.addEdge('includedIn', artOfFrenchCookingVolOne)
carrotSoup.addEdge('includedIn', artOfSimpleFood)

// meal - recipe edges
beefBourguignon.addEdge('includedIn', meal1)
Using DataStax Enterprise advanced functionality

```java
saladeNicoise.addEdge('includedIn', meal1)
carrotSoup.addEdge('includedIn', meal4)
roastPorkLoin.addEdge('includedIn', meal4)

// meal - book edges
meal7.addEdge('includedIn', artOfFrenchCookingVolOne)
meal8.addEdge('includedIn', artOfSimpleFood)
meal5.addEdge('includedIn', frenchChefCookbook)
g.V()
```

In Studio:

Figure 22: Data for the Recipe Toy Graph

```
In Gremlin console:

// A series of returns for vertices and edges will mark the successful completion of the script
// Sample vertex
==>v[~label=meal, type="dinner", mealId=4008]
// Sample edge
==>e[~label=includedIn, ~out_vertex={~label=meal, type="dinner", mealId=4008},
  ~in_vertex={~label=book, bookId=1004},
  ~local_id=5dec6ef7-0562-11e8-a4a1-4b3271ac7767}]
```

The `g.V()` command at the end of the script displays all the vertices created.

In Gremlin console:
2. If a vertex count is run as either a transactional query or analytical query, there is now a higher count of 61 vertices. Run the vertex count again:

```
g.V().count()
```

The DSE Graph Loader (page 484) is the recommended method for scripting data loading. Using `graph.addVertex` or `g.addV()` are only practical for small toy graphs like the recipe example.

3. Similarly, the edge count can be run, to discover the higher edge count of 67:

```
g.E().count()
```

QuickStart Exploring traversals

Exploring the graph with graph traversals can lead to interesting conclusions. Here we’ll explore a number of traversals, to show off the power of Gremlin in creating simple queries.

1. All queries can be profiled to see what the query path is and how the query performs.

```
g.V().has('person', 'name', 'Julia Child').profile()
```

In Studio:
Using DataStax Enterprise advanced functionality

Clicking on the bars in the graph in Studio will show more detail about underlying processes in the database.

In Gremlin console:

```plaintext
===> Traversal Metrics
Step      Count  Traversers Time (ms) % Dur

DsegGraphStep(vertex, [], (label = person & name ... 1 1 10.097 65.69
query-optimizer 1.848
   \_condition=((label = person & name = Julia Child) & (true))
query-setup 0.065
   \_isFitted=true
   \_isSorted=false
   \_isScan=false
index-query 1.645
   \_indexType=Materialized
   \_usesCache=false
   \_statement=SELECT "personId" FROM "DSE_GRAPH_QUICKSTART"."person_p_byName" WHERE "name" = ? LIMIT ?; with params (java.lang.String) Julia Child,
   (java.lang.Integer) 50000
   \_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async=true}
DsegPropertyLoadStep 1 1 5.274 34.31
> TOTAL - - 15.372 -
```

DSE 6.7 Developer Guide (Latest version)
In all the following queries, to investigate what happens, and why some queries are more efficient than others, try adding `.profile()` to any query will show you information similar to the information above.

2. With several `person` vertices in the graph, a specific `name` must be given to find a particular vertex. This traversal gets the stored vertex information for the vertex that has the name of Julia Child. Note that the constraint that the vertex is an `author` is also included in the `has()` clause. Graph queries will have lower latency if the query is more specific, and the `has()` step is an more tool for narrowing the search.

```plaintext
g.V().has('person', 'name', 'Julia Child')
```

Running the query in Studio will display the vertex id, label and all property values. In Gremlin console, this query will only display the vertex id, and the `valueMap()` step must be appended to get the property values.

3. In this next traversal, `has()` filters vertex properties by `name = Julia Child` as seen above. The traversal step `outE()` discovers the outgoing edges from that vertex with the `authored` label.

```plaintext
g.V().has('name', 'Julia Child').outE('authored')
```

In Studio, either the listing of the Raw JSON view edge information:

or the Graph view graph visualization where scrolling over a vertex provides additional information.
4. Spark SQL can also be used to discover information for a set of vertices or edges that match particular conditions. Here, all the edges with a `createdate` greater than May 1, 1975 are returned. Note the lack of camel case column names in Spark SQL.

```sql
SELECT * FROM DSE_GRAPH_QUICKSTART_edges WHERE createdate > '1975-05-01';
```

In Studio:

The data presented in Spark SQL is different than the data stored in the database tables for graph. In Spark SQL tables, the source and destination vertices are listed for an edge, along with the edge label and properties.

5. If instead, you want to query for the books that all people have written, the query must be modified. The previous example retrieved edges, but not the adjacent
book vertices. Add a traversal step `inV()` to find all the vertices that connect to the outgoing edges, then print the book titles of those vertices. Notice how the chained traversal steps go from the vertices along outgoing edges to the adjacent vertices with `V().outE().inV()`. The outgoing edges are given a particular filter value, *authored*.

```plaintext
g.V().outE('authored').inV().values('name')
```

6. Notice that the book titles are duplicated in the resulting list, because a listing is returned for each author. If a book has three authors, three listings are returned. The traversal step `dedup()` can eliminate the duplication.

```plaintext
g.V().outE('authored').inV().values('name').dedup()
```

7. Refine the traversal by reinserting the `has()` step for a particular author. Find all the books authored by *Julia Child*.

```plaintext
g.V().has('name','Julia Child').outE('authored').inV().values('name')
```
Using DataStax Enterprise advanced functionality

8. The previous example and this example accomplish the same result. However, the number of traversal steps and the type of traversal steps can affect performance. The traversal step `outE()` should be only used if the edges are explicitly required. In this example, the edges are traversed to get information about connected vertices, but the edge information is not important to the query.

```
g.V().has('name','Julia Child').out('authored').values('name')
```

In Studio:

and a similar listing in Gremlin console.

The traversal step `out()` retrieves the connected book vertices based on the edge label `authored` without retrieving the edge information. In a larger graph traversal, this subtle difference in the traversal can become a latency issue.

9. Additional traversal steps continue to fine-tune the results. Adding another chained `has` traversal step finds only books authored by Julia Child published after 1967. This example also displays the use of the `gt`, or `greater than` function.

```
g.V().has('name','Julia Child').out('authored').has('publishYear', gt(1967)).values('name')
```

In Studio:
Using DataStax Enterprise advanced functionality

10. When developing or testing, oftentimes checking the number of vertices with each vertex label can confirm that data was read. To find the number of vertices by vertex label, use the traversal step `label()` followed by the traversal step `groupCount()`. The step `groupCount()` is useful for aggregating results from a previous step. Although this query can be run in real-time, it is an excellent example of a query that should be run in analytic (OLAP) mode. In Studio, under the run arrow, select **Execute using analytic engine (Spark)** before running.

```
g.V().label().groupCount()
```

11. An alternative method for getting the group count with Spark SQL uses:

```
SELECT `~label` AS label, COUNT(*) AS label_count FROM DSE_GRAPH_QUICKSTART_vertices GROUP BY label;
```

**QuickStart Writing and reading data**

Writing data from DSE Graph to a file is most easily accomplished with the `graph.io()` command. The **DSE Graph Loader (page 484)** is the most appropriate tool for reading in data from files or other sources.

1. Write your data to an output file to save or exchange information. A Gryo file is a binary format file that can reload data to DSE Graph. In this next command, graph I/O writes

```
"g.V().has('name', 'Julia Child').out('authored').has('publishYear', gt(1967)).values('name')"
```
the entire graph to a file. Other file formats can be written by substituting `gryo()` with `graphml()` or `graphson()`.

```java
graph.io(gryo()).writeGraph("/tmp/recipe.gryo")
```

**Note:** `graph.io()` is disabled in sandbox mode.

**In Studio:**

```java
null
```

**In Gremlin console:**

```java
===>null
```

2. To load a Gryo file, use the `graphloader`, after creating a mapping script:

```bash
$ graphloader mappingGryo.groovy -graph recipe -address localhost
```

Details about loading Gryo data are found in Loading Gryo Data (page 517), in Using DSE Graph Loader (page 484).

### QuickStart Listing graphs

1. To discover all graphs that exist, use a system command:

```java
system.graphs()
```

2. To display all the tables within Spark SQL:

```java
SHOW TABLES FROM DSE_GRAPH_QUICKSTART;
```

### Increase your knowledge

Further adventures in traversing can be found in Creating queries using traversals (page 575). If you want to explore various loading options, check out DSE Graph Loader (page 484).
Using DataStax Enterprise advanced functionality

DataStax also hosts a DSE Graph self-paced course on DataStax Academy; register for a free account to access the course.

**DSE Graph, OLTP, and OLAP**

Online transactional processing (OLTP) is characterized by a large number of short, online transactions for very fast query processing. OLTP is typically used for data entry and retrieval with transaction-oriented applications. Online analytical processing (OLAP) is typically used to perform multidimensional analysis of data, doing complex calculations on aggregated historical data.

OLTP applications require sub-second response times, whereas OLAP applications take much longer to finish queries. Graph databases are a random access data system. In these databases, OLAP traversals do a linear scan of all vertices in the graph. Conversely, OLTP traversals are localized to a particular subgraph of the global graph. OLTP traversals leverage indexes to "jump" in to a particular vertex in the graph before starting a scan on the subgraph.

**OLTP queries**

OLTP queries are best for questions that require access to a limited subset of the entire graph. OLTP queries use filters to limit the number of vertices that will be walked to find answers. DSE Graph co-locates vertices with their edges and adjacent neighbors. When a subgraph is specified in a traversal using indexes, the number of requests to disk are reduced to locate and write the requested subgraph to memory. Once in memory, the traversal performs a link walk from vertex to vertex along the edges.

**OLAP queries**

OLAP queries are best for questions that must access a significant portion of the data stored in a graph. Using the previous method to evaluate OLAP queries will not be efficient, so a different process is used. When OLAP queries are processed, the entire graph is interpreted as a sequence of star graphs, each composed of a single vertex, along with its properties, incident edges, and the edges’ properties. The star graphs are linearly processed, jumping from one star graph to the next until all star graphs are processed and an aggregation of the discovered data is completed.

**Principles for writing graph traversals**

Understanding these underlying principles can lead to writing better graph traversals to query the graph data. A simple example illustrates the differences. Using the food graph, the query is “How many recipes has Julia Child created?”

Consider the following graph traversal:

```java
g.V().in().has('name','Julia Child').count()
```

This traversal completes the following processing:
1. Looks at all vertices.

2. Walks the incoming edges.

3. Finds the adjacent vertices that have the property key of `name` and property value of Julia Child.

4. Counts the number of vertices.

This graph traversal is a classic OLAP traversal, which must touch all vertices and does not use indexing. The count returned includes all vertices with edges to Julia Child, and not just the recipes, so as shown later, the count is incorrect and too high.

Consider the number of elements that must be traversed to complete this query. DSE Graph has profiling that aids in analyzing the traversal:

```plaintext
gremlin> g.V().in().has('name','Julia Child').count().profile()
```

---

**Traversal Metrics**

<table>
<thead>
<tr>
<th>Step</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traversers Time (ms) % Dur</td>
<td>-------</td>
</tr>
<tr>
<td>DsegGraphStep(vertex,[])</td>
<td>61</td>
</tr>
<tr>
<td>query-optimizer</td>
<td>0.563</td>
</tr>
<tr>
<td>query-setup</td>
<td>0.048</td>
</tr>
<tr>
<td>index-query</td>
<td>0.979</td>
</tr>
<tr>
<td>query-optimizer</td>
<td>0.862</td>
</tr>
</tbody>
</table>

Consider the number of elements that must be traversed to complete this query. DSE Graph has profiling that aids in analyzing the traversal:
Using DataStax Enterprise advanced functionality

```java
_options=Options{consistency=Optional[ONE],
 serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1,
 user=Optional.empty, waitForSchemaAgreement=true, async=true}
index-query
0.679

_options=Options{consistency=Optional[ONE],
 serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1,
 user=Optional.empty, waitForSchemaAgreement=true, async=true}
index-query
1.344

_options=Options{consistency=Optional[ONE],
 serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1,
 user=Optional.empty, waitForSchemaAgreement=true, async=true}
index-query
1.053

_options=Options{consistency=Optional[ONE],
 serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1,
 user=Optional.empty, waitForSchemaAgreement=true, async=true}
index-query
4.173
```

Page 418
Using DataStax Enterprise advanced functionality

```java
String _statement="SELECT "community_id", "member_id" FROM "DSEQuickStart"."reviewer_p" WHERE "~~vertex_exists" = ? LIMIT ? ALLOW FILTERING; with params (java.lang.Boolean) true, (java.lang.Integer) 50000

Options _options=newOptions(consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Optional.empty,
user=Optional.empty, waitForSchemaAgreement=true, async=true)

78 DseVertexStep(IN,vertex)
    78          95.721    61.90
query-optimizer
    0.305

(String _statement="SELECT * FROM "DSEQuickStart"."author_e" WHERE "community_id" = ? AND "member_id" = ? LIMIT ?
ALLOW FILTERING; with params (java.lang.Integer) 588941056, (java.lang.Long) 0, (java.lang.Integer) 50000

Options _options=newOptions(consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Optional.empty,
user=Optional.empty, waitForSchemaAgreement=true, async=true)

    

    


```

DSE 6.7 Developer Guide (Latest version)  Page 419
Using DataStax Enterprise advanced functionality

\_options=\{consistency=Optional[ONE],
    serialConsistency=Optional.empty, fallbackConsistency=Optional.empty,
    pagingState=null, pageSize=-1,
    user=Optional.empty, waitForSchemaAgreement=true, async=true\}
\_isPartitioned=false
\_usesIndex=false
query-setup 0.941
\_isFitted=false
\_isSorted=true
\_isScan=false
query-setup 0.015
\_isFitted=false
\_isSorted=true
\_isScan=false
vertex-query 1.966
\_usesCache=false
\_statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
    "community_id" = ? AND "member_id" = ? LIMIT ?
    ALLOW FILTERING; with params (java.lang.Integer)
    138026496, (java.lang.Long) 0, (java.lang.I
    nteger) 50000
\_options=\{consistency=Optional[ONE],
    serialConsistency=Optional.empty, fallbackConsistency=Optional.empty,
    pagingState=null, pageSize=-1,
    user=Optional.empty, waitForSchemaAgreement=true, async=true\}
\_isPartitioned=false
\_usesIndex=false
query-setup 0.015
\_isFitted=false
\_isSorted=true
\_isScan=false
query-setup 0.013
\_isFitted=false
\_isSorted=true
\_isScan=false
query-setup 0.016
\_isFitted=false
\_isSorted=true
\_isScan=false
NoOpBarrierStep(2500) 78
   25  2.877  1.86
HasStep([name.=\{(Julia Child)\}]) 5
   1  25.242  16.32
CountGlobalStep 1
   1  1.859  1.20
**Note:** The time each step takes depends on caching and other factors. For the purposes of this discussion, ignore the times reported. The `profile()` method now includes CQL commands that are executed due to Gremlin commands.

Figure 32: Studio profile output for Traversal 1

Looking at the first step, all vertices in the graph are traversed. This graph is very small, so the number of vertices is negligible compared to production graphs. In the next step, the traversal must find all incoming edges to the vertices. Again, for a small graph, the number of edges is negligible, but in production graphs, edges can number in the millions to billions. Now, the adjacent vertices are filtered for the property key information specified, narrowing the number of vertices to 6. The last two steps accomplish the count and profiling metrics.

**Specifying an edge label**

Now consider a modification to the original traversal that specifies the edge label for the incoming edges:

```cql
g.V().in('created').has('name','Julia Child').count()
===>3
```
Using DataStax Enterprise advanced functionality

This modified traversal still looks at all vertices, but in walking the incoming edges, it is limited to those that are labeled as `created`. The following profile shows an improved picture:

```sql
gremlin> g.V().in('created').has('name','Julia Child').count().profile()
```

<table>
<thead>
<tr>
<th>Step</th>
<th>Count</th>
<th>Time (ms)</th>
<th>% Dur</th>
</tr>
</thead>
<tbody>
<tr>
<td>DsegGraphStep(vertex,[])</td>
<td>61</td>
<td>22.251</td>
<td>16.91</td>
</tr>
<tr>
<td>query-optimizer</td>
<td>1.760</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_condition=((label = FridgeSensor</td>
<td>label = author</td>
<td>label = book</td>
<td>label = ingredient</td>
</tr>
<tr>
<td>query-setup</td>
<td>0.071</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_isFitted=true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_isSorted=false</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_isScan=true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index-query</td>
<td>1.139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_usesCache=false</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_statement=SELECT &quot;city_id&quot;, &quot;sensor_id&quot; FROM &quot;DSEQuickStart&quot;.&quot;FridgeSensor_p&quot; WHERE &quot;~~vertex_exists&quot; = ? LIMIT ? ALLOW FILTERING; with params (java.lang.Boolean) true, (java.lang.Integer) 50000 _options=Options{consistency=Optional[ONE], serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1, user=Optional.empty, waitForSchemaAgreement=true, async=true}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index-query</td>
<td>2.012</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_usesCache=false</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_statement=SELECT &quot;community_id&quot;, &quot;member_id&quot; FROM &quot;DSEQuickStart&quot;.&quot;author_p&quot; WHERE &quot;~~vertex_exists&quot; = ? LIMIT ? ALLOW FILTERING; with params (java.lang.Boolean) true, (java.lang.Integer) 50000 _options=Options{consistency=Optional[ONE], serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1, user=Optional.empty, waitForSchemaAgreement=true, async=true}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index-query</td>
<td>0.549</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_usesCache=false</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Using DataStax Enterprise advanced functionality

```java
.Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async=true}

index-query

0.849

.Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async=true}

index-query

0.887

.Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async=true}

index-query

0.889

.Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async=true}

index-query

0.499
```

DSE 6.7 Developer Guide (Latest version)
Using DataStax Enterprise advanced functionality

DsegVertexStep(IN,[created],vertex)

query-optimizer
0.618
\_condition=(((label = created) & (true)) & direction = IN)
vertex-query
0.261
\_usesCache=false
\_statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE "community_id" = ? AND "member_id" = ? AND "--
edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Integer) 143208000, (java
.lang.Long) 1, (java.lang.Integer) 65577,
(java.lang.Integer) 50000
\_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async
c=true}
\_isPartitioned=false
\_usesIndex=false
vertex-query
0.200
\_statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE "community_id" = ? AND "member_id" = ? AND "--
edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Integer) 153541376, (java
.lang.Long) 1, (java.lang.Integer) 65577,
(java.lang.Integer) 50000
\_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async
c=true}
\_isPartitioned=false
\_usesIndex=false
query-setup
0.017
\_isFitted=true
\_isSorted=true
\_isScan=false
vertex-query
6.140
\_usesCache=false
\_statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE "community_id" = ? AND "member_id" = ? AND "--
edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Integer) 58891056, (java
.lang.Long) 0, (java.lang.Integer) 65577,
(java.lang.Integer) 50000
\_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
Using DataStax Enterprise advanced functionality

```
  al.empty, pagingState=null, pageSize=-1,
  user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
  _isPartitioned=false
  _usesIndex=false
query-setup
  0.017
  _isFitted=true
  _isSorted=true
  _isScan=false
vertex-query
  0.201
  _usesCache=false
  _statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
  "community_id" = ? AND "member_id" = ? AND "~
  edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params
  (java.lang.Integer) 771301632, (java.
  lang.Long) 0, (java.lang.Integer) 65577,
  (java.lang.Integer) 50000
  _options=Options{consistency=Optional[ONE],
  serialConsistency=Optional.empty, fallbackConsistency=Option
  al.empty, pagingState=null, pageSize=-1,
  user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
  _isPartitioned=false
  _usesIndex=false
query-setup
  0.012
  _isFitted=true
  _isSorted=true
  _isScan=false
vertex-query
  0.173
  _usesCache=false
  _statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
  "community_id" = ? AND "member_id" = ? AND "~
  edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params
  (java.lang.Integer) 994194304, (java.
  lang.Long) 0, (java.lang.Integer) 65577,
  (java.lang.Integer) 50000
  _options=Options{consistency=Optional[ONE],
  serialConsistency=Optional.empty, fallbackConsistency=Option
  al.empty, pagingState=null, pageSize=-1,
  user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
  _isPartitioned=false
  _usesIndex=false
query-setup
  0.012
  _isFitted=true
  _isSorted=true
  _isScan=false
NoOpBarrierStep(2500)                                                  8
  4  0.910  0.69
```
As with the original traversal, the first step still finds all the vertices. In the next step, however, the number of edges walked is significantly decreased. However, in a production graph, finding all the vertices in the entire graph will take a long time. The third step now reflects the true answer for how many recipes Julia Child has created; in the first traversal, other incoming edges for Julia Child's books were included in the count.

This graph traversal is still an OLAP traversal that touch all vertices and does not use indexes.

Specifying the vertex label

What effect does specifying the vertex label have on improving the traversal?

g.V().hasLabel('recipe').in().has('name', 'Julia Child').count()

====>3
This modified traversal now is limited to the recipe vertices, but walks all incoming edges. The profile shows a somewhat better picture:

```sql
 gremlin> g.V().hasLabel('recipe').in().has('name','Julia Child').count().profile()
```

<table>
<thead>
<tr>
<th>Step</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traversers</td>
<td>8</td>
</tr>
<tr>
<td>Time (ms)</td>
<td>2.598</td>
</tr>
<tr>
<td>% Dur</td>
<td>9.25</td>
</tr>
<tr>
<td>DsegGraphStep([~label.=(recipe)])</td>
<td></td>
</tr>
<tr>
<td>query-optimizer</td>
<td>0.241</td>
</tr>
<tr>
<td>_condition=((label = recipe) &amp; (true))</td>
<td></td>
</tr>
<tr>
<td>query-setup</td>
<td>0.187</td>
</tr>
<tr>
<td>_isFitted=true</td>
<td></td>
</tr>
<tr>
<td>_isSorted=false</td>
<td></td>
</tr>
<tr>
<td>_isScan=true</td>
<td></td>
</tr>
<tr>
<td>index-query</td>
<td>1.225</td>
</tr>
<tr>
<td>_usesCache=false</td>
<td></td>
</tr>
<tr>
<td>_statement=SELECT &quot;community_id&quot;, &quot;member_id&quot; FROM &quot;DSEQuickStart&quot;.&quot;recipe_p&quot; WHERE &quot;~~vertex_exists&quot; = ? LIMIT ? ALLOW FILTERING; with params (java.lang.Boolean) true, (java.lang.Integer) 50000 _options=Options{consistency=Optional[ONE], serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1, user=Optional.empty, waitForSchemaAgreement=true, async=true}</td>
<td></td>
</tr>
<tr>
<td>DsegVertexStep(IN,vertex)</td>
<td>15</td>
</tr>
<tr>
<td>query-optimizer</td>
<td>0.150</td>
</tr>
<tr>
<td>_condition=((true) &amp; direction = IN)</td>
<td></td>
</tr>
<tr>
<td>query-setup</td>
<td>0.047</td>
</tr>
<tr>
<td>_isFitted=false</td>
<td></td>
</tr>
<tr>
<td>_isSorted=true</td>
<td></td>
</tr>
<tr>
<td>_isScan=false</td>
<td></td>
</tr>
<tr>
<td>vertex-query</td>
<td>0.896</td>
</tr>
<tr>
<td>_usesCache=false</td>
<td></td>
</tr>
<tr>
<td>_statement=SELECT * FROM &quot;DSEQuickStart&quot;.&quot;recipe_e&quot; WHERE &quot;community_id&quot; = ? AND &quot;member_id&quot; = ? LIMIT ? ALLOW FILTERING; with params (java.lang.Integer) 1315507840, (java.lang.Long) 1, (java.lang.Integer) 50000 _options=Options{consistency=Optional[ONE], serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1, user=Optional.empty, waitForSchemaAgreement=true, async=true}</td>
<td></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

```java

_isPartitioned=false
_isUsesIndex=false

vertex-query

1.415
inesisCache=false
_isUsesIndex=false

_statement=SELECT * FROM "DSEQuickStart"."recipe_e" WHERE "community_id" = ? AND "member_id" = ? LIMIT ?
    ALLOW FILTERING; with params (java.lang.Integer) 96517120, (java.lang.Long) 1, (java.lang.Integer) 50000

_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
_isPartitioned=false
_isUsesIndex=false

vertex-query

2.846
inesisCache=false
_isUsesIndex=false

_statement=SELECT * FROM "DSEQuickStart"."recipe_e" WHERE "community_id" = ? AND "member_id" = ? LIMIT ?
    ALLOW FILTERING; with params (java.lang.Integer) 1598713728, (java.lang.Long) 1, (java.lang.Integer) 50000

_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
_isPartitioned=false
_isUsesIndex=false

query-setup

0.038
_isFitted=false
_isSorted=true
_isScan=false

vertex-query

0.364
inesisCache=false
_isUsesIndex=false

_statement=SELECT * FROM "DSEQuickStart"."recipe_e" WHERE "community_id" = ? AND "member_id" = ? LIMIT ?
    ALLOW FILTERING; with params (java.lang.Integer) 1146421632, (java.lang.Long) 1, (java.lang.Integer) 50000

_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
_isPartitioned=false
_isUsesIndex=false

query-setup

0.014
```
Using DataStax Enterprise advanced functionality

```java
_isFitted=false
_isSorted=true
_isScan=false
vertex-query
  0.431
_isCache=false
_statement=SELECT * FROM "DSEQuickStart"."recipe_e" WHERE "community_id" = ? AND "member_id" = ? LIMIT ?
  ALLOW FILTERING; with params (java.lang.Integer) 384373760, (java.lang.Long) 2, (java.lang.Integer) 50000
_options=Options{consistency=Optional[ONE], serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1, user=Optional.empty, waitForSchemaAgreement=true, async=true}
_isPartitioned=false
_isIndex=false
query-setup
  0.014
_isFitted=false
_isSorted=true
_isScan=false
HasStep([name.=(Julia Child)])                                         3
  3          15.765    56.10
CountGlobalStep
  1          0.068     0.24
>TOTAL                     -          28.100        -
```

Figure 34: Studio profile output for Traversal 3
A limited number of vertices are found in the first step. A number of edges are walked. However, in a production graph, finding even a limited number of vertices will take some time without indexing, and the number of edges walked could be quite large.

This graph traversal is still an OLAP traversal that does not use indexes. Although this traversal narrows the query by limiting the vertex label initially, an index is not used to find the starting point for the traversal.

Using an edge label plus a vertex label

Indexes are identified by vertex label and property key. The following graph traversal twists the direction of the query:

```java
g.V().has('author', 'name', 'Julia Child').outE('created').count() => 3
```

This traversal starts at a single vertex by specifying both vertex label `author` and a specific property key and value `Julia Child`, and walks only the outgoing edges that have an edge label `created`.

```java
gremlin> g.V().has('author','name','Julia Child').outE('created').count().profile()
Traversal Metrics
Step Traversers Time (ms) % Dur
DsegGraphStep([~label.=(author), name.=(Julia C... 1 29.049 84.45
query-optimizer 7.673
\_condition=(((label = author) & (true)) & name = Julia Child)
query-setup 0.033
\_isFitted=true
\_isSorted=false
\_isScan=false
index-query 17.694
\_indexType=Secondary
\_usesCache=false
\_statement=SELECT "community_id", "member_id" FROM "DSEQuickStart"."author_p" WHERE "name" = ? LIMIT ?;
with params (java.lang.String) Julia Child,
(java.lang.Integer) 50000
\_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
DsegVertexStep(OUT,[created],edge) 3 5.265 15.31
```
<table>
<thead>
<tr>
<th>CountGlobalStep</th>
<th>1</th>
<th>0.081</th>
<th>0.24</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td>1</td>
<td>34.397</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 35: Studio profile output for Traversal 4

A single vertex starts the traversal. An edge label filters the edges.

This graph traversal is an OLTP traversal. An index on the vertex label `author` and property key `name` can be used to start the traversal directly at an indexed vertex. This example results in a single vertex, but queries that use indexing to limit the starting point to even several vertices will be more efficient than a linear scan that must check all vertices in the graph. Thus, a subgraph, or portion of the graph is traversed.
The key to creating OLTP graph traversals is considering how the graph will be traversed. Use of indexing is critical to the success of fast transactional processing. The profiling tool included with DSE Graph is valuable to analyzing how the traversal performs.

For information on running OLAP queries using Spark, see DSE Graph and Graph Analytics (page 620).

**Graph anti-patterns**

Some common mistakes are made with DSE Graph. Examining best practices can ease the learning curve and improve graph application performance.

**Not using indexing**

Indexing is a key feature in decreasing the latency of queries in a distributed database. DSE Graph relies on indexing to speed up OLTP read latency for complex graph traversals. What is key to understand is that global indexing in DSE Graph involves both a vertex label and a property key. The vertex label narrows the search in the underlying DSE datastore to a partition, which in turn narrows the search to one or a small number of DSE database nodes in the cluster. Indexing a property key that is used for more than one vertex label and not supplying the vertex label in the query amounts to an almost full scan of the cluster. Thus, using this query:

```java
  g.V().has('name','James Beard')...
```

requires the traversal to check all vertices that use the property key `name`. Changing this query to:

```java
  g.V().has('author', 'name', 'James Beard')...
```

allows the query to consult an index that can be built for all names in author records, and retrieve just one vertex to start the traversal. The index would be added during schema creation (page 447):

```java
  schema.vertexLabel('author').index('byName').secondary().by('name').add()
```

In fact, this one change in the traversal will change the query from an OLAP query into an OLTP query.

**Property key creation**

Property key creation can affect the performance of DSE Graph. Using unique property key names can seem beneficial at first, but reusing property keys for different vertex labels can improve the storage of property keys for the graphs. For example, consider the following:

```java
  schema.propertyKey('recipeCreationDate').Timestamp().create()
  schema.propertyKey('mealCreationDate').Timestamp().create()
  schema.propertyKey('reviewCreationDate').Timestamp().create()
```
While these property key names make code readable and ease tracking in graph traversals, each additional property key stored requires resources. Use one property key instead, such as:

```java
schema.propertyKey('timestamp').Timestamp().create()
```

to decrease overhead. Since property keys are mostly used in graph traversals along with vertex labels, `timestamp` will be uniquely identified by the combination of vertex label and property key.

**Vertex label creation**

Vertex label creation can affect the performance of DSE Graph. Using many unique vertex labels can seem useful, but like property keys, the fewest vertex labels created can improve the storage requirements. For example, consider the following:

```java
schema.vertexLabel('recipeAuthor').create()
schema.vertexLabel('bookAuthor').create()
schema.vertexLabel('mealAuthor').create()
schema.vertexLabel('reviewAuthor').create()
```

While these vertex labels again have the advantage of readability, unless a vertex label will be uniquely queried, it is best to roll the functionality into a single vertex label. For instance, in the above code, it is likely that recipes, meals, and books will have the same authors, whereas reviews are likely to have a different set of writers and types of queries. Use two vertex labels instead of four:

```java
schema.vertexLabel('author').create()
schema.vertexLabel('reviewer').create()
```

In fact, this case may even be better suited to using only one vertex label `person`, if the overlap in authors and reviewers is great enough. In some cases, a property key that identifies whether a `person` is an author or a reviewer is a viable option.

```java
schema.propertyKey('type').Text().create()
schema.vertexLabel('person').create()

graph.addVertex(label, 'person', 'type', 'author', 'name', 'Jamie Oliver')
```

**Mixing schema creation or configuration setting with traversal queries**

Consider the following statements. The first statement configures a graph setting for read consistency. The second statement executes a count on a field `name` with a value `read vertex` for all vertices.

```java
schema.config().option('graph.tx_groups.default.read_consistency').set('ALL');
g.V().has('name', 'read vertex').count()
```

In Gremlin Server, both statements are run in one transaction. Any changes made during this transaction are applied when it successfully commits both actions. The change in read consistency is not actually applied until the end of a transaction and thereby only affects the next transaction. The statements are not processed sequentially as individual requests.
To avoid such errors in processing, avoid mixing schema creation or configuration setting with traversal queries in applications. Best practice is to create schema and set configurations before querying the graph database with graph traversals.

InterruptedException indicates OLTP query running too long

In general, seeing logs with this exception are indicative that an OLTP query is running too long. The typical cause is that indexes have not been created for elements used in graph traversal queries. Create the indexes (page 472) and retry the queries.

g.V().count() and g.E().count() can cause long delays

Running a count on a large graph can cause serious issues. The command basically must iterate through all the vertices, taking hours if the graph is large. Any table scan (iterating all vertices) is simply not an OLTP process. Doing the same process on edges is essentially the same, a full table scan, as well. Using Spark commands are currently the recommended method to get these counts.

Setting replication factor too low for graph_name_system

Each graph created in turn creates three DSE database keyspaces, graph_name, graph_name_system and graph_name_pvt. The graph_name_system stores the graph schema, and loss of this data renders the entire graph inoperable. Be sure to set the replication factor appropriately (page 664) based on cluster configuration.

Using string concatenation in application instead of parameterized queries

String concatenation in graph applications will critically impair performance. Each unique query string creates an object that is cached on a node, using up node resources. Use parameterized queries (DSE Java Driver, DSE Python Driver, DSE Ruby Driver, DSE Node.js Driver, DSE C# Driver, DSE C/C++ Driver) to prevent problems due to resource allocation.

DSE Graph data modeling

Graph data modeling introduction

Data modeling for graph databases is generally a simple process. Imagine information written on a whiteboard as vertices and lines connecting them, and you are 90% done with a graph database data model.

Figure 36: Julia Child creates beef bourguignon

Julia Child was a famous chef who created many recipes. One of the recipes she created for an American audience in 1961 was beef bourguignon. In the diagram above, a person,
Julia Child, is linked to a recipe, beef bourguignon. Person and recipe are two types of vertex, and the line adjoining the vertices, or edge, identifies the relationship as "created". Vertices and edges have associated properties, such as a person's name, a recipe name, and the date associated with the edge. Properties are a basic element that are used in a query about the graph, and consist of a property key and property value. In graph databases, a vertex is incident to an edge, and an edge is incident to a vertex. A vertex is adjacent to another vertex if they share an edge. A generalized view of this data model is shown below:

![Generalized data model for author and recipe](image.png)

Each vertex is assigned a vertex label to identify a specific type of vertex. The vertex labels shown here are person and recipe. Each edge must also have an edge label specifying its type. The edge label shown is created. The properties shown are name and createDate.

DSE Graph limits the number of vertex labels to 200 per graph.

For more complex graphs, multiple edges (page 466) can connect vertices, and multiple properties can be assigned to vertices and edges. Both properties and edges can have multiple cardinality (page 466). Vertex properties can have meta-properties, a property on a property (page 459).

An important concept to be aware of is the nature of vertices and edges as addressable elements. Indexes (page 469) play a critical role in querying graphs, and vertex labels must be a part of every index. Only vertices are globally addressable, whereas edges are only locally addressable. In practice, what this situation means is that edges can only be indexed locally for a particular vertex label. Edges are about the relationship of vertices, and are classified as second-class citizens; vertices are entities and are first-class citizens for which all graph operations are available. To illustrate the nature of the second-class citizenry of edges, meta-properties of edges cannot be indexed and used to narrow queries, making those edges better modeled as vertices if the data stored in the meta-properties must be used to narrow down a query.

For the remaining 10% of your effort, optimization of whether an aspect of your whiteboard graph should be a vertex or an edge is the most pressing factor. If an aspect used as an edge begins to be used more than a few times, it should become a vertex instead. For instance, we could add a vertex property to the author to add their country of origin. However, since many authors will come from the same country, such as the China or
Using DataStax Enterprise advanced functionality

France, creating a location vertex type can be more advantageous to later querying operations.

**Graph data modeling example**

Let's consider the example of recipes further to create a more complex data model. This example will go through some of the thinking behind creating a graph database data model.

1. Obviously, we will need vertices that are connected by edges. What is a possible additional type of vertex besides person and recipe?

   Not surprisingly, we can add an *ingredient* vertex label. This vertex will have some properties. Can you think of the possibilities for vertex properties?

   The most likely property for an ingredient vertex is the *name* of the ingredient. While we could use *ingredientName* to identify the name of the ingredient, keeping the schema small has advantages in DSE Graph. We'll reuse *name* for every vertex label in our example.

   ![Ingredient vertex with properties](image)

2. There are other possibilities that might be important for the ingredient vertex properties. Think about it and write down some more possibilities. We will add them later. Let's move on to considering the edges that will connect persons, recipes, and ingredients.

   What are the edge labels we can use to identify different types of edges? Previously, you've seen that persons and recipe are connected by an edge *created*.

   An ingredient must be included in a recipe, so an edge *includes* can connect the two vertices.

   ![Graph with edge labels](image)

3. Edges also have attached properties that can be used later in narrowing queries.

   What edge property is appropriate for *includes*?
The amount of ingredient included in a recipe is important! One cup of salt instead of one teaspoon of salt will make a big difference in the results.

4. Today, people publish their recipes online and in cookbooks. Restaurants create fixed price meals from recipes. Consumers review the recipes they try. The results are an intertwined graph of data.

5. The additional vertices and edges that can be added to this graph are numerous. For instance, the gender of the recipe authors and reviewers (both persons) can be included. Nutritional information for the ingredients can be derived from the calories for a recipe. The number of servings that a recipe makes is useful to cooks. The resulting web of data can grow quickly.
Using DataStax Enterprise advanced functionality

Add a hundred authors, a thousand recipes, ten thousand reviews, and the enormity of the graph becomes obvious. However, as you will see in later sections, DSE Graph can transform complex searches and pattern matching into simple and powerful solutions.

**What's next:**

The data model is the first step in creating a graph. Using the data model, a schema can be created ([page 447](#)) that defines how DSE Graph will store the data.

**Further data modeling concepts**

Graph data models can be expanded to encompass complex relationships. The whole graph can be digested better if subgraphs are considered. The recipe data model can be modified to include new layers of data.
Consider an ingredient. Many additional properties can be added to an ingredient:

**category**
vegetable, fruit, pasta, meat

**nutritional value**
% of vitamins, protein, carbohydrate, fat

**calories**
number of kcals

While it may seem simple to choose the property values for an ingredient, there can be more to consider. For instance, consider category. Depending on the number of categories used to describe the ingredients, it can be more advantageous to create a vertex...
Using DataStax Enterprise advanced functionality

label or a property for category. Vertices can be the starting point for a graph traversal, but vertex properties cannot. In order to ask the question "what ingredients are dairy products?", a starting point at the dairy vertex requires one edge hop per ingredient to find all the ingredients categorized as dairy.

However, if too many ingredients are dairy, a super node, or node that is a hotspot with too many edges attached, can slow down queries that are searching for dairy ingredients. Using property indexing, an ingredient category can be better modeled as a property rather than a vertex label.

Nutrients are a set number of items, such as vitamin C, vitamin D, calcium, and sodium. Creating a vertex label for nutrient and weighting the edges between ingredient and nutrient with the percentage adds another dimension to the graph.

Look at the relationships that result for just one ingredient:
Using DataStax Enterprise advanced functionality

and imagine the graph resulting for even one hundred ingredients, let alone thousands of ingredients. Examine whether it is better to create a nutrient vertex label or nutrient vertex properties.

Imagine the possibilities for applications built using the ingredient properties. Look in the refrigerator and discover that you have mushrooms and beef, and query the graph database to find a recipe to cook, such as Beef Stroganoff. With the coming possibility of
Using DataStax Enterprise advanced functionality

tagged food in your refrigerator, you could even have your fridge tell you what's for dinner tonight, given the items stored.

Using DSE Graph

Getting started with graph databases

Graph databases are useful for discovering simple and complex relationships between objects. These things can be people, software, locations, automobiles, or anything else you can think of. Relationships are fundamental to how objects interact with one another and their environment. Graph databases are the perfect representation of the relationships between objects.

Graph databases consist of three elements:

**vertex**
A vertex is an object, such as a person, location, automobile, recipe, or anything else you can think of as nouns.

**edge**
An edge defines the relationship between two vertices. A person can create software, or an author can write a book. Think verbs when defining edges.

**property**
A key-value pair that describes some attribute of either a vertex or an edge. Property key is used to describe the key in the key-value pair. All properties are global in DSE Graph, meaning that a property can be used for any vertices. For example, "name" can be used for all vertices in a graph.

Vertices, edges and properties can have properties; for this reason, DSE Graph is classified as a property graph. The properties for elements are an important element of storing and querying information in a property graph.

Property graphs are typically quite large, although the nature of querying the graph varies depending on whether the graph has large numbers of vertices, edges, or both vertices and edges. To get started with graph database concepts, a toy graph is used for simplicity. The example used here explores the world of food.
Elements are labeled to distinguish the type of vertices and edges in a graph database. A vertex that will hold information about a person is labeled `person`. An edge in the graph is labeled `authored`. Labels specify the types of vertices and edges that make up the graph. Specifying appropriate labels is an important step in graph data modeling (page 434).

Vertices and edges generally have properties. For instance, an `person` vertex can have a `name`. Gender and current job are examples of additional properties for a `author` vertex. Edges also have properties. A `created` edge can have a `createDate` property that identifies when the adjoining `recipe` vertex was created.

Properties can also have properties. Consider the locations that an author may have lived in while authoring books. While knowing the writing location may be interesting by itself, generally an inquirer is interested in the dates that a person lived in a particular location. Would it be interesting to know if Julia Child lived in France or the United States while writing her first cookbook? It could be relevant if the cookbook is on French cuisine.

There are a variety of methods for ingesting data into DSE Graph.

**DSE Graph Loader**
Data can be loaded using the DSE Graph Loader (page 484). CSV, JSON, text parsed with regular expressions, and data selected from a JDBC compliant database can be loaded using a command line tool.

**DataStax Studio**
Graph API and Traversal API can be used.

**Gremlin commands**
Data can be added using Gremlin commands. This is a useful method for toy (small graphs) used for development and test. An API exists for adding data using Gremlin commands as well, so Gremlin is common in scripts. The Quick Start (page 378) shows some of the common Gremlin commands for creating a graph and running traversals.

**Gryo**
Data can be loaded using Gryo (page 517), a binary format, if the data was previously stored in Titan or TinkerGraph. Gryo files can be transferred directly using the schema from the original database.

**GraphSON**
Data can be entered with GraphSON (page 519), a JSON format that is useful for transferring human-readable data. GraphSON files can lose data type information in transfer unless lossless data is generated.

**GraphML**
Data can be entered using GraphML (page 521), an XML format that is useful for transferring graph data. However, data type information is lost with GraphML data transfer.

After loading data, **graph traversals** are executed to retrieve filtered information. In relational databases, **queries** are retrieved that combine and filter information. In graph databases, the vertex properties, edge connections, and edge properties all play a role in picking a starting point in the graph and traversing the connections to provide a particular answer to a query. Several **TraversalSources**, that supply a traversal strategy and traversal engine to use in executing traversals, can be generated for any **Graph**. Queries in graph databases can consist of several traversals if a complex question is asked, or trivially include no traversals, if a mathematical calculation like $1 + 1$ is submitted.

### Managing graphs

#### Creating a graph

Depending on the DSE Graph schema mode, DataStax Studio will have differing behavior. **In Production mode**, DataStax Studio will not auto-create a graph, and the **graph must be created in the Gremlin console** (page 444). **In Development mode**, DataStax Studio creates a graph and aliases the graph to a graph traversal automatically for each connection that is created.

DataStax Studio creates a graph automatically for each connection that is created. In Gremlin console, a graph must be manually created. In addition to creating the graph, a graph traversal must be aliased to the graph in order to run queries.

**Studio**

1. Start DSE Graph (page 1437).
2. Install and start Studio. Also create a Studio notebook, if needed.
3. In DataStax Studio, **create a connection**. Choose a graph name; any graph previously unused will work.

4. In DataStax Studio, **create a notebook**. Select the connection created in the last step. A blank notebook will open with a single cell. DSE Graph runs a Gremlin Server \texttt{tinkerpop.server} on each DSE node. DataStax Studio automatically connects to the Gremlin Server, and if it doesn't exist, creates a graph using the connection information. The graph is stored as one graph instance per DSE database keyspace with a replication factor of 1 and a strategy of \texttt{SimpleStrategy}. Once a graph exists, a graph traversal \texttt{g} is configured that will allow graph traversals to be executed. Graph traversals are used to query the graph data and return results. A graph traversal is bound to a specific traversal source which is the standard OLTP traversal engine.

**Gremlin console**

5. **Start the Gremlin console** (*page 638*).

6. Create a simple graph with default settings to hold the data.

   ```java
   system.graph('food').create()
   =>null
   ```

7. Create a graph with non-default replication (*page 664*), sysre\texttt{temReplication} (*page 665*), and configuration settings (*page 645*):

   ```java
   system.graph('food2').
   replication("{'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }").
   systemReplication("{'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }").
   option("graph.schema_mode").set("Production").
   option("graph.allow_scan").set("false").
   option("graph.default_property_key_cardinality").set("multiple").
   option("graph.tx_groups.*.write_consistency").set("QUORUM").create()
   ```

   **Caution:** For graphs created in multi-datacenter clusters, the DSE \texttt{database settings} must use \texttt{NetworkTopologyStrategy} and a replication factor greater than one. If the graph is created with a replication setting of \texttt{SimpleStrategy} and a replication factor of 1, the graph data will be stored across the multiple datacenters rather than localizing the data in the graph datacenter.

   The default replication strategy for a multi-node or multi-datacenter graph is \texttt{NetworkTopologyStrategy}, whereas for a single node, the replication strategy will default to \texttt{SimpleStrategy}. The number of nodes will determine the default replication factor:
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>number of nodes per datacenter</th>
<th>graph_name replication factor</th>
<th>graph_name_system replication factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>number of nodes per datacenter</td>
<td>number of nodes per datacenter</td>
</tr>
<tr>
<td>greater than 3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

8. On the remote Gremlin Server, set the alias for the graph traversal `g` to the graph traversal specified in `food`. To run traversals, the graph traversal must be aliased to a graph.

```plaintext
:remote config alias g food.g
```

```
g=food.g
```

**Examining graphs**

1. A list of all graphs can be retrieved with the following command:

```plaintext
system.graphs()
```

In Studio and Gremlin console, a list is retrieved, although the presentation is different. Here is a Gremlin console result:

```
=> food
=> test
```

**Dropping graphs**

Graphs can be dropped (deleted). All schema and data for the graph will be lost, so be sure that you intend to remove a graph before using the steps below.

- A system command is required to drop a graph. If using the Gremlin console, the graph traversal alias can be cleared, to be certain no action effects the currently selected graph in the Gremlin console:

```
gremlin> :remote config alias reset
```

```
=>Aliases cleared
```

- Optional: If unsure of the graph name, examine what graphs exist *(page 446).*
- Drop the desired graph by running the `drop()` command:

```
system.graph('food').drop()
```

Use the `ifExists()` step to check if a graph exists before dropping.
Managing graph schema

Creating graph schema

Creating a data model (page 436) for a graph database is the critical first step towards creating a schema. Once the data model is designed and a graph is created, defining the schema for the vertices and edges and their properties is the next step in creating a graph database. Gremlin-Groovy is the language used to create scripts; Gremlin-Groovy is packaged with the Apache TinkerPop engine, and can be used with either DataStax Studio or the Gremlin console (dse gremlin-console) installed with DataStax Enterprise.

Graph schema can be created with create() or added to existing schema with add().

Prerequisites:

Create a graph (page 444).

1. Optional. If you are reusing a graph that you previously created, drop the graph schema and data (page 446).

2. Optional. If running large scripts in Gremlin console, set the timeout value to max to prevent client-side time outs. Use this setting to ensure that script processing will complete. This step cannot be completed in Studio.

```
gremlin> :remote config timeout max
```

3. Optional. If running large scripts, set the evaluation_timeout value to max to prevent server-side timeouts. Use this setting to ensure that script processing will complete.

```
graph.schema().config().option("graph.traversal_sources.g.evaluation_timeout").set("PT10M")
```

**Important:** Setting a timeout value greater than 1095 days (maximum integer) can exceed the limit of a graph session. Starting a new session and setting the timeout to a lower value can recover access to a hung session. This caution is applicable for all timeouts: evaluation_timeout, system_evaluation_timeout, analytic_evaluation_timeout, and realtime_evaluation_timeout

4. Load the example schema listed in the Example below:

   a. In Studio, copy and paste the entire code block into a single cell and execute the cell.

   b. In Gremlin console, copy and paste the example schema to a schema file. Two choices for loading are shown:
Using DataStax Enterprise advanced functionality

Use the :load command by specifying the location of the schema file:

```plaintext
gremlin> :load /tmp/RecipeSchema.groovy
```

Use a cat command in the machine shell to load the file, specifying the location of the schema file, and pass to the Gremlin console:

```plaintext
$ cat /tmp/RecipeSchema.groovy | dse gremlin-console
```

**NOTE:** Each command submitted is within a single session, so from cell to cell (Studio) or line to line (Gremlin console), the Gremlin server is not aware of any variables set on the previous line. If any of the lines in the Recipe Schema are entered separately, an error will occur on the edge creation commands.

5. The following steps show the details of the full script broken down into sections.

6. Define the properties for the vertices and the edges. The data type of the property is specified in addition to a key name. All properties created in this example are Text, Integers, or Timestamps. Other data types (page 754) are available. Properties will be used to retrieve selective subsets of the graph and to retrieve stored values. Properties are global in nature, and the pairing of a vertex label and a property will uniquely identify a property for use in traversals. Edge properties are expensive to update, as because the whole edge with all its properties are deleted and recreated to update edge properties. Use edge properties only in situations that warrant their use.

```plaintext
// Property Keys
// Check for previous creation of property key with ifNotExists()

schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().create()
schema.propertyKey('instructions').Text().create()
schema.propertyKey('category').Text().create()
schema.propertyKey('year').Int().create()
schema.propertyKey('timestamp').Timestamp().create()
schema.propertyKey('ISBN').Text().create()
schema.propertyKey('calories').Int().create()
schema.propertyKey('amount').Text().create()
schema.propertyKey('stars').Int().create()
schema.propertyKey('comment').Text().create() // single()
    is optional - default
// Example of multiple property
// schema.propertyKey('nickname').Text().multiple().create();
// Example meta-property added to property:
// schema.propertyKey('livedIn').Text().create()
  //
  // schema.propertyKey('country').Text().multiple().properties('livedIn').create()
```

Property keys can be checked for prior existence with ifNotExists(). Property keys can be created with either single or multiple cardinality with single() or multiple().
The default is single cardinality which does not have to be specified, but it can be explicitly stated as in the example.

Meta-properties, or properties of properties, can be created using `propertyKey()` followed by `properties()`. The property key must exist prior to the creation of a meta-property. Meta-properties cannot be nested, i.e., a meta-property cannot have a meta-property. In this example, `country` is the property that has a meta-property `livedIn`. This property and meta-property are used to represent the countries that an author has lived in at various times in their life.

```json
{
  "name":"Julia Child",
  "gender":"F",
  [ {"country": "United States", "livedIn": "1929-1949" },
    {"country": "France", "livedIn": "1949-1952" } ],
  "authored":[{
    "book":{
      "label":"book",
      "bookTitle":"Art of French Cooking Volume One",
      "publishDate":1968
    },
    "book":{
      "label":"book",
      "bookTitle":"The French Chef Cookbook",
      "publishDate":1968,
      "ISBN": "0-394-40135-2"
    }
  }],
  "created": [{
    "type": "recipe",
    "recipeTitle": "BeefBourguignon",
    "instructions": "Braise the beef.",
    "createDate":1967
  },
  {"type": "recipe",
    "recipeTitle": "Salade Nicoise",
    "instructions": "Break the lettuce into pieces.",
    "createDate": 1970
  }]
}
```

7. Define the vertex labels. The vertex labels identify the type of vertices that can be created.

```java
// Vertex Labels
schema.vertexLabel('author').ifNotExists().create()
schema.vertexLabel('recipe').create()
// Example of creating vertex label with properties
```
Using DataStax Enterprise advanced functionality

Vertex labels can be checked for prior existence using `ifNotExists()`. Vertex labels can be created along with properties. Vertex labels can be created with user-defined vertex ids (page 462), rather than the autogenerated vertex ids (page 462).

Note: Auto-generated vertex ids are deprecated with DSE 6.0.

DSE Graph limits the number of vertex labels to 200 per graph.

8. Define the edge labels. The edge labels identify the type of edges that can be created.

```java
// Edge Labels
schema.edgeLabel('authored').ifNotExists().create()
schema.edgeLabel('created').create()
schema.edgeLabel('includes').create()
schema.edgeLabel('includedIn').create()
schema.edgeLabel('rated').properties('rating').connection('reviewer','recipe').create()
```

Edge labels can be checked for prior existence using `ifNotExists()`. Edge labels can be created with adjacent vertex labels identified using `connection()` (page 644). Edge labels can identify properties that an edge has using `properties()` (page 657).

9. Define indexes that can speed up the query processing. All types of indexes are presented here. Indexing (page 469) has more information.

```java
// Vertex Indexes
// Secondary
schema.vertexLabel('author').index('byName').secondary().by('name').add()
// Materialized
schema.vertexLabel('recipe').index('byRecipe').materialized().by('name').add()
schema.vertexLabel('meal').index('byMeal').materialized().by('name').add()
schema.vertexLabel('ingredient').index('byIngredient').materialized().by('name').add()
schema.vertexLabel('reviewer').index('byReviewer').materialized().by('name').add()
// Search
//
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().add()
//
schema.vertexLabel('recipe').index('search').search().by('instructions').asString().add()
// If more than one property key is search indexed
```
Using DataStax Enterprise advanced functionality

```java
// schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('category').add()

// Edge Index
schema.vertexLabel('reviewer').index('ratedByStars').outE('rated').by('stars').add()

// Example of property index using meta-property 'livedIn':
// schema.vertexLabel('author').index('byLocation').property('country').by('livedIn').add
```

These indexes are included to make the schema for the food example more efficient for data loading.

**Note:** The difference between `create()` and `add()` is subtle but important. If an entity (vertex label or edge label) has been created and already exists, if an index or property keys are associated with the entity, then an `add()` command is used. For example, a vertex label and property keys can be created, and then the property keys can be added to the vertex label.

10. After creating the graph schema, examine the schema to verify. A portion of the output is shown.

```java
schema.describe()
```

// RECIPE SCHEMA

// To run in Studio, copy and paste all lines to a cell and run.

// To run in Gremlin console, use the next two lines:
// script = new File('/tmp/RecipeSchema.groovy').text; []
// :> @script

// Property Keys
// Check for previous creation of property key with ifNotExists()
schema.propertyKey('name').Text().ifNotExists().create()
Using DataStax Enterprise advanced functionality

```java
// Using DataStax Enterprise advanced functionality

// Schema

// Example of creating a property
// propertyKey('gender').Text().create()

// propertyKey('instructions').Text().create()

// propertyKey('category').Text().create()

// propertyKey('year').Int().create()

// propertyKey('timestamp').Timestamp().create()

// propertyKey('ISBN').Text().create()

// propertyKey('calories').Int().create()

// propertyKey('amount').Text().create()

// propertyKey('stars').Int().create()

// propertyKey('comment').Text().single().create() // single()

// is optional - default

// Example of multiple property
// propertyKey('nickname').Text().multiple().create();

// Example meta-property added to property:
// propertyKey('livedIn').Text().create()

// propertyKey('country').Text().properties('livedIn').create()

// Vertex Labels

// vertexLabel('author').ifNotExists().create()

// vertexLabel('recipe').create()

// Example of creating vertex label with properties

// vertexLabel('recipe').properties('name','instructions').create()

// vertexLabel('ingredient').create()

// vertexLabel('book').create()

// vertexLabel('meal').create()

// vertexLabel('reviewer').create()

// Example of custom vertex id:

// vertexLabel('city_id').Int().create()

// vertexLabel('sensor_id').Uuid().create()

// vertexLabel('FridgeSensor').partitionKey('city_id').clusteringKey('sensor_id').create()

// Edge Labels

// edgeLabel('authored').ifNotExists().create()

// edgeLabel('created').create()

// edgeLabel('includes').create()

// edgeLabel('includedIn').create()

// edgeLabel('rated').properties('stars').connection('reviewer','recipe').create()

// Vertex Indexes

// Secondary

// vertexLabel('author').index('byName').secondary().by('name').add()

// Materialized

// vertexLabel('recipe').index('byRecipe').materialized().by('name').add()

// vertexLabel('meal').index('byMeal').materialized().by('name').add()

// vertexLabel('ingredient').index('byIngredient').materialized().by('name').add()

// vertexLabel('reviewer').index('byReviewer').materialized().by('name').add()

// Search

// vertexLabel('recipe').index('search').search().by('instructions').asText().add()

// vertexLabel('recipe').index('search').search().by('instructions').asString().add()
```
// If more than one property key is search indexed
//
// schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('category').asString().add()

// Edge Index
schema.vertexLabel('reviewer').index('ratedByStars').outE('rated').by('stars').add()

// Example of property index using meta-property 'livedIn':
//
// schema().vertexLabel('author').index('byLocation').property('country').by('livedIn').add()

// Schema description
// Use to check that the schema is built as desired
schema.describe()

### Examining schema

Examining schema can be done to either verify the existence or non-existence of schema before creation.

1. A list of all schema can be retrieved with the following command:

```java
schema.describe()
```

In Studio and Gremlin console, a list is retrieved, although the presentation is different. Here is a portion of a Studio result:

![Studio Result](image)

2. The schema for a particular element can be examined by appending `describe()` to a more specific schema:

```java
schema.edgeLabel('includedIn').describe()
```

```java
schema.edgeLabel("includedIn").multiple().properties("amount").create()
schema.edgeLabel("includedIn").connection("recipe", "meal").connection("recipe", "book").connection("recipe", "book")
```
Using DataStax Enterprise advanced functionality

```java
"ingredient").connection("ingredient",
"recipe").connection("meal", "book").add()
```

3. Some groovy steps are useful in the Gremlin query to find specific schema descriptions. For instance, to inspect only the vertex labels and their indexes, use the `split()` and `grep()` steps:

```java
schema.describe().split(\"\n\").grep(-/.*vertexLabel.*/)```

In Studio:

Creating property key schema

Property keys (page 657), as discussed in the data model (page 434), allow attribute values to be assigned to either vertices or edges. Property keys can also have property keys assigned to them as meta-properties using `properties()`. Property keys are defined before assigning them to vertex or edge labels, and indexes can be created based on property keys.

Properties will be used to retrieve selective subsets of the graph and to retrieve stored values. Properties are global in nature, and the pairing of a vertex label and a property will uniquely identify a property for use in traversals. Edge properties are expensive to update, because the whole edge, with all its properties, are deleted and recreated to update edge properties. Use edge properties only in situations that warrant their use.

The cardinality of a property key is single `cardinality` by default, but multiple cardinality can be assigned. Additionally, property keys can be created with a specific Time-To-Live
Using DataStax Enterprise advanced functionality

(TTL) value. When creating property keys, prior existence of a key can be checked with `ifNotExists()`.

Property keys *(page 657)* must have a key name and data type *(page 754)* assigned, at a minimum. Several data types exist to accommodate data storage in flexible formats. Some warrant special attention, such as geospatial and Cartesian. Three geospatial data types, *point*, *linestring*, and *polygon*, store data that can be searched with geospatial shapes. For most geospatial queries that look for geospatial points, points or linestrings within circles or polygons, DSE Search indexes *(page 473)* must also be created; the same is true for Cartesian data types.

Property key schema are created with `create()`. Either DataStax Studio or the Gremlin console *(dse gremlin-console)* installed with DataStax Enterprise can be used to create or modify property keys.

**Prerequisites:**

*Create a graph*(page 444).

**Single and multiple cardinality property keys**

1. Define a single cardinality property key:

   The name of the propertyKey is *personId*, it is defined as an *Integer* type with `Int()`, with single cardinality explicitly stated with `single()`:

   ```java
   schema.propertyKey('personId').Int().single().create()
   ```

   Since single cardinality is the default, `single()` can be omitted:

   ```java
   schema.propertyKey('personId').Int().create()
   ```

   To check if a property key already exists before creating it, use `ifNotExists()`:

   ```java
   schema.propertyKey('createDate').Date().single().ifNotExists().create()
   ```

2. Define a multiple cardinality property key:

   The name of the propertyKey is *nickname*, it is defined as an *Text* type with `Text()`, with multiple cardinality:

   ```java
   schema.propertyKey('nickname').Text().multiple().create()
   ```

   Since multiple cardinality is not the default, `multiple()` must be included.

**Meta-properties**

3. Define a single cardinality meta-property to a multiple cardinality property key. First create the property keys for the meta-property, then create the property key that has the meta-property.
The name of the meta-property propertyKey is *since*, it is defined as an *Integer* type with `Int()`, with single cardinality. The name of the propertyKey that uses this meta-property is *badge*, it is defined as an *Text* type with `Text()`, with multiple cardinality. The meta-property is defined with `properties`.

```java
// MULTIPLE CARDINALITY PROPERTY KEY WITH SINGLE CARDINALITY META-PROPERTY
schema.propertyKey('since').Int().single().create() // meta-property
schema.propertyKey('badge').Text().multiple().properties('since').create()
```

4. Both properties and meta-properties can be defined with multiple cardinality. Define a multiple cardinality meta-property to a multiple cardinality property key. First create the property keys for the meta-property, then create the property key that has the meta-property.

The name of the meta-property propertyKey is *startYear*, it is defined as an *Integer* type with `Int()`, with multiple cardinality. The name of the propertyKey that uses this meta-property is *country*, it is defined as an *Text* type with `Text()`, with multiple cardinality. The meta-property is defined with `properties`.

```java
// MULTIPLE CARDINALITY PROPERTY KEY WITH MULTIPLE CARDINALITY META-PROPERTY
schema.propertyKey('startYear').Int().multiple().create() // meta-property
schema.propertyKey('endYear').Int().multiple().create() // meta-property
schema.propertyKey('country').Text().multiple().properties('startYear','endYear').create()
```

This example includes two meta-properties, *startYear* and *endYear*.

**Property keys with Time-To-Live (TTL)**

5. Property keys can be defined with a Time-To-Live (TTL) value; once the specified time is reached, the value is deleted from the graph. Define a single cardinality property key with TTL:

The name of the propertyKey is *bookDiscount*, it is defined as an *Text* type with `Text()`, single cardinality (default), and a TTL of 604,800 seconds with `ttl()`:

```java
// PROPERTY KEY WITH TTL
schema.propertyKey('bookDiscount').Text().ttl(604800).create()
```

**Geospatial property keys**

6. Create schema for *point* property key:

```java
schema.propertyKey('point').Point().withGeoBounds().create()
```
Note: For all geospatial elements, the `withGeoBounds()` method limits searches to a default valid range of latitude in degrees from -90 to +90 (South Pole to North Pole) and a valid range of longitude in degrees from -180 to +180 (east to west from the Greenwich Meridian). The point is specified using `Geo.point(longitude, latitude)` when adding points, using WellKnownText (WKT) format. Note that is specifies longitude first, then latitude.

7. Create schema for a `linestring` property key:
   ```java
   schema.propertyKey('line').Linestring().withGeoBounds().create()
   ```
   The same boundary limits (page 457) imposed on points are imposed on linestrings.

8. Create schema for a `polygon` property key:
   ```java
   schema.propertyKey('polygon').Polygon().withGeoBounds().create()
   ```
   The same boundary limits (page 457) imposed on points are imposed on polygons.

Cartesian property keys

9. Create schema for a `point` property key:
   ```java
   schema.propertyKey('point').Point().withBounds(-3, -3, 3, 3).create()
   ```
   Note: For Cartesian spatial types, the `withBounds(x1, y1, x2, y2)` method limit searches to a default valid range of values in the x-y grid.

10. Create schema for a `linestring` property key:
    ```java
    schema.propertyKey('line').Linestring().withBounds(-3, -3, 3, 3).create()
    ```
    The same boundary limits (page 457) imposed on points are imposed on linestrings.

11. Create schema for a `polygon` property key:
    ```java
    schema.propertyKey('polygon').Polygon().withBounds(-3, -3, 3, 3).create()
    ```
    The same boundary limits (page 457) imposed on points are imposed on polygons.

12. Define the properties for the vertices and the edges. The data type of the property is specified in addition to a key name. All properties created in this example are Text,
Integers, or Timestamps. Other data types (page 754) are available. Properties will be used to retrieve selective subsets of the graph and to retrieve stored values. Properties are global in nature, and the pairing of a vertex label and a property will uniquely identify a property for use in traversals. Edge properties are expensive to update, as because the whole edge with all its properties are deleted and recreated to update edge properties. Use edge properties only in situations that warrant their use.

```java
// ********
// PROPERTY KEYS
// ********
// SYNTAX:
// propertyKey('name').
//    type().
//    [ single() | multiple() ].
//    [ ttl ].
//    [ properties(metadata_property) ].
//    [ ifNotExists() ].
//    [ create() | add() | describe() | exists() ]
// DEFAULT IS SINGLE CARDINALITY
// ********
// SINGLE CARDINALITY PROPERTY KEY
schema.propertyKey('personId').Int().single().create()
schema.propertyKey('mealId').Int().single().create()
schema.propertyKey('itemId').Int().single().create()
schema.propertyKey('recipeId').Int().single().create()
schema.propertyKey('bookId').Int().single().create()
schema.propertyKey('ingredId').Int().single().create()
schema.propertyKey('homeId').Int().single().create()
schema.propertyKey('storeId').Int().single().create()
schema.propertyKey('locId').Text().single().create()
schema.propertyKey('stateId').Int().single().create()
schema.propertyKey('cityId').Int().single().create()
schema.propertyKey('sensorId').Int().single().create()
schema.propertyKey('name').Text().single().create()
schema.propertyKey('gender').Text().single().create()
schema.propertyKey('calGoal').Int().single().create()
schema.propertyKey('macroGoal').Text().single().create()
schema.propertyKey('publishYear').Int().single().create()
schema.propertyKey('ISBN').Text().single().create()
// PROPERTY KEY WITH TTL
schema.propertyKey('bookDiscount').Text().ttl(604800).create()
schema.propertyKey('instructions').Text().single().create()
schema.propertyKey('notes').Text().single().create()
schema.propertyKey('type').Text().single().create()
schema.propertyKey('servAmt').Text().single().create()
schema.propertyKey('macro').Text().single().create()
schema.propertyKey('calories').Int().single().create()
schema.propertyKey('geoPoint').Point().withGeoBounds().create()
schema.propertyKey('address').Text().single().create()
schema.propertyKey('amount').Text().single().create()
// MULTIPLE CARDINALITY PROPERTY KEY
schema.propertyKey('nickname').Text().multiple().create()
```
Using DataStax Enterprise advanced functionality

```java
// MULTIPLE CARDINALITY PROPERTY KEY WITH SINGLE CARDINALITY META-PROPERTY
schema.propertyKey('cuisine').Text().multiple().create()

// MULTIPLE CARDINALITY PROPERTY KEY WITH MULTIPLE CARDINALITY META-PROPERTY
schema.propertyKey('since').Int().single().create() // meta-property
schema.propertyKey('badge').Text().multiple().properties('since').create()

// MULTIPLE CARDINALITY PROPERTY KEY WITH MULTIPLE CARDINALITY META-PROPERTY
schema.propertyKey('startYear').Int().multiple().create() // meta-property
schema.propertyKey('endYear').Int().multiple().create() // meta-property
schema.propertyKey('country').Text().multiple().properties('startYear','endYear').create()

// EDGE PROPERTIES
schema.propertyKey('numServ').Int().single().create()

// Property Keys
// Check for previous creation of property key with ifNotExists()
// Property keys can be created with either single or multiple cardinality with single() or multiple(). The default is single cardinality which does not have to be specified, but it can be explicitly stated as in the example.

// Example of multiple property
// schema.propertyKey('nickname').Text().multiple().create();

// Example meta-property added to property:
// schema.propertyKey('livedIn').Text().create()

// Property keys can be checked for prior existence with ifNotExists()

Property keys can be checked for prior existence with ifNotExists(). Property keys can be created with either single or multiple cardinality with single() or multiple(). The default is single cardinality which does not have to be specified, but it can be explicitly stated as in the example.

Meta-properties, or properties of properties, can be created using propertyKey() followed by properties(). The property key must exist prior to the creation of a
```
Using DataStax Enterprise advanced functionality

meta-property. Meta-properties cannot be nested, i.e., a meta-property cannot have a meta-property. In this example, country is the property that has a meta-property livedIn. This property and meta-property are used to represent the countries that an author has lived in at various times in their life.

```json
{
    "name": "Julia Child",
    "gender": "F",
    [ "country": "United States", "livedIn": "1929-1949" },
    { "country": "France", "livedIn": "1949-1952" } ],
    "authored": [{
        "book":{
            "label": "book",
            "bookTitle": "Art of French Cooking Volume One",
            "publishDate": 1968
        },
        "book":{
            "label": "book",
            "bookTitle": "The French Chef Cookbook",
            "publishDate": 1968,
            "ISBN": "0-394-40135-2"
        }
    }],
    "created": [{
        "type": "recipe",
        "recipeTitle": "Beef Bourguignon",
        "instructions": "Braise the beef."
    },
    {
        "type": "recipe",
        "recipeTitle": "Salade Nicoise",
        "instructions": "Break the lettuce into pieces."
    }]
}
```

// RECIPE SCHEMA

// To run in Studio, copy and paste all lines to a cell and run.

// To run in Gremlin console, use the next two lines:
// script = new File('/tmp/RecipeSchema.groovy').text; []
// :> @script

// Property Keys
// Check for previous creation of property key with ifNotExists()
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().create()
Using DataStax Enterprise advanced functionality

```java
// Using DataStax Enterprise advanced functionality

// Example of creating a vertex label with properties

// Vertex Labels

// Example of creating vertex label with properties

// Edge Labels

// Secondary

// Search
```

DSE 6.7 Developer Guide (Latest version)
Creating vertex label schema

Vertex labels (page 659), as discussed in the data model (page 434), define the vertex id and associated property keys for each type of vertex created. Property keys must be created (page 454) prior to using them in vertex label creation. Vertex label schema can be created with create() or property keys can be added to existing schema with add(). Vertex labels can be created with a specific Time-To-Live (TTL) value, or prior existence of a vertex label can be checked using ifNotExists().

A key component of a vertex label is the vertex id which identifies the data locality with which vertices with a particular vertex label will be stored. User-defined vertex ids (UDV ids) are analogous to a primary key in RDBMS, and map directly to the underlying data representation of the graph in DataStax’s distribution of Apache Cassandra™. UDV ids identify the unique property keys that define the partitionKey and clusteringKey of a vertex label. The values associated with the UDV ids define the node in a DSE cluster that vertices will be partitioned (partitionKey), and the order in which the data is stored in the associated tables (clusteringKey).

A UDV id can be defined using three different arrangements for the vertex id:

**Single-key: composed of a single property in a partitionKey**
Maps every instance of a vertex label to a distinct DSE partition and distributes the data around the DSE cluster based on DSE distribution methodologies.

**Multiple-key: composed of more than one property in a partitionKey**
Maps a particular vertex label and associated properties to a distinct DSE partition, but contains more than one property key to identify the uniqueness.

**Composite key: composed of a partitionKey and a clusteringKey**
Includes both a partitionKey which maps a particular vertex label to a distinct DSE partition and one or more clusteringKeys to group data within a partition.

Caution: Keep in mind that UDV ids must be globally unique within the graph.

Auto-generated vertex ids also exist, but are discouraged. If a partitionKey or clusteringKey are not specified, an auto-generated vertex id will be created that assigns values to two internal properties, community_id as a partitionKey and member_id as a
Using DataStax Enterprise advanced functionality

clusteringKey. Because a unique id is created for every vertex, duplicate elements can be created with the same property values accidentally, leading to confusion.

**Note:** Auto-generated vertex ids are deprecated with DSE 6.0.

Caching can improve query performance and is configurable. DSE Graph has two types of cache: adjacency list cache and index/property cache. Either edges or properties can be cached using the `cache()` option with `vertexLabel()` *(page 659).* Caching can be configured for all edges, all properties, or a filtered set of edges. Vertices are not cached directly, but caching properties and edges that define the relationship between vertices essentially accomplishes the same operation. The best use of caching is for static values.

Property caching is enabled if indexes exist and are used in the course of queries. Full graph scan queries will not be cached. If an index does not exist, then caching does not occur. Adjacency list caching is enabled if caching is configured for edges.

The caches are local to a node and data is loaded into cache when it is read with a query. Both caches are set to a default size of 128 MB in the `dse.yaml` file. The settings are `adjacency_cache_size_in_mb` and `index_cache_size_in_mb`. Both caches utilize off-heap memory implemented as Least Recently Used (LRU) cache.

Graph cache is local to each node in the cluster, so the cached data can be different between nodes. Thus, a query can use cache on one node, but not on another. The caches are updated only when the data is not found. Graph caching does not have any means of eviction. No flushing occurs, and the cache is not updated if an element is deleted or modified. The cache will only evict data based on the time-to-live (TTL) value set when the cache is configured for an element. Set a low TTL value for elements *(property keys *(page 657)*, vertex labels *(page 659)*, edge labels *(page 652)*) that change often to avoid stale data.

Caching is intended to help make queries more efficient if the same information is required in a later query. For instance, caching the `calories` property for `meal_item` vertices will improve the retrieval of a query asking for all meal items with a calorie count less than 850 calories. However, it is useful only for rarely changed graph data, and queries run in the same sort order. Caching `calories` for the query above will not reduced query latency if the query asks for all recipes with a calorie count greater than 850 calories.

DSE Graph limits the number of vertex labels to 200 per graph.

**Prerequisites:**

*Create property key schema *(page 454).*

**User-defined vertex ids**

1. Create a vertex label with a single-key vertex id of `sensorId`. The property key `sensorId` must exist prior to use in creating the vertex label and cannot be a multiple cardinality property.

   ```
   schema.vertexLabel('FridgeSensor').partitionKey('sensorId').create()
   ```
Using DataStax Enterprise advanced functionality

This vertex id will store data based on the unique sensorId value for each FridgeSensor, distributing the data throughout the entire DSE cluster.

2. Create a vertex label with a composite key vertex id of cityId and clustering key sensorId.

```java
schema.vertexLabel('FridgeSensor').partitionKey('cityId').clusteringKey('sensorId').create()
```

The vertex id in this example will store all data for FridgeSensors with a particular cityId on the same partition, but order the data based on the sensorId. If the city has a large number of sensors, the table storing these vertices could grow quite big.

3. Create a vertex label with a multiple-key vertex id using both cityId and sensorId as part of the partitioning key.

```java
schema.vertexLabel('FridgeSensor').partitionKey('cityId', 'sensorId').create()
```

This vertex id will hash both property keys before distributing the data in the cluster, so that each is uniquely stored based on more information.

### Auto-generated vertex ids

4. If no partitionKey or clusteringKey are specified, an auto-generated vertex id will be generated when data is created:

```java
schema.vertexLabel('anAutoId').create()
```

The vertex id consists of the label plus the two attributes community_id and a member_id:

```
{~label=anAutoId, community_id=1270013568, member_id=0}
```

### Associating property keys with vertex labels

5. Properties can be defined in either the `create()` or `add()` statement:

```java
schema.vertexLabel('book').partitionKey('bookId').create()
schema.vertexLabel('book').properties('name','publishYear','ISBN','bookDiscount').add()
```

or

```java
schema.vertexLabel('book').partitionKey('bookId').properties('publishYear', 'ISBN', 'name', 'bookDiscount').create()
```

### Caching

6. Cache all properties for person vertices up to an hour (3600 seconds):
Using DataStax Enterprise advanced functionality

Enabling property cache causes index queries to use an index cache for the specified vertex label.

7. Cache both incoming and outgoing created edges for `person` vertices up to a minute (60 seconds):

```java
schema.vertexLabel('person').cache().bothE('created').ttl(60).add()
```

The vertex labels used for the DSE QuickStart example used throughout the documentation:

```java
// ********
// VERTEX LABELS
// ********
// SYNTAX:
// schema.vertexLabel('vertexLabel').
//    [ partitionKey(propertyKey,
//                  [ partitionKey(propertyKey) ]) ].
//    [ clusteringKey(propertyKey) ].
//    [ ttl ].
//    [ properties(property, property) ].
//    [ index ].
//    [ cache() ].
//    [ ifNotExists() ].
//    [ create() | add() | describe() | exists() ]
// ********

// SINGLE-KEY VERTEX ID
schema.vertexLabel('person').partitionKey('personId').create()
schema.vertexLabel('person').properties('name','nickname','gender','calGoal','macroGoal').add()
schema.vertexLabel('book').partitionKey('bookId').create()
schema.vertexLabel('book').properties('name','publishYear','ISBN','bookDiscount').add()
schema.vertexLabel('meal_item').partitionKey('itemId').create()
schema.vertexLabel('meal_item').properties('name','servAmt','macro','calories').add()
schema.vertexLabel('ingredient').partitionKey('ingredId').create()
schema.vertexLabel('ingredient').properties('name').add()
schema.vertexLabel('home').partitionKey('homeId').create()
schema.vertexLabel('home').properties('name','address').add()
schema.vertexLabel('store').partitionKey('storeId').create()
schema.vertexLabel('store').properties('name','address').add()
schema.vertexLabel('location').partitionKey('locId').create()
schema.vertexLabel('location').properties('name','geoPoint').add()
schema.vertexLabel('recipe').partitionKey('recipeId').create()
schema.vertexLabel('recipe').properties('name','cuisine','instructions','notes').add()
```
Using DataStax Enterprise advanced functionality

```java
// MULTIPLE-KEY VERTEX ID
schema.vertexLabel('meal').partitionKey('type', 'mealId').create()
// COMPOSITE KEY VERTEX ID
schema.vertexLabel('fridgeSensor').partitionKey('stateId', 'cityId').clusteringKey('sensorId').create()
```

Creating edge label schema

Edge labels (page 652), as discussed in the data model (page 434), define the name of the edge label, incoming and outgoing vertex labels connected by an edge, and associated property keys for the edge. Property keys (page 454) and vertex labels (page 462) must be created prior to using them in edge label creation. Edge label schema can be created with `create()`, and property keys or connections can be added to existing schema with `add()`. The cardinality of an edge label is multiple cardinality by default, but single cardinality can be assigned. Edge labels can be created with a specific Time-To-Live (TTL) value, or prior existence of an edge label can be checked using `ifNotExists()`.

A key component of an edge label is the `connection()` which identifies the two types of vertices that are connected with an edge using the edge label. The connection has directionality specified by the order, going from the outgoing vertex label listed first to the incoming vertex label listed second.

**Prerequisites:**

Create property key schema (page 454).

**Single and multiple cardinality edge labels**

1. Define a multiple cardinality edge label:

   The name of the edgeLabel is `ate`, it is defined with multiple cardinality explicitly stated with `multiple()`:

   ```java
   schema.edgeLabel('ate').multiple().create()
   ```

   Since multiple cardinality is the default, `multiple()` can be omitted:

   ```java
   schema.edgeLabel('ate').create()
   ```

   To check if an edge label already exists before creating it, use `ifNotExists()`:

   ```java
   schema.edgeLabel('knows').multiple().create().ifNotExists().create()
   ```

2. Define a single cardinality edge label:

   The name of the edgeLabel is `has`, it is defined with single cardinality:

   ```java
   schema.edgeLabel('has').single().create()
   ```
Since single cardinality is not the default, `single()` must be included.

### Associating property keys with edge labels

3. Properties can be defined in either the `create()` or `add()` statement:

```java
schema.edgeLabel('ate').multiple().create()
schema.edgeLabel('ate').properties('mealDate').add()
```

or

```java
schema.edgeLabel('ate').multiple().properties('mealDate').create()
```

**Important:** If cardinality (`single` or `multiple`) or `properties()` are specified, those options must precede the `connection()` option in the schema statement.

### Connections

4. Connections can be defined in either the `create()` or `add()` statement:

```java
schema.edgeLabel('ate').multiple().create()
schema.edgeLabel('ate').connection('person', 'meal').add()
```

or

```java
schema.edgeLabel('ate').multiple().connection('person', 'meal').create()
```

5. Multiple connections between vertex labels can be created for the same edge label:

```java
schema.edgeLabel('includedIn').connection('recipe', 'meal').add()
schema.edgeLabel('includedIn').connection('meal', 'book').add()
schema.edgeLabel('includedIn').connection('recipe', 'book').add()
```

These schema statements will allow edges to be inserted that use the edge label `includedIn` between recipes and meals, meals and books, and recipe and books.

### Order restrictions

6. If cardinality (`single` or `multiple`) or `properties()` are specified, those options must precede the `connection()` option in the schema statement.

**Multiple steps:**

```java
schema.edgeLabel('includedIn').multiple().create()
schema.edgeLabel('includedIn').properties('amount').add()
schema.edgeLabel('includedIn').connection('recipe', 'meal').add()
schema.edgeLabel('includedIn').connection('meal', 'book').add()
schema.edgeLabel('includedIn').connection('recipe', 'book').add()
```
Using DataStax Enterprise advanced functionality

```java
schema.edgeLabel('includedIn').connection('ingredient', 'recipe').add()
```

Single step:

```java
schema.edgeLabel('includedIn').multiple().properties('amount').connection('recipe', 'meal').add()
```

**Edge labels with Time-To-Live (TTL)**

7. Edge labels can be defined with a Time-To-Live (TTL) value; once the specified time is reached, the value is deleted from the graph. Define an edge label with TTL:

   The name of the edge label is `createDate` is defined with a TTL of 60 seconds with `ttl()`:

```java
schema.edgeLabel('createDate').ttl(60).create()
```

The edge labels used for the DSE QuickStart example used throughout the documentation:

```java
// ********
// EDGE LABELS
// ********
// SYNTAX:
// schema.edgeLabel('edgeLabel').
//   [ single() | multiple() ].
//   [ connection( outVertex, inVertex) ].
//   [ ttl ].
//   [ properties(property[, property]) ].
//   [ ifNotExists() ].
//   [ create() | add() | describe() | exists() ]
// DEFAULT IS MULTIPLE CARDINALITY
// ********

schema.edgeLabel('ate').multiple().create()
schema.edgeLabel('ate').properties('mealDate').add()
schema.edgeLabel('ate').connection('person', 'meal').add()
schema.edgeLabel('knows').multiple().create()
schema.edgeLabel('knows').properties('since').add()
schema.edgeLabel('knows').connection('person', 'person').add()
schema.edgeLabel('includes').multiple().create()
schema.edgeLabel('includes').properties('numServ').add()
schema.edgeLabel('includes').connection('meal', 'meal_item').add()
schema.edgeLabel('includedIn').multiple().create()
schema.edgeLabel('includedIn').properties('amount').add()
schema.edgeLabel('includedIn').connection('recipe', 'meal').add()
schema.edgeLabel('includedIn').connection('meal', 'book').add()
schema.edgeLabel('includedIn').connection('recipe', 'book').add()
schema.edgeLabel('includedIn').connection('ingredient', 'recipe').add()
schema.edgeLabel('created').multiple().create()
schema.edgeLabel('created').properties('createDate').add()
schema.edgeLabel('created').connection('person', 'recipe').add()
```
Indexing

Indexes play a significant role in making DSE Graph queries performant. Graph queries that must traverse the entire graph to find information will have poor performance, which explains why full-scan queries are disallowed in production environments. Two aspects of querying a graph can be improved with indexing: the initial vertex or vertices from which to start a traversal, and the narrowing of the edges and vertices to traverse from this starting point. DSE Graph implements two types of indexes, global indexes and vertex-centric indexes (VCIs) to address these different aspects of query processing. Global indexes are used to find the starting point for a query and involve finding a matching value for a value of a vertex property. Vertex-centric indexes are used to narrow down the scope of a query after a starting point is defined.

Global indexing overview

Global indexes identify the starting point for a graph traversal query using a vertex label (page 462) and a property (page 454). It is important to understand that graph queries must start from a vertex, not an edge. Although a vertex-centered index using an edge property can be used to narrow a traversal, a traversal cannot start from an edge. In a distributed graph database like DSE Graph, the most efficient traversal would start with a vertex identified by its vertex id (page 462), such as this query that uses the vertex id for Julia Child:

```g.V([\~-label\:'person', \'personId\:1])```

However, identifying a vertex by vertex id is rather restrictive. Using a vertex label and a property in a traversal allows DSE Graph to identify the DSE node where the vertex data resides without reading all data from all DSE nodes. Most graph queries will first use a global index to find a starting vertex with a friendlier property:

```g.V().has('person', 'name', 'Julia Child')```

Since the property name is not part of the vertex id, an index is required to match the search conditions with the correct vertex, and that index is a global index.
Global indexing in DSE Graph can be accomplished with one of three DSE indexing methods: a materialized view (MV), a search index (page 312), or a secondary index.

Materialized views are tables generated from a base table to provide a query based on a different primary key than the base table. This type of index is best used for values of high cardinality of nearly unique values, or high selectivity. Selectivity is derived from cardinality, using the following formula:

\[
\text{selectivity} = \left( \frac{\text{cardinality}}{\text{number of rows}} \right) \times 100\%
\]

In general, low cardinality results in low selectivity, and high cardinality results in high selectivity. Searching materialized views yields similar response times to searching base tables, although writing the data incurs a small time penalty. When data is written or updated in the graph, the index information is updated in the MV table along with the graph tables. A consequence of using a MV table is higher write latencies, but results in lower read latencies for graph traversals.

Search indexes are used when textual, numeric or geospatial indexing are required and rely on DSE Search (page 312). Since graph data is stored in DSE database tables, one search core is available per vertex label. For each vertex label that will be indexed with search, all properties must be added to a single search index named search. Because search is implemented with DSE Search, all data types can be indexed. For two indexing options, full text and string, the property key must be defined, as different indexing results. Full text indexing performs tokenization and secondary processing such as case normalization. Full text indexing is useful for queries where partial match of text is required, and lends itself to regular expressing (regEx) searching. String indexing is useful for queries where an exact string is sought and no tokenization is required, similar to Solr faceting. This type of index is best for low selectivity, but lends itself to fuzzy matching for both tokenized and non-tokenized indexing.

Secondary indexing in DSE Graph follows the same rule of thumb as DSE secondary indexing. This type of index is meant for lower cardinality values, or alternatively, for low selectivity values. The number of values for indexing should number in the tens to hundreds at most; for instance, searching by country is a good candidate for secondary indexing. In addition, only equality conditions can be used to match values, and no ordering or range queries on values can be used. If more complex value matching is required, search indexes are the superior choice.

To summarize global indexes:

- Composite index keys are not currently supported in DSE Graph.

Vertex-centric indexing (VCI) overview

Vertex-centric indexes (VCI) are used to narrow the traversal based on an additional property. Global indexes can be applied across all vertices with a specified vertex label, as opposed to VCIs which apply to a filtered set of vertices. Vertex-centric indexes are especially important in reducing the complexity of a traversal from \(O(n)\) to \(O(1)\) or \(O(\log n)\), using Big O notation. Two types of VCI exist, edge indexes and property indexes. Edges indexes are useful for traversing edges based on associated properties, to avoid linear
scans of all incident edges of a vertex, since traversing all incident edges can quickly compound the cost of a traversal if many incident edges exist. For instance, an edge index is useful in picking just certain edges once a global index has initiated the traversal at a particular vertex (in this case, Julia Child):

```
g.V().has('person', 'name', 'Julia Child').outE('created').has('createDate', gt(1960-01-01))
```

Property indexes are created to index meta-properties. Property indexes can support both equality and inequality predicates, and are useful in cases where a range of values must be returned by a query. This example will find all the countries that Fritz Streiff lived in and order them by the year he started living in the country:

```
g.V().has('person', 'name', 'Fritz Streiff').properties('country').has('startYear', order().by(decr))
```

Vertex-centric indexing in DSE Graph is accomplished with materialized views (MVs) for both edge and property indexes, and have the same properties as described above for global indexes.

**Indexing best practices**

The most important fact to remember is that a search index is the only choice for indexing two or more properties that define the starting point for a query. Multiple materialized view or secondary indexes cannot be used for global indexing. For instance, `g.V().has('person', 'gender', 'F').has('person', 'country', 'France')` will only use one index, not both, if the indexes are materialized view or secondary indexes. If a search index is defined, both properties, `country` and `gender`, are used. Once the starting point is defined, a vertex-centric index can be used to narrow the query.

More than one index can be created on the same property, such as creating both a materialized view (MV) index and a search index on the property `amount`. The DSE Graph query optimizer automatically uses the appropriate index when processing a query; designation of an index type to use is not a feature. The order of preference that DSE Graph uses is MV index > secondary index > DSE Search index to ensure best performance. However, choosing the optimal type of index is key to good performance. For instance, it is important to understand the limitations of materialized views, and base the number of MV indexes on that understanding. See Understanding materialized views.

Different index types may be created on different properties as appropriate, based on the selectivity. In general, secondary indexes in DSE Graph are limited in usefulness, for the same reasons that constrict their general use in DSE. Materialized view indexing should be the first choice, unless textual search is required and a search index is selected.

If a search index is created, be aware that building the index can take time, and that until the index is available, queries that depend on the index can fail. Applications that create schema, immediately followed by data insertion that require search indexes will likely experience errors. Also, queries that use search indexes should be run on DSE Search-enabled nodes in the cluster. Search indexes also require extra resources. Each index allocates a minimum of 256MB of memory by default, and each index will require
Using DataStax Enterprise advanced functionality

two physical cores. For a typical 32GB node, 16 search indexes would be a reasonable number to create.

Queries that use textual predicates (regex, tokenRegex, prefix, tokenPrefix, token, and eq/neq) can be accomplished without DSE search indexes. However, such queries will not make use of secondary or materialized indexes and will instead use full graph scans to return results. By default, Production mode does not allow full graph scans, so such queries will fail. If such matching search methods are required, search indexes are strongly suggested.

Caution: tokenRegex will display case insensitivity in queries, whether a search index is used or not.

Textual search indexes are by default indexed in both tokenized (TextField) and non-tokenized (StrField) forms. This means that all textual predicates (token, tokenPrefix, tokenRegex, eq, neq, regex, prefix) will be usable with all textual vertex properties indexed. Practically, search indexes should be created using the asString() method only in cases where there is absolutely no use for tokenization and text analysis, such as for inventory categories (silverware, shoes, clothing). The asText() method is used if searching tokenized text, such as long multi-sentence descriptions. The query optimizer will choose whether to use analyzed or non-analyzed indexing based on the textual predicate used.

It is possible to modify search index schema (page 324) to change search characteristics. Although DSE Graph will not overwrite these out-of-band changes, it is recommended that you do not add or remove fields in this manner - only DSE Graph commands should be used. The general use of this feature is mainly to change the behavior of a search, such as adding case sensitivity to a type of search.

**Adding index schema**

All index schema is based on previously created properties and vertex labels and added to existing schema with add().

**Prerequisites:**

Create property key schema (page 454) and vertex label schema (page 462).

**Materialized index**

1. Create a materialized view index for a global index:

```
schema.vertexLabel('person').index('byName').materialized().by('name').add()
```

Identify the vertex label and property key for the index, in the vertexLabel() and by() steps, respectively. In the index() step, name the index. The materialized() step identifies the index as a materialized view index.
2. Create a materialized view for an edge index. Edges indexes are vertex-centric to a particular vertex label. For instance, the example below indexes anything that a reviewer rates.

```java
schema.vertexLabel('person').index('ratedByStars').outE('reviewed').by('stars').ifNotExists().add()
```

Identify the vertex label and property keys for the index, in the `vertexLabel()` and `by()` steps, respectively. In the `index()` step, name the index. The `outE()` step is used to define the direction of the edge.

**Note:** For edge and property indexes, the `materialized()` step is not included, because all edge indexes are created as materialized views.

3. Create an edge index that indexes both incoming and outgoing edges:

```java
schema.vertexLabel('person').index('ratedByStars').bothE('reviewed').by('stars').ifNotExists().add()
```

Identify the vertex label and property keys for the index, in the `vertexLabel()` and `by()` steps, respectively. In the `index()` step, name the index. The `bothE()` step is used to define the direction of the edge.

4. Create a materialized view for a property index:

```java
schema.vertexLabel('person').index('byStartYear').property('country').by('startYear').add()
```

Identify the vertex label for the index in the `vertexLabel()` step. In the `index()` step, name the index. The `property()` and `by()` steps identify the property key and its meta-property, respectively.

**Note:** For edge and property indexes, the `materialized()` step is not included, because all edge indexes are created as materialized views.

**Search index**

5. Most commonly, a search index is created with multiple columns, chained together as multiple `by()` steps in one statement:

```java
schema.vertexLabel('recipe').index('search').search().by('name').by('instructions').add()
```

Identify the vertex label and property keys for the index, in the `vertexLabel()` and `by()` steps, respectively. In the `index()` step, name the index search; only this naming convention can be used. The `search()` step identifies the index as a search index. Since no option is specified, the property will be indexed both as Text and String.
Using DataStax Enterprise advanced functionality

**Tip:** Only one search index can be created per vertex label.

Textual search indexes are by default indexed in both tokenized (TextField) and non-tokenized (StrField) forms. This means that all textual predicates (token, tokenPrefix, tokenRegex, eq, neq, regex, prefix) will be usable with all textual vertex properties indexed. Practically, search indexes should be created using the `asString()` method only in cases where there is absolutely no use for tokenization and text analysis, such as for inventory categories (silverware, shoes, clothing). The `asText()` method is used if searching tokenized text, such as long multi-sentence descriptions. The query optimizer will choose whether to use analyzed or non-analyzed indexing based on the textual predicate used.

6. Create a search index with only one property key indexed as Text():

```java
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().add()
```

7. A search index can specify the string indexing option along with the text indexing option:

```java
schema.vertexLabel('recipe').index('search').search().by('instructions').asString().by('name').asText().add()
```

This example specifically states how the search index will be constructed for each property.

8. Search indexes can also include non-text data types:

```java
schema.vertexLabel('recipe').index('search').search().by('year').by('name').asString().asInt().add()
```

Data types other than text are inferred from the schema and DSE Search uses a comparable Solr data type. In this example, `year` is indexed as an integer.

**Caution:** The Decimal data type will index as a SolrDecimalStrField. Use Int, Long, Float, or Double to ensure that the Solr data types are used for sorting and range querying.

9. Create a search index for geospatial data:

```java
schema.propertyKey("coordinates").Point().single().create()
schema.propertyKey("name").Text().single().create()

schema.vertexLabel("place").properties("coordinates", "name").create()
schema.vertexLabel("place").index("search").search().by("name").asText().by("coordinate").add()
```
In this example, the property `coordinates` is a point defining a longitude and latitude. The search index includes `coordinates` without a qualifying `asText()` or `asString()` method. See Geospatial Schema (page 456) for additional information.

### 10. While DSE Graph natively supports geospatial searches, performing them without a Search index does not scale as the number of vertices in the graph increases. Doing such queries without a search index results in very inefficient query performance because full scans are required. DSE Search indexes can index points and linestrings, but not polygons.

```csharp
//SEARCH INDEX ONLY WORKS FOR POINT AND LINestring
schema.vertexLabel('location').index('search').search().by('point').add()
schema.vertexLabel('lineLocation').index('search').search().by('line').add()
```

Without a search index, spatial queries always return exact results. DSE Search indexes, however, can trade off performance for accuracy.

**Note:** A point of confusion can occur if the same geospatial query is run with or without a DSE Search index. Without a search index, geospatial queries always return exact results. DSE Search indexes, however, trade off write performance and index size for query accuracy with two tunable parameters, `maxDistErr` (default: 0.000009) and `distErrPct` (default: 0.025). Inconsistent results in these two cases are due to the distance calculation algorithm variation of the default values of these parameters. DSE Graph can pass values for these two parameters when creating the search index. Change `maxDistErr` in `withError(maxDistErr, distErrPct)` to 0.0 to force both index-backed and non-index-backed queries to yield the same value:

```csharp
schema.vertexLabel('location').index('search').search().by('point').withError(0.000009, 0.0).add()
```

### 11. Create a search index for timestamp data:

```csharp
schema.propertyKey('review_ts').Timestamp().create()
schema.propertyKey('name').Text().create()
schema.vertexLabel('rating').properties('name', 'review_ts').create()
schema.vertexLabel('rating').index('search').search().by('name','review_ts').add()
```

### Secondary index

### 12. Create a global index as a secondary index:

```csharp
schema.vertexLabel('recipe').index('byRecipe').secondary().by('name').add()
```
Using DataStax Enterprise advanced functionality

Identify the vertex label and property key for the index, in the `vertexLabel()` and `by()` steps, respectively. In the `index()` step, name the index. The `secondary()` step identifies the index as a secondary index.

The edge labels used for the DSE QuickStart example used throughout the documentation:

```java
// ********
// VERTEX INDEX
// ********
// SYNTAX:
// index('index_name').
//    [secondary() | materialized() | search()].
//    by('propertykey_name').
//    [ asText() | asString() ].
//    add()
// ********

schema.vertexLabel('person').index('byName').materialized().by('name').add()
schema.vertexLabel('meal_item').index('byName').materialized().by('name').add()
schema.vertexLabel('ingredient').index('byName').materialized().by('name').add()
// schema.vertexLabel('recipe').index('byCuisine').materialized().by('cuisine').add()
// schema.vertexLabel('book').index('byName').materialized().by('name').add()

schema.vertexLabel('meal').index('byType').secondary().by('type').add()

schema.vertexLabel('recipe').index('byStars').inE('reviewed').by('stars').ifNotExists().add()

// ********
// EDGE INDEX
// ********
// SYNTAX:
// index('index_name').
//    [outE('edgeLabel') | inE('edgeLabel') ].
//    by('propertykey_name').
//    add()
// ********

schema.vertexLabel('recipe').index('byStars').inE('reviewed').by('stars').ifNotExists().add()
```
Using DataStax Enterprise advanced functionality

```java
// ********
// PROPERTY INDEX using meta-property 'livedIn'
// ********
// SYNTAX:
// index('index_name').
//    property('propertykey_name').
//    by('meta-propertykey_name').
//    add()
// ********

// PROPERTY INDEX using meta-property 'livedIn'

// ********
// SYNTAX:
// index('index_name').
//    property('propertykey_name').
//    by('meta-propertykey_name').
//    add()
// ********

Dropping graph schema

Schema can be dropped (deleted) using the `drop()` step; see `propertyKey (page 657)`, `vertexLabel (page 659)`, and `edgeLabel (page 652)` for more information.

Drop schema

- To drop the schema and all data without dropping the graph, use a `drop()` step. Running `describe()` after will verify that the schema is dropped. After the schema is dropped, new schema and data can be loaded to the graph.

  ```java
  schema.drop()
  =>null
  ```

- To drop schema for a particular property key, specify the property key and then use the `drop()` step:

  ```java
  schema.propertyKey('aBadProperty').drop()
  ```

  **Warning:** To drop a property key for a vertex label that has a materialized view index, additional steps are required to prevent data loss or cluster errors. **Drop any materialized indexes (page 478) for the vertex label, drop the property key, and then rebuild the schema for the materialized view indexes. The MV index data will be recreated from the base table, with the exception of the data for the dropped property key.**
Using DataStax Enterprise advanced functionality

- To drop schema for a particular vertex label, specify the vertex label and then use the `drop()` step:

  ```java
  schema.vertexLabel('aBadVertexLabel').drop()
  ```

### Drop index

- To drop an index from the schema, such as the `byMeal` index, identify the index by name. Use `describe()` to examine all indexes for the desired vertex label first.

  ```java
  gremlin> schema.vertexLabel('meal').describe()
  ==>schema.vertexLabel('meal').properties("name").create()
  schema.vertexLabel('meal').index('byMeal').materialized().by('name').add()
  ```

  - Using the vertex label and index name, remove the index. Running `describe()` again will verify that the index is removed.

    ```java
    gremlin> schema.vertexLabel('meal').index('byMeal').drop()
    ==>null
    ```

  - Remove only one property from a search index:

    ```java
    schema.vertexLabel('author').index('search').search().properties('nick_name').drop()
    schema.vertexLabel('author').describe()
    ```

---

### Managing graph data

#### Data Formats

DSE Graph handles many different formats. Each format has advantages, and the choice of which to use is a matter of preference.

DSE Graph can ingest comma-delimited (CSV), JSON, or text data using the DSE Graph Loader. The DSE Graph Loader is a tool that can transform data upon ingestion if required, but complex transformations will make the data ingestion slow.

Gryo is a compact binary format, the fastest and most space-saving format for populating DSE Graph with data. This method can be used if the data has been generated from DSE Graph or another graph database.

GraphSON is a JSON style file format that passes both data and schema in human-readable format. It is easy to work with, as the structure is evident, but it results in large files. GraphML is an XML-based file format that similarly passes both data and schema. It is widely supported by graph-related tools and libraries making it a solid interchange format for DSE Graph. GraphSON and GraphML are common formats and useful for importing data into DSE Graph from another graph database.
CSV

A CSV file is a common file format that can be input into DSE Graph. A sample of CSV graph data:

<table>
<thead>
<tr>
<th>personId</th>
<th>name</th>
<th>nickname</th>
<th>gender</th>
<th>CALORIES</th>
<th>macroGoal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Julia Child</td>
<td>null</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Simone Beck</td>
<td>null</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

JSON

A JSON file is a common file format that can be input into DSE Graph. A sample of JSON graph data:

```json
{
  "personId": 1,
  "country": {
    "value": "USA",
    "startYear": 1930,
    "endYear": 1949
  }
}
```

Gryo

Writing data out of the graph into a Gryo file can be accomplished with a simple `graph.io()` command:

```java
graph.io(gryo()).writeGraph("/tmp/recipe.gryo")
```

GraphSON

Writing data out of the graph into a GraphSON file can be accomplished with a simple `graph.io()` command:

```java
graph.io(graphson()).writeGraph("/tmp/recipe.gson")
```

A sample of the output of a GraphSON file:

```json
{"id":{"@type":"g:Map","@value":{"~label":"recipe","recipeId":{"@type":"g:Int32","@value":2003} } },"label":"recipe","inE":{"created": [{"id":{"@type":"g:Map","@value":{"~label":"created","~out_vertex":{"@type":"g:Map","@value":{"~label":"person","personId":{"@type":"g:Int32","@value":1} } },"~in_vertex":{"@type":"g:Map","@value":{"~label":"recipe","recipeId":{"@type":"g:Int32","@value":2003} } },"~local_id":{"@type":"g:UUID","@value":"f38f9dd1-2978-11e8-8043-6bfe97ac83b9"} } },"outV": {"@type":"g:Map","@value":{"~label":"person","personId":{"@type":"g:Int32","@value":1} } },"properties":{"createDate":{}} },"outE":{"includedIn": [{"id":{"@type":"g:Map","@value":{"~label":"includedIn","~out_vertex":{"@type":"g:Map","@value":{"~label":"recipe","recipeId":{"@type":"g:Int32","@value":2003} } },"~in_vertex":{"@type":"g:Map","@value":{"~label":"book","bookId":{"@type":"g:Int32","@value":1001} } },"~local_id":{"@type":"g:UUID","@value":"f38f9dd1-2978-11e8-8043-6bfe97ac83b9"} } } }
```

Using DataStax Enterprise advanced functionality
Using DataStax Enterprise advanced functionality

GraphML

Writing data out of the graph into a GraphML file can be accomplished with a simple `graph.io()` command:

```java
graph.io(graphml()).writeGraph("/tmp/recipe.gml")
```

A sample of the output of a GraphSON file:

```xml
<?xml version='1.0' encoding='UTF-8'?><graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd"><key id="instructions" for="node" attr.name="instructions" attr.type="string"/><key id="notes" for="node" attr.name="notes" attr.type="string"/><key id="mealId" for="node" attr.name="mealId" attr.type="int"/><key id="gender" for="node" attr.name="gender" attr.type="string"/><key id="ingredId" for="node" attr.name="ingredId" attr.type="int"/><key id="type" for="node" attr.name="type" attr.type="string"/><key id="recipeId" for="node" attr.name="recipeId" attr.type="int"/><key id="bookId" for="node" attr.name="bookId" attr.type="int"/><key id="labelV" for="node" attr.name="labelV" attr.type="string"/><key id="publishYear" for="node" attr.name="publishYear" attr.type="int"/><key id="ISBN" for="node" attr.name="ISBN" attr.type="string"/><key id="personId" for="node" attr.name="personId" attr.type="int"/><key id="labelE" for="edge" attr.name="labelE" attr.type="string"/><key id="amount" for="edge" attr.name="amount" attr.type="string"/>
```
Using DataStax Enterprise advanced functionality

r.name="createDate" attr.type="string"/><graph id="G" 
edgedefault="directed"><node id="{~label=recipe, recipeId=2003}"><data 
key="labelV">recipe</data><data key="instructions">Take a sal 
ad bowl or platter and line it with lettuce leaves, shortly before 
serving. Drizzle some olive oil on the leaves and dust them with 
salt.</data><data key="notes"></data><data key="name">Salade Nicoise</data><data key="recipeId">2003</data></node><node 
id="{~label=recipe, recipeId=2005}"><data key="labelV">recipe</data><data 
key="instructions">Preheat the oven to 375 deg 
rees F. Cook bacon in a large skillet over medium heat until very 
crisp and fat has rendered, 8-10 minutes.</data><data key="notes"> </data><data key="name">Spicy Meatloaf</data><data key="recipeId">2005</data></node><node id="{~label=recipe, recipeId=2006}"><data key="labelV">recipe</data><data 
key="instructions">Saute the shallots, celery, herbs, and seasonings 
ablespoons of the butter for 3 minutes. Add the watercress and let 
it wilt.</data><data key="notes"> </data><data key="name">Oysters Rockefeller</data><data key="recipeId">2006</data></node><node id="{~label=recipe, recipeId=2007}"><data key="labelV">recipe</data><data 
key="instructions">In a heavy-bottomed pot, melt the butter. When it starts to foam, add the onions an 
d thyme and cook over medium-low heat until tender, about 10 minutes.</data><data key="notes">Quick and easy.</data><data key="name">Carrot Soup</data><data key="recipeId">2007</data></node><node id="{~label=recipe, recipeId=2001}"><data key="labelV">recipe</data><data 
key="instructions">Braise the beef. 
aut the onions and carrots. Add wine and cook in a dutch oven at 
425 degrees for 1 hour.</data><data key="notes">Takes a long 
time to make.</data><data key="name">Beef Bourguignon</data><data key="recipeId">2001</data></node><node id="{~label=recipe, recipeId=2002}"><data key="labelV">recipe</data><data 
key="instructions">Peel and cut the eggplant. Make sure you cut 
eyplant into lengthwise slices that are about 1-inch wide, 3-inch 
es long, and 3/8-inch thick.</data><data key="notes">I've made this 13 
times.</data>

Inserting data with traversal API

The **Traversal API** *(page 666)* can be used to insert data into DSE Graph.

1. **Add a vertex:**

   ```java
   g.addV('person').property('personId', 1).property('name', 'Julia Child').property('gender', 'F')
   ```

   The `addV()` step must identify the vertex label, and can be followed by key and value pairs in `property()` steps for any properties that are added.

2. **To add an edge using only the graph API, the two vertices that are connected by the edge can be assigned variables that can be used in the `add()` statement:**

   ```java
   juliaChild = g.V().has('person', 'personId', 1).next()
   ```
Using DataStax Enterprise advanced functionality

artOfFrenchCookingVolOne = g.V().has('book', 'bookId', 1001).next()
g.V(juliaChild).addE('authored').to(artOfFrenchCookingVolOne)

or alternatively, the query can be constructed without variables:

[g.V().has('person','name','Julia Child')
  .addE('authored')
  .to(g.V().has('book','name','The Art of French Cooking, Vol. 1')]

The outgoing vertex, juliaChild, is connected to the incoming vertex, artOfFrenchCookingVolOne with a to() step, to create an authored edge with addE(),

If the edge has properties, the key and value pairs are appended to the addE() statement, similar to the addV() statement with property():

[g.V().has('person','name','Julia Child')
  .addE('created')
  .to(g.V().has('recipe','name','Beef Bourguignon'))
  .property('createDate', '1956-01-10')]

3. A property can also be added to a previously created vertex, like jamieOliver:

jamieOliver = g.V().has('person', 'name', 'Jamie Oliver').next()
jamieOliver.property('gender', 'M').property('nickname', 'jimmy')

A property() value can also be modified by simply doing a similar traversal with a different value.

4. A property can also be added to a previously created edge by constructing a query that finds the edge and changes or adds the property with property():

[g.V().has('person','name','Julia Child')
  .outE('created')
  .has('createDate', '1956-01-10')
  .property('createDate', '1956-09-09')]

Inserting data with graph API

The Graph API (page 641) can be used to insert data into DSE Graph.

1. Add a vertex:

    graph.addVertex(label, 'person', 'personId', 1, 'name', 'Julia Child', 'gender', 'F')

    The literal label followed by the vertex label are the first two items in the statement, followed by key and value pairs for any properties that are added.
2. To add an edge using only the graph API, the two vertices that are connected by the edge must be assigned variables that can be used in the addEdge statement:

```java
juliaChild = graph.addVertex(label, 'person', 'personId', 1, 'name','Julia Child', 'gender', 'F')
juliaChild.addEdge('authored', artOfFrenchCookingVolOne)
```

The outgoing vertex, juliaChild, is connected to the incoming vertex, artOfFrenchCookingVolOne, to create an authored edge.

If the edge has properties, the key and value pairs are appended in the addEdge statement, similar to the addVertex statement:

```java
beefBourguignon.addEdge('includedIn', beef, 'amount', '2 lbs')
```

3. A property can also be added to a previously created vertex, like jamieOliver:

```java
jamieOliver.property('gender', 'M', 'nickname', 'jimmy')
```

4. To add geospatial data:

```java
graph.addVertex(label, 'location', 'name', 'Paris', 'point', Geo.point(2.352222, 48.856614))
graph.addVertex(label, 'lineLocation', 'name', 'ParisLondon', 'line', "LINESTRING(2.352222 48.856614, -0.127758 51.507351)"")
graph.addVertex(label, 'polyLocation','name', 'ParisLondonDublin', 'polygon',Geo.polygon(2.352222, 48.856614, -0.127758, 51.507351, -6.26031, 53.349805))
```

A vertex label is created for location that has a point property. A vertex label is created for lineLocation that has a LineString property. A vertex label is created for polyLocation that has a Polygon property.

**Note:** For all geospatial elements, the withGeoBounds() method limits searches to a default valid range of latitude in degrees from -90 to +90 (South Pole to North Pole) and a valid range of longitude in degrees from -180 to +180 (east to west from the Greenwich Meridian). The point is specified using Geo.point(longitude, latitude) when adding points,
Using DataStax Enterprise advanced functionality

using WellKnownText (WKT) format. Note that is specifies longitude first, then latitude.

Using the DSE Graph Loader

How to load schema and data using the DSE Graph Loader.

DSE Graph Loader overview

DSE Graph Loader is a customizable, highly tunable command line utility for loading graph datasets into DSE Graph from various input sources. It is not included as part of DataStax Enterprise installations and must be installed separately (page 485).

DSE Graph Loader is built to load datasets containing hundreds of millions (10^8) of vertices and billions (10^9) of edges. DSE Graph Loader is efficient, using parallel loading and persistent cache to store vertices, provided a sufficient machine (page 485) is used to run the program.

Data can be loaded from CSV files, JSON files, delimited text (CSV with a header line to identify the fields), text parsed by regular expressions, and binary Grro files. Distributed filesystem support exists to read input files from Hadoop Distributed File Systems (HDFS) and AWS S3 sources. In addition, DSE Graph Loader supports reading input data directly from a JDBC compatible database or a Neo4J database. Input files can be uncompressed or compressed files. All data can be transformed (page 541) upon reading to manipulate the data that is loaded into a graph.

Data from an input source file can be mapped to define vertices or edges, along with properties for both. The mapping script configures loading parameters, defines the input parameters, and identifies the mapping from each input record to graph element. Both vertex and edge properties can be included in the data that is loaded.

DSE Graph Loader processes input data with three stages:

**Preparation**
Reads entire input data to check for graph schema conformity. Suggests graph schema updates, or if enabled, changes graph schema. Supplies statistics about how much data will be added to graph when loaded. The dryrun configuration option (page 485) can be used to stop the loading process at this stage.

**Vertex loading**
Adds or retrieves all of the vertices in the input data and caches them locally to speed up subsequent edge loading.
Vertex validation is enabled unless the data is identified as new data with load_new. If data is new, validation is not executed, and performance improvement will be seen.

**Edge and property loading**
Adds all edges and properties from the input data to the graph.
Edge validation is enabled unless the data is identified as new data with load_new. If data is new, validation is not executed, and performance
improvement will be seen. Another method of handling mixed new and existing data is the use of `isNew()` (page 561) and `exists()` (page 560). If duplicate edges are required, `isNew()` must be used to designate those edges as additive to the edges that already exist.

**Note:** Multiple cardinality input data must have graph schema created prior to data loading.

A critical feature to keep in mind when using DSE Graph Loader is the upsert nature of the underlying DSE database. If a vertex already exists, DSE Graph Loader updates the stored data with the new property values depending on the configuration choices made. Configuration (page 485) can be used to identify if the data loaded is new or will overwrite data that currently exists. Edges will be duplicated if the same edge is loaded multiple times and the edge label is set to the default of multiple cardinality.

It is strongly recommended that graph schema is created (page 447) before loading data using DSE Graph Loader. Without schema, the correct data types for the data are not enforced. Creating indexes (page 472) will greatly speed up the loading process, and are necessary to achieve acceptable performance for loading.

**Installing DSE Graph Loader**

DSE Graph Loader is not included as part of DataStax Enterprise installations. Follow the instructions for installing DSE Graph Loader in the DataStax Installation Guide.

**Configuring DSE Graph Loader**

Before loading data using any of the methods detailed in the next topics, decide which configuration items to include in the mapping script.

The configuration settings can be applied in the command line using a "-" command, like `-read_threads`, or the settings can be included in the mapping script. All configuration settings are shown in the DSE Graph Loader reference (page 549) including security options (page 556).

- The `dryrun` setting will run the DSE Graph Loader with a mapping script, and output the results, but will not execute the loading process. It is useful for spotting potential errors in the mapping script or graphloader command.

```bash
config dryrun: true
```

This command may be more useful to use as a command line option, since it is not common to leave in after checking a mapping script:

```bash
graphloader map.groovy -graph food -address localhost -dryrun true
```

**Notice:** This configuration option discovers schema and suggests missing schema without executing any changes. In DSE 6.0, this option is deprecated and may possibly be removed in a future release.
• The **preparation** setting is a validity checking mechanism. If **preparation** is true, then a sample of the data is analyzed for whether or not the schema is valid. This setting is used in conjunction with **create_schema**. If **create_schema** and **preparation** are both true, then the data is analyzed, compared to the schema, and new schema is created if found missing.

```c
/* CONFIGURATION */
/* Configures the data loader to analyze the schema */
config preparation: true
```

See the table below *(page 486)* for all permutations.

**Notice:** This configuration option validates and creates schema if used in conjunction with **create_schema**. The default will be set to **false**, and this option is deprecated with DSE 6.0. In a future release, it may be removed.

• This example sets **create_schema** to true, so that schema is created from the data. Setting **create_schema** to true is a good method of inputting new data, to get feedback on what schema may be required for the data. It is not recommended for Production data loading.

```c
/* CONFIGURATION */
/* Configures the data loader to create the schema */
config create_schema: true
```

**Notice:** It is strongly recommended that **schema is created** *(page 447)* prior to data loading, so that the correct data types are enforced and indexes created. Setting **create_schema** to true is recommended only for testing. In DSE 6.0, this configuration option is deprecated and will be removed in a future release.

**preparation** and **create_schema** must be considered together.

• The **load_new** setting is used if vertex records do not yet exist in the graph at the beginning of the loading process, such as for a new graph. Configuring **load_new** can significantly speed up the loading process. However, it is important that the user guarantee that the vertex records are indeed new, or duplicate vertices can be created in the graph. Edges that are created in the same script will use the newly created vertices for the outgoing vertex **outV** and incoming vertex **inV**.

```c
config load_new: true
```

**Warning:** Duplicate vertices will be created if **load_new** is set to **false** and the data being loaded contain any vertex that already exists in the graph.

• Setting the number of threads used for loading vertices or edges uses **load_vertex_threads** and **load_edge_threads**, respectively; the default is 0, which will set **load_vertex_threads** to the number of cores divided by 2, and **load_edge_threads** to the number of nodes in the datacenter multiplied by six.
Using DataStax Enterprise advanced functionality

```plaintext
config load_vertex_threads: 3 load_edge_threads: 0

- Multiple configuration settings can be listed together.
  ```plaintext
  config load_new: true, dryrun: true, schema_output: '/tmp/loader_output.txt'
  ```

**What's next:** Load data. *(page 487)*

**Loading data**

DSE Graph Loader can load data from many different input data formats. Pick the option that most resembles your data source:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV</td>
<td>Strict format, with the first line of the file identifying the property keys used in the graph.</td>
<td>Loading CSV data <em>(page 488)</em></td>
</tr>
<tr>
<td>Text</td>
<td>Delimited text data of any format.</td>
<td>Loading TEXT data <em>(page 503)</em></td>
</tr>
<tr>
<td>Text with regular expressions</td>
<td>Delimited text data parsed using regular expressions (regex).</td>
<td>Loading TEXT data using regular expressions (regex) <em>(page 507)</em></td>
</tr>
<tr>
<td>JSON</td>
<td>Data stored in JSON (JavaScript Object Notation) format.</td>
<td>Loading JSON data <em>(page 496)</em></td>
</tr>
<tr>
<td>JDBC-compatible database</td>
<td>Data stored in a JDBC-compatible database</td>
<td>Loading data from a JDBC compatible database. <em>(page 510)</em></td>
</tr>
<tr>
<td>HDFS file</td>
<td>Data file stored in a Hadoop Distributed File System (HDFS) of any format.</td>
<td>Loading data from Hadoop (HDFS) <em>(page 512)</em></td>
</tr>
<tr>
<td>AWS S3 file</td>
<td>Data file stored in AWS S3 storage of any format.</td>
<td>Loading data from AWS S3 <em>(page 515)</em></td>
</tr>
<tr>
<td>Gryo</td>
<td>Data stored in a binary Gryo format.</td>
<td>Loading Gryo data <em>(page 517)</em></td>
</tr>
<tr>
<td>GraphSON</td>
<td>Data stored in GraphSON format.</td>
<td>Loading GraphSON data <em>(page 519)</em></td>
</tr>
<tr>
<td>GraphML</td>
<td>Data stored in GraphML format.</td>
<td>Loading GraphML data <em>(page 521)</em></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

**Note:** Fields that contain NULL, null, or empty fields in text and CSV files will be pruned by DSE Graph Loader. A transform must be used if a different behavior is desired.

**Warning:** When loading user-defined vertex ids (page 462), the vertex cache that DSE Graph Loaders uses will be bypassed to facilitate faster write throughput. The client must ensure vertices are unique because no logic will validate the existence of a vertex with custom ids. To ensure the fastest performance, the DSE Graph configuration option `external_vertex_verify` (page 646) should be set to false.

The DSE Graph Loader also supports loading several files of the same format from a single directory. Example mapping scripts are shown for CSV (page 491) and JSON (page 498), but will work for all formats.

**Loading CSV data**

A common file format for loading graph data is CSV (comma-delimited data). An input CSV file generally identifies the property keys in the first line of the file with a header line. However, the mapping script can also identify the property keys to be read with `header()` in the data input line. If more flexibility is desired, such as manipulation of the vertex labels using `labelField` (page 525), use Loading TEXT data (page 503).

**Mapping several different CSV files**

DSE Graph Loader can load several different CSV files that exist in a directory using the following steps. Sample input data:

```
// For the author.csv file:
name|gender
Julia Child|F
// For the book.csv file:
name|year|ISBN
Simca's Cuisine: 100 Classic French Recipes for Every Occasion|1972|0-394-40152-2
// For the authorBook.csv file:
bname|aname
Simca's Cuisine: 100 Classic French Recipes for Every Occasion|Simone Beck
```

Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

1. If desired, add configuration (page 485) to the mapping script.

2. Specify the data input files. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.
It is important to note that CSV files can have a header line that shows the field names. For example, the `authorInput` will have the following as the first line in the file:

```
name|gender
```

If a `header()` is used in the mapping script and a header line is used in the data file, then both must match. Either a header line in the data file or a `header()` is required.

3. In each line, the file is specified as a `csv` file, the file name is specified, and a delimiter is set. A map, `authorInput`, is created that will be used to process the data. The map can be manipulated before loading using transforms (page 541).

```
authorInput = File.csv(inputfiledir + 'author.csv').delimiter('|')
```

**Tip:** If you need to trim excess whitespace from data, use `trimWhitespace(true)` in the `File.csv()` statement.

4. Create the main body of the mapping script. (page 522) This part of the mapping script is the same regardless of the file format.

5. To run DSE Graph Loader for CSV loading as a dry run, use the following command:

```
$ graphloader authorBookMappingCSV.groovy -graph testCSV -address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema must be created prior to using `graphloader`.

The fullscript is shown:

```
/* SAMPLE INPUT

// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/CSV/'
authorInput = File.csv(inputfiledir + 'author.csv').delimiter('|
bookInput = File.csv(inputfiledir + 'book.csv').delimiter('|
authorBookInput = File.csv(inputfiledir + 'authorBook.csv').delimiter('|
```

The fullscript is shown:
Using DataStax Enterprise advanced functionality

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true, 
  load_vertex_threads: 3

// DATA INPUT
// Define the data input source (a file which can be specified 
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/CSV/'
authorInput = File.csv(inputfiledir + 
  "author.csv").delimiter('|')
bookInput = File.csv(inputfiledir + "book.csv").delimiter('|')
authorBookInput = File.csv(inputfiledir + 
  "authorBook.csv").delimiter('|')

//Specifies what data source to load using which mapper (as 
// defined inline)

load(authorInput).asVertices {
  label "author"
  key "name"
}

load(bookInput).asVertices {
  label "book"
  key "name"
}

load(authorBookInput).asEdges {
  label "authored"
  outV "aname", {
    label "author"
    key "name"
  }
  inV "bname", {
    label "book"
    key "name"
  }
}
Mapping several files with same format from a directory

DSE Graph Loader can load several CSV files with same format that exist in a directory using the following steps. Sample input data:

SAMPLE INPUT
// For the author.csv file:
name|gender
Julia Child|F
Simone Beck|F

// For the knows.csv file:
aname|bname
Julia Child|James Beard

A number of files with the same format exist in a directory. If the files differ, the graphloader will issue an error and stop:

java.lang.IllegalArgumentException: /tmp/dirSource/data has more than 1 input type.

1. If desired, add configuration (page 485) to the mapping script.

2. Specify the data input directory. The variable inputfiledir specifies the directory for the input files. Each of the identified files will be used for loading.

// DATA INPUT
// Define the data input source (a file which can be specified via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/dirSource/data'
personInput = 
  File.directory(inputfiledir).delimiter('|').header('name','gender')

//Specifies what data source to load using which mapper (as defined inline)
load(personInput).asVertices {
  label "author"
  key "name"
}

The important element is File.directory(); this defines the directory where the files are stored.

It is important to note that CSV files must have a header line that shows the field names. For example, the authorInput will have the following as the first line in the file:
3. Note that two directories could be used to load vertices and edges:

```java
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/dirSource/data'
vertexfiledir = inputfiledir+'_vertices'
edgefiledir = inputfiledir+'_edges'
personInput = File.directory(vertexfiledir).delimiter('|').header('name','gender')
personEdgeInput = File.directory(edgefiledir).delimiter('|').header('aname','bname')

//Specifies what data source to load using which mapper (as
//defined inline)
load(personInput).asVertices {
    label "author"
    key "name"
}
load(personEdgeInput).asEdges {
    label "knows"
    outV "aname", {
        label "author"
        key "name"
    }
    inV "bname", {
        label "book"
        key "name"
    }
}
```

4. To run DSE Graph Loader for CSV loading from a directory, use the following command:

```bash
$ graphloader dirSourceMapping.groovy -graph testdirSource -address localhost
```
For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

### Mapping files from a directory using a file pattern

DSE Graph Loader can load several files from a directory using file pattern matching.

**Sample input files:**

```
ls data
```

```
badOne.csv person1.csv person2.csv
```

A number of files with the same format exist in a directory. If the files differ, DSE Graph Loader will only load the files that match the pattern in the map script.

**Several file patterns are defined for use:**

#### Mapping using *

- If desired, add configuration *(page 485)* to the mapping script.
- Sample input file:

```
/* SAMPLE CSV INPUT:
id|name|gender
001|Julia Child|F
*/
```

- Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```java
// DATA INPUT
// Define the data input source (a file which can be specified via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/filePattern'
inputfileCSV = inputfiledir+'data'
personInput = 
  File.directory(inputfileCSV).fileMatches("person*.csv").delimiter('|').header('id','name','gender')

//Specifies what data source to load using which mapper (as defined inline)

load(personInput).asVertices {
  label "person"
  key "name"
}
```

/* RESULT: */

person1.csv and person2.csv will be loaded, but not badOne.csv
The important element is `fileMatches("person*.csv");` this defines the pattern that will be matched for loaded files. The file `badOne.csv` will not be loaded, because the pattern does not match. Note that a file `personExtra.csv` would also be loaded, as it would match the pattern.

This same pattern matching can be used for JSON input files, by substituting `person*.json` for `person*.csv` and using JSON input file parameters.

- To run DSE Graph Loader for CSV loading from a directory, use the following command:

  ```
  $ graphloader filePatternCSV.groovy -graph testPattCSV -address localhost
  ```

  For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

Mapping using [ ]

- If desired, add configuration (page 485) to the mapping script.
- Sample input file:

  ```
  /* SAMPLE CSV INPUT:
  id|name|gender
  001|Julia Child|F
  */
  ```

- Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

  ```
  // DATA INPUT
  // Define the data input source (a file which can be specified via command line arguments)
  // inputfiledir is the directory for the input files
  inputfiledir = '/tmp/filePattern'
  inputfileCSV = inputfiledir+'/data'
  personInput = File.directory(inputfileCSV).fileMatches("person[1-9].csv").delimiter('|').header('id','name','gender')

  //Specifies what data source to load using which mapper (as defined inline)
  load(personInput).asVertices {
    label "person"
    key "name"
  }
  ```

  /* RESULT:
Using DataStax Enterprise advanced functionality

person1.csv and person2.csv will be loaded, but not badOne.csv
*/

The important element is fileMatches("person[1-9].csv"); this defines the pattern that will be matched for loaded files. All files person1.csv through person9.csv will be loaded, but person15.csv doesn't match the pattern and will not be loaded, as well as badOne.csv. Note that fileMatches("person?.csv") would achieve the same result.

This same pattern matching can be used for JSON input files, by substituting person[1-9].json for person[1-9].csv and using JSON input file parameters.

• Run DSE Graph Loader for this example use the following command:

```
$ graphloader filePatternRANGE.groovy -graph testPattRANGE -address localhost
```

For testing purposes, the graph specified does not have to exist prior to running graphloader. However, for production applications, the graph and schema should be created prior to using graphloader.

Mapping using { } with multiple patterns
• If desired, add configuration (page 485) to the mapping script.
• Sample input file:

/* SAMPLE CSV INPUT:
id|name|gender
001|Julia Child|F
*/

• Specify the data input directory. The variable inputfiledir specifies the directory for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/filePattern/data'
personInput = File.directory(inputfiledir).fileMatches("(person*,badOne).csv").delimiter('|').header('id','name','gender')

//Specifies what data source to load using which mapper (as defined inline)

load(personInput).asVertices {
    label "person"
    key "name"
}
```
The important element is `fileMatches("{person*,badOne}.csv")`; this defines the pattern that will be matched for loaded files. The files `person1.csv`, `person2.csv`, and `badOne.csv` will be loaded, because the pattern matches all three files. This same pattern matching can be used for JSON input files, by substituting `person*.json` for `person*.csv` and using JSON input file parameters.

- To run DSE Graph Loader for this example using the following command:

  ```
  $ graphloader filePatternMULT.groovy -graph testPattMULT -
  address localhost
  ```

  For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

### Loading JSON data

A common file format for loading graph data is JSON. An input JSON file holds all key and value information in a nested structure.

### Mapping several different JSON files

DSE Graph Loader can load several different CSV files that exist in a directory using the following steps. Sample input data:

```perl
// DATA INPUT
// For the author.json file:
{"author_name":"Julia Child","gender":"F"}
// For the book.json file:
// For the authorBook.json file:
{"name":"The Art of French Cooking, Vol. 1","author":"Julia Child"}
```

Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

1. If desired, add configuration (page 485) to the mapping script.

2. Specify the data input files. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```perl
// DATA INPUT
```
// Define the data input source (a file which can be specified via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/JSON/
authorInput = File.json(inputfiledir + 'author.json')
bookInput = File.json(inputfiledir + 'book.json')
authorBookInput = File.json(inputfiledir + 'authorBook.json')

3. In each line, the file is specified as a json file and the file name is specified. The JSON format for File.json is one JSON object per line. A map, authorInput, is created that will be used to process the data. The map can be manipulated before loading using transforms (page 541).

   authorInput = File.json(inputfiledir + 'author.json')

4. Create the main body of the mapping script. (page 522) This part of the mapping script is the same regardless of the file format.

5. To run DSE Graph Loader for JSON loading as a dry run, use the following command:

   $ graphloader authorBookMappingJSON.groovy -graph testJSON -address localhost -dryrun true

For testing purposes, the graph specified does not have to exist prior to running graphloader. However, for production applications, the graph and schema should be created prior to using graphloader.

The fullscript is shown:

   /* SAMPLE INPUT
   author: {"name":"Julia Child","gender":"F"}
   authorBook: {"bname":"The Art of French Cooking, Vol. 1","aname":"Julia Child"
   */

   // CONFIGURATION
   // Configures the data loader to create the schema
   config create_schema: true, load_new: true,
   load_vertex_threads: 3

   // DATA INPUT
   // Define the data input source (a file which can be specified via command line arguments)
   // inputfiledir is the directory for the input files that is given in the commandline
   // as the "-filename" option
Using DataStax Enterprise advanced functionality

```java
inputfiledir = '/tmp/JSON/'
authorInput = File.json(inputfiledir + 'author.json')
bookInput = File.json(inputfiledir + 'book.json')
authorBookInput = File.json(inputfiledir + 'authorBook.json')

//Specifies what data source to load using which mapper (as defined inline)
load(authorInput).asVertices {
    label "author"
    key "name"
}
load(bookInput).asVertices {
    label "book"
    key "name"
}
load(authorBookInput).asEdges {
    label "authored"
    outV "aname", {
        label "author"
        key "name"
    }
    inV "bname", {
        label "book"
        key "name"
    }
}
```

Mapping several files with same format from a directory

DSE Graph Loader can load several JSON files with same format that exist in a directory using the following steps. Sample input data:

**SAMPLE INPUT**

// For the author.json file:
{"author_name":"Julia Child","gender":"F"}

// For the book.json file:

// For the authorBook.json file:
{"name":"The Art of French Cooking, Vol. 1","author":"Julia Child"}

A number of files with the same format exist in a directory. If the files differ, the graphloader will issue an error and stop:

```java
java.lang.IllegalArgumentException: /tmp/dirSource/data has more than 1 input type.
```
1. If desired, add configuration *(page 485)* to the mapping script.

2. Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

   // DATA INPUT
   // Define the data input source (a file which can be specified via command line arguments)
   // inputfiledir is the directory for the input files

   inputfiledir = '/tmp/dirSource/data'
   personInput = File.directory(inputfiledir)

   // Specifies what data source to load using which mapper (as defined inline)

   load(personInput).asVertices {
     label "author"
     key "name"
   }

   The important element is `File.directory()`; this defines the directory where the files are stored.

3. Note that two directories could be used to load vertices and edges:

   // DATA INPUT
   // Define the data input source (a file which can be specified via command line arguments)
   // inputfiledir is the directory for the input files

   inputfiledir = '/tmp/dirSource/data'
   vertexfiledir = inputfiledir+"/vertices"
   edgefiledir = inputfiledir+"/edges"
   personInput = File.directory(vertexfiledir)
   personEdgeInput = File.directory(edgefiledir)

   // Specifies what data source to load using which mapper (as defined inline)

   load(personInput).asVertices {
     label "author"
     key "name"
   }

   load(personEdgeInput).asEdges {
     label "knows"
     outV "aname", {
     label "author"
     key "name"
   }
Using DataStax Enterprise advanced functionality

```java
inV "bname", {
    label "book"
    key "name"
}
```

4. To run DSE Graph Loader for JSON loading from a directory, use the following command:

```
$ graphloader dirSourceJSONMapping.groovy -graph testdirSource -address localhost
```

For testing purposes, the graph specified does not have to exist prior to running graphloader. However, for production applications, the graph and schema should be created prior to using graphloader.

**Mapping files from a directory using a file pattern**

DSE Graph Loader can load several files from a directory using file pattern matching. Sample input files:

```
ls data
badOne.csv person1.csv person2.csv
```

A number of files with the same format exist in a directory. If the files differ, DSE Graph Loader will only load the files that match the pattern in the map script.

Several file patterns are defined for use:

**Mapping using ***

- If desired, add configuration (page 485) to the mapping script.
- Sample input file:

  ```
  /* SAMPLE CSV INPUT: 
  id|name|gender
  001|Julia Child|F 
  */
  ```

- Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

  ```java
  // DATA INPUT
  // Define the data input source (a file which can be specified via command line arguments)
  // inputfiledir is the directory for the input files
  inputfiledir = '/tmp/filePattern'
  inputfileCSV = inputfiledir+'data'
  ```
Using DataStax Enterprise advanced functionality

```java
personInput = File.directory(inputfileCSV).fileMatches("person*.csv").delimiter('|').header('id','name','gender')

//Specifies what data source to load using which mapper (as defined inline)

load(personInput).asVertices {
    label "person"
    key "name"
}

/* RESULT:
person1.csv and person2.csv will be loaded, but not badOne.csv
*/

The important element is fileMatches("person*.csv"); this defines the pattern that will be matched for loaded files. The file badOne.csv will not be loaded, because the pattern does not match. Note that a file personExtra.csv would also be loaded, as it would match the pattern.

This same pattern matching can be used for JSON input files, by substituting person*.json for person*.csv and using JSON input file parameters.

• To run DSE Graph Loader for CSV loading from a directory, use the following command:

  ```
  $ graphloader filePatternCSV.groovy -graph testPattCSV -address localhost
  ```

  For testing purposes, the graph specified does not have to exist prior to running graphloader. However, for production applications, the graph and schema should be created prior to using graphloader.

Mapping using []

• If desired, add configuration (page 485) to the mapping script.

• Sample input file:

  ```
  /* SAMPLE CSV INPUT:
   id|name|gender
   001|Julia Child|F
  */
  ```

• Specify the data input directory. The variable inputfiledir specifies the directory for the input files. Each of the identified files will be used for loading.

  ```
  // DATA INPUT
  // Define the data input source (a file which can be specified via command line arguments)
  // inputfiledir is the directory for the input files
  inputfiledir = '/tmp/filePattern'
  ```
inputfileCSV = inputfiledir+'/data'
personInput =
    File.directory(inputfileCSV).fileMatches("person[1-9].csv").delimiter('|').header('id','name','gender')

//Specifies what data source to load using which mapper (as defined inline)
load(personInput).asVertices {
    label "person"
    key "name"
}

/* RESULT:
    person1.csv and person2.csv will be loaded, but not badOne.csv
*/

The important element is fileMatches("person[1-9].csv"); this defines the pattern that will be matched for loaded files. All files person1.csv through person9.csv will be loaded, but person15.csv doesn't match the pattern and will not be loaded, as well as badOne.csv. Note that fileMatches("person?.csv") would achieve the same result.

This same pattern matching can be used for JSON input files, by substituting person[1-9].json for person[1-9].csv and using JSON input file parameters.

• Run DSE Graph Loader for this example use the following command:

    $ graphloader filePatternRANGE.groovy -graph testPattRANGE -address localhost

For testing purposes, the graph specified does not have to exist prior to running graphloader. However, for production applications, the graph and schema should be created prior to using graphloader.

Mapping using { } with multiple patterns
• If desired, add configuration (page 485) to the mapping script.
• Sample input file:

    /* SAMPLE CSV INPUT:
    id|name|gender
    001|Julia Child|F
    */

• Specify the data input directory. The variable inputfiledir specifies the directory for the input files. Each of the identified files will be used for loading.

    // DATA INPUT
    // Define the data input source (a file which can be specified via command line arguments)
    // inputfiledir is the directory for the input files
inputfiledir = '/tmp/filePattern/data'
personInput = 
  File.directory(inputfiledir).fileMatches("{person*,badOne}.csv").delimiter('|').header('id','name','gender')

// Specifies what data source to load using which mapper (as defined inline)

load(personInput).asVertices {
  label "person"
  key "name"
}

/* RESULT: 
  person1.csv, person2.csv and badOne.csv will all be loaded */

The important element is fileMatches("{person*,badOne}.csv"); this defines the pattern that will be matched for loaded files. The files person1.csv, person1.csv, and badOne.csv will be loaded, because the pattern matches all three files. This same pattern matching can be used for JSON input files, by substituting person*.json for person*.csv and using JSON input file parameters.

- To run DSE Graph Loader for this example using the following command:

  $ graphloader filePatternMULT.groovy -graph testPattMULT - address localhost

  For testing purposes, the graph specified does not have to exist prior to running graphloader. However, for production applications, the graph and schema should be created prior to using graphloader.

Loading TEXT data

The data mapping script for delimited text data is shown with explanation. The full script is found at the bottom of the page.

- If desired, add configuration (page 485) to the mapping script.
- A sample of the data for load looks like the following:

  SAMPLE INPUT
  // For the author.dat file:
  Julia Child|F
  // For the book.dat file:
  Simca's Cuisine: 100 Classic French Recipes for Every Occasion| 1972|0-394-40152-2
  // For the authorBook.dat file:
  Simca's Cuisine: 100 Classic French Recipes for Every Occasion| Simone Beck
Using DataStax Enterprise advanced functionality

- Specify the data input files. The variable `inputfiledir` specifies the directory name for the input files. Each of the identified files will be used for loading.

```java
// DATA INPUT
// Define the data input source (a file which can be specified via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/TEXT/
authorInput = File.text(inputfiledir + "author.dat").delimiter("|").header('name', 'gender')
authorBookInput = File.text(inputfiledir + "authorBook.dat").delimiter("|").header('bname', 'aname')
```

Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

- In each line, the file is specified as a text file, the file name is specified, a delimiter is set, and a header can be specified to identify the fields that will be read. The header can alternatively be specified on the first line of the data file. A map, `authorInput`, is created that will be used to process the data. The map can be manipulated before loading using transforms (page 541).

```java
authorInput = File.text(inputfiledir + "author.dat").delimiter("|").header('name', 'gender')
```

If a `header()` is used in the mapping script and a header line is used in the data file, then both must match. Either a header line in the data file or a `header()` is required.

- Create the main body of the mapping script. (page 522) This part of the mapping script is the same regardless of the file format.

- To run DSE Graph Loader for text loading as a dry run, use the following command:

```bash
$ graphloader authorBookMappingTEXT.groovy -graph testTEXT -address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

- The full loading script is shown.

```java
/** SAMPLE INPUT
author: Julia Child|F
```
Using DataStax Enterprise advanced functionality

authorBook: Simca's Cuisine: 100 Classic French Recipes for Every Occasion | Simone Beck
*/

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true, load_vertex_threads: 3

// DATA INPUT
// Define the data input source (a file which can be specified via command line arguments)
// inputfiledir is the directory for the input files that is given in the commandline
// as the "-filename" option

inputfiledir = '/tmp/CSV/
authorInput = File.text(inputfiledir + "author.dat")
    .delimiter("|")
    .header('name', 'gender')
bookInput = File.text(inputfiledir + "book.dat")
    .delimiter("|")
    .header('name', 'year', 'ISBN')
authorBookInput = File.text(inputfiledir + "authorBook.dat")
    .delimiter("|")
    .header('bname', 'aname')

// Specifies what data source to load using which mapper (as defined inline)

load(authorInput).asVertices {
    label "author"
    key "name"
}

load(bookInput).asVertices {
    label "book"
    key "name"
}

load(authorBookInput).asEdges {
    label "authored"
    outV "aname", {
        label "author"
        key "name"
    }
    inV "bname", {
        label "book"
        key "name"
    }
}

Mapping several files with same format from a directory

- A sample of the data for load looks like the following:

  ```
  SAMPLE INPUT
  // For the author.text file:
  name|gender
  Julia Child|F
  Simone Beck|F
  
  // For the knows.text file:
  aname|bname
  Julia Child|James Beard
  ```

A number of files with the same format exist in a directory. If the files differ, the graphloader will issue an error and stop:

```java
java.lang.IllegalArgumentException: /tmp/dirSource/data has more than 1 input type.
```

- Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

  ```
  // DATA INPUT
  // Define the data input source (a file which can be specified via command line arguments)
  // inputfiledir is the directory for the input files
  inputfiledir = '/tmp/dirSource/data'
  personInput = File.directory(inputfiledir).delimiter('|').header('name','gender')
  
  //Specifies what data source to load using which mapper (as defined inline)
  load(personInput).asVertices {
    label "author"
    key "name"
  }
  ```

The important element is `File.directory()`; this defines the directory where the files are stored.

- Note that two directories could be used to load vertices and edges:

  ```
  // DATA INPUT
  // Define the data input source (a file which can be specified via command line arguments)
  // inputfiledir is the directory for the input files
  inputfiledir = '/tmp/dirSource/data'
  vertexfiledir = inputfiledir+"/vertices"
  edgefiledir = inputfiledir+"/edges"
  ```
Using DataStax Enterprise advanced functionality

```java
personInput = File.directory(vertexfiledir).delimiter('|').header('name','gender')
personEdgeInput = File.directory(edgefiledir).delimiter('|').header('aname','bname')

// Specifies what data source to load using which mapper (as defined inline)
load(personInput).asVertices {
    label "author"
    key "name"
}
load(personEdgeInput).asEdges {
    label "knows"
    outV "aname", {
        label "author"
        key "name"
    }
    inV "bname", {
        label "book"
        key "name"
    }
}
```

- To run DSE Graph Loader for text file loading from a directory, use the following command:

  ```
  $ graphloader dirSourceMapping.groovy -graph testdirSource -address localhost
  ```

  For testing purposes, the graph specified does not have to exist prior to running graphloader. However, for production applications, the graph and schema should be created prior to using graphloader.

**Loading TEXT data using regular expressions (regex)**

The data mapping script for text data parsed using regular expressions (regex) is shown with explanation. The full script is found at the bottom of the page.

- If desired, add configuration (page 485) to the mapping script.
- A sample of the data for load looks like the following:

```text
SAMPLE INPUT
// This file uses tabs between fields
// For the authorREGEX.data file:
name: Julia Child gender:F
// For the bookREGEX.dat file:
// For the authorBookREGEX.dat file:
```
• Specify the data input files. The variable `inputfiledir` specifies the directory name for the input files. Each of the identified files will be used for loading.

    // DATA INPUT
    // Define the data input source
    // inputfiledir is the directory for the input files

    inputfiledir = '/tmp/REGEX/
    authorInput = File.text(inputfiledir + "authorREGEX.dat").
        regex("name:(.*)\tgender:([MF])").
        header('name', 'gender')
    bookInput = File.text(inputfiledir + "bookREGEX.dat").
        regex("name:(.*)\tyear:([0-9]{4})\tISBN:([0-9]{1}-[0-9]{3}-[0-9]{5}-[0-9]{0,1})").
        header('name', 'year', 'ISBN')
    authorBookInput = File.text(inputfiledir + 
        "authorBookREGEX.dat").
        regex("bname:(.*)\taname:(.*)."
        header('bname', 'aname')

Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

• In each line, the file is specified as a text file, the file name is specified, a delimiter is set, and a header must be specified to identify the fields that will be read. In addition, to parse each line of the text file using regex, the regex logic is included. A map, `authorInput`, is created that will be used to process the data. The map can be manipulated before loading using transforms (page 541).

    authorInput = File.text(inputfiledir + 
        "authorREGEX.dat").regex("name:(.*)\tgender:([MF])").header('name', 'gender')

• Create the main body of the mapping script. (page 522) This part of the mapping script is the same regardless of the file format.

• To run DSE Graph Loader for text loading as a dry run, use the following command:

    $ graphloader authorBookMappingREGEX.groovy -graph testREGEX -
        address localhost -dryrun true

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

• The full loading script is shown:

    /* SAMPLE INPUT - uses tabs
Using DataStax Enterprise advanced functionality

```
// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true, load_vertex_threads: 3

// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files that is
given in the commandline
// as the "-filename" option
inputfiledir = '/tmp/REGEX/'
authorInput = File.text(inputfiledir + "authorREGEX.dat").
   regex("name:(.*)\tgender:(\[MF\])").
   header('name', 'gender')
bookInput = File.text(inputfiledir + "bookREGEX.dat").
   regex("name:(.*)\tyear:([0-9]{4})\tISBN:([0-9]{1}[-]{1}[0-9]{3}[-]{1}[0-9]{5}[-]{1}[0-9]{0,1})").
   header('name', 'year', 'ISBN')
authorBookInput = File.text(inputfiledir +
   "authorBookREGEX.dat").
   regex("bname:(.*)\taname:(.*)").
   header('bname', 'aname')

// Specifies what data source to load using which mapper (as
defined inline)
load(authorInput).asVertices {
   label "author"
   key "name"
}
load(bookInput).asVertices {
   label "book"
   key "name"
}
load(authorBookInput).asEdges {
   label "authored"
   outV "aname", {
      label "author"
      key "name"
   }
   inV "bname", {
```
Using DataStax Enterprise advanced functionality

Loading data from a JDBC compatible database.

The data mapping script for loading from a JDBC compatible database is shown with explanation. The full script is found at the bottom of the page.

**Note:** Using DSE Graph Loader to load directly from a JDBC compatible database is convenient, but very slow for a large database. Test a small dataset first, to see if the time required to move a larger dataset makes this method efficient.

- If desired, add configuration *(page 485)* to the mapping script.
- A sample of the data for load looks like the following:

```java
// For the author data:
name:Julia Child gender:F
// For the book data:
// For the authorBook data:
name:Simca's Cuisine: 100 Classic French Recipes for Every Occasion aname:Simone Beck
```

Because the property key `name` is used for both vertex labels author and book, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between author and book vertices.

- Some databases will need a driver installed in the same directory as the `graphloader` script. For the following example using MySQL, the driver can be downloaded. Unzip the file and copy the `mysql-connector-java-5.1.44-bin.jar` file to the correct directory. A similar download would be required for the other databases as well.
- Specify the data input database with JDBC information. The variable `inputDatabase` specifies the data input database. This example uses the MySQL database, but any JDBC-compliant database (H2, MySQL, Postgres, Oracle) can be used. The connection to a localhost and a MySQL database `sample` are specified. In addition, `user` and `password` are defined. The `MySQL()` step denotes the data connection to a MySQL database. The connection can alternatively define a remote machine address.

```java
// DATA INPUT
// Define the data input source (a database connection and SQL statements for data selection)
// inputDatabase is the database name
```
inputDatabase = 'localhost/sample'
db = Database.connection('jdbc:mysql://' +
    inputDatabase).user('root').password('foo').MySQL()
// Define multiple data inputs from the database source via SQL queries
authorInput = db.query "select * from author";
bookInput = db.query "select * from book";
authorBookInput = db.query "select * from authorbook";

**Note:** To load data from H2, the connection line could be:

```java
inputDatabase = '~/test'
db = Database.connection("jdbc:h2:" +
    inputDatabase).H2().user("sa")
```

For Postgres, `Postgre()` is used, and for Oracle, `Oracle()`.

- In each line, the database query is specified that will be used to retrieve the data. A map, `authorInput`, is created that will be used to process the data. The map can be manipulated before loading using transforms (page 541).

```java
authorInput = db.query "SELECT * FROM AUTHOR";
```

**Important:** DSE Graph Loader will retrieve all column names from the database with lower-cased names. Create the graph schema with corresponding lower-cased names to avoid read errors.

- Create the main body of the mapping script. (page 522) This part of the mapping script is the same regardless of the file format.
- To run DSE Graph Loader for text loading as a dry run, use the following command:

```
$ graphloader authorBookMappingJDBC.groovy -graph testJDBC -
    address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

- The full loading script is shown.

```groovy
/* SAMPLE INPUT
author:
    name:Julia Child gender:F
book:
authorBook:
    bname:Simca's Cuisine: 100 Classic French Recipes for Every Occasion aname:Simone Beck
*/
```

// CONFIGURATION
Using DataStax Enterprise advanced functionality

// Configures the data loader to create the schema
config create_schema: true, load_new: true, load_vertex_threads: 3

// DATA INPUT
// Define the data input source (a database connection and SQL
// statements for data selection)
inputDatabase = 'localhost/sample'
db = Database.connection('jdbc:mysql://' +
 inputDatabase).user('root').password('foo').MySQL()

// Define multiple data inputs from the database source via SQL
queries
authorInput = db.query "select * from author";
bookInput = db.query "select * from book";
authorBookInput = db.query "select * from authorbook";

// Specifies what data source to load using which mapper (as
defined inline)
load(authorInput).asVertices {
  label "author"
  key "name"
}
load(bookInput).asVertices {
  label "book"
  key "name"
}
load(authorBookInput).asEdges {
  label "authored"
  outV "aname", {
    label "author"
    key "name"
  }
  inV "bname", {
    label "book"
    key "name"
  }
}

Loading data from Hadoop (HDFS)

The data mapping script for loading from HDFS is shown with explanation. The full
script is found at the bottom of the page.

• If desired, add configuration (page 485) to the mapping script.
• A sample of the CSV data residing on HDFS:

// SAMPLE INPUT
// For the author.csv file:
// name|gender
Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

- Specify the data inputs using a HDFS reference `dfs_uri` and the filenames:

```java
// DATA INPUT
// Define the data input sources /
// dfs_uri specifies the URI to the HDFS directory in which the files are stored

dfs_uri = 'hdfs://hadoopNode:9000/food/
authorInput = File.csv(dfs_uri + 'author.csv.gz').
gzip().
delimiter('|')
bookInput = File.csv(dfs_uri + 'book.csv.gz').
gzip().
delimiter('|')
authorBookInput = File.csv(dfs_uri + 'authorBook.csv.gz').
gzip().
delimiter('|')
```

This example uses compressed files and the additional step `gzip()`.

- Create the main body of the mapping script. *(page 522)* This part of the mapping script is the same regardless of the file format.

- To run DSE Graph Loader for text loading as a dry run, use the following command:

```bash
$ graphloader authorBookMappingHDFS.groovy -graph testHDFS -
address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`. The `-dryrun true` option runs the command without loading data.

- The full loading script is shown.

```java
// SAMPLE INPUT
// For the author.csv file:
// name|gender
```
// Julia Child
// For the book.csv file:
// name|year|ISBN
// Simca's Cuisine: 100 Classic French Recipes for Every Occasion|1972|0-394-40152-2
// For the authorBook.csv file:
// bname|aname
// Simca's Cuisine: 100 Classic French Recipes for Every Occasion|Simone Beck

// CONFIGURATION
// Configures the data loader to create the schema

config create_schema: true, load_new: true, preparation: true

// DATA INPUT
// Define the data input sources
// dfs_uri specifies the URI to the HDFS directory in which the files are stored

dfs_uri = 'hdfs://hadoopNode:9000/food/
authorInput = File.csv(dfs_uri + 'author.csv.gz').
gzip().
delimiter('|')
bookInput = File.csv(dfs_uri + 'book.csv.gz').
gzip().
delimiter('|')
authorBookInput = File.csv(dfs_uri + 'authorBook.csv.gz').
gzip().
delimiter('|')

// Specifies what data source to load using which mapper (as defined inline)

load(authorInput).asVertices {
    label "author"
    key "name"
}

load(bookInput).asVertices {
    label "book"
    key "name"
}

load(authorBookInput).asEdges {
    label "authored"
    outV "aname", {
        label "author"
        key "name"
    }
    inV "bname", {
        label "book"
        key "name"
    }
}
Loading data from AWS S3

The data mapping script for loading from AWS S3 is shown with explanation. The full script is found at the bottom of the page.

- If desired, add configuration (page 485) to the mapping script.
- A sample of the CSV data residing on AWS S3:

```java
// SAMPLE INPUT
// For the author.csv file:
// name|gender
// Julia Child|F
// For the book.csv file:
// name|year|ISBN
// Simca's Cuisine: 100 Classic French Recipes for Every Occasion|1972|0-394-40152-2
// For the authorBook.csv file:
// bname|aname
// Simca's Cuisine: 100 Classic French Recipes for Every Occasion|Simone Beck
```

Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

- Specify the data inputs using a AWS S3 reference `dfs_uri` that defines `s3://[bucket]` and the filenames:

```java
// DATA INPUT
// Define the data input sources /
// `dfs_uri` specifies the URI to the HDFS directory in which the files are stored

dfs_uri = 's3://food/
authorInput = File.csv(dfs_uri +
    'author.csv.gz').gzip().delimiter('|')
bookInput = File.csv(dfs_uri +
    'book.csv.gz').gzip().delimiter('|')
authorBookInput = File.csv(dfs_uri +
    'authorBook.csv.gz').gzip().delimiter('|')
```

This example uses compressed files and the additional step `gzip()`.

- Create the main body of the mapping script. (page 522) This part of the mapping script is the same regardless of the file format.
- To run DSE Graph Loader for text loading as a dry run, use the following command:
Using DataStax Enterprise advanced functionality

$ graphloader authorBookMappingS3.groovy -graph testS3 -address localhost -dryrun true

For testing purposes, the graph specified does not have to exist prior to running graphloader. However, for production applications, the graph and schema should be created prior to using graphloader. The -dryrun true option runs the command without loading data.

- The full loading script is shown.

```groovy
// SAMPLE INPUT
// For the author.csv file:
// name|gender
// Julia Child|F
// For the book.csv file:
// name|year|ISBN
// Simca's Cuisine: 100 Classic French Recipes for Every Occasion|1972|0-394-40152-2
// For the authorBook.csv file:
// bname|aname
// Simca's Cuisine: 100 Classic French Recipes for Every Occasion|Simone Beck

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true, preparation: true

// DATA INPUT
// Define the data input sources
// dfs_uri specifies the URI to the HDFS directory in which the files are stored
dfs_uri = 's3://food/
authorInput = File.csv(dfs_uri +
  'author.csv.gz').gzip().delimiter('|')
bookInput = File.csv(dfs_uri +
  'book.csv.gz').gzip().delimiter('|')
authorBookInput = File.csv(dfs_uri +
  'authorBook.csv.gz').gzip().delimiter('|')

// Specifies what data source to load using which mapper (as defined inline)
load(authorInput).asVertices {
    label "author"
    key "name"
}
load(bookInput).asVertices {
    label "book"
}
```
Using DataStax Enterprise advanced functionality

```
key "name"
}

load(authorBookInput).asEdges {
label "authored"
outV "aname", {
  label "author"
  key "name"
}
inV "bname", {
  label "book"
  key "name"
}
}
```

**Loading Gryo data**

One file format for importing and exporting data to and from DSE Graph is Gryo, a binary file format. Gryo is a Gremlin variant of Kryo, a fast and efficient object graph serialization framework for Java.

The data mapping script for Gryo data is shown with explanation. The full script is found at the bottom of the page.

**Note:** DSE Graph Loader can load Gryo files generated with DSE Graph or with TinkerGraph, the in-memory graph database included with Apache TinkerPop. The Gryo files generated with DSE Graph have a different format from TinkerGraph Gryo files, and the mapping script is different (page 538) for loading data from each source.

- If desired, add configuration (page 485) to the mapping script.
- Specify the data input file. The variable `inputfiledir` specifies the directory for the input file. The identified file will be used for loading.

```
// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/Gryo/
recipeInput = Graph.file(inputfiledir + 'recipe.gryo').gyro()
```

If the Gryo input file is generated from DSE Graph, an additional step `dse()` will allow the input data to be streamed, facilitating large file transfers.

```
// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/Gryo/
recipeInput = Graph.file(inputfiledir + 'recipe.gryo').gyro().dse()
```
Using DataStax Enterprise advanced functionality

- The file is specified as a gryo file and an additional step gryo() identifies that the file should be processed as a Gryo file. A map, recipeInput, is created that will be used to process the data.

```java
recipeInput = Graph.file(inputfiledir + 'recipe.gryo')
```

**Note that** `Graph.file` is used, in contrast to `File.csv` or `File.json`.

**Tip:** If you wish to access a java.io.File object, fully namespace the first call; otherwise, DSE Graph Loader overrides the File object:

```java
currentDir = new java.io.File('.').getCanonicalPath() + '/'
source = Graph.file(currentDir + 'myfile.kryo').gryo()
```

- Create the main body of the mapping script. * (page 522) This part of the mapping script is the same regardless of the file format, although Gryo files use a slightly modified version * (page 538).

- To run DSE Graph Loader for Gryo loading as a dry run, use the following command:

```
$ graphloader recipeMappingGRYO.groovy -graph testGRYO -address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

- The full loading script is shown:

```java
/* SAMPLE INPUT
Gryo file is a binary file
*/

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true

// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/GRYO/
recipeInput = Graph.file(inputfiledir + 'recipe.gryo').gryo()

load(recipeInput.vertices()).asVertices {
  labelField "-label"
  key "-id", "id"
}

load(recipeInput.edges()).asEdges {
  labelField "-label"
```
Loading GraphSON data

The data mapping script for GraphSON data is shown with explanation. The full script is found at the bottom of the page.

Note: DSE Graph Loader can load GraphSON files generated with Apache TinkerGraph, the in-memory graph database included with Apache TinkerPop. GraphSON files generated with DSE Graph cannot be loaded using DSE Graph Loader.

- If desired, add configuration (page 485) to the mapping script.
- Specify the data input file. The variable inputfiledir specifies the directory for the input file. The identified file will be used for loading.

```java
// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/GraphSON/'
recipeInput = Graph.file(inputfiledir + 'recipe.json').graphson().dse()
```

If the GraphSON input file is generated from DSE Graph, an additional step dse() will allow the input data to be streamed, facilitating large file transfers.

```java
// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/GraphSON/'
recipeInput = Graph.file(inputfiledir + 'recipe.json').graphson().dse()
```

- The file is specified as a json file and an additional step graphson() identifies that the file should be processed as a GraphSON file. A map, recipeInput, is created that will be used to process the data.

```java
recipeInput = Graph.file(inputfiledir + 'recipe.json')
```

Note that Graph.file is used, in contrast to File.csv or File.json.
• **Create the main body of the mapping script. (page 522)** This part of the mapping script is the same regardless of the file format, although GraphSON files use a slightly modified version (page 540).

• To run DSE Graph Loader for GraphSON loading as a dry run, use the following command:

```
$ graphloader recipeMappingGraphSON.groovy -graph testGraphSON -address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

• The full loading script is shown:

```java
/* SAMPLE INPUT
GraphSON file is a JSON-like file
*/

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true

// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/GraphSON/
recipeInput = Graph.file(inputfiledir + 'recipe.json').graphson()

// Specifies what data source to load using which mapper (as defined inline)
load(recipeInput.vertices()).asVertices {
    labelField "~label"
    key "~id", "id"
}

load(recipeInput.edges()).asEdges {
    labelField "~label"
    outV "outV", {
        labelField "~label"
        key "~id", "id"
    }
    inV "inV", {
        labelField "~label"
        key "~id", "id"
    }
}
```
Loading GraphML data

The data mapping script for GraphML data is shown with explanation. The full script is found at the bottom of the page.

**Note:** DSE Graph Loader can load GraphML files generated with TinkerGraph, the in-memory graph database included with Apache TinkerPop. GraphML files generated with DSE Graph cannot be loaded using DSE Graph Loader.

- If desired, add configuration (*page 485*) to the mapping script.
- Specify the data input file. The variable `inputfiledir` specifies the directory for the input file. The identified file will be used for loading.

```java
// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/GraphML/
recipeInput = Graph.file(inputfiledir + 'recipe.xml').graphml()
```

- The file is specified as an `xml` file and an additional step `graphml()` identifies that the file should be processed as a GraphML file. A map, `recipeInput`, is created that will be used to process the data.

```java
recipeInput = Graph.file(inputfiledir + 'recipe.xml')
```

*Note that* `Graph.file` *is used, in contrast to* `File.csv` *or* `File.json` *

- Create the main body of the mapping script. (*page 522*) This part of the mapping script is the same regardless of the file format, although GraphML files use a slightly modified version (*page 539*).

- To run DSE Graph Loader for GraphML loading as a dry run, use the following command:

```bash
$ graphloader recipeMappingGraphML.groovy -graph testGraphML -address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

- The full loading script is shown:

```java
/* SAMPLE INPUT
GraphML file is an XML file */
```

```java
// CONFIGURATION
// Configures the data loader to create the schema
```
Using DataStax Enterprise advanced functionality

```plaintext
config create_schema: true, load_new: true

// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/GraphML/
recipeInput = Graph.file(inputfiledir + 'recipe.xml').graphml()

// Specifies what data source to load using which mapper (as defined inline)
load(recipeInput.vertices()).asVertices {
    labelField "-label"
    key "-id", "id"
}
load(recipeInput.edges()).asEdges {
    labelField "-label"
    outV "outV", {
        labelField "-label"
        key "-id", "id"
    }
    inV "inV", {
        labelField "-label"
        key "-id", "id"
    }
}
```

**Mapping script**

Regardless of the file format selected, the main body of the mapping script is the same. After setting configuration and adding a data input source, the mapping commands are specified.

- Vertices are loaded from `authorInput`, with the vertex label `author` and the property key `name` which uniquely identifies a vertex listed for the `key`. Note that, in this example, if `gender` were chosen for the key, it would not be unique enough to load each record from the data file. Using the configuration setting `load_new: true` can significantly speed up the loading process, but a duplicate vertex will be created if the record already exists. All other property keys will be loaded, but do not have to be identified in the loading script. For `author` vertices, `gender` will also be loaded.

```plaintext
load(authorInput).asVertices {
    label "author"
    key "name"
}
```

**Note:** If more than 256 property key values are present in the input file, see important information (page 147) on the `max_query_params` value in the `dse.yaml` file.
One load statement must be created for each vertex loaded, even if the same file is reused for one or more vertices. When using the same input file for multiple vertices, sometimes a field exists in the input file that should be ignored for a particular vertex. See the instructions for ignoring a field (page 524). If an input file includes multiple types of lines, for instance, authors and reviewers, that should be read into different vertex labels, see the instructions for labelField (page 525).

• Loading the book vertices follows a similar pattern. Note that both vertex labels author and book use name as the unique key for identifying a vertex. This declares that the vertex record does not yet exist in the graph at the beginning of the loading process.

```java
load(bookInput).asVertices {
    label "book"
    key "name"
}
```

• After vertices are loaded, edges are loaded. Similar to the vertex mapping, an edge label is specified. In addition, the outgoing vertex (outV) and incoming vertex (inV) for the edge must be identified. For each vertex in outV or inV, the vertex label is specified with label, and the unique key is specified with key.

```java
load(authorBookInput).asEdges {
    label "authored"
    outV "aname", {
        label "author"
        key "name"
    }
    inV "bname", {
        label "book"
        key "name"
    }
}
```

Note the naming convention used for the outV and inV designations. Because both the outgoing vertex and the incoming vertex keys are listed as name, the designators aname and bname are used to distinguish between the author name and the book name as the field names in the input file.

• An alternative to the definitions shown above is to specify the mapping logic with variables, and then list the load statements separately.

```java
authorMapper = {
    label "author"
    key "name"}
bookMapper = {
    label "book"
    key "name"
}
authorBookMapper = {
```
Using DataStax Enterprise advanced functionality

label "authored"
outV "aname", {
    label "author"
    key "name"
}
inV "bname", {
    label "book"
    key "name"
}

load(authorInput).asVertices(authorMapper)
load(bookInput).asVertices(bookMapper)
load(authorBookInput).asEdges(authorBookMapper)

Ignoring a field in input file

If the input file includes a field that should be ignored for a particular vertex load, use `ignore`.

1. Create a map script that ignores the field `restaurant`:

```java
// authorInput includes name, gender, and restaurant
// but restaurant is not loaded
/* Sample input:
name|gender|restaurant
Alice Waters|F|Chez Panisse
*/
load(authorInput).asVertices {
    label "author"
    key "name"
    ignore "restaurant"
}
```

2. An additional example shows the use of `ignore` where two different types of vertices are created, `book` and `author`, using the same input file.

```java
/* Sample input:
name|gender|bname
Julia Child|F|The French Chef Cookbook
Simone Beck|F|The Art of French Cooking, Vol. 1
*/

//inputfiledir = '/tmp/TEXT/'
authorInput = File.text("author.dat").
    delimiter("|").
    header('name', 'gender','bname')

//Specifies what data source to load using which mapper (as defined inline)
```
Using labelField to parse input into different vertex labels

Oftentimes, an input file includes a field that is used to identify the vertex label. In order to load the file and create different vertex labels on-the-fly, labelField is used to identify that particular field.

1. Create a map to input both authors and reviewers from the same file using labelField:

```plaintext
/* SAMPLE INPUT
The input personInput includes type of person, name, gender; type can be either author or reviewer.
type::name::gender
author::Julia Child::F
reviewer::Jane Doe::F
*/

personInput = File.text('people.dat').delimiter('::').header('type','name','gender')

load(personInput).asVertices{
    labelField "type"
    key "name"
}
```

Running this map script using the sample data results in two different vertex labels, with one record for each.

```plaintext
g.V().hasLabel('author').valueMap()
gender=[F], name=[Julia Child]
g.V().hasLabel('reviewer').valueMap()
```
Using DataStax Enterprise advanced functionality

{gender=[F], name=[Jane Doe]}

Using compressed files to load data

Compressed files can be loaded using DSE Graph Loader to load both vertices and edges. This example loads vertices and edges, as well as edge properties, using gzipped files.

1. Create a map script that specifies the input files as compressed *.gz files:

```plaintext
/* SAMPLE INPUT
rev_name|recipe_name|timestamp|stars|comment
John Doe|Beef Bourguignon|2014-01-01|5|comment
*/

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: false, load_new: false

// DATA INPUT
// Define the data input source (a file which can be specified via command line arguments)
// inputfiledir is the directory for the input files that is given in the commandline
// as the "-filename" option
inputfiledir = '/tmp/CSV/'

// This next file is not required if the reviewers already exist
reviewerInput = File.csv(inputfiledir + "reviewers.csv.gz").
gzip().
delimiter('|')

// This next file is not required if the recipes already exist
recipeInput = File.csv(inputfiledir +"recipes.csv.gz").
gzip().
delimiter('|')

// This is the file that is used to create the edges with edge properties
reviewerRatingInput = File.csv(inputfiledir +"reviewerRatings.csv.gz").
gzip().
delimiter('|')

// Specifies what data source to load using which mapper (as defined inline)
load(reviewerInput).asVertices {
    label "reviewer"
    key "name"
}

load(recipeInput).asVertices {
    label "recipe"
    key "name"
}
```
load(reviewerRatingInput).asEdges {
    label "rated"
    outV "rev_name", {
        label "reviewer"
        key "name"
    }
    inV "recipe_name", {
        label "recipe"
        key "name"
    }
    // properties are automatically added from the file, using the
    // header line as property keys
    // from previously created schema
}

The compressed files are designated as .gz files, followed by a gzip() step for
processing. Edge properties are loaded from one of the input files based on the
header identifying the property keys to use for the values listed in each line of
the CSV file. The edge properties populate a rated edge between a reviewer
vertex and a recipe vertex with the properties timestamp, stars, and comment.

Mapping data with a composite custom id

Data with a composite primary key (page 462) requires some additional definition
when specifying the key for loading, if the custom id uses multiple keys for definition
(either partitionKeys and/or clusteringKeys).

1. Inserting data for vertices with a composite custom id requires the declaration of two
or more keys:

/* SAMPLE INPUT
cityId|sensorId|fridgeItem
santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus
*/

load(fridgeItemInput).asVertices {
    label "fridgeSensor"
    // The vertexLabel schema for fridgeSensor includes two keys:
    // partition key: cityId and clustering key: sensorId
    key cityId: "cityId", sensorId: "sensorId"
}

Tip: The schema for the composite custom id must be created prior to using
DSE Graph Loader, and cannot be inferred from the data. In addition, create
a search index (page 472) that includes all properties in the composite
key. The search index is required to use DSE Graph Loader for inserting
composite custom id data.
Using DataStax Enterprise advanced functionality

Check the vertex id results with \texttt{id()} to retrieve the full primary key definition:

\begin{verbatim}
gremlin> g.V().hasLabel('fridgeSensor').id() 
==>{~label=fridgeSensor, 
sensorId=93c4ec9b-68ff-455e-8668-1056ebc3689f, 
cityId=santaCruz} 
==>{~label=fridgeSensor, sensorId=9c23b683-1de2-4c97-a26a-277b3733732a, cityId=sacramento} 
==>{~label=fridgeSensor, sensorId=eff4a8af-2b0d-4ba9-a063-c170130e2d84, cityId=sacramento}
\end{verbatim}

Each vertex stores \texttt{fridgeItem} as data:

\begin{verbatim}
gremlin> g.V().valueMap() 
==>{fridgeItem=[asparagus]} 
==>{fridgeItem=[ham]} 
==>{fridgeItem=[eggs]}
\end{verbatim}

2. To load edges based on a composite key, a transformation is required:

\begin{verbatim}
/* SAMPLE EDGE DATA 
  cityId|sensorId|name 
santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus 
*/ 
the_edges = File.csv(inputfiledir + 
"fridgeItemEdges.csv").delimiter('|') 

the_edges = the_edges.transform { 
  it['fridgeSensor'] = [ 
    'cityId' : it['cityId'], 
    'sensorId' : it['sensorId'] ]; 
  it['ingredient'] = [ 
    'name' : it['name'] ]; 
  it 
}
load(the_edges).asEdges { 
  label "contains" 
  outV "ingredient", { 
    label "ingredient" 
    key "name" 
  } 
  inV "fridgeSensor", { 
    label "fridgeSensor" 
    key cityId:"cityId", sensorId:"sensorId" 
  } 
}
\end{verbatim}

The edge file transforms the partition key and clustering key into a map of \texttt{cityId} and \texttt{sensorId}. This map can then be used to designate the key for a \texttt{fridgeSensor} vertex when the edges are loaded.
The resulting map shows the edges created between ingredient and fridgeSensor vertices.

3. For DSE 5.1.3 and later, an alternative method of loading edge data from CSV files can be used:

/* SAMPLE EDGE DATA
cityId|sensorId|homeId
100|001|9001
*/

isLocatedAt_fridgeSensor = File.csv(/tmp/data/edges/" +
"isLocatedAt_fridgeSensor.csv").delimiter('|')

load(isLocatedAt_fridgeSensor).asEdges {
  label "isLocatedAt"
  outV {
    label "fridgeSensor"
    key cityId: "cityId", sensorId: "sensorId"
    exists()
    ignore "homeId"
  }
  inV {
    label "home"
    key "homeId"
    exists()
    ignore "cityId"
    ignore "sensorId"
  }
  ignore "cityId"
  ignore "sensorId"
  ignore "homeId"
}
In this example, no transform is required, but *ignore* statements are required in both the *inV* and *outV* declarations, as well as the edge properties section. Removing the *exists()* statement in the incoming and outgoing vertex declarations can enable loading the vertices as well as the edges in this mapping script.

**Important:** There is a new subtle change in the *inV* and *outV* declarations. An input field name is no longer used, such as *inV* "home", }, due to the requirement to support multiple-key custom ids.

The resulting map:

**Mapping multi-cardinality edges**

Multiple cardinality edges are a common type of data that is inserted into graphs. Often, the input file has both vertex and edge information for loading.

1. Inserting vertices and multi-cardinality edges can be accomplished from one file with judicious use of *ignore* while loading vertices:

```plaintext
/* SAMPLE INPUT
authorCity:
author|city|dateStart|dateEnd
Julia Child|Paris|1961-01-01|1967-02-10
*/

// CONFIGURATION
// Configures the data loader to create the schema
config dryrun: false, preparation: true, create_schema: false,
load_new: true, schema_output: 'loader_output.txt'

// DATA INPUT
// Define the data input source (a file which can be specified via command line arguments)
```
Using DataStax Enterprise advanced functionality

```java
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/multiCard/
authorCityInput = File.csv(inputfiledir + "authorCity.csv").delimiter('|')

// Specifies what data source to load using which mapper (as defined inline)

// Ignore city, dateStart, and dateEnd when creating author vertices
load(authorCityInput).asVertices {
    label "author"
    key "author"
    ignore "city"
    ignore "dateStart"
    ignore "dateEnd"
}

// Ignore author, dateStart, and dateEnd when creating city vertices
load(authorCityInput).asVertices {
    label "city"
    key "city"
    ignore "author"
    ignore "dateStart"
    ignore "dateEnd"
}

// create edges from author -> city and include the edge properties dateStart and dateEnd
load(authorCityInput).asEdges {
    label "livedIn"
    outV "author", {
        label "author"
        key "author"
    }
    inV "city", {
        label "city"
        key "city"
    }
}
```

**Mapping meta-properties**

If the input file includes meta-properties, or properties that have properties, use `vertexProperty`.

The schema for this data load should be created prior to running `graphloader`

```java
// PROPERTY KEYS
schema.propertyKey('name').Text().single().create()
schema.propertyKey('gender').Text().single().create()
```
Using DataStax Enterprise advanced functionality

```
schema.propertyKey('badge').Text().single().create()
schema.propertyKey('since').Int().single().create()
// Create the meta-property since on the property badge
schema.propertyKey('badge').properties('since').add()
// VERTEX LABELS
schema.vertexLabel('reviewer').properties('name','gender','badge').create()
// INDEXES
schema.vertexLabel('reviewer').index('byname').materialized().by('name').add()
```

1. The mapping script uses `vertexProperty` to identify `badge` as a vertex property. Note the structure of the nested fields for `badge` in the JSON file.

```
* SAMPLE INPUT
reviewer: { "name":"Jon Doe", "gender":"M", "badge" : { "value": "Gold Badge","since" : 2012 } }
*/

// CONFIGURATION
// Configures the data loader to create the schema
config dryrun: false, preparation: true, create_schema: true, load_new: true, load_vertex_threads: 3, schema_output: 'loader_output.txt'

// DATA INPUT
// Define the data input source (a file which can be specified via command line arguments)
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/
reviewerInput = File.json(inputfiledir + "reviewer.json")

// Specifies what data source to load using which mapper (as defined inline)
load(reviewerInput).asVertices{
    label "reviewer"
    key "name"
    vertexProperty "badge", {
        value "value"
    }
}
```

Running this mapping script using the sample data results in a `reviewer` vertex where the property `badge` has a meta-property `since`.

```
g.V().valueMap()
{badge=[Gold Badge], gender=[M], name=[Jane Doe]}
g.V().properties('badge').valueMap()
```
Mapping multiple meta-properties

If the input file includes multiple meta-properties, or properties that have multiple properties, use `vertexProperty`.

The schema for this data load should be created prior to running `graphloader`.

```java
// PROPERTY KEYS
schema.propertyKey('badge').Text().multiple().create()
schema.propertyKey('gender').Text().single().create()
schema.propertyKey('name').Text().single().create()
schema.propertyKey('since').Int().single().create()

// VERTEX LABELS
schema.vertexLabel('reviewer').properties('name', 'gender', 'badge').create()
schema.propertyKey('badge').properties('since').add()

// INDEXES
schema.vertexLabel('reviewer').index('byname').materialized().by('name').add()
```

1. The mapping script uses `vertexProperty` to identify `badge` as a vertex property. Note the structure of the nested fields for `badge` in the JSON file.

```java
/* SAMPLE INPUT
*/
```

1. The mapping script uses `vertexProperty` to identify `badge` as a vertex property. Note the structure of the nested fields for `badge` in the JSON file.
Using DataStax Enterprise advanced functionality

```plaintext
vertexProperty "badge", {
    value "value"
}
}
```

Optionally, the data can be loaded from a CSV file if a transform is used before loading:

```plaintext
/* SAMPLE INPUT
name|gender|value|since
Jane Doe|F|Gold Badge|2011
Jane Doe|F|Silver Badge|2005
Jon Doe|M|Gold Badge|2012
*/

// CONFIGURATION
// Configures the data loader to create the schema
config dryrun: false, preparation: true, create_schema:
    false, load_new: true, load_vertex_threads: 3, schema_output:
    'loader_output.txt'

// DATA INPUT
// Define the data input source (a file which can be specified
via command line arguments)
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/'
reviewerInput = File.csv(inputfiledir +
    "reviewerMultiMeta.csv").delimiter('|')

// Specifies what data source to load using which mapper (as defined inline)
reviewerInput = reviewerInput.transform {
    badge1 = [
        "value": it.remove("value"),
        "since": it.remove("since")
    ]
    it["badge"] = [badge1]
    it
}

load(reviewerInput).asVertices{
    label "reviewer"
    key "name"
    vertexProperty "badge", {
        value "value"
    }
}
```

Running this mapping script using the sample data results in a `reviewer` vertex where the property `badge` has multiple values.
Using DataStax Enterprise advanced functionality

Choosing the pop-up link for `badge` reveals the meta-property values:

Mapping geospatial and Cartesian data

Geospatial and Cartesian data can be loaded with DSE Graph Loader. The DSE Graph Loader is not capable of creating schema for geospatial (page 456) and Cartesian (page 457) data, so schema must be created before loading and the `create_schema` configuration must be set to `false`. 
An example of geospatial schema for the example:

```java
//SCHEMA
schema.propertyKey('name').Text().create()
schema.propertyKey('point').Point().withGeoBounds().create()
schema.vertexLabel('location').properties('name','point').create()
schema.propertyKey('line').Linestring().withGeoBounds().create()
schema.vertexLabel('lineLocation').properties('name','line').create()
schema.propertyKey('polygon').Polygon().withGeoBounds().create()
schema.vertexLabel('polyLocation').properties('name','polygon').create()

schema.vertexLabel('location').index('byname').materialized().by('name').add()
schema.vertexLabel('lineLocation').index('byname').materialized().by('name').add()
schema.vertexLabel('polyLocation').index('byname').materialized().by('name').add()
schema.vertexLabel('location').index('search').search().by('point').add()
schema.vertexLabel('lineLocation').index('search').search().by('line').add()
schema.vertexLabel('polyLocation').index('search').search().by('polygon').add()
```

Search indexes *(page 473)* must be used for geospatial and Cartesian points, linestrings or polygons in graph queries. DSE Graph uses one index per query, and because geospatial data consists of latitude and longitude (two parameters), only search indexes can be used to optimize query performance.

1. If desired, *add configuration (page 485)* to the mapping script.

2. Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```java
/* SAMPLE DATA
name|point
New York|POINT(74.0059 40.7128)
Paris|POINT(2.3522 48.8566)
*/

// DATA INPUT
// Define the data input source (a file which can be specified via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/geo_dgl/data/
ptsInput = File.csv(inputfiledir + "vertices/place.csv").delimiter('|')
linesInput = File.csv(inputfiledir + "vertices/place_lines.csv").delimiter('|')
polysInput = File.csv(inputfiledir + "vertices/place_polys.csv").delimiter('|')

// Specifies what data source to load using which mapper (as defined inline)

load(ptsInput).asVertices {
    label "location"
    key "name"
}```
A transformation of the input data is required, converting the point from the WKT format into the format DSE Graph stores. For a point, the transformation imports a `Point` library and uses the `fromWellKnownText` method:

```
import com.datastax.driver.dse.geometry.Point
ptsInput = ptsInput.transform {
  it['point'] = Point.fromWellKnownText(it['point']);
  return it;
}
```

Linestrings and polygons use the same library and method, respectively.

3. To run DSE Graph Loader for CSV loading from a directory, use the following command:
Using DataStax Enterprise advanced functionality

Mapping Gryo data generated from DSE Graph

Inserting Gryo binary data requires a slightly modified map script. To load Gryo data, allow DSE Graph Loader to create schema and load new data. Loading will require a graph `schema_mode` set to Development.

1. Create a map script for DSE Graph generated Gryo input:

```java
inputfiledir = '/tmp/Gryo/'
recipeInput =
    com.datastax.dsegraphloader.api.Graph.file(inputfiledir + 'recipesDSEG.gryo').gryo().dse()
load(recipeInput.vertices()).asVertices {
    labelField '~label'
    key 'name'
}
load(recipeInput.edges()).asEdges {
    labelField '~label'
    outV 'outV', {
        labelField '~label'
        key 'name' : 'name', 'personId' : 'personId'
    }
    inV 'inV', {
        labelField '~label'
        key 'name' : 'name', 'bookId' : 'bookId'
    }
}
```

The Gryo data format will include `~label` and `name` field values that must be used to create the vertices. For instance, a record that is an author will have a `~label` of `person` and property `name`. For the edges, notice that a user-defined vertex ID consisting of both `name` and `bookId` is used to identify the vertex to use as the incoming vertex for the edge.

Mapping Gryo data generated with TinkerGraph

Inserting Gryo binary data requires a slightly modified map script. To load Gryo data, allow DSE Graph Loader to create schema and load new data. Loading will require a graph `schema_mode` set to Development.

1. Create a map script for TinkerGraph generated Gryo input:

```java
//Specifies what data source to load using which mapper (as defined inline)
load(recipeInput.vertices()).asVertices {
    labelField "~label"
```
The Gryo data format will include `~label` and `name` field values that must be used to create the vertices and edges. For instance, a record that is an author will have a `~label` of `author` and property `name`. The `vertexKeyMap` creates a map of each vertex label to a unique property. This map is used to create unique keys used while loading vertices from the binary file.

**Mapping GraphML binary data**

Inserting GraphML binary data requires a slightly modified map script. To load GraphML data, allow DSE Graph Loader to create schema and load new data. Loading will require a graph `schema_mode` set to Development.

1. Create a map script for GraphML data:

```java
// Specifies what data source to load using which mapper (as defined inline)
load(recipeInput.vertices()).asVertices {
    labelField "~label"
    key "~id", "id"
}
load(recipeInput.edges()).asEdges {
    labelField "~label"
    outV "outV", {
        labelField "~label"
        key "~id", "id"
    }
    inV "inV", {
        labelField "~label"
        key "~id", "id"
    }
}
```

The GraphML data format will include `~label` and `~id` field values that must be used to create the label and key for each record loaded. For instance, a record
Using DataStax Enterprise advanced functionality

that is an author will have a ~label of author. The ~id will similarly be set in the
record, a difference from other data. The difference can be seen by looking at a
record and noting the presence of the id field, based on the second item in each
key setting in the mapping script:

```java
g.V().hasLabel('author').valueMap()
{gender=[F], name=[Julia Child], id=[0])
{gender=[F], name=[Simone Beck], id=[3]}
```

Mapping GraphSON binary data

Inserting GraphSON data requires a slightly modified map script. To load GraphSON
data, allow DSE Graph Loader to create schema and load new data. Loading will
require a graph schema_mode set to Development.

1. Create a map script for GraphML data:

```java
//Specifies what data source to load using which mapper (as
defined inline)
load(recipeInput.vertices()).asVertices {
  labelField "~label"
  key "~id", "id"
}

load(recipeInput.edges()).asEdges {
  labelField "~label"
  outV "outV", {
    labelField "~label"
    key "~id", "id"
  }
  inV "inV", {
    labelField "~label"
    key "~id", "id"
  }
}
```

The GraphSON data format will include ~label and ~id field values that must be
used to create the label and key for each record loaded. For instance, a record
that is an author will have a ~label of author. The ~id will similarly be set in the
record, a difference from other data. The difference can be seen by looking at a
record and noting the presence of the id field, based on the second item in each
key setting in the mapping script:

```java
g.V().hasLabel('author').valueMap()
{gender=[F], name=[Julia Child], id=[0])
```
Using transforms (filter, flatMap, and map) with DSE Graph Loader

All data inputs support arbitrary user transformations to manipulate or truncate the input data according to a user provided function. The available transforms for DSE Graph Loader are:

- **filter** *(page 541)*
- **flatMap** *(page 544)*
- **map** *(page 547)*

**Notice:** As of DSE Graph Loader 6.0, transformation functions may be deprecated; be aware that changes may occur.

The data record for each data input is a document structure or nested map defined from an input file. A transformation acts upon the nested map and returns a nested map. Any provided transformation function must be thread-safe or the behavior of the data loader becomes undefined.

The transforms used are Groovy closures, or open anonymous blocks of code that can take arguments, return values and be assigned for a variable. These closures often make use of a Groovy implicit parameter, it. When a closure does not explicitly define a parameter list, it is always a defined parameter that can be used. In the following examples, it is used to get each record in an input file and apply the transformation.

The placement of the transform in the mapping script is arbitrary; as long as the input file is defined before the transform is defined, a transform may be placed anywhere in the mapping script.

Here's a simple introduction to Groovy for those unfamiliar with it.

**filter**

The *filter* function can apply criteria to the input file, selecting only the objects that meet the criteria and loading them. The criteria can match any data type used in a field.

Filter based on inequality operation on integer

The defined input file in this example is *chefs*. The filter is applied to the input file using the syntax `<input_file_name>.filter { ... }`. Given an integer field for *age*, all chefs 41 years old and younger can be filtered, and loaded into the graph with vertex label *chefYoung*:

```java
/** SAMPLE INPUT
name|gender|status|age
Jamie Oliver|M|alive|41
*/

inputfiledir = '/tmp/filter_map_flatmap/
```
Using DataStax Enterprise advanced functionality

```java
chefs = File.csv(inputfiledir + "filterData.csv").delimiter('|')

// filter
def chefsYoung = chefs.filter { it["age"].toInteger() <= 41 }

// Specifies what data source to load using which mapper (as defined inline)
load(chefsYoung).asVertices {
  label "chefYoung"
  key "name"
}

The value for age is converted to an Integer for the function operation, and compared to the value of 41.

Only the records that match the criteria will create vertices, as reflected in the resulting values:

```java
==>{gender=[M], name=[Jamie Oliver], age=[41], status=[alive]}
==>{gender=[F], name=[Amanda Cohen], age=[35], status=[alive]}
==>{gender=[M], name=[Patrick Connolly], age=[31], status=[alive]}
```

Filter based on equality match operation on string

Another example of two filters finds all the chefs who are alive and who are deceased:

```java
/** SAMPLE INPUT
name|gender|status|age
Jamie Oliver|M|alive|41
**/

inputfiledir = '/tmp/filter_map_flatmap/'
chefs = File.csv(inputfiledir + "filterData.csv").delimiter('|')
def chefsAlive = chefs.filter { it["status"] == "alive" }
def chefsDeceased = chefs.filter { it["status"] == "deceased" }

load(chefsAlive).asVertices {
  label "chefAlive"
  key "name"
}
load(chefsDeceased).asVertices {
  label "chefDeceased"
  key "name"
}

The filter checks the value of the string status and creates two new inputs, chefsAlive and chefsDeceased to use for loading the vertices, with the respective vertex labels chefAlive and chefDeceased.

The resulting vertices are:
// List all the living chefs
```java
G.V().hasLabel('chefAlive').valueMap()
```
```java
==>{gender=[F], name=[Alice Waters], age=[73], status=[alive]}
==>{gender=[F], name=[Patricia Curtan], age=[66], status=[alive]}
==>{gender=[F], name=[Kelsie Kerr], age=[57], status=[alive]}
==>{gender=[M], name=[Fritz Streiff], age=[500], status=[alive]}
==>{gender=[M], name=[Emeril Lagasse], age=[57], status=[alive]}
==>{gender=[M], name=[Jamie Oliver], age=[41], status=[alive]}
==>{gender=[M], name=[Amanda Cohen], age=[35], status=[alive]}
==>{gender=[M], name=[Patrick Connolly], age=[31], status=[alive]}
```

// List all the deceased chefs
```java
G.V().hasLabel('chefDeceased').valueMap()
```
```java
==>{gender=[F], name=[Julia Child], age=[500], status=[deceased]}
==>{gender=[F], name=[Simone Beck], age=[500], status=[deceased]}
==>{gender=[F], name=[Louisette Bertholie], age=[500], status=[deceased]}
==>{gender=[F], name=[Patricia Simon], age=[500], status=[deceased]}
==>{gender=[M], name=[James Beard], age=[500], status=[deceased]}
```

Full filter data set

The full sample data set used in this example:

<table>
<thead>
<tr>
<th>name</th>
<th>gender</th>
<th>status</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia Child</td>
<td>F</td>
<td>deceased</td>
<td>500</td>
</tr>
<tr>
<td>Simone Beck</td>
<td>F</td>
<td>deceased</td>
<td>500</td>
</tr>
<tr>
<td>Louisette Bertholie</td>
<td>F</td>
<td>deceased</td>
<td>500</td>
</tr>
<tr>
<td>Patricia Simon</td>
<td>F</td>
<td>deceased</td>
<td>500</td>
</tr>
<tr>
<td>Alice Waters</td>
<td>F</td>
<td>alive</td>
<td>73</td>
</tr>
<tr>
<td>Patricia Curtan</td>
<td>F</td>
<td>deceased</td>
<td>66</td>
</tr>
<tr>
<td>Kelsie Kerr</td>
<td>F</td>
<td>alive</td>
<td>57</td>
</tr>
<tr>
<td>Fritz Streiff</td>
<td>M</td>
<td>alive</td>
<td>500</td>
</tr>
<tr>
<td>Emeril Lagasse</td>
<td>M</td>
<td>alive</td>
<td>57</td>
</tr>
<tr>
<td>James Beard</td>
<td>M</td>
<td>deceased</td>
<td>500</td>
</tr>
<tr>
<td>Jamie Oliver</td>
<td>M</td>
<td>alive</td>
<td>41</td>
</tr>
<tr>
<td>Amanda Cohen</td>
<td>F</td>
<td>alive</td>
<td>35</td>
</tr>
<tr>
<td>Patrick Connolly</td>
<td>M</td>
<td>alive</td>
<td>31</td>
</tr>
</tbody>
</table>

Note the use of 500 as a placeholder for the age of deceased chefs.

Full filter mapping script

The full map script with all three filters:

```java
/** SAMPLE INPUT
name|gender|status|age
Jamie Oliver|M|alive|41
**/
```

```java
// SCHEMA
```
```java
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().ifNotExists().create()
```
Using DataStax Enterprise advanced functionality

```java
schema.propertyKey('status').Text().ifNotExists().create()
schema.propertyKey('age').Int().ifNotExists().create()

schema.vertexLabel('chefAlive').properties('name', 'gender', 'status', 'age').create()
schema.vertexLabel('chefAlive').index('byname').materialized().by('name').add()
schema.vertexLabel('chefDeceased').properties('name', 'gender', 'status', 'age').create()
schema.vertexLabel('chefDeceased').index('byname').materialized().by('name').add()
schema.vertexLabel('chefYoung').properties('name', 'gender', 'status', 'age').create()
schema.vertexLabel('chefYoung').index('byname').materialized().by('name').add()

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: false, load_new: true

// DATA INPUT
// Define the data input source (a file which can be specified via
// command line arguments)
// inputfiledir is the directory for the input files that is given in
// the commandline
// as the "-filename" option

inputfiledir = '/tmp/filter_map_flatmap/
chefs = File.csv(inputfiledir + "filterData.csv").delimiter('|')
def chefsYoung = chefs.filter { it['age'].toInteger() <= 41 }
def chefsAlive = chefs.filter { it['status'] == "alive" }
def chefsDeceased = chefs.filter { it['status'] == "deceased" }

// Specifies what data source to load using which mapper (as defined
// inline)

load(chefsYoung).asVertices {
    label "chefYoung"
    key "name"
}
load(chefsAlive).asVertices {
    label "chefAlive"
    key "name"
}
load(chefsDeceased).asVertices {
    label "chefDeceased"
    key "name"
}
```

**flatMap**

The `flatMap` function (also called `expand`) can break a single field in the input file into separate objects before loading them. In general, this function is used to convert more compacted data into an expanded form.
FlatMap based on multiple cuisine values for a recipe

The input file for this example is recipes. The flatMap is applied to the input file using the syntax `<input_file_name>.flatMap { ... }`. Given a field for cuisine that identifies all the possible cuisine choices for a recipe, a record for each vertex can be created using the recipe name and the cuisine type as a separate vertex when loading the vertices into the graph:

```groovy
/** SAMPLE INPUT
name|cuisine
Beef Bourguignon|English::French 
**/

inputfiledir = '/tmp/filter_map_flatmap/
recipes = File.csv(inputfiledir + "flatmapData.csv").delimiter('|')
def recipesCuisine = recipes.flatMap {
def name = it["name"];it["cuisine"].split("::").collect {
    it = [ 'name': name, 'cuisine': it ]
}
}
//Specifies what data source to load using which mapper (as defined inline)
load(recipesCuisine).asVertices {
    label "recipe"
    key name: "name", cuisine: "cuisine"
}
```

The flatMap function gets each record, retrieves the recipe name, splits the cuisine field, and then collects each name/cuisine pair to use as the composite key for identifying each separate vertex. The Groovy `split` method splits a string (cuisine) using the supplied delimiter (::) and returns an array of strings (each cuisine). The Groovy `collect` method iterates over a collection and transforms each element of the collection.

The result of the loading reflects all the possible vertices based on cuisine:

```
g.V().valueMap()
==>{name=[Beef Bourguignon], cuisine=[English]}
==>{name=[Beef Bourguignon], cuisine=[French]}
==>{name=[Nicoise Salade], cuisine=[French]}
==>{name=[Wild Mushroom Stroganoff], cuisine=[American]}
==>{name=[Wild Mushroom Stroganoff], cuisine=[English]}
```

Full flatMap data set

The full sample data set used in this example:
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>name</th>
<th>cuisine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beef Bourguignon</td>
<td>English::French</td>
</tr>
<tr>
<td>Nicoise Salade</td>
<td>French</td>
</tr>
<tr>
<td>Wild Mushroom Stroganoff</td>
<td>American::English</td>
</tr>
</tbody>
</table>

Full flatMap mapping script

The full map script with flatMap:

```java
/** SAMPLE INPUT
name|cuisine
Beef Bourguignon|English::French */

// SCHEMA
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('cuisine').Text().ifNotExists().create()

schema.vertexLabel('recipe').properties('name', 'cuisine').create()

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: false, load_new: true

// DATA INPUT
// Define the data input source (a file which can be specified via command line arguments)
// inputfiledir is the directory for the input files that is given in the commandline
// as the "-filename" option

inputfiledir = '/tmp/filter_map_flatmap/'
recipes = File.csv(inputfiledir + "flatmapData.csv").delimiter('|')

def recipesCuisine = recipes.flatMap {
    def name = it['name'];
    it['cuisine'].split('::').collect {
        it = [ 'name': name, 'cuisine': it ]
    }
}

// Specifies what data source to load using which mapper (as defined inline)
load(recipesCuisine).asVertices {
    label "recipe"
    key name: "name", cuisine: "cuisine"
}
Using DataStax Enterprise advanced functionality

map

The map() (also called transform()) applies a function to a field's values before loading the data.

map converts gender field from to lower case from any case

The input file for this example is authorInput. The map is applied to the input file using the syntax `<input_file_name>.map { ... }`. Given a field gender, the Groovy toLowerCase() method is performed on each gender value in the nested map

```groovy
inputfiledir = '/tmp/TEXT/
authorInput = File.text(inputfiledir + "author.dat").
    delimiter("|"
    header('name', 'gender')

authorInput = authorInput.map { it['gender'] =
    it['gender'].toLowerCase(); it }
```

This map() transformation ensures that the gender values in the graph are only lowercase.

The result of the loading reflects the change to the case of gender:

```text
V().valueMap()
---
{gender=[f], name=Julia Child, age=500}
{gender=[f], name=Simone Beck, age=500}
{gender=[f], name=Louisette Bertholie, age=500}
{gender=[f], name=Patricia Simon, age=500}
{gender=[f], name=Alice Waters, age=73}
{gender=[f], name=Kelsie Kerr, age=66}
{gender=[m], name=Fritz Streiff, age=500}
{gender=[m], name=Emeril Lagasse, age=57}
{gender=[m], name=James Beard, age=500}
{gender=[m], name=Jamie Oliver, age=41}
{gender=[f], name=Amanda Cohen, age=35}
{gender=[m], name=Patrick Connolly, age=31}
```

Full map data set

The full sample data set used in this example:

<table>
<thead>
<tr>
<th>name</th>
<th>gender</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia Child</td>
<td>F</td>
<td>500</td>
</tr>
<tr>
<td>Simone Beck</td>
<td>F</td>
<td>500</td>
</tr>
<tr>
<td>Louisette Bertholie</td>
<td>F</td>
<td>500</td>
</tr>
<tr>
<td>Patricia Simon</td>
<td>F</td>
<td>500</td>
</tr>
<tr>
<td>Alice Waters</td>
<td>F</td>
<td>73</td>
</tr>
<tr>
<td>Patricia Curtan</td>
<td>F</td>
<td>66</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

Full map mapping script

The full map script with `map`:

```java
/** SAMPLE INPUT
name|gender|age
Jamie Oliver|M|41
**/

// SCHEMA
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().ifNotExists().create()
schema.propertyKey('age').Int().ifNotExists().create()

schema.vertexLabel('chef').properties('name','gender','age').create()
schema.vertexLabel('chef').index('byname').materialized().by('name').add()

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: false, load_new: true

// DATA INPUT
// Define the data input source (a file which can be specified via command line arguments)
// inputfiledir is the directory for the input files that is given in the commandline
// as the "-filename" option

inputfiledir = '/tmp/filter_map_flatmap/
chefs = File.csv(inputfiledir + "mapData.csv").delimiter('|')
chefInput = chefs.map { it['gender'] = it['gender'].toLowerCase(); it }

// Specifies what data source to load using which mapper (as defined inline)
load(chefInput).asVertices {
    label "chef"
    key "name"
}
## DSE Graph Loader reference

### Synopsis

$ graphloader loadingScript [[-option value]...]

### Table 14: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
Options can be invoked in the command line or included in the loading script. Required options are marked.

<table>
<thead>
<tr>
<th>Option</th>
<th>Data type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-abort_on_num_failures</td>
<td>Integer</td>
<td>100</td>
<td>Number of failures after which loading is aborted.</td>
</tr>
<tr>
<td>-abort_on_prep_errors</td>
<td>Boolean</td>
<td>true</td>
<td>Normally if errors occur in the preparation, or during the vertex insertion phase we abort, setting this to false will force the loader to continue up to the maximum number of allowed failures.</td>
</tr>
<tr>
<td>-address</td>
<td>String</td>
<td></td>
<td>The IP address (and port) of the DSE Graph instance to connect to. REQUIRED</td>
</tr>
<tr>
<td>-allow_remote_hosts_in_quorum</td>
<td>Boolean</td>
<td>false</td>
<td>Allows hosts in a different datacenter to participate in a local consistency level, so that a node from a remote datacenter can be used to reach a consistency level of QUORUM, for instance, for a query. Choices are: true, false.</td>
</tr>
<tr>
<td>-batch-size</td>
<td>Integer</td>
<td>100</td>
<td>Size of loading batches.</td>
</tr>
<tr>
<td>-compress</td>
<td>String</td>
<td>none</td>
<td>The compression of the file. Choices are none, gzip, and xzip.</td>
</tr>
<tr>
<td>Option</td>
<td>Data type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-consistency_level</td>
<td>CL</td>
<td>ONE</td>
<td>Choices are: ANY, ONE, TWO, THREE, QUORUM, ALL, LOCAL_QUORUM, EACH_QUORUM, SERIAL, LOCAL_SERIAL, LOCAL_ONE.</td>
</tr>
<tr>
<td>-create_graph</td>
<td>Boolean</td>
<td>true</td>
<td>Check if the target graph exists, and if it doesn't, creates it if true. Note that this option can fail on the default consistency level of QUORUM if a datacenter is unreachable.</td>
</tr>
<tr>
<td>-create_schema</td>
<td>Boolean</td>
<td>true</td>
<td>Whether to update or create the schema for missing schema elements. Notice: It is strongly recommended that schema is created (page 447) prior to data loading, so that the correct data types are enforced and indexes created. Setting create_schema to true is recommended only for testing. In DSE 6.0, this configuration option is deprecated and will be removed in a future release.</td>
</tr>
<tr>
<td>Option</td>
<td>Data type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>---------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-driver_retry_attempts</td>
<td>Integer</td>
<td>3</td>
<td>Number of retry attempts. If greater than zero, requests will be resubmitted after some recoverable failures.</td>
</tr>
<tr>
<td>-driver_retry_delay</td>
<td>milliseconds</td>
<td>1000</td>
<td>Number of milliseconds between driver retries.</td>
</tr>
<tr>
<td>-dryrun</td>
<td>Boolean</td>
<td>false</td>
<td>Whether to only conduct a trial run to verify data integrity and schema consistency. Does not create a graph if it doesn't exist. Notice: This configuration option discovers schema and suggests missing schema without executing any changes. In DSE 6.0, this option is deprecated and may possibly be removed in a future release.</td>
</tr>
<tr>
<td>-filename</td>
<td>String</td>
<td></td>
<td>The file to load the vertex data from. REQUIRED if not defined in the mapping script.</td>
</tr>
<tr>
<td>-graph</td>
<td>String</td>
<td></td>
<td>The name of the graph to load into. REQUIRED</td>
</tr>
<tr>
<td>Option</td>
<td>Data type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-label</td>
<td>String</td>
<td></td>
<td>The label of the vertex to be populated with data. If left blank, the name of the input file is used as the vertex label name.</td>
</tr>
<tr>
<td>-load_failure_log</td>
<td>String</td>
<td>load_failures.txt</td>
<td>Name and location of the file where failed records will be stored.</td>
</tr>
<tr>
<td>-load_new</td>
<td>Boolean</td>
<td>false</td>
<td>Whether the vertices loaded are new and do not yet exist in the graph.</td>
</tr>
<tr>
<td>-load_edge_threads</td>
<td>Integer</td>
<td>0</td>
<td>Number of threads to use for loading edge and property data into the graph (0 will force the value to be the number of nodes in the DC * 6).</td>
</tr>
<tr>
<td>-load_vertex_threads</td>
<td>Integer</td>
<td>0</td>
<td>Number of threads to use for loading vertices into the graph (0 will force the value to the number of cores/2).</td>
</tr>
<tr>
<td>Option</td>
<td>Data type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>---------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-preparation</td>
<td>Boolean</td>
<td>true</td>
<td>Whether to do a preparation run to analyze the data and update the schema, if necessary. Notice: This configuration option validates and creates schema if used in conjunction with create_schema. The default will be set to false, and this option is deprecated with DSE 6.0. In a future release, it may be removed.</td>
</tr>
<tr>
<td>-preparation_limit</td>
<td>Integer</td>
<td>0</td>
<td>The number of records that the preparation phase will use to attempt to determine if the schema should be updated. Zero indicates no limit.</td>
</tr>
<tr>
<td>-queue-size</td>
<td>Integer</td>
<td>10000</td>
<td>Data retrieval queue size.</td>
</tr>
<tr>
<td>-read_threads</td>
<td>Integer</td>
<td>1</td>
<td>Number of threads to use for reading data from data input.</td>
</tr>
<tr>
<td>-remote_hosts_in_dc</td>
<td>Integer</td>
<td>2</td>
<td>Number of remote nodes that can participate in the consistency level for a query.</td>
</tr>
<tr>
<td>-reporting_interval</td>
<td>Integer</td>
<td>1</td>
<td>Number of seconds between each progress report written to the log.</td>
</tr>
<tr>
<td>Option</td>
<td>Data type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-schema_output</td>
<td>String</td>
<td>proposed_schema.txt</td>
<td>The name of the file to save the proposed schema in when executing a dry-run. Leave blank to disable.</td>
</tr>
<tr>
<td>-skip_blank_values</td>
<td>Boolean</td>
<td>true</td>
<td>When false, loader will insert a blank (&quot;&quot;) for all unspecified (empty/blank) property values in a CSV file.</td>
</tr>
<tr>
<td>-timeout</td>
<td>Integer</td>
<td>120000</td>
<td>Number of milliseconds until a connection times out.</td>
</tr>
<tr>
<td>-v</td>
<td>--version</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>-vertex_complete</td>
<td>Boolean</td>
<td>false</td>
<td>The loader assumes that all vertices referenced by properties and edges in this load are also included as vertexes of this load. No new vertices will be created from edge data or property data files.</td>
</tr>
<tr>
<td>-username</td>
<td>String</td>
<td></td>
<td>Username for DSE authentication.</td>
</tr>
<tr>
<td>-password</td>
<td>String</td>
<td></td>
<td>Password for DSE authentication.</td>
</tr>
<tr>
<td>-ssl</td>
<td>Boolean</td>
<td>false</td>
<td>Enable SSL.</td>
</tr>
<tr>
<td>-kerberos</td>
<td>Boolean</td>
<td>false</td>
<td>Enable kerberos.</td>
</tr>
<tr>
<td>-sasl</td>
<td>String</td>
<td></td>
<td>An optional sasl protocol name used in conjunction with kerberos.</td>
</tr>
</tbody>
</table>

Security options:
### Description

DSE Graph Loader is an utility for loading up to 100 million vertices and 1 billion edges. The utility runs on a sufficiently powerful computer that can cache all vertices in memory and includes enough cores to parallelize the loading process. For larger loads, the utility must be run on a different machine.

DSE Graph Loader is invoked on the command line with a loading script as argument and a variable number of configuration option-value pairs. The loading script specifies what input data is being loaded and how that data maps onto the graph. The loading script can also configure the option-value pairs.

The three stages of load processing are:

#### Preparation

Reads entire input data. This stage either ensures that the data conforms to the graph schema, or the stage updates the graph schema according to the provided data (if enabled). At the end of this stage, statistical estimates are provided on how much data will be added to the graph but no data is loaded. Set

```
-dryrun true
```

to abort the loading process after the preparation stage and before any changes are made. Inspect the output and verify that it matches your expectations. For large datasets, doing a dry run is important for spotting errors.

#### Vertex Loading

The second stage adds or retrieves all of the vertices in the input data and caches them locally to speed up the subsequent edge loading.

#### Edge and Property Loading

Adds all edges and properties from the input data to the graph.

---

<table>
<thead>
<tr>
<th>Option</th>
<th>Data type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-kerberos</td>
<td>Boolean</td>
<td>false</td>
<td>Enable kerberos.</td>
</tr>
<tr>
<td>-password</td>
<td>String</td>
<td></td>
<td>Password for DSE authentication.</td>
</tr>
<tr>
<td>-sasl</td>
<td>String</td>
<td></td>
<td>An optional sasl protocol name used in conjunction with kerberos.</td>
</tr>
<tr>
<td>-ssl</td>
<td>Boolean</td>
<td>false</td>
<td>Enable SSL.</td>
</tr>
<tr>
<td>-username</td>
<td>String</td>
<td></td>
<td>Username for DSE authentication.</td>
</tr>
</tbody>
</table>
A loading, or mapping, script is required to specify the particular mapping used to load the data from the input file to the graph. DSE Graph Loader supports four file-based data input types: CSV, JSON, delimited text, and text parsed by regular expressions. All file-based input formats support compression of the input data files.

Logging during the loading process can provide useful information if troubleshooting is required. The three stages of load processing are detailed in the log.

Examples

To get the listing of possible options, use -help.

$ graphloader -help

This example will use the loading script mymapscript.groovy to read data from a file /tmp/recipe/all.dat into the graph test that is running on the localhost. Dry run is specified to test the loading without inserting the data.

$ graphloader mymapscript.groovy -filename /tmp/recipe/all.dat -graph test -address localhost -dryrun true

This example will use the loading script csv2Vertex.groovy to read data from a file MyUsers.csv into the graph csvTest that is running on the localhost. The -label option specifies that the vertex label will be User, rather than the filename MyUsers.

$ graphloader ./scripts/csv2Vertex.groovy -filename MyUsers.csv -graph csvTest -label User -address 127.0.0.1

The configuration settings can also be specified in the loading script. A fragment of a loading script is shown here that sets create_schema to true and load_vertex_threads to 3.

// CONFIGURATION
// Configures the data loader to create the schema and set load_vertex_threads to 3
config load_new: true, load_vertex_threads: 3

By default, the graphloader logs debug information to the file loader.log in the directory from which graphloader is run. The location of the log can be specified with -load_failure_log:

$ graphloader mymapscript.groovy -graph test -address localhost -load_failure_log /tmp/dgl.log

If log4j modifications are desired to log information differently, a configuration file can be created, and used in conjunction with the -load_failure_log. Here is a sample configuration file:

# Set root logger level to the designated level and its appenders to F1 and stdout
log4j.rootLogger=INFO, WARN, A1, stdout
# /dev/stdout
Using DataStax Enterprise advanced functionality

```java
# Log INFO messages to A1. A1 is set to be a ConsoleAppender.
log4j.appender.A1.Target=System.out
log4j.appender.A1.Threshold=INFO
# A1 uses PatternLayout.
log4j.appender.A1.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
# Direct INFO log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.Threshold=INFO
# stdout uses PatternLayout.
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

and a sample `graphloader` command:

```
$ java -Dlog4j.configuration=file:./lib/log4j.properties -jar
  graphLoaderJar mymapscript.groovy -graph test -address localhost -
  load_failure_log /dev/stdout
```

that will write the log information to stdout.

The preparation stage has additional options. To use the input data to discover the schema, use `-preparation true`. If preparation discovers missing elements in the schema, those elements can be added if `-create_schema true`. If desired, preparation can be performed, but schema creation must be manually created if `-create_schema false`. Setting `-create_schema true` without `-preparation true` will result in a stopped job. Without sampling the data to discover the schema that the data describes, `graphloader` cannot create schema because the manner of the schema is unknown. To summarize, if you wish to create schema manually, use `-preparation true -create_schema false`. If you wish `graphloader` to automatically create schema, use `-preparation true -create_schema true`.

To use authentication, configure `graphloader` with `-user` and `-password`:

```
$ graphloader mymapscript.groovy -graph test -address localhost -
  username myName -password myPasswd
```

To configure `graphloader` with SSL encryption and using Kerberos:

```
$ java -Djavax.net.ssl.trustStore=<TRUSTSTORE_PATH>
  -Djavax.net.ssl.trustStorePassword=<PASSWORD> -
  -Djavax.net.ssl.keyStore=<KEYSTORE_PATH> \
  -Djavax.net.ssl.keyStorePassword=<PASSWORD> -jar dse-graph-
  loader-5.0.3-uberjar.jar -kerberos true -sasl dsename -graph new -
  address localhost mymapscript.groovy
```

If the truststore and keystore java options are set in `cassandra-env.sh`, the command is simplified:
Using DataStax Enterprise advanced functionality

```
$ java -jar dse-graph-loader.jar -kerberos true -sasl dsename -graph new -address localhost mymapscript.groovy
```

Runtime parameters

Some modifications are necessary if certain conditions must be set. For instance, the JAR file can be run directly to use Java modifiers, or the graphloader script may be modified to allow additional parameters to be set.

If a large data set is loaded, configure the heap space to cache all vertices. This command runs Java and calls the jar file for DSE Graph Loader. For example:

```
$ java -Xmx10g -jar dse-graph-loader.jar
```

Vertex caching uses a temporary directory to store data during loading. If the temporary directory is not large enough, loading is blocked. To change the location of the temporary directory, use a runtime variable `LOADER_TMP_DIR`:

```
$ LOADER_TMP_DIR=/home/user ./graphloader -graph new -address localhost mymapscript.groovy
```

Successful loading

When graphloader has successfully loaded the data specified, notification of the results are logged to `/var/lib/cassandra/system.log`:

```
2017-02-09 23:27:22 INFO Reporter:97 - Current total additions:
1155735 vertices 1982536 edges 6583940 properties 0 anonymous
```

Tuning graphloader JVM options

The DSE Graph Loader is written in Java and has some configurable JVM tuning in the `graphloader` script.

The default maximum heap size is 10G, generally a good heap size for appropriately sized machine used with graphloader. Two environment variables, `MAX_HEAP_SIZE` and `HEAP_NEWSIZE` were added in DSE 5.0.5 and later. `graphloader` now calculates the values of these two environment variables in the same manner as the DSE database.
If a particular value is desired for either variable, the value can be set directly in the `graphloader` script.

**graphloader API**

`graphloader` mapping options are used to designate the manner in which a data file will be parsed for loading.

**`exists()`**

**Synopsis**

```java
exists()
```

**Description**

When loading edges, often the specified vertices for incoming or outgoing endpoints already exist in the database. The `exists()` method will identify that the vertices do not need creation when the edges are created.

The `exists()` method can also be used to specify that edges already exist.

**Examples**

Identify that the vertices for the outgoing vertices identified in the field `aname` in `outV` already exist in the database and do not need to be created:

```java
load(authorBookInput).asEdges {
  label "authored"
  outV "aname", {
    label "author"
    key "name"
    exists()
  }
  inV "bname", {
    label "book"
    key "name"
  }
}
```

**`ignore`**

**Synopsis**

```java
ignore "fieldName"
```

**Description**

Each record read from an input data file will insert every field included unless `ignore` is used.
Examples

Ignore the field gender in the input data file:

```
ignore "gender"
```

**inE**

**Synopsis**

```
inE "edgeLabel" {
   labelField "fieldName"
   vertex "vertexLabel" {
      label "labelName"
      key "fieldName"
   }
}
```

**Description**

Sets the information for an incoming edge to the given edge label and vertex. The edge label must already exist. `labelField` is optional.

**Examples**

Set the incoming edge in a mapping script to FridgeSensor.

```
inE "authored", {
   vertex "author", {
      label "author"
      key "name"
   }
}
```

The vertex with its `label (page 564)` and `key (page 563)` must be set along with `inE`.

**isNew()**

**Synopsis**

```
isNew()
```

**Description**

The `isNew()` method will identify that vertices or edges need creation during the loading process. This method is used instead of the graphloader parameter `load_new` when only a portion of the loading needs identification. `load_new` requires either the entire creation of all vertices and edges during loading to be true or false.

**Examples**

Identify that the edges between existing author vertices and existing book vertices will be created as new edges during the loading into the database:
Using DataStax Enterprise advanced functionality

```java
load(authorBookInput).asEdges {
  isNew()
  label "authored"
  outV "aname", {
    label "author"
    key "name"
    exists()
  }
  inV "bname", {
    label "book"
    key "name"
    exists()
  }
}
```

**inV**

**Synopsis**

**DSE5.1.2 and earlier:**

```java
inV "field_name", {
  label "field_name"
  [ key "key_name" | key key1_name: "key1_name", key2_name: "key2_name" ]
}
```

**DSE5.1.3 and later:**

```java
inV {
  label "field_name"
  [ key "key_name" | key key1_name: "key1_name", key2_name: "key2_name" ]
  ignore "field_name"
}
```

**Description**

In DSE versions 5.1.2 and earlier, sets the field name in the input file that will define the incoming vertex of an edge. Both `inV` and `outV` (page 565) must be defined in an edge mapping statement. In DSE 5.1.3 and later, the `field_name` is deleted from between the `inV` keyword and the `{`.

**Examples**

DSE 5.1.2 and earlier: Sets the field name for the incoming vertex in a mapping script to `fridgeSensor`.

```
//Sample line read:
// cityId|sensorId|name
// santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus
// or JSON
// {"sensor": {"cityId": "santaCruz", "sensorId": "93c4ec9b-68ff-455e-8668-1056ebc3689f"}, "name": "asparagus"}
```
Using DataStax Enterprise advanced functionality

The incoming vertex has a vertex label of `fridgeSensor`, the particular vertex is defined as the one with
the cityId of santaCruz and a sensorId of
93c4ec9b-68ff-455e-8668-1056ebc3689f
inV "fridgeSensor", {
    label "fridgeSensor"
    key cityId:"cityId", sensorId:"sensorId"
}

The field name in the input file that defines the outgoing vertex is `fridgeSensor`, the vertex has a vertex label of `fridgeSensor`, and the composite key value `cityId, sensorId` is supplied in the input file field set in this statement. The label (page 564) and key (page 563) must be set along with inV.

DSE5.1.3 and later:

```
//Sample line read:
// cityId|sensorId|homeId|name
// santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus
// or JSON
// {"sensor": {"cityId": "santaCruz", "sensorId":
// "93c4ec9b-68ff-455e-8668-1056ebc3689f"}, "name": "asparagus"}
```

```
// The incoming vertex has a vertex label of `fridgeSensor`, the
particular vertex is defined as the one with
the cityId of santaCruz and a sensorId of
93c4ec9b-68ff-455e-8668-1056ebc3689f
inV {
    label "fridgeSensor"
    key cityId: "cityId", sensorId: "sensorId"
    exists()
    ignore "homeId"
    ignore "name"
}
```

**key**

**Synopsis**

```
key "fieldName"
```

**Description**

Each record read from an input data file must be unique to avoid duplication. `key` defines a simple unique key for this element comprised of a single field and associated property key name.

**Examples**

Set the key in a mapping script to `name`:

```
key "name"
```
If the data file includes unique ids, such as a GraphSON or Gryo file written from DataStax Enterprise, the key can be set to identify the id:

```plaintext
key "-id" "id"
```

where `-id` defines that the id is found in the data file, and `id` renames the field to `id` in the loaded file.

Set a key in a mapping script to a composite custom id:

```plaintext
key city_id: "city_id", sensor_id: "sensor_id"
```

This definition uses the following pattern:

```plaintext
key <csv_column_name1>: "vertex_property_key1", <csv_column_name2>: "vertex_property_key2"
```

where `<csv_column_name>` is the column in the input file that specifies the value to be assigned to the `vertex_property_key` in the graph.

**label**

**Synopsis**

```plaintext
label "labelName"
```

**Description**

Sets the label of the vertex to the given name. The vertex label must already exist.

`label` can be used in both vertex and incident edge mapping (`inE (page 561)`, `outE (page 565)`).

**Examples**

Set the label in a mapping script to `recipe`.

```plaintext
label "recipe"
```

**labelField**

**Synopsis**

```plaintext
labelField "fieldName"
```

**Description**

Sets the label of the vertex to the name associated with the given field in the input data file. The vertex label must already exist.

`labelField` can be used in both vertex and incident edge mapping (`inE (page 561)`, `outE (page 565)`).
Examples

Set the label in a mapping script to the field name `type`.

```
labelField "type"
```

The contents of the field `type` will designate the vertex label. For instance, if a record in the data file has the field `type` entered as `author`, then the record will be read into a vertex with the vertex label set to `author`. The next record might instead have a value of `recipe` for the `type` field, and the data will be read into a vertex with a vertex label set to `recipe`. Thus, mixed sets of data can be read from a single input data file.

**outE**

Synopsis

```
outE "edgeLabel" {
    labelField "fieldName"
    vertex "vertexLabel" {
        label "labelName"
        key "fieldName"
    }
}
```

Description

Sets the information for an outgoing edge to the given edge label and vertex. The edge label must already exist. `labelField` is optional.

Examples

Set the outgoing edge in a mapping script to `ingredient`.

```
outE "authored", {
    vertex "book", {
        label "book"
        key "name"
    }
}
```

The vertex with its label (page 564) and key (page 563) must be set along with `outE`.

**outV**

Synopsis

```
DSE5.1.2 and earlier:
outV "fieldName", {
    label "fieldName"
    [ key "key_name" | key key1_name: "key1_name", key2_name: "key2_name" ]
    [ exists() ]
```

DSE 6.7 Developer Guide (Latest version)  Page 565
Using DataStax Enterprise advanced functionality

DSE5.1.3 and later:

```c
outV {  
    label "field_name"
    [ key "key_name" | key key1_name: "key1_name", key2_name: "key2_name"
    ignore "field_name"
    [ exists() ]
}
```

Description

In DSE versions 5.1.2 and earlier, sets the field name in the input file that will define the outgoing vertex of an edge. Both `outV` and `inV` (page 562) must be defined in an edge mapping statement. In DSE 5.1.3 and later, the `field_name` is deleted from between the `outV` keyword and the `{`.

Examples

DSE 5.1.2 and earlier: Set the field name for the outgoing vertex of an edge in a mapping script to `ingredient`.

```c
//Sample line read:
// city_id|sensor_id|name
// santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus

// The outgoing vertex has a vertex label of ingredient, the particular vertex is defined as the one with
// the name of asparagus

outV "ingredient", {  
    label "ingredient"
    key "name"
}
```

The field name in the input file that defines the outgoing vertex is `ingredient`, the vertex has a vertex label of `ingredient`, and the key value `name` is supplied in the input file field set in this statement. The `label` (page 564) and `key` (page 563) must be set along with `outV`.

DSE 5.1.3 and later:

```c
//Sample line read:
// cityId|sensorId|homeId|name
// santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus
// or JSON
// {"sensor": {"cityId": "santaCruz", "sensorId":  
// "93c4ec9b-68ff-455e-8668-1056ebc3689f"}, "name": "asparagus"}

// The incoming vertex has a vertex label of fridgeSensor, the particular vertex is defined as the one with
```
Using DataStax Enterprise advanced functionality

// the cityId of santaCruz and a sensorId of 93c4ec9b-68ff-455e-8668-1056ebc3689f
outV {
  label "ingredient"
  key "name"
  exists()
  ignore "homeId"
  ignore "cityId"
  ignore "sensorId"
}

**property**

**Synopsis**

```
property [ "csv_column" ] "propertyName"
```

**Description**

Identifies a property of a vertex to the given field name that will be mapped onto a property from an input data file. If a CSV column is named differently than the field in the graph, the CSV column name may be optionally set. If no property is set, all properties are read. If `ignore` ([page 560](#)) is used, a property will be bypassed.

During the graphloader stage 1 schema discovery, if a property key sequentially occurs multiple times on the same vertex, then the property is considered to be a multi-property. This is a reasonable deduction; however, it might change the discovered schema in certain fringe cases. Typically, in JSON input data, nested lists are mapped to multi-cardinality properties and should be read in that manner. If multi-cardinal fields exist in the input file, schema must define those elements prior to loading. Another example is a JDBC column that is of data type `java.sql.Array`.

**Examples**

Set `property` in a mapping script to `gender`. The values of the property will be read based on the values set in the `value` field of `badge`.

```
property "gender"
```

For this mapping, the data could be:

```
{ "name":"Jane Doe", "gender":"M" }
```

Set `property` to read a column `nick` in a CSV file to `nickname` in a mapping script.

```
property "nick" "nickname"
```

For this mapping, the data could be:

```
name, nick
```
vertex

Synopsis

vertex "fieldName", {
  label "labelName"
  key "fieldName"
}

Description

Each record read from an input data file will have vertex information. vertex defines the label and key for a vertex that will be read. If no property values are set, all properties in the record will be input.

Examples

Set the parameters for a vertex in a mapping script:

```
vertex "recipe", {
  label "recipe"
  key "name"
}
```

For this vertex, each vertex created with the vertex label recipe will be created from a record in the input file in which the vertex label is defined as recipe and the key field as name.

vertexProperty

Synopsis

vertexProperty "propertyName"

Description

Sets the vertex property of a vertex to the given name.

Examples

Set vertexProperty in a mapping script to badge. The values of the vertex property will be read based on the values set in the value field of badge.

```
vertexProperty "badge", {
  "value" "since"
}
```

For this mapping, the data could be:
Dropping graph data

Data can be dropped as follows:

**Drop data**

- To drop all data without dropping a graph and schema, drop all vertices.

```java
g.V().drop().iterate()
```

- To drop specific data, such as all *person* vertices, identify the vertices along with a drop traversal step.

```java
g.V().hasLabel('person').drop()
```

**Note:** If a very large number of vertices will be dropped with the command shown above, DSE Graph may complain. In that case, modify the `drop()` command in the following manner:

```java
g.V().hasLabel('person').limit(100).drop()
```

and repeat until all vertices are dropped.

- To drop a specific value, such as *person* vertices, identify the vertices along with a drop traversal step.

```java
g.V().hasLabel('person').properties('gender').hasValue('M').drop()
```

This query will drop the gender value for all person vertices that have a `gender` value of `M`.

```sql
gremlin> g.V().hasLabel('person').valueMap()
==>{gender=[F], name=[Julia Child]}
==>{gender=[F], name=[Patricia Curtan]}
==>{gender=[F], name=[Kelsie Kerr]}
==>{gender=[F], name=[Simone Beck]}
==>{gender=[F], name=[Alice Waters]}
==>{name=[James Beard]}
==>{name=[Fritz Streiff]}
==>{name=[Emeril Lagasse]}
```

- To drop a property key from an edge, such as `reviewed` edges, identify the edges, the property key `stars` along with a drop traversal step.

```java
g.E().hasLabel('reviewed').properties('stars').drop()
```
This query will drop the property key \texttt{stars} for all edges that have a \texttt{rated} edge label.

\begin{verbatim}
g.E().hasLabel('reviewed').properties('stars').valueMap()
\end{verbatim}

returns no values.

\textbf{Important:} When deleting schema elements, data for edge labels and meta-properties will not always be removed from the underlying database tables. If vertex labels are removed, edge data to vertices with that vertex label will still exist, but will be filtered out during any traversal. If those edge labels are added back to the graph, then the stale edge data will reappear. The same is true for meta-properties; if meta-properties are readded to property keys, then stale meta-property data will reappear. Dropping vertices with this command will also drop all edges associated with the vertices. Any vertex at the other end of an edge dropped will remain, but the edges and edge properties will be hidden from traversals.

\textbf{Warning:} For data created earlier than DSE 5.0.5, conditions exist that will drop all edges as well as the edge property during a property key drop. See Dropping edge property drops edges

\section*{Discovering properties about graphs and traversals}

After schema and data are inserted into a graph, it is important to verify that the information is correct. Checking simple information about inserted data is a good way to get started with traversals. The \texttt{graph.schema()} calls can be used to check how the graph is storing data.

- Use the graph traversal instance \texttt{g} to check if data is loaded by checking the count of vertices. Note that the command is a remote command to Gremlin Server, as are all commands of discovery shown below.

\begin{verbatim}
g.V().count()
\end{verbatim}

\begin{verbatim}\
==>56
\end{verbatim}

- Check the properties of a loaded vertex. Find all the information for the vertex with an name value of Julia Child.

\begin{verbatim}
g.V().has('name','Julia Child').valueMap()
\end{verbatim}

\begin{verbatim}\
==>\{gender=[F], name=[Julia Child]\}
\end{verbatim}

- Check the properties of a loaded edge. Find all the information for the edges with a label of rated.

\begin{verbatim}
g.E().hasLabel('rated').values()
\end{verbatim}
Find the id information for vertices:

```java
g.V().hasLabel('FridgeSensor').id()
```

```java
// FridgeSensor vertices:
(~label=FridgeSensor,
 sensor_id=93c4ec9b-68ff-455e-8668-1056ebc3689f,
 city_id=santaCruz)

(~label=FridgeSensor, sensor_id=9c23b683-1de2-4c97-a26a-277b3733732a, city_id=sacramento)

(~label=FridgeSensor, sensor_id=eff4a8af-2b0d-4ba9-a063-c170130e2d84, city_id=sacramento)
```

Discover schema information using a `describe()` step. This traversal step provides a sorted list of the same information as the next alternative below.

```java
schema.describe()
```

```java
// Schema properties:
schema.propertyKey("member_id").Smallint().single().create()
schema.propertyKey("instructions").Text().single().create()
schema.propertyKey("amount").Text().single().create()
schema.propertyKey("gender").Text().single().create()
schema.propertyKey("year").Int().single().create()
schema.propertyKey("calories").Int().single().create()
schema.propertyKey("stars").Int().single().create()
schema.propertyKey("community_id").Int().single().create()
schema.propertyKey("ISBN").Text().single().create()
schema.propertyKey("name").Text().single().create()
schema.propertyKey("comment").Text().single().create()
schema.propertyKey("timestamp").Timestamp().single().create()
schema.edgeLabel("authored").multiple().create()
schema.edgeLabel("rated").multiple().properties("timestamp", "stars", "comment").create()
schema.edgeLabel("includedIn").multiple().create()
schema.edgeLabel("created").multiple().properties("year").create()
schema.edgeLabel("includes").multiple().properties("amount", "calories").create()
schema.propertyKey("meal").properties("name", "timestamp", "calories").create()
schema.propertyKey("ingredient").properties("name").create()
schema.propertyKey("author").properties("name", "gender").create()
schema.propertyKey("book").properties("name", "year", "ISBN").create()
schema.propertyKey("recipe").properties("name", "instructions").create()
schema.propertyKey("reviewer").properties("name").create()
schema.edgeLabel("rated").connection("recipe", "reviewer").connection("reviewer", "recipe").add()
```
Using DataStax Enterprise advanced functionality

```java
schema.edgeLabel("created").connection("recipe", "author").add()
schema.edgeLabel("includes").connection("ingredient", "recipe").connection("recipe", "ingredient").add()
gremlin> schema.edgeLabel('includes').describe()
==>
```

• An alternative to discover schema information uses a `valueMap()` step on the traversal.

```java
schema.traversal().V().valueMap()
```
Caution: Using `valueMap()` without specifying properties can result in slow query latencies, if a large number of property keys exist for the queried vertex or edge. Specific properties can be specified, such as `valueMap('name')`.

- Changing `valueMap()` to `valueMap(true)` adds the id for each field.

```plaintext
graph.schema().traversal().V().valueMap(true)
```
Using DataStax Enterprise advanced functionality

• Running `traversal()` will supply information about the number of schema element exist for vertices and edges, as well as the `TraversalSource` type.

```java
schema.traversal()
```

```java
==>graphtraversalsource[tinkergraph[vertices:58 edges:106], standard]
```

• A list of all vertex labels using utilities `split()` and `grep()`.

```java
schema.describe().split('\n').grep(\-/.*vertexLabel.*\/)
```

```
gremlin> schema.describe().split('\n').grep(\-/.*vertexLabel.*\/)
```
Creating queries using traversals

DSE Graph can create complex queries that traverse the relationships of the graph structure. If the complex queries require real-time results, DSE Graph is the best product for discovering answers. Start with the Quick Start (page 378) traversals that increase in complexity in a stepwise fashion. The examples shown here will continue with the Recipe Toy Graph example (page 443).

Additional complex Gremlin recipes can also be found at Apache TinkerPop Recipes.

Anatomy of a graph traversal

Structure of a graph traversal

Simple traversals can be complex, but generally do not employ specialized techniques such as recursion or branching.

Break down the chain of a graph traversal into traversal steps:

```
g.V().hasLabel('recipe').count()
```

This graph traversal to find the number of recipes in the graph has four parts:

The graph traversal `g`

`g` will return an error if run alone

All vertices are gathered with `v()`

All the vertices will be returned. A sample of the result:

```
gremlin> g.V().hasLabel('recipe').count()
gremlin> g.V().hasLabel('recipe').count()
gremlin> g.V().hasLabel('recipe').count()
gremlin> g.V().hasLabel('recipe').count()
gremlin> g.V().hasLabel('recipe').count()
```
Filter out the vertices labeled as a recipe with `hasLabel('recipe')`
Only the vertices that are recipes will be returned:

```gremlin
// Filter out the vertices labeled as a recipe with hasLabel('recipe')

g.V().hasLabel('recipe')
  // The result
  .where(out().label('recipe').has('recipeId', 2003))
  .where(out().label('recipe').has('recipeId', 2005))
  .where(out().label('recipe').has('recipeId', 2006))
  .where(out().label('recipe').has('recipeId', 2007))
  .where(out().label('recipe').has('recipeId', 2001))
  .where(out().label('recipe').has('recipeId', 2002))
  .where(out().label('recipe').has('recipeId', 2004))
  .where(out().label('recipe').has('recipeId', 2008))
```

Count the number of vertices with `count()`
The number of vertices returned from the last traversal step is totalled:

```gremlin
// Count the number of vertices with count()

g.V().hasLabel('recipe').count()
  // The result
  .where(out().label('recipe').has('recipeId', 2003))
  .where(out().label('recipe').has('recipeId', 2005))
  .where(out().label('recipe').has('recipeId', 2006))
  .where(out().label('recipe').has('recipeId', 2007))
  .where(out().label('recipe').has('recipeId', 2001))
  .where(out().label('recipe').has('recipeId', 2002))
  .where(out().label('recipe').has('recipeId', 2004))
  .where(out().label('recipe').has('recipeId', 2008))
```

Graph traversal with edges

Before trying the traversals displayed below, run the following script either in Studio (copy and paste) or Gremlin console (`:load /tmp/generateReviews.groovy`):

```java
// reviewer vertices
johnDoe = graph.addVertex(label, 'person', 'personId', 11, 'name', 'John Doe')
johnSmith = graph.addVertex(label, 'person', 'personId', 12, 'name', 'John Smith')
janeDoe = graph.addVertex(label, 'person', 'personId', 13, 'name', 'Jane Doe')
sharonSmith = graph.addVertex(label, 'person', 'personId', 14, 'name', 'Sharon Smith')
betsyJones = graph.addVertex(label, 'person', 'personId', 15, 'name', 'Betsy Jones')

beefBourguignon = g.V().has('recipe', 'recipeId', 2001, 'name', 'Beef Bourguignon').tryNext().orElseGet {graph.addVertex(label, 'recipe', 'recipeId', 2001, 'name', 'Beef Bourguignon')}
spicyMeatLoaf = g.V().has('recipe', 'recipeId', 2005, 'name', 'Spicy Meatloaf').tryNext().orElseGet {graph.addVertex(label, 'recipe', 'recipeId', 2005, 'name', 'Spicy Meatloaf')}
carrotSoup = g.V().has('recipe', 'recipeId', 2007, 'name', 'Carrot Soup').tryNext().orElseGet {graph.addVertex(label, 'recipe', 'recipeId', 2007, 'name', 'Carrot Soup')}

// reviewer - recipe edges
johnDoe.addEdge('reviewed', beefBourguignon, 'timestamp', Instant.parse('2014-01-01T00:00:00.000Z'), 'stars', 5, 'comment', 'Pretty tasty!')
johnSmith.addEdge('reviewed', beefBourguignon, 'timestamp', Instant.parse('2014-01-23T00:00:00.000Z'), 'stars', 4)
```

```gremlin
// reviewer - recipe edges

johnDoe.addEdge('reviewed', beefBourguignon, 'timestamp', Instant.parse('2014-01-01T00:00:00.000Z'), 'stars', 5, 'comment', 'Pretty tasty!')
johnSmith.addEdge('reviewed', beefBourguignon, 'timestamp', Instant.parse('2014-01-23T00:00:00.000Z'), 'stars', 4)
```
janeDoe.addEdge('reviewed', beefBourguignon, 'timestamp', Instant.parse('2014-02-01T00:00:00.00Z'), 'stars', 5, 'comment', 'Yummy!')

sharonSmith.addEdge('reviewed', beefBourguignon, 'timestamp', Instant.parse('2015-01-01T00:00:00.00Z'), 'stars', 3, 'comment', 'It was okay.]

johnDoe.addEdge('reviewed', spicyMeatLoaf, 'timestamp', Instant.parse('2015-12-31T00:00:00.00Z'), 'stars', 4, 'comment', 'Really spicy - be careful!')

sharonSmith.addEdge('reviewed', spicyMeatLoaf, 'timestamp', Instant.parse('2014-07-23T00:00:00.00Z'), 'stars', 3, 'comment', 'Too spicy for me. Use less garlic.')

test

janeDoe.addEdge('reviewed', carrotSoup, 'timestamp', Instant.parse('2015-12-30T00:00:00.00Z'), 'stars', 5, 'comment', 'Loved this soup! Yummy vegetarian!')

Any number of traversal steps can be chained into a traversal, filtering and transforming the graph data as required. In some cases, edges will be the result, and perhaps unexpected. Consider the following traversal:

g.V().hasLabel('recipe').has('name', 'Beef Bourguignon').inE().values('comment')

This graph traversal begins as the last traversal did with g.V().hasLabel('recipe'). It is then followed by:

**A traversal step to pick only the vertices with the recipe title specified**

The filter should capture one recipe if recipe titles are unique.

gremlin> g.V().hasLabel('recipe').has('name', 'Beef Bourguignon')

```
==>v[~label=recipe, recipeId=2001]
```

**A traversal step that retrieves incoming edges**

Notice from the two edges sampled from the complete result that edges with any label are filtered with this step. Using inE('reviewed') would be more precise if the target result are only ratings.

gremlin> g.V().hasLabel('recipe').has('name', 'Beef Bourguignon').inE()

```
===>e[~label=includedIn, ~out_vertex={~label=ingredient, ingredId=3001}, ~in_vertex={~label=recipe, recipeId=2001}, ~local_id=95672dd-2e2d-11e8-8043-bf87ac83b9] [{~label=ingredient, ingredId=3001}-includedIn->{~label=recipe, recipeId=2001}]
===>e[~label=includedIn, ~out_vertex={~label=ingredient, ingredId=3002}, ~in_vertex={~label=recipe, recipeId=2001}, ~local_id=9567ca17-2e2d-11e8-8043-bf87ac83b9] [{~label=ingredient, ingredId=3002}-includedIn->{~label=recipe, recipeId=2001}]
===>e[~label=includedIn, ~out_vertex={~label=ingredient, ingredId=3003},
```
Parsing out the comment property from the rated edges

Here, the \texttt{inE()} is specified with the edge label \texttt{reviewed}. The property values are retrieved for the property key \texttt{comment}:

```gremlin
gremlin> g.V().hasLabel('recipe').has('name', 'Beef Bourguignon').inE('reviewed').values('comment')
==>
Yummy!
==>
Pretty tasty!
```
Building graph traversals one step at a time can yield interesting results and insight into how to create traversals.

The path of a graph traversal

A traversal step exists that will show the path taken by a graph traversal. First, find the results for a traversal that answers the question about what recipes that list beef and carrots as ingredients are included in the cookbooks, given the cookbook and recipe title?

```groovy
g.V().hasLabel('ingredient').has('name', within('beef', 'carrots')).in().as('Recipe').
   out().hasLabel('book').as('Book').
   select('Book', 'Recipe').by('name').
   by('name')

==> {Book=The Art of French Cooking, Vol. 1, Recipe=Beef Bourguignon}
==> {Book=The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution, Recipe=Carrot Soup}
```

One expects that the traversal path will be from ingredient to recipe to book. To check if this assumption is correct, add `path()` to the end of the traversal.

```groovy
g.V().hasLabel('ingredient').has('name', within('beef', 'carrots')).
   in().as('Recipe').
   out().hasLabel('book').as('Book').
   select('Book', 'Recipe').
   by('name').by('name').path()

==> [v[~label=ingredient, ingredId=3001]], v[~label=recipe, recipeId=2001],
   v[~label=book, bookId=1001],
   {Book=The Art of French Cooking, Vol. 1, Recipe=Beef Bourguignon}]
==> [v[~label=ingredient, ingredId=3028]], v[~label=recipe, recipeId=2007],
   v[~label=book, bookId=1004],
   {Book=The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution, Recipe=Carrot Soup}]
```

For each case, notice that the traversal does follow the expected path.

Traversal metrics

In addition to tracing the output of each graph traversal step, metrics can produce interesting insights as well. To add metrics to the last traversal shown, use some additional chained steps:

```groovy
g.V().has('recipe', 'name', 'Beef Bourguignon').inE().values('comment').profile()
```

---

**Traversal Metrics**

<table>
<thead>
<tr>
<th>Step</th>
<th>Count</th>
<th>Traversers</th>
<th>Time (ms)</th>
<th>% Dur</th>
</tr>
</thead>
</table>

---
The type of traversal step is listed, along with the number of traversers and the time to complete the traversal step. If a traversal step can be processed in parallel, multiple traversers will be employed to retrieve data. Some traversal steps are graph-global.
requiring retrieval from the entire graph; DsegGraphStep is a graph-global retrieval that
finds vertices that match certain conditions. Other traversal steps are graph-local walks
and can be processed in parallel; DsegVertexStep is a graph-local walk that walks through
the graph along constrained paths. DSE Graph uses automatic query optimization to
determine the traversal strategies to efficiently use any index structures that exist.

Looking at the metrics, the question of performance comes to mind. For instance, is there
any way to optimize the traversal shown above? In fact, a simple modification results in a
time savings:

```
gremlin> g.V().has('recipe', 'name', 'Beef Bourguignon').inE('reviewed').values('comment').profile()
```

<table>
<thead>
<tr>
<th>Step</th>
<th>Count</th>
<th>Traversers</th>
<th>Time (ms)</th>
<th>% Dur</th>
</tr>
</thead>
<tbody>
<tr>
<td>DsegGraphStep(vertex,[],(label = recipe &amp; name ...</td>
<td>1</td>
<td>1</td>
<td>7.109</td>
<td>83.34</td>
</tr>
<tr>
<td>query-optimizer</td>
<td></td>
<td></td>
<td>0.342</td>
<td></td>
</tr>
<tr>
<td>_condition=((label = recipe &amp; name = Beef Bourguignon) &amp; (true)) query-setup</td>
<td></td>
<td></td>
<td>0.013</td>
<td></td>
</tr>
<tr>
<td>_isFitted=true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_isSorted=false</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_isScan=false</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index-query</td>
<td></td>
<td></td>
<td>6.110</td>
<td></td>
</tr>
<tr>
<td>_indexType=Search</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_usesCache=false</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| _statement=SELECT "recipeId" FROM "DSE_GRAPH_QUICKSTART"."recipe_p" WHERE "solr_query" = '{"q":"*:*","fq":['name:Beef Bourguignon"
   f\\ Bourguignon"]'} LIMIT ?; with params (java.lang.Integer) 50000 |       |            |           |       |
| _options=Options{consistency=Optional[ONE], serialConsistency=Optional.empty, fallbackConsistency=Option |       |            |           |       |
| al.empty, pagingState=null, pageSize=-1, user=Optional.empty, waitForSchemaAgreement=true, async=c=true} |       |            |           |       |
| DsegVertexStep(IN,[reviewed],edge,(direction = ...) | 4     | 4          | 1.234    | 14.47 |
| query-optimizer                           |       |            | 0.187     |       |
| _condition=((direction = IN & label = reviewed) & (true)) query-setup |       |            | 0.009    |       |
| _isFitted=true                             |       |            |           |       |
| _isSorted=false                           |       |            |           |       |
| _isScan=false                             |       |            |           |       |
| vertex-query                              |       |            | 0.474     |       |
| _usesCache=false                          |       |            |           |       |
Using DataStax Enterprise advanced functionality

```sql
\_statement=SELECT * FROM "DSE_GRAPH_QUICKSTART"."recipe_e" WHERE
"recipeId" = ? AND "~edge_label_id" = ? LIMIT ? ALL
OW FILTERING; with params (java.lang.Integer) 2001,
(java.lang.Integer) 65633, (java.lang.Integer) 50000
\_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async
c=true}
\_usesIndex=false
DsegPropertiesStep([comment],value,label = comm...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>0.186</th>
<th>2.19</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>&gt;TOTAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>8.530</td>
<td>-</td>
</tr>
</tbody>
</table>
```

The change made is subtle. The traversal steps, `inE()` has been replaced by `inE('reviewed')`, to find only the edges that are reviews. Although each measurement can vary, generally the second traversal will outperform the first traversal.

DataStax Studio provides more easy to use metrics, in which scrolling over or selecting any phase of the traversal can be explored:

**Using user-defined vertex ids**

Vertices can be filtered using a `hasId()` traversal step and providing a particular user-defined vertex id:

```java
g.V().hasId(['~label':'fridgeSensor', 'stateId':31, 'cityId':100, 'sensorId':1]).valueMap()
```

A vertex can be queried directly using a user-defined vertex id:

```java
g.V(['~label':'fridgeSensor', 'stateId':31, 'cityId':100, 'sensorId':1]).valueMap()
```

**Using indexes**

Global indexes can be used in graph traversal queries for the first traversal step reached after the `V()` step, and are used to trim down the number of vertices that are initially fetched. Remember that a search index must be used if two or more properties are used for global indexing. In general, the traversal step involves a vertex label and can include
a property key and a particular property value. In a traversal, the step following `g.V()` is generally the step in which an index will be consulted. If a mid-traversal `V()` step is called, then an additional indexed step can be consulted to narrow the list of vertices that will be traversed.

**Note:** Graph traversals will only use indexes if the both the vertex label and property key are specified. If both are not specified, indexing will not be used and a full graph scan for the property key can result. If full graph scan is disabled, a query will fail, as shown in this example where a property is specified, but a vertex label is not specified:

```
g.V().has('name', 'Julia Child')
```

Could not find an index to answer query clause and `graph.allow_scan` is disabled:

```
((label = FridgeSensor & name WITHIN [Julia Child]) | (label = author & name WITHIN [Julia Child]) |
(label = book & name WITHIN [Julia Child]) | (label = ingredient & name WITHIN [Julia Child]) |
(label = meal & name WITHIN [Julia Child]) | (label = recipe & name WITHIN [Julia Child]) |
(label = reviewer & name WITHIN [Julia Child]))
```

Edge indexes and property indexes (vertex-centric indexes) can be used to narrow the query after a global index has found the starting vertex. They allow definition of the edges that will be followed or the meta-properties that will be used to further restrict the query.

**Global index**

- The graph traversal shown uses an index to discover certain person vertices to start the query.

```
g.V().has(person, 'name', 'Emeril Lagasse').out('created').values('name')
```

This graph traversal uses an index, if the index exists, because the traversal step `has('person', 'name', 'Emeril Lagasse')` identifies the vertex label and the property key indexed. After finding the initial vertex to traverse from, the outgoing `created` edges are walked and the adjacent vertices are listed by `name`. This graph traversal shows the importance of using the vertex label in combination with
Using DataStax Enterprise advanced functionality

the property key, as two different elements, persons and recipes, use the same property key name.

Checking for the use of indexing can be accomplished with the profile() method:

```gremlin
gremlin> g.V().has('person', 'name', 'Emeril Lagasse').out('created').values('name').profile()
==>
Traversal Metrics
Step                                      Count  Traversers       Time (ms)    % Dur
=============================================================================================================  
DsegGraphStep(vertex, [], (label = person & name ...
  1           1           8.427    27.42
query-optimizer
  \_condition=((label = person & name = Emeril Lagasse) &
(true))
query-setup
  \_isFitted=true
  \_isSorted=false
  \_isScan=false
index-query
  6.514
  \_indexType=Materialized
  \_usesCache=false
  \_statement=SELECT "personId" FROM "dse60"."person_p_byName" WHERE "name" = ? LIMIT ?; with params (java.
  lang.String) Emeril Lagasse, (java.lang.Integer) 50000  
  \_options=Options{consistency=Optional[ONE],
  serialConsistency=Optional.empty, fallbackConsistency=Option
  al.empty, pagingState=null, pageSize=-1,
  user=Optional.empty, waitForSchemaAgreement=true, async
  c=true} ...
```

Note the index-query used in the first step DsegGraphStep identifies the index type as materialized. If an index was not used, index-query would be missing from the profile output.

**Edge index**
- An edge index can narrow the query, such as this one that finds all the outgoing edges for reviews that John Doe wrote that have a rating of greater or equal to 3 stars:

```gremlin
g.V().has('person', 'name', 'John Doe').outE().has('stars', gte(3))```
Using profile() on the query shows that a global index query was used in the initial step, and the output shown here shows that in the second step, the ratedByStars edge index was used to cut the latency of the query.

**Tip:** The local() (page 720) step can be used to affect how an edge index narrows a query.

**Property index**

- A property index can narrow the query, such as this one that finds the countries that Julia Child lived in, starting in the year 1961 (in this case, only one country):

```plaintext
g.V().has('person', 'name','Julia Child').as('person').local(properties('country').has('startYear', 1961)).value().as('country').select('person','country').by('name').by().profile()
```

```sql
gremlin> g.V().has('person', 'name','Julia Child').as('person').
......1>    local(properties('country').has('startYear', 1961)).value().as('country').
......2>    select('person','country').
......3>    by('name').by().profile()
```

```
<table>
<thead>
<tr>
<th>Step</th>
<th>Count</th>
<th>Traversers</th>
<th>Time (ms)</th>
<th>% Dur</th>
</tr>
</thead>
<tbody>
<tr>
<td>DsegGraphStep(vertex,[](label = person &amp; name ...</td>
<td>1</td>
<td>1</td>
<td>1.274</td>
<td>37.35</td>
</tr>
<tr>
<td>query-optimizer</td>
<td></td>
<td></td>
<td>0.253</td>
<td></td>
</tr>
<tr>
<td>_condition=((label = person &amp; name = Julia Child) &amp; (true))</td>
<td></td>
<td></td>
<td>0.008</td>
<td></td>
</tr>
<tr>
<td>query-setup</td>
<td></td>
<td></td>
<td>0.008</td>
<td></td>
</tr>
<tr>
<td>_isFitted=true</td>
<td></td>
<td></td>
<td>0.008</td>
<td></td>
</tr>
<tr>
<td>_isSorted=false</td>
<td></td>
<td></td>
<td>0.008</td>
<td></td>
</tr>
<tr>
<td>_isScan=false</td>
<td></td>
<td></td>
<td>0.008</td>
<td></td>
</tr>
<tr>
<td>index-query</td>
<td></td>
<td></td>
<td>0.557</td>
<td></td>
</tr>
<tr>
<td>_indexType=Materialized</td>
<td></td>
<td></td>
<td>0.557</td>
<td></td>
</tr>
<tr>
<td>_usesCache=false</td>
<td></td>
<td></td>
<td>0.557</td>
<td></td>
</tr>
</tbody>
</table>
```
```plaintext
\_statement=SELECT "personId" FROM "newComp"."person_p_byName" WHERE "name" = ? LIMIT ?; with params (java.lang.String) Julia Child, (java.lang.Integer) 50000
\_options=Options{consistency=Optional[ONE], serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1, user=Optional.empty, waitForSchemaAgreement=true, async=true}

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>DsegHasStep@[person]</td>
<td>0.060</td>
<td>1.76</td>
</tr>
<tr>
<td>LocalStep([DsegPropertiesStep([country],property,(label ...</td>
<td>1.300</td>
<td>38.12</td>
</tr>
<tr>
<td>DsegPropertiesStep([country],property,(label ...</td>
<td>1.149</td>
<td></td>
</tr>
<tr>
<td>query-optimizer</td>
<td>0.239</td>
<td></td>
</tr>
<tr>
<td>_condition=((label = country &amp; startYear = 1961) &amp; (true))</td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>_isFitted=true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_isSorted=false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_isScan=false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vertex-query</td>
<td>0.564</td>
<td></td>
</tr>
<tr>
<td>_usesCache=false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_options=Options{consistency=Optional[ONE], serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1, user=Optional.empty, waitForSchemaAgreement=true, async=true}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_usesIndex=true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DsegHasStep([startYear.eq(1961)])</td>
<td>0.081</td>
<td></td>
</tr>
<tr>
<td>PropertyValueStep</td>
<td>0.026</td>
<td></td>
</tr>
<tr>
<td>SelectStep(last,[person, country],[value(name),...</td>
<td>0.720</td>
<td>21.13</td>
</tr>
<tr>
<td>NoOpBarrierStep(2500)</td>
<td>0.032</td>
<td>0.95</td>
</tr>
<tr>
<td>DsegPropertyLoadStep</td>
<td>0.023</td>
<td>0.69</td>
</tr>
</tbody>
</table>

>TOTAL - - 3.411 -
```
Using profile() on the query shows that a global index query was used in the initial step, and the output shown here shows that in the second SELECT step, the byStartYear property index was used to cut the latency of the query.

Tip: The local() (page 720) step can also be handy for use with property indexes.

Using search indexes

DSE Graph leverages DSE Search indexes (page 469) to efficiently filter vertices by properties, and reducing query latency. DSE Search uses a modified Apache Solr (page 313) to create the search indexes. Graph search indexes can be created using textual, numeric and geospatial data.

It is important to note that traversal queries with search predicates can be completed whether a search index exists or not. However, full graph scans will occur without a search index and performance will degrade severely as the graph grows, an unacceptable solution in a production environment. Create search indexes during schema creation before inserting data and querying the graph. Search indexes will only be created if DSE Search is started in conjunction with DSE Graph. If search indexes are used, the queries must be run on DSE Search nodes in the cluster.

In general, the traversal step will involve a vertex label and can include a property key and a particular property value. In a traversal, the step following g.V() is generally the step in which an index will be consulted. If a mid-traversal V() step is called, then an additional indexed step can be consulted to narrow the list of vertices that will be traversed.

Textual search indexes are by default indexed in both tokenized (TextField) and non-tokenized (StrField) forms. This means that all textual predicates (token, tokenPrefix, tokenRegex, eq, neq, regex, prefix) will be usable with all textual vertex properties indexed. Practically, search indexes should be created using the asString() method only in cases where there is absolutely no use for tokenization and text analysis, such as for inventory categories (silverware, shoes, clothing). The asText() method is used if searching tokenized text, such as long multi-sentence descriptions. The query optimizer will choose whether to use analyzed or non-analyzed indexing based on the textual predicate used.

Property key indexes defined with asText() or undefined (since this is the default) can use the following options for search:

- token (page 588)
- tokenPrefix (page 589)
- tokenRegex (page 589)

Property key indexes defined with asString() can use the following options for search:

- eq/neq (page 590)
- prefix (page 591)
- regex (page 591)
Note: The `eq()` search cannot be used with property key indexes created with `asText()` because they contain tokenized data and are therefore not suitable for exact text matches.

In addition, in DSE 5.1 and later, fuzzy search predicates have been added:

- **phrase (page 591)**
- **fuzzy (page 592)**
- **tokenFuzzy (page 593)**

Two of the predicates, fuzzy and tokenFuzzy, can be used with TextFields and StrFields, respectively, while phrase can be used only with TextFields.

Creating a textual search index

- An example search index from Creating indexes (page 472) for vertex label `recipe` that will be used for all examples below:

  ```java
  schema.vertexLabel('recipe').index('search').search().
  by('instructions').asText().
  by('name').asString().add()
  ```

  This search index uses DSE Search to index `instructions` as full text using tokenization, and `name` as a string. Note that, as of DSE 5.1, only those properties that specifically should be indexed as non-tokenized data must specify `asString()`. If there are properties that specifically should be indexed only as tokenized data, specify `asText()`.

Search using token() methods on full text

- In a traversal query, use a token search to find list the names of all recipes that have the word `Saute` in the instructions. The method `token()` is used with a supplied word.

  ```java
  g.V().has('recipe','instructions', token('Saute')).values('name')
  ```

<table>
<thead>
<tr>
<th>index 1</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Wild Mushroom Stroganoff</td>
</tr>
<tr>
<td>1</td>
<td>Beef Bourguignon</td>
</tr>
<tr>
<td>2</td>
<td>Oysters Rockefeller</td>
</tr>
</tbody>
</table>

Displaying 1 - 3 of 3
Why does this search find these three recipes? Because the instructions for each meet the search requirements:

```
g.V().has('recipe').has('instructions', tokenPrefix('Sea')).values('name','instructions')
```

---

- Raise the beef. Sauté the onions and carrots. Add wine and cook in a Dutch oven at 425 degrees for 1 hour.
- Sauté the shallots, celery, herbs, and seasonings in 3 tablespoons of the butter for 3 minutes. Add the watercress and let it wilt.
- Cook the egg noodles according to the package directions and keep warm. Heat 1 1/2 tablespoons of the oil/wall in a large sauté pan over medium-high heat.

**Search using tokenPrefix() methods on full text**

- In a traversal query, use a token prefix search to list the names of all recipes that have a word that includes a prefix of Sea in the instructions. The method tokenPrefix() is used with a supplied prefix (a set of alphanumeric characters).

```
g.V().hasLabel('recipe').has('instructions', tokenPrefix('Sea')).values('name','instructions')
```

Two recipes are returned, one with the word **Season** in the instructions, and one with the word **seasonings** in the instructions. Case is insensitive in tokenPrefix() indexing.

**Search using tokenRegex() methods on full text**

- In a traversal query, use a token regular expression (regex) search to find all recipes that have a word that includes the regular expression specified. The regex, `.*sea.*in.*`, looks for the letters sea preceded by any number of other characters and followed by any number of other characters until the letters in are found and also followed by any number of other characters in the instructions and list the recipe names. The method tokenRegex() is used with a supplied regex.

```
g.V().hasLabel('recipe').has('instructions', tokenRegex('.*sea.*in.*')).values('name','instructions')
```
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>index</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Oysters Rockefeller</td>
</tr>
<tr>
<td>1</td>
<td>Saute the shallots, celery, herbs, and seasonings in 3 tablespoons of the butter for 3 minutes. Add the watercress and let it wilt.</td>
</tr>
</tbody>
</table>

Note that in this query, only the Oysters Rockefeller recipe is returned because the word **Season** in the Roast Pork Loin recipe does not meet the requirements for the regular expression.

**Search using eq() on non-token methods on strings**

- In a traversal query, use a non-token search to list all recipes that have **Carrot Soup** in the recipe name. Note that this search is case-sensitive, so using **carrot soup** would not find a vertex. The method `eq()` is used with a supplied name.

```g.V().hasLabel('recipe').has('name', eq('Carrot Soup')).values('name')
```

The match is found for the full author name listed. Note that `neq()` can also be used to find all strings that do not match the specified string.

- In a traversal query, use a non-token search to list all recipes that have **Carrot** in the recipe name. The method `eq()` is used with a supplied name.

```g.V().hasLabel('recipe').has('name', eq('Carrot')).valueMap()
```
No match is found, because only a partial name was specified. For `asString()` indexes, the string must match.

Search using `prefix()` on non-token methods on strings

- In a traversal query, use a non-token search to find all authors that have a name beginning with the letter `R`. The method `prefix()` is used with a supplied string.

```gremlin
g.V().hasLabel('recipe').has('name', prefix('R')).values('name')
```

Matches are found for each author name that begins with `R`, provided the recipe name was designated with `asString()` in the search index.

Search using `regex()` on non-token methods on strings

- In a traversal query, use a non-token search to find all recipes that have a name that includes a specified regular expression. The method `regex()` is used with a supplied regex.

```gremlin
g.V().hasLabel('recipe').has('name', regex('.*ee.*')).values('name')
```

Matches are found for each author name that include the regex `.*ee.*` to find all strings that include `ee` preceded and followed by any number of other characters, provided the recipe name was designated with `asString()` in the search index.

Search using `phrase()`

- The `phrase()` predicate is used with properties designated as `TextFields`.

Find the exact phrase `Wild Mushroom Stroganoff` in a recipe name:
Using DataStax Enterprise advanced functionality

```cypher
g.V().hasLabel('recipe').has('name', phrase('Wild Mushroom Stroganoff', 0))
```

The 0 designates that the result must be an exact phrase.

```cypher
v[{~label=recipe, community_id=2123369856, member_id=0}]
```

The vertex for the correct recipe is returned.

- The `phrase()` predicate can be used for proximity searches, to discover phrases that have terms that are within a certain distance of one another in the tokenized text.

```cypher
g.V().hasLabel('recipe').has('name', phrase('Wild Stroganoff', 1))
```

The value of 1 designates that the result must only have words in the recipe name that are one term away from one another. In this example, the variation is the addition of the word `Mushroom`.

```cypher
v[{~label=recipe, community_id=2123369856, member_id=0}]
```

The vertex for the correct recipe is returned. A match for `g.V().hasLabel('recipe').has('name', phrase('Wild Mushroom', 1))` will also return the correct vertex, but `g.V().hasLabel('recipe').has('name', phrase('Mushroom Wild', 1))` will not.

**Search using fuzzy()**

- The `fuzzy()` predicate uses optimal string alignment distance calculations to match properties designated as StrFields. Variations in the letters used in words, such as misspellings, are the focus of this predicate. The edit distance specified refers to the number of transpositions of letters, with a single transposition of letters constituting one edit.

Find the exact name of *James Beard* in an author name:

```cypher
g.V().hasLabel('author').has('name', fuzzy('James Beard', 0)).values('name')
```

The 0 designates that the result must be an exact match.

```cypher
James Beard
```

- Changing the last value in a `fuzzy()` predicate will find misspellings:

```cypher
g.V().hasLabel('author').has('name', fuzzy('James Beard', 1)).values('name')
```

The 1 designates that the result matches with an edit distance of at most one.

```cypher
James Beard, Jmaes Beard
```

If an author vertex exists with the misspelling *Jmaes Beard*, the query shown will find both vertices. The value of 1 finds this misspelling because of the single transposition of the letters *a* and *m*. 
• Note that searching for a misspelling will find the records with the correct spelling, as well as the misspelled name

```java
G.V().hasLabel('author').has('name', fuzzy('Jmase Beard', 2)).values('name')
```

The 2 designates that the result must match with at most two transpositions.

James Beard, Jmaes Beard

If an author vertex exists with the misspelling Jmaes Beard, the query shown will find both vertices. The value of 2 finds both the misspelling because of the single transposition of letters, e and s in Jmaes Beard, as well as the correct spelling with a second transposition of letters from Jmase Beard to James Beard.

Caution: Specifying an edit distance of 3 or greater matches too many terms for useful results. The resulting search index will be too large to efficiently filter queries.

Search using tokenFuzzy()

• The tokenFuzzy() predicate similar to fuzzy(), but searches for variation across individual tokens in analyzed textual data (TextFields).

Find the recipe name that includes the word Wild while searching for the word with a one-letter misspelling:

```java
G.V().hasLabel('recipe').has('name',
  tokenFuzzy('Wlid',1)).values('name')
```

The 1 designates that one letter misspelling (one transposition) is acceptable.

Wild Beef Stroganoff

Using two search indexes for a single traversal query

• Create a second search index like an example search index from Creating indexes (page 472) for vertex label author.

```java
schema.vertexLabel('author').index('search').search().
  by('name').asString().
  by('nickname').ifNotExists().add()
```

This search index will use DSE Search to index nickname as full text using tokenization, and name as a string.

• This traversal query demonstrates a mid-traversal \( V() \) that allows a search index for author as well as a search index for recipe to be used to execute the query. The first index uses a tokenRegex() to find recipe instructions that start with the word Braise; this part of the query is labeled as \( r \) for use later in the query. Then the search index for author is searched for an author name that starts with the letter \( J \), and traversed through an outgoing edge to a vertex where the search found in the first part of the query is found with where(eq('r')).
Using DataStax Enterprise advanced functionality

This query traversal finds the recipe Beef Bourguignon authored by Julia Child, and illustrates some of the complexity that can be successfully used with search indexes.

**Search using geospatial values**

- Geospatial search is used to discover geospatial relationships. Search indexes are used to make such searches possible. First, a search index must be created.

```
schema.vertexLabel('FridgeSensor').index('search').search().by('location').ifNotExists().add()
```

- Some sample data will be helpful for understanding the search results. Two vertices are entered for fridge sensor:

```
graph.addVertex(label, 'FridgeSensor', 'name', 'jones1', 'city_id', 100, 'sensor_id', '60bcae02-f6e5-11e5-9ce9-5e5517507c66', 'location', Geo.point(-118.359770, 34.171221))
graph.addVertex(label, 'FridgeSensor', 'name', 'smith1', 'city_id', 100, 'sensor_id', '61deada0-3bb2-4d6d-a606-a44d963f03b5', 'location', Geo.point(-115.655068, 35.163427))
```

The sensors are named and given a city ID and sensor ID in addition to the location with data type `Point`.

- A query can find all sensors that meet the requirement of being inside the described polygon `Distance` that is designated as a circle with a center at (-110, 30) and a radius of 20 degrees with the method `Geo.inside()`.

```
Distance d = Geo.point(-110,30),20, Geo.Unit.DEGREES
graph.V().hasLabel('FridgeSensor').has('location', Geo.inside(d)).values('name')
```
Using DataStax Enterprise advanced functionality

More information on geospatial queries can be found in Geospatial traversals (page 604).

Search using numerical values

- Search indexes can also be used for non-textual values:

```
schema.propertyKey('name').Text().create()
schema.propertyKey('age').Int().create()
schema.vertexLabel('person').properties('name','age').create()
schema.vertexLabel('person').index('search').search().by('name').by('age').add()
```

This example includes a search index by the integer property age. Here is data to query:

```
graph.addVertex(label, 'person','name','Julia','age',56)
graph.addVertex(label, 'person','name','Emeril','age',48)
graph.addVertex(label, 'person','name','Simone','age',50)
graph.addVertex(label, 'person','name','James','age',52)
```

and the query itself:

```
g.V().has('person','age', gt(50)).values()
```

to find all persons over the age of 50.

```json
=>Julia
=>56
=>James
=>52
```

- To sort the previous search, add additional methods:

```
g.V().hasLabel("person").has("age", gt(50)).order().by("age", incr).values()
```

to get:
Using DataStax Enterprise advanced functionality

Simple Traversals

Returning to the Recipe Toy Graph, let's expand the graph to include reviewers and ratings. Load the following script to add the reviewer vertices and recipe-reviewer edges. You must have inserted the DSE Graph QuickStart data (page 401) previously, so that the recipe vertices exist before loading this script:

```java
// Generates review vertices and edges for Recipe Toy Graph
// :load /tmp/generateReviews.groovy

// reviewer vertices
johnDoe = graph.addVertex(label, 'reviewer', 'name','John Doe')
johnSmith = graph.addVertex(label, 'reviewer', 'name', 'John Smith')
janeDoe = graph.addVertex(label, 'reviewer', 'name','Jane Doe')
sharonSmith = graph.addVertex(label, 'reviewer', 'name','Sharon Smith')
betsyJones = graph.addVertex(label, 'reviewer', 'name','Betsy Jones')
beefBourguignon = g.V().has('recipe', 'name','Beef Bourguignon').tryNext().orElseGet {graph.addVertex(label, 'recipe', 'name', 'Beef Bourguignon')}
spicyMeatLoaf = g.V().has('recipe', 'name','Spicy Meatloaf').tryNext().orElseGet {graph.addVertex(label, 'recipe', 'name', 'Spicy Meatloaf')}
carrotSoup = g.V().has('recipe', 'name','Carrot Soup').tryNext().orElseGet {graph.addVertex(label, 'recipe', 'name', 'Carrot Soup')}

// reviewer - recipe edges
johnDoe.addEdge('rated', beefBourguignon, 'timestamp', '2014-01-01T05:15:00.00Z', 'stars', 5, 'comment', 'Pretty tasty!')
johnSmith.addEdge('rated', beefBourguignon, 'timestamp', '2014-01-23T00:00:00.00Z', 'stars', 4)
janeDoe.addEdge('rated', beefBourguignon, 'timestamp', '2014-02-01T00:00:00.00Z', 'stars', 5, 'comment', 'Yummy!')
sharonSmith.addEdge('rated', beefBourguignon, 'timestamp', '2015-01-01T00:00:00.00Z', 'stars', 3, 'comment', 'It was okay.')
johnDoe.addEdge('rated', spicyMeatLoaf, 'timestamp', '2015-12-31T10:56:00.00Z', 'stars', 4, 'comment', 'Really spicy - be careful!')
sharonSmith.addEdge('rated', spicyMeatLoaf, 'timestamp', '2014-07-23T00:00:00.00Z', 'stars', 3, 'comment', 'Too spicy for me. Use less garlic.')
janeDoe.addEdge('rated', carrotSoup, 'timestamp', '2015-12-30T01:20:00.00Z', 'stars', 5, 'comment', 'Loved this soup! Yummy vegetarian!')
```

Run the script by first identifying the script, and then remotely executing it.
The recipes that were previously entered are queried to assign the result to recipe variables. The variables are then used to create the reviewer-recipe edges. These queries make use of two Apache TinkerPop methods, `tryNext()` and `orElseGet()`; see the Apache TinkerPop Java API for more information.

Exploring recipe ratings

Check if the vertices are created by counting the number of vertices with the `reviewer` label.

```groovy
g.V().hasLabel('reviewer').count()
```

List all the reviewer using `values`:

```groovy
// Get the names of all the reviewers
g.V().hasLabel('reviewer').values('name')
    .asList()
    .collect{
        println it
    }
```

Verifying that the reviewers are created is useful, but creating traversals that answer queries is more important. For instance, what does John Doe say about recipes?

Use a query that identifies a vertex label as `reviewer` with a `name` value of `John Doe`.

```groovy
g.V().has('reviewer', 'name', 'John Doe').outE('rated').values('comment')
```

The use of the outgoing edges command `outE('rated')` to find all the recipes that John Doe has rated allows the value of the property `comments` to be retrieved:

```groovy
Pretty tasty!
Really spicy - be careful!
```
It might be nice to know which recipes John Doe reviewed, so another traversal can be used.

```gremlin
g.V().has('reviewer', 'name', 'John Doe').outE('rated').inV().values('name')
```

resulting in:

```text
=> Beef Bourguignon
=> Spicy Meatloaf
```

To find all the reviews that give a recipe more than 3 stars is a reasonable question to ask. Try a traversal using `gt(3)`, or `greater than 3` to filter the `stars` values:

```gremlin
g.E().hasLabel('rated').has('stars', gt(3)).valueMap()
```

```text
==>[stars:4, timestamp:2014-01-23T00:00:00Z]
==>[comment:Loved this soup! Yummy vegetarian!, stars:5, timestamp:2015-12-30T00:00:00Z]
==>[comment:Yummy!, stars:5, timestamp:2014-02-01T00:00:00Z]
==>[comment:Pretty tasty!, stars:5, timestamp:2014-01-01T00:00:00Z]
==>[comment:Really spicy - be careful!, stars:4, timestamp:2015-12-31T00:00:00Z]
```
The traversal shown finds each edge that is labeled `rated` and filters the edges found to output only those edges with a star rating of 4 or 5. But this traversal doesn't output the answer to the original question. The traversal needs modification to get the incoming vertices with `inV()`, and to list those incoming vertices by name with `values('name')`:

```gremlin
g.E().hasLabel('rated').has('stars', gt(3)).inV().values('name')
```

The results indicate that Beef Bourguignon has been rated three times, although we don't have any reviewer information, just duplication of the recipe title in the list.

Returning to the previous query, let's look for more recent reviews. Adding an additional traversal step to filter by the timestamp can find the 4 and 5 star ratings using `gte(4)` or `greater than or equal to 4`, with a review date of Jan 1, 2015 or later.

```gremlin
g.E().hasLabel('rated').has('stars', gte(4)).has('timestamp', gte(Instant.parse('2015-01-01T00:00:00.00Z'))).valueMap()
```

Chaining traversal steps together can yield very exacting results. For instance, if we added the `inV().values('name')` to the last query, we'd now refine the results to find all 4-5 star reviews since the beginning of the year 2015.

Manipulating the ratings with statistical functions yields interesting answers. For instance, what is the `mean` value of all the recipe ratings?

```gremlin
g.E().hasLabel('rated').values('stars').mean()
```

The results show that the reviewers like the recipes they reviewed, and establishes that reviewers in this sample did not write reviews for recipes that they did not like.

Perhaps a prolific reviewer would have a wider range of reviews. Find the maximum number of reviews that a single reviewer has written.

```gremlin
g.V().hasLabel('reviewer').map(outE('rated').count()).max()
```

This traversal maps all the outgoing edges using `outE('rated')` of each reviewer and counts them, then determines which count has the highest value using `max()`.

Another measure that can be investigated is the mean rating of each reviewer. This traversal query uses a number of Apache TinkerPop traversal steps.
The `as()` step allows display labels to be created for the two items that will be lists, the reviewer's name and the mean stars value for each reviewer. These display labels, reviewer and starCount are then used in a `select()` step that gets each value, first the reviewer's name using `by('name')` and then the starCount using `by(outE('rated').values('stars').mean())`. The `select()` step checks each reviewer vertex and then traverses to discover the associated starCount value.

```
gremlin> g.V().hasLabel('reviewer').as('reviewer','starCount').
    select('reviewer','starCount').
    by('name').
    by(outE('rated').values('stars').mean())
==>[reviewer:Jane Doe, starCount:5.0]
==>[reviewer:Betsy Jones, starCount:NaN]
==>[reviewer:John Doe, starCount:4.5]
==>[reviewer:John Smith, starCount:4.0]
==>[reviewer:Sharon Smith, starCount:3.0]
```

Notice that Betsy Jones is listed as a reviewer, but has not reviewed any recipes. Her starCount lists NaN (not a number). It is clear from the results that Jane Doe really likes at least one recipe, while Sharon Smith does not.

Ordering the results by the starCount, or mean star rating, can allow the highest rater and the lowest rater to be discovered. Here, the traversal steps `order().by(select('starCount'), decr)` use the output of the `select('starCount')` step to order the display in decremental order.

```
gremlin> g.V().hasLabel('reviewer').as('reviewer','starCount').
    select('reviewer','starCount').
    by('name').
    by(outE('rated').values('stars').mean()).
    order().by(select('starCount'), decr)
==>[reviewer:Betsy Jones, starCount:NaN]
==>[reviewer:Jane Doe, starCount:5.0]
==>[reviewer:John Doe, starCount:4.5]
==>[reviewer:John Smith, starCount:4.0]
==>[reviewer:Sharon Smith, starCount:3.0]
```
Betsy Jones and her lack of ratings still cause the listing to be incorrect. We could add a traversal step `limit(1)` to the traversal and get the highest rater, Jane Doe, if Betsy were not listed.

A tricky traversal step, `coalesce()`, is used to change NaN to a zero value.

```gremlin
g.V().hasLabel('reviewer').as('reviewer','starCount').
  select('reviewer','starCount').
  by('name').
  by(coalesce(outE('rated').values('stars'),constant(0)).mean()).
  order().by(select('starCount'), decr)
==>[reviewer:Jane Doe, starCount:5.0]
==>[reviewer:John Doe, starCount:4.5]
==>[reviewer:John Smith, starCount:4.0]
==>[reviewer:Sharon Smith, starCount:3.0]
==>[reviewer:Betsy Jones, starCount:0.0]
```

Note that now Betsy Jones has a `starCount` of 0.0, the true value.

Find the star rating each reviewer has given to recipes:

```gremlin
g.V().hasLabel('reviewer').as('reviewer','rating').out().as('recipe').
  select('reviewer','rating','recipe').
  by('name').
  by(outE('rated').values('stars')).
  by(values('name'))
```

Note how the recipe name is traversed and named with the step modulator `as('recipe')` after the reviewer and rating are labeled from the reviewer vertices with `as('reviewer','rating')`. The first two items in the output listing are retrieved starting at the reviewer vertex while the third item is retrieved from the adjacent recipe vertex.

```text
==>[reviewer=John Doe, rating=5, recipe=Beef Bourguignon]
==>[reviewer=John Doe, rating=5, recipe=Spicy Meatloaf]
==>[reviewer=John Smith, rating=4, recipe=Beef Bourguignon]
==>[reviewer=Jane Doe, rating=5, recipe=Beef Bourguignon]
==>[reviewer=Jane Doe, rating=5, recipe=Carrot Soup]
==>[reviewer=Sharon Smith, rating=3, recipe=Beef Bourguignon]
==>[reviewer=Sharon Smith, rating=3, recipe=Spicy Meatloaf]
```

In general, the most interesting statistic from the reviews answers the question about how many people rated a particular recipe, and what the mean rating is for that particular recipe. The graph traversal starts from a recipe vertex this time, and retrieves the recipe name, the number of reviews by counting the incoming edges with `inE('rated').count()`, and the mean value of the incoming edges with `inE('rated').values('stars').mean()`. The `coalesce()` step shown earlier could be used to change all `NaN` values for `meanRating` into zeroes.

```gremlin
g.V().hasLabel('recipe').as('recipe','numberOfReviews','meanRating').
  select('recipe','numberOfReviews','meanRating').
  by('name').
  by(inE('rated').count()).
```

```text
==>[recipe=Beef Bourguignon, numberOfReviews=1, meanRating=5.0]
==>[recipe=Spicy Meatloaf, numberOfReviews=1, meanRating=5.0]
==>[recipe=Beef Bourguignon, numberOfReviews=3, meanRating=5.0]
==>[recipe=Carrot Soup, numberOfReviews=3, meanRating=5.0]
==>[recipe=Spicy Meatloaf, numberOfReviews=3, meanRating=5.0]
```
**Using DataStax Enterprise advanced functionality**

```java
by(inE('rated').values('stars').mean())
```

```java
==>{recipe=Beef Bourguignon, numberOfReviews=4, meanRating=4.25}
==>{recipe=Wild Mushroom Stroganoff, numberOfReviews=0, meanRating=NaN}
==>{recipe=Spicy Meatloaf, numberOfReviews=2, meanRating=3.5}
==>{recipe=Ratatouille, numberOfReviews=0, meanRating=NaN}
==>{recipe=Salade Nicoise, numberOfReviews=0, meanRating=NaN}
==>{recipe=Roast Pork Loin, numberOfReviews=0, meanRating=NaN}
==>{recipe=Oysters Rockefeller, numberOfReviews=0, meanRating=NaN}
==>{recipe=Carrot Soup, numberOfReviews=1, meanRating=5.0}
```

**Searching recipes**

A common query for recipes is finding recipes that contain a certain ingredient.

```java
g.V().hasLabel('recipe').out().has('name','beef').in().hasLabel('recipe').values('name')
```

```java
==>Beef Bourguignon
```

A modification allows a query that includes either one ingredient or another.

```java
g.V().hasLabel('recipe').out().has('name',within('beef','carrots')).in().hasLabel('recipe').values('name')
```

```java
==>Beef Bourguignon
==>Carrot Soup
```

**Finding all the ingredients for a particular recipe is a common query.**

```java
g.V().match(
  __.as('a').hasLabel('ingredient'),
  __.as('a').in('includes').has('name','Beef Bourguignon')).select('a').by('name')
```

This query uses a `match()` step to find a match for the ingredients used to make Beef Bourguignon. The traversal starts by filtering all vertices to find the ingredients, then traverses to the recipe vertices along the `includes` edges using `in('includes')`. This query also uses a Groovy double underscore variable as a private variable for the match method. The results are:

```java
==>tomato paste
==>beef
==>onion
==>mashed garlic
==>butter
```

Although `inside()` is most commonly used for geospatial searches, the method can be used to find anything that falls within a particular range of values. An example is finding books that have a publishing date between 1960 and 1970:

```java
g.V().has('book', 'year', inside(1960,1970)).valueMap()
```

The results are:
Grouping output

Group output from a graph traversal using the `group()` traversal step. For example, display all the vertices by name, grouped by label:

```java
g.V().group().by(label).by('name')
```

Note that the meals, ingredients, authors, books, recipes, and reviewers are all grouped in the results:

```java
==>{meal:[JuliaDinner, Saturday Feast, EverydayDinner], ingredient: [olive oil, chicken broth, eggplant, pork sausage, green bell pepper, yellow onion, celery, hard-boiled egg, shallots, zucchini, butter, green beans, mashed garlic, onion, mushrooms, bacon, parsley, oyster, tomato, thyme, pork loin, tuna, tomato paste, ground beef, red wine, fennel, Pernod, chervil, egg noodles, carrots, beef], author:[Louisette Bertholie, Kelsie Kerr, Alice Waters, Julia Child, Emeril Lagasse, Simone Beck, Patricia Curtan, Patricia Simon, James Beard, Fritz Streiff], book:[Simca's Cuisine: 100 Classic French Recipes for Every Occasion, The French Chef Cookbook, The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution, The Art of French Cooking, Vol. 1], recipe:[Wild Mushroom Stroganoff, Roast Pork Loin, Spicy Meatloaf, Rataouille, Beef Bourguignon, Oysters Rockefeller, Salade Nicoise, Carrot Soup], reviewer:[Sharon Smith, John Smith, Jane Doe, Betsy Jones, John Doe]}
```

Another example groups all books by year and displays a listing of each year books were published followed by the book titles:

```java
g.V().hasLabel('book').group().by('year').by('name')
```

and lists:

```java
```
Grouping for processing using `local()`

Oftentimes, it is critical to do local processing for a particular step in the graph traversal. The next two examples use the `limit()` command to show how `local()` can change the processing from the whole stream entering the query to a portion of the query. First, find just two authors and the year that they have published books:

```java
g.V().hasLabel('author').as('author').out().properties('year').as('year').
    select('author','year').
    by('name').
    by().
    limit(2)
```

This query results in returning the first two records in the database:

```java
==>{author=Julia Child, year=vp[year->1961]}
==>{author=Julia Child, year=vp[year->1968]}
```

Using `local()`, change this query to find the first two books that each author in the graph has published:

```java
g.V().hasLabel('author').as('author').
    local(out()).properties('year').as('year').limit(2)).
    select('author','year').
    by('name').
    by()
```

Note that up to two books are displayed for each author:

```java
==>{author=Julia Child, year=vp[year->1961]}
==>{author=Julia Child, year=vp[year->1968]}
==>{author=Simone Beck, year=vp[year->1961]}
==>{author=Simone Beck, year=vp[year->1972]}
==>{author=Louisette Bertholie, year=vp[year->1961]}
==>{author=Patricia Simon, year=vp[year->1972]}
==>{author=Alice Waters, year=vp[year->2007]}
==>{author=Patricia Curtan, year=vp[year->2007]}
==>{author=Kelsie Kerr, year=vp[year->2007]}
==>{author=Fritz Streiff, year=vp[year->2007]}
```

The traversal step `local()` has many applications for processing a subsection of a graph within a graph traversal to return results before moving on to further processing.

**Geospatial traversals**

Geospatial queries are used to discover geospatial information. All geospatial data types (points, linestrings, and polygons) can be searched for specified values with simple queries. More interesting traversal queries discover points or linestrings within a radius from a specified point or within a specified geospatial polygon.
Using DataStax Enterprise advanced functionality

**Important:** All points must be specified in \((\text{longitude}, \text{latitude})\) following WKT format.

Distance calculations are crucial to proper results. DSE Search indexes can be created for geospatial data in DSE Graph, and DSE Search uses the [Haversine formula](https://en.wikipedia.org/wiki/Haversine_formula) to determine the great-circle distance between two points. DSE Search indexes cannot be created for polygons, but are essential to making geospatial point and linestring queries performant.

**Note:** A point of confusion can occur if the same geospatial query is run with or without a DSE Search index. Without a search index, geospatial queries always return exact results. DSE Search indexes, however, trade off write performance and index size for query accuracy with two tunable parameters, `maxDistErr` (default: 0.000009) and `distErrPct` (default: 0.025). Inconsistent results in these two cases are due to the distance calculation algorithm variation of the default values of these parameters. DSE Graph can pass values for these two parameters when creating the search index. Change `maxDistErr` in `withError(maxDistErr, distErrPct)` to 0.0 to force both index-backed and non-index-backed queries to yield the same value:

```java
schema.vertexLabel('location').index('search').search().by('point').withError(0.000009,0.0).add()
```

**Schema and data**

The examples here use the following schema:

```java
// SCHEMA
// POINT
schema.propertyKey('name').Text().create()
schema.propertyKey('point').Point().withGeoBounds().create()
schema.vertexLabel('location').properties('name','point').create()
// LINESTRING
schema.propertyKey('line').Linestring().withGeoBounds().create()
schema.vertexLabel('lineLocation').properties('name','line').create()
// POLYGON
schema.propertyKey('polygon').Polygon().withGeoBounds().create()
schema.vertexLabel('polyLocation').properties('name','polygon').create()
// MATERIALIZED VIEW INDEXES
schema.vertexLabel('location').index('byname').materialized().by('name').add()
schema.vertexLabel('lineLocation').index('byname').materialized().by('name').add()
schema.vertexLabel('polyLocation').index('byname').materialized().by('name').add()
//SEARCH INDEX - ONLY WORKS FOR POINT AND LINESTRING
schema.vertexLabel('location').index('search').search().by('point').add()
schema.vertexLabel('lineLocation').index('search').search().by('line').add()
```

The example use the following data:

```java
// Create a point
graph.addVertex(label,'location','name','Paris','point',Geo.point(2.352222, 48.856614))
graph.addVertex(label,'location','name','London','point',Geo.point(-0.127758, 51.507351))
```
Using DataStax Enterprise advanced functionality

```java
graph.addVertex(label, 'location', 'name', 'Dublin', 'point', Geo.point(-6.26031, 53.349805))
graph.addVertex(label, 'location', 'name', 'Aachen', 'point', Geo.point(6.083887, 50.775346))
graph.addVertex(label, 'location', 'name', 'Tokyo', 'point', Geo.point(139.691706, 35.689487))

// Create a linestring
graph.addVertex(label, 'lineLocation', 'name', 'ParisLondon', 'line', "LINESTRING(2.352222 48.856614, -0.127758 51.507351)"
graph.addVertex(label, 'lineLocation', 'name', 'LondonDublin', 'line', "LINESTRING(-0.127758 51.507351, -6.26031 53.349805)"
graph.addVertex(label, 'lineLocation', 'name', 'ParisAachen', 'line', "LINESTRING(2.352222 48.856614, 6.083887 50.775346)"
graph.addVertex(label, 'lineLocation', 'name', 'AachenTokyo', 'line', "LINESTRING(6.083887 50.775346, 139.691706 35.689487)"

// Create a polygon
graph.addVertex(label, 'polyLocation','name', 'ParisLondonDublin', 'polygon', Geo.polygon(2.352222, 48.856614, -0.127758, 51.507351, -6.26031, 53.349805))
graph.addVertex(label, 'polyLocation','name', 'LondonDublinAachen', 'polygon', Geo.polygon(-0.127758, 51.507351, -6.26031, 53.349805, 6.083887, 50.775346))
graph.addVertex(label, 'polyLocation','name', 'DublinAachenTokyo', 'polygon', Geo.polygon(-6.26031, 53.349805, 6.083887, 50.775346, 139.691706, 35.689487))
```

The example data has the following approximate distances:

```java
// PARIS TO LONDON: 3.08 DEGREES; 344 KM; 214 MI; 344,000 M
// PARIS TO AACHEN: 3.07 DEGREES; 343 KM; 213 MI; 343,000 M
// PARIS TO DUBLIN: 7.02 DEGREES; 781 KM; 485 MI; 781,000 M
// PARIS TO TOYKO: 86.3 DEGREES; 9713 KM; 6035 MI; 9,713,000 M
```

Find stored geospatial data that matches specified information

Find the stored data that matches a point mapped to the specified (longitude, latitude):

```
g.V().
  has('location','point', Geo.point(2.352222, 48.856614)).
  valueMap()
```

results in:

```java
{name=[Paris], point=[POINT (2.352222 48.856614)]}
```

Find the stored data that matches a line mapped to the specified points:

```
g.V().
  has('lineLocation','line',Geo.lineString(2.352222, 48.856614, -0.127758, 51.507351)).
  valueMap()
```
results in:

```json
{line=[LINESTRING (2.352222 48.856614, -0.127758 51.507351]),
 name=[ParisLondon]}
```

Find the stored data that matches a polygon mapped to the specified points:

```cypher
g.V().
has('polyLocation', 'polygon', Geo.polygon(2.352222, 48.856614,
-0.127758, 51.507351, -6.26031, 53.349805)).
valueMap()
```

results in:

```json
{polygon=[POLYGON ((2.352222 48.856614, -0.127758 51.507351, -6.26031 53.349805, 2.352222 48.856614))], name=[ParisLondonDublin]}
```

Find stored geospatial points or linestrings within a specified radius from a specified point

These queries, as well as the queries that use a specified geospatial polygon use a method `Geo.inside()` that specifies a point, a radius, and the units to be used.

Several units are available with use of the `Geo.inside()` method:

**DEGREES**

Degrees of distance. One degree of latitude is approximately 111.2 kilometers, whereas one degree of longitude depends on the distance from the equator. At the equator, one degree of longitude equals 111.2 kilometers, but at 45 degrees of latitude, one degree of longitude is 78.6 kilometers. While the physical distance over a single degree of longitude changes with latitude, we calculate only great-circle distances in degrees.

**KILOMETERS**

Kilometers of distance.

**MILES**

Miles of distance.

**METERS**

Meters of distance.

Find all the cities (points) within a radius from a particular location (centerpoint):

```cypher
g.V().
has('location', 'point', Geo.inside(Geo.point(2.352222, 48.856614),
 4.2, Geo.Unit.DEGREES)).
values('name')
```

```
=>Paris
=>London
=>Aachen
```
Using DataStax Enterprise advanced functionality

Centering the query on Paris and searching within 4.2 degrees returns three cities: Paris, London, and Aachen from the dataset.

Find all the linestrings within a radius from a particular location (centerpoint):

```
g.V().
has('lineLocation', 'line', Geo.inside(Geo.point(2.352222, 48.856614), 9713, Geo.Unit.KILOMETERS)).
values('name')
```

lists:

```===>ParisLondon
===>LondonDublin
===>AachenTokyo
===>ParisAachen```

Centering the query on Paris and searching within 9713 kilometers returns four stored linestrings: Paris to London, London to Dublin, Aachen to Tokyo, and Paris to Aachen. Note that London to Dublin was not a stored linestring.

Find stored geospatial points or linestrings within a specified geospatial polygon

Polygons may be used in these queries with a search index on point.

**Note:** If a query is not backed by a search index, the results are consistent with geospatial coordinates, automatically defined using withGeoBounds(). This means that queries will return accurate results if the polygon crosses the international dateline. A query not backed by a search index will only use Cartesian coordinates if the underlying graph property keys are declared using withBounds().

Find all cities (points) within a specified geospatial polygon:

```
g.V().
has('location', 'point', Geo.inside(Geo.polygon(-6.26031, 53.349805, 6.083887, 50.775346, 139.691706, 35.689487))).
values('name')
```

lists:

```===>Dublin
===>Aachen
===>Tokyo```

This results is not surprising, since the three points used to create the polygon represent the three cities discovered.

Find all cities (points) within a specified geospatial polygon generated with a WKT tool:

```
g.V().
has('location', 'point', Geo.inside(Geo.polygon(-7.9541015625, 55.148273231753834, -9.6240234375, 51.47539580264131, 1.0986328125, 50.86924482345238, 0.5712890625, 53.29887631763788, -7.9541015625, 55.148273231753834))).
```
values('name')

lists:

–>London
–>Dublin

The polygon used encompasses most of the Republic of Ireland as well as the southern half of the United Kingdom, and finds London and Dublin within the polygon.

find linestrings within a polygon

g.V().has('lineLocation', 'line', Geo.inside(Geo.polygon(-6.26031, 53.349805, 6.083887, 50.775346, 139.691706, 35.689487))).values('name')

lists:

–>AachenTokyo

Since two of the points in the specified polygon represent Aachen and Tokyo, it is reassuring that the linestring of Aachen to Tokyo is found.

Schema and data

The examples here use the following schema:

```javascript
//SCHEMA
// PROPERTY KEYS
// Check for previous creation of property key with ifNotExists()
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().ifNotExists().create()
schema.propertyKey('location').Point().withGeoBounds().ifNotExists().create()

// VERTEX LABELS
schema.vertexLabel('author').properties('name','gender').ifNotExists().create()
schema.vertexLabel('place').properties('name','location').create()

// EDGE LABELS
schema.edgeLabel('livesIn').connection('author','place').ifNotExists().create()

// VERTEX INDEXES
// Secondary
schema.vertexLabel('author').index('byName').secondary().by('name').add()

// Search
schema.vertexLabel('author').index('search').search().by('name').asString().ifNotExists().add()
schema.vertexLabel('place').index('search').search().by('location').ifNotExists().add();
```

The examples use the following data:

```javascript
//VERTICES
// AUTHOR VERTICES
juliaChild = graph.addVertex(label, 'author', 'name','Julia Child', 'gender', 'F')
simoneBeck = graph.addVertex(label, 'author', 'name', 'Simone Beck', 'gender', 'F')
```
Using DataStax Enterprise advanced functionality

```javascript
louissetteBertholie = graph.addVertex(label, 'author', 'name', 'Louissette Bertholie', 'gender', 'F')
patriciaSimon = graph.addVertex(label, 'author', 'name', 'Patricia Simon', 'gender', 'F')
aliceWaters = graph.addVertex(label, 'author', 'name', 'Alice Waters', 'gender', 'F')
patriciaCurtan = graph.addVertex(label, 'author', 'name', 'Patricia Curtan', 'gender', 'F')
kelsieKerr = graph.addVertex(label, 'author', 'name', 'Kelsie Kerr', 'gender', 'F')
fritzStreiff = graph.addVertex(label, 'author', 'name', 'Fritz Streiff', 'gender', 'M')
emerilLagasse = graph.addVertex(label, 'author', 'name', 'Emeril Lagasse', 'gender', 'M')
jamesBeard = graph.addVertex(label, 'author', 'name', 'James Beard', 'gender', 'M')

// PLACE VERTICES
newYork = graph.addVertex(label, 'place', 'name', 'New York', 'location', Geo.point(74.0059, 40.7128));
paris = graph.addVertex(label, 'place', 'name', 'Paris', 'location', Geo.point(2.3522, 48.8566));
newOrleans = graph.addVertex(label, 'place', 'name', 'New Orleans', 'location', Geo.point(90.0715, 29.9511));
losAngeles = graph.addVertex(label, 'place', 'name', 'Los Angeles', 'location', Geo.point(118.2437, 34.0522));
london = graph.addVertex(label, 'place', 'name', 'London', 'location', Geo.point(-0.1278, 51.5074));
chicago = graph.addVertex(label, 'place', 'name', 'Chicago', 'location', Geo.point(-87.6298, 41.8781136));
tokyo = graph.addVertex(label, 'place', 'name', 'Tokyo', 'location', Geo.point(139.6917, 35.6895));

// EDGES
juliaChild.addEdge('livesIn', newYork);
simoneBeck.addEdge('livesIn', paris);
louissetteBertholie.addEdge('livesIn', london);
patriciaSimon.addEdge('livesIn', newYork);
aliceWaters.addEdge('livesIn', losAngeles);
patriciaCurtan.addEdge('livesIn', chicago);
kelsieKerr.addEdge('livesIn', tokyo);
fritzStreiff.addEdge('livesIn', tokyo);
emerilLagasse.addEdge('livesIn', newOrleans);
jamesBeard.addEdge('livesIn', london);
```

Of course, this data can be loaded using the DSE Graph Loader (page 484) as well, from CSV or other formatted files.

Find authors who live within a certain distance from a specified city in sorted order

First list the place names for all cities within the given radius (50 degrees) from New York (the approximate centerpoint listed:

```javascript
g.V()
```
Using DataStax Enterprise advanced functionality

```java
has('place', 'location', Geo.inside(Geo.point(74.0, 40.5), 50, Geo.Unit.DEGREES)).values('name')
```

results in:

```plaintext
===> New York
===> Paris
===> New Orleans
===> Los Angeles
```

Now list the place names and authors who live in those cities for all cities within the given radius (50 degrees) from New York (the approximate centerpoint), sorted in alphabetical order:

// Order by name, not by distance from location point given
g.V().has('place', 'location', Geo.inside(Geo.point(74.0, 40.5), 50, Geo.Unit.DEGREES)).
order().by('name').
as('Location').
in().as('Author').
select('Location','Author').
by('name').
by('name')

finds:

```plaintext
==>{Location=Los Angeles, Author=Alice Waters}
==>{Location=New Orleans, Author=Emeril Lagasse}
==>{Location=New York, Author=Patricia Simon}
==>{Location=New York, Author=Julia Child}
==>{Location=Paris, Author=Simone Beck}
```

This query uses some additional methods such as order() and select() that are explained in Simple Traversals (page 596).

Now list the place names and authors who live in those cities for all cities within the given radius (50 degrees) from New York (the approximate centerpoint), sorted by the distance from the centerpoint:

// Order by distance from NYC
g.V().has('place', 'location', Geo.inside(Geo.point(74.0, 40.5), 50, Geo.Unit.DEGREES)).
order().by{it.value('location').getOgcGeometry().distance(Geo.point(74.0059, 40.7128)).getOgcGeometry() as 'Location').
in().as('Author').
select('Location', 'Author').
by('name').
by('name')

```plaintext
==>{Location=New York, Author=Patricia Simon}
==>{Location=New York, Author=Julia Child}
==>{Location=New Orleans, Author=Emeril Lagasse}
==>{Location=Los Angeles, Author=Alice Waters}
```
This query introduces some additional methods that must be imported in order for the query to succeed: `getOgcGeometry()` and `distance()`. Importing the library is accomplished in the original script using:

```java
import com.esri.core.geometry.ogc.OGCGeometry;
```

**Cartesian spatial traversals**

Cartesian spatial queries are used to discover Cartesian (graphable) information. All Cartesian data types (points, linestrings, and polygons) can be searched for specified values with simple queries. More interesting traversal queries discover points or linestrings within a radius from a specified point or within a specified spatial polygon.

DSE Search indexes can be created to decrease the latency in response time, but are not required. Create schema to use a search index for point and linestring properties in the Cartesian schema (page 457).

**Schema and data**

The examples here use the following schema:

```java
// SCHEMA
// POINT
schema.propertyKey('name').Text().create()
schema.propertyKey('point').Point().withBounds(-3, -3, 3, 3).create()
schema.vertexLabel('location').properties('name','point').create()
// LINESTRING
schema.propertyKey('line').Linestring().withBounds(-3, -3, 3, 3).create()
schema.vertexLabel('lineLocation').properties('name','line').create()
// POLYGON
schema.propertyKey('polygon').Polygon().withBounds(-3, -3, 3, 3).create()
schema.vertexLabel('polyLocation').properties('name','polygon').create()
// MATERIALIZED VIEW INDEXES
schema.vertexLabel('location').index('byname').materialized().by('name').add()
schema.vertexLabel('lineLocation').index('byname').materialized().by('name').add()
schema.vertexLabel('polyLocation').index('byname').materialized().by('name').add()
// SEARCH INDEX - ONLY WORKS FOR POINT AND LINESTRING
schema.vertexLabel('location').index('search').search().by('point').add()
schema.vertexLabel('lineLocation').index('search').search().by('line').add()
```

The example use the following data:

```java
/// Create a point
graph.addVertex(label,'location','name','p0','point',Geo.point(0.5,0.5))
graph.addVertex(label,'location','name','p1','point',Geo.point(1,1))
graph.addVertex(label,'location','name','p2','point',Geo.point(-1,1))
graph.addVertex(label,'location','name','p3','point',Geo.point(-2,-2))
graph.addVertex(label,'location','name','p4','point',Geo.point(2,2))
```
Using DataStax Enterprise advanced functionality

```java
// Create a linestring
graph.addVertex(label, 'lineLocation', 'name', 'l1', 'line',
    "LINESTRING(0 0, 1 1)"
)  
graph.addVertex(label, 'lineLocation', 'name', 'l2', 'line',
    "LINESTRING(0 0, -1 1)"
)  
graph.addVertex(label, 'lineLocation', 'name', 'l3', 'line',
    "LINESTRING(0 0, -2 -2)"
)  
graph.addVertex(label, 'lineLocation', 'name', 'l4', 'line',
    "LINESTRING(0 0, 2 -2)"
)

// Create a polygon
graph.addVertex(label, 'polyLocation', 'name', 'g1',
    'polygon', Geo.polygon(0,0,1,1,0,1,0,0)
)  
graph.addVertex(label, 'polyLocation', 'name', 'g2',
    'polygon', Geo.polygon(0,0,0,1,-1,1,0,0)
)  
graph.addVertex(label, 'polyLocation', 'name', 'g3',
    'polygon', Geo.polygon(0,0,-2,0,-2,-2,0,0)
)  
graph.addVertex(label, 'polyLocation', 'name', 'g4',
    'polygon', Geo.polygon(0,0,2,0,2,-2,0,0)
)

Find stored Cartesian spatial data that matches specified information

Find the stored data that matches a point mapped to the specified (x, y):

```java
g.V().
    has('location','point', Geo.point(0.5, 0.5)).
    valueMap()
```  
results in:

```java
{name=[p0], point=[POINT (0.5 0.5)]}
```

Find the stored data that matches a line mapped to the specified points:

```java
g.V().
    has('lineLocation', 'line', Geo.lineString(0, 0, 1, 1)).
    valueMap()
```  
results in:

```java
{line=[LINESTRING (0 0, 1 1)], name=[l1]}
```

Find the stored data that matches a polygon mapped to the specified points:

```java
g.V().
    has('polyLocation', 'polygon', Geo.polygon(0,0,1,1,0,1,0,0)).
    valueMap()
```  
results in:

```java
{polygon=[POLYGON ((0 0, 1 1, 0 1, 0 0))], name=[g1]}
```
Using DataStax Enterprise advanced functionality

Find stored Cartesian spatial points or linestrings within a specified radius from a specified point

These queries, as well as the queries that use a specified Cartesian spatial polygon use a method `Geo.inside()` that specifies a point and a radius.

Find all the points within a radius from a particular point (centerpoint):

```java
g.V().
  has('location', 'point', Geo.inside(Geo.point(0, 0), 1)).
  values('name')
```

lists:

```java
==> p0
```

Centering the query on (0, 0) and searching within 1 unit returns one point, p0, from the dataset.

Find all the linestrings within a radius from a particular location (centerpoint):

```java
g.V().has('lineLocation', 'line', Geo.inside(Geo.point(0.0, 0.0),
  1.415)).valueMap()
```

lists:

```java
==> {line=[LINESTRING (0 0, 1 1)], name=[l1]}
==> {line=[LINESTRING (0 0, -1 1)], name=[l2]}
```

Centering the query on (0,0) and searching within 1.415 units returns two stored linestrings: l1 and l2.

Find stored Cartesian spatial points or linestrings within a specified Cartesian spatial polygon

Polygons may be used in these queries to find points with a polygon.

Find all points within a specified Cartesian spatial polygon:

```java
g.V().
  has('location', 'point', Geo.inside(Geo.polygon(0, 0, 1, 0, 1, 1, 0, 1,
    0, 0))).
  values('name')
```

lists:

```java
==> p0
```

find linestrings within a polygon

```java
g.V().
  has('lineLocation', 'line', Geo.inside(Geo.polygon(0, 0, 1, 0, 1, 1, 0,
    1, 0, 0))).
```
Using DataStax Enterprise advanced functionality

values('name')

lists:

==>l1

### Schema and data

The examples here use the following schema:

```java
// SCHEMA
// PROPERTY KEYS
// Check for previous creation of property key with ifNotExists()
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('address').Text().ifNotExists().create()
schema.propertyKey('location').Point().withBounds(-100,-100,100,100).ifNotExists().create()

// VERTEX LABELS
schema.vertexLabel('person').properties('name').ifNotExists().create()
schema.vertexLabel('home').properties('address','location').ifNotExists().create()
schema.vertexLabel('store').properties('name','location').ifNotExists().create()
schema.vertexLabel('ingredient').properties('name').ifNotExists().create()

// EEDGE LABELS
schema.edgeLabel('livesIn').connection('person','home').ifNotExists().create()
schema.edgeLabel('isStockedWith').connection('store','ingredient').multiple().ifNotExists().create()

// SEARCH INDEXES
schema.vertexLabel('person').index('search').search().by('name').asString().ifNotExists().add()
schema.vertexLabel('home').index('search').search().by('name').by('location').add();
schema.vertexLabel('store').index('search').search().by('name').by('location').add();
schema.vertexLabel('ingredient').index('search').search().by('name').add();
```

The examples use the following data:

```java
// VERTICES
// PERSON VERTICES
pam = graph.addVertex(label, 'person', 'name','Pam')
les = graph.addVertex(label, 'person', 'name','Les')
paul = graph.addVertex(label, 'person', 'name','Paul')
victoria = graph.addVertex(label, 'person', 'name','Victoria')
terri = graph.addVertex(label, 'person', 'name','Terri')

// HOME VERTICES
home1 = graph.addVertex(label, 'home', 'address', '555 4th St',
                        'location', Geo.point(7,2));
home2 = graph.addVertex(label, 'home', 'address', '1700 Coyote Rd',
                        'location', Geo.point(-2,1));
home3 = graph.addVertex(label, 'home', 'address', '99 Mountain Pass Hwy',
                        'location', Geo.point(0,0));

// STORE VERTICES
store1 = graph.addVertex(label, 'store', 'name', 'ZippyMart',
                          'location', Geo.point(1,5));
```

DSE 6.7 Developer Guide (Latest version)
Using DataStax Enterprise advanced functionality

```java
store2 = graph.addVertex(label, 'store', 'name', 'Quik Station', 'location', Geo.point(7,-1));
store3 = graph.addVertex(label, 'store', 'name', 'Mamma's Grocery', 'location', Geo.point(-3,-3));

// INGREDIENT VERTICES
celery = graph.addVertex(label, 'ingredient','name', 'celery');
milk = graph.addVertex(label, 'ingredient','name', 'milk');
bokChoy = graph.addVertex(label, 'ingredient','name', 'bok choy');
steak = graph.addVertex(label, 'ingredient','name', 'steak');
carrots = graph.addVertex(label, 'ingredient','name', 'carrots');
porkChops = graph.addVertex(label, 'ingredient','name', 'pork chops');

// PERSON - HOME EDGES
pam.addEdge('livesIn', home1);
les.addEdge('livesIn', home1);
paul.addEdge('livesIn',home3);
victoria.addEdge('livesIn',home3);
terri.addEdge('livesIn',home2);

// STORE - INGREDIENT EDGES
store1.addEdge('isStockedWith', milk);
store1.addEdge('isStockedWith', bokChoy);
store1.addEdge('isStockedWith', steak);
store2.addEdge('isStockedWith', steak);
store2.addEdge('isStockedWith', carrots);
store2.addEdge('isStockedWith', porkChops);
store3.addEdge('isStockedWith', milk);
store3.addEdge('isStockedWith', carrots);
store3.addEdge('isStockedWith', celery);
```

Finding celery

You are a mathematics teacher writing simple Cartesian problem for your students. They are great fans of *ants on a log*, a snack made with celery, cream cheese, and raisins. So, you decide to help them find the nearest store to their house which has celery in stock.

Paul is the student whose home we'll use as the starting point. First, list all stores within the given radius (10 units of distance) from Paul's home (the centerpoint):

```java
g.V().has('store', 'location', Geo.inside(Geo.point(0,0),10)).values('name')
```

results in:

```java
==>ZippyMart
==>Quik Station
==>Mamma's Grocery
```

Note that this exercise is using Cartesian coordinates and distances calculated between Cartesian points, but a similar exercises can use geospatial data *(page 610)*.

Now list the stores within a 10 unit radius from Paul's home that have celery:
Using DataStax Enterprise advanced functionality

```java
g.V().has('store', 'location', Geo.inside(Geo.point(0,0),10)).as('Store').
out().has('name','celery').as('Ingred').
select('Store', 'Ingred').
by('name').
by('name')
```

finds:

```java
==>{Store=ZippyMart, Ingred=celery}
==>{Store=Quik Station, Ingred=celery}
==>{Store=Mamma's Grocery, Ingred=celery}
```

This query uses methods that are common, such as `as()`, `out()`, and `select()` that are explained in Simple Traversals (page 596) to narrow the query.

Finally, list the stores within a 10 unit radius of Paul's home that have celery, and sort them by the distance from Paul's home:

```java
// List store name, location, and ingredient in order by distance from the store
g.V().has('store', 'location', Geo.inside(Geo.point(0,0),25)).as('Store').
order().by{it.value('location').getOgcGeometry().distance(Geo.point(0,0).getOgcGeometry())}.as('Location').
out().has('name','celery').as('Ingred').
select('Store', 'Location','Ingred').
by('name').
by('location').
by('name')
```

```java
==>{Store=Mamma's Grocery, Location=POINT (-3 -3), Ingred=celery}
==>{Store=ZippyMart, Location=POINT (1 5), Ingred=celery}
==>{Store=Quik Station, Location=POINT (7 -1), Ingred=celery}
```

This query adds the method `order()` to sort the results; it is also explained in Simple Traversals (page 596). The query must also use a method that must be imported in order for the query to succeed: `getOgcGeometry()` and `distance()`. Importing the library is accomplished in the original script using:

```java
import com.esri.core.geometry.ogc.OGCGeometry;
```

The students working on this problem now know that Mamma's Grocery is the place to head to get the celery they need to make their favorite snack!

**Branching Traversals**

Branching traversals allow decision points to be inserted into the traversal processing. Prior to trying out branching traversals shown here, you must create the data as described in Simple Traversals (page 596).
Using DataStax Enterprise advanced functionality

This branching traversal example chooses between two labels, either author or reviewer to fork the traversal. If the vertex label is author, the edges labeled created are counted. If the vertex label is reviewer, the edges labeled rated are counted.

```
g.V().choose(label()).
  option('author', out('created').count()).
  option('reviewer', out('rated').count())
```

The output for this traversal lists each result, the count returned. This type of traversal is useful as an intermediary step in a query process, but clearly the output is not useful without reference.

```
=>0
=>0
=>2
=>0
=>0
=>0
=>2
=>1
=>0
=>0
=>0
=>3
=>0
=>2
=>2
=>1
=>2
```

**Recursive Traversals**

Recursive traversals allow iterative processing over traversal paths. Prior to trying out branching traversals shown here, you must create the data as described in Simple Traversals (page 596).

This recursive traversal example returns the names of vertices that are two outgoing steps from the author vertex named Julia Child using the times(2) step. Books, meals, and ingredients are returned by this query.

```
g.V().has('name','Julia Child').repeat(out()).times(2).valueMap()
```

The output for this traversal lists each result:

```
=>{name=[onion]}
=>{name=[beef]}
=>{name=[mashed garlic]}
=>{name=[butter]}
=>{name=[tomato paste]}
=>{name=[JuliaDinner], calories=[900],
   timestamp=[2016-01-14T00:00:00Z]}
=>{year=[1961], name=[The Art of French Cooking, Vol. 1]}
```
==>{name=[Saturday Feast], calories=[1000],
    timestamp=[2015-11-30T00:00:00Z]}
==>{name=[olive oil]}
==>{name=[green beans]}
==>{name=[tuna]}
==>{name=[hard-boiled egg]}
==>{name=[tomato]}
==>{name=[JuliaDinner], calories=[900],
    timestamp=[2016-01-14T00:00:00Z]}
==>{year=[1961], name=[The Art of French Cooking, Vol. 1]}
==>{name=[olive oil]}
==>{name=[yellow onion]}
==>{name=[zucchini]}
==>{name=[mashed garlic]}
==>{name=[eggplant]}

Path Traversals

Path traversals map traversal steps to a location to use in the event that a previous
location must be revisited.

This path traversal starts at an ingredient, traverses to a recipe, and eventually finds a
book that contains the recipe with the ingredients specified.

```
g.V().has('ingredient',
    'name',within('beef','carrots')).in('includes').as('Recipe').
out().hasLabel('book').as('Book').
select('Book','Recipe').
by('name').by('name').path()
```

The output for this traversal lists each result:

```
[{{label=ingredient, member_id=2, community_id=1442590464}},
  {{label=recipe, member_id=2, community_id=473764096}},
  {{label=book, member_id=0, community_id=568859392}},
  (Book=The Art of French Cooking, Vol. 1, Recipe=Beef Bourguignon)]
[{{label=ingredient, member_id=1, community_id=684566272}},
  {{label=recipe, member_id=0, community_id=1462084224}},
  {{label=book, member_id=1, community_id=1620680576}},
  (Book=The Art of Simple Food: Notes, Lessons, and Recipes from a
   Delicious Revolution, Recipe=Carrot Soup)]
```

Another path traversal creates a tree that emanates from a vertex label, in this case a
book.

```
g.V().hasLabel('book').in().tree().by('name').next()
```

The output for this traversal lists each result:

```
Simca's Cuisine: 100 Classic French Recipes for Every Occasion=
  (Patricia Simon={}, Simone Beck={})
The Art of French Cooking, Vol. 1=
  (Simone Beck={}, Julia Child={}, Beef Bourguignon={}, Louisette
   Bertholle={}, Salade Nicoise={})
```
Using DataStax Enterprise advanced functionality

Each book lists the authors and recipes that are included in the book.

Another tree traversal discovers all the vertices that are on outgoing tree branch from a recipe.

```java
g.V().hasLabel('recipe').out().tree().by('name').next()
```

The output for this traversal lists each result:

- **Roast Pork Loin**
  ```java
  {red wine={}, pork loin={}, chicken broth={}, The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution={})
  ```

- **Spicy Meatloaf**
  ```java
  {bacon={}, celery={}, pork sausage={}, onion={}, ground beef={}, green bell pepper={}
  ```

- **Beef Bourguignon**
  ```java
  {mashed garlic={}, butter={}, The Art of French Cooking, Vol. 1={}, onion={}, tomato paste={}, beef={}
  ```

- **Carrot Soup**
  ```java
  {butter={}, onion={}, chicken broth={}, carrots={}, The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution={}, thyme={}
  ```

- **Ratatouille**
  ```java
  {mashed garlic={}, yellow onion={}, olive oil={}, zucchini={}, eggplant={}
  ```

- **Salade Nicoise**
  ```java
  {tuna={}, The Art of French Cooking, Vol. 1={}, hard-boiled egg={}, olive oil={}, tomato={}, green beans={}
  ```

- **Wild Mushroom Stroganoff**
  ```java
  {mushrooms={}, yellow onion={}, egg noodles={}
  ```

- **Oysters Rockefeller**
  ```java
  {oyster={}, chervil={}, parsley={}, celery={}, fennel={}, shallots={}, Pernod={}
  ```

Each recipe lists the ingredients for the recipe and the books that include the recipe.

### DSE Graph Analysis with DSE Analytics

#### DSE Graph and Graph Analytics

Many local graph traversals can be executed in real time at high transactional loads. When the density of the graph is too high or the branching factor too large (the number of connected nodes at each level of the graph), the memory and computation requirements to answer OLTP queries go beyond what is acceptable under typical application workloads. These type of queries are called *deep queries*.
Scan queries are queries that touch either an entire graph or large parts of the graph. They typically traverse a large number of vertices and edges. For example, a query on a social network graph that searches for friends of friends is a scan query.

For applications that use deep and scan queries, using a OLAP query will result in better performance.

Performing OLAP queries using DSE Graph

Every graph created in DSE Graph has an OLAP traversal source that is available to gremlin-console and DataStax Studio. This traversal source uses the SparkGraphComputer to analyze queries and execute them against the underlying DSE Analytics nodes. The nodes must be started with Graph and Spark enabled to access the OLAP traversal source. You must connect to the Spark Master (page 225) node for the datacenter by either running the console from the Spark Master or specifying the Spark Master in the hosts field of the Gremlin console yaml file (page 149). For one-off or single-session OLAP queries, alias database.a to g and create the query. For example in the Gremlin console:

```plaintext
:remote config alias g database.a

g.V().count()
```

If you are performing multiple queries against different parts of the graph, use graph.snapshot() to return an OLAP traversal source for each part of the graph. For example, in the Gremlin console:

```plaintext
categories = graph.snapshot().vertices('category1', 'category2').create()
```

To create a snapshot, supply all the vertices the snapshot will traverse. For example, the following query touches both the Person and Address vertices.

```plaintext
def person = graph.snapshot().vertices('Person', 'Address').create()
person.V().hasLabel('Person').out('HAS_ADDRESS').count()
```

Use the conf() method on the snapshot before you call create() to set TinkerPop's SparkGraphComputer configuration options. For example, to explicitly set the storage level for the snapshot to MEMORY_ONLY:

```plaintext
graph.snapshot().vertices("vertexlabel_alice", "vertexlabel_bob").edges("edgelabel_carol").conf("gremlin.spark.persistStorageLevel", "MEMORY_ONLY").create()
```

Setting Spark properties from Gremlin

Spark properties (page 235) can be set from Gremlin using the graph.configuration.setProperty method on the graph.
Using DataStax Enterprise advanced functionality

By default, Spark applications will use all available resources on the node, so no other Spark application can run. Limit the application's resources before running OLAP traversals by setting the maximum number of cores and the amount of memory used by the traversal. This is particularly important on servers with very large amounts of cores and memory.

For example this request sets 10 executors with 1 core and 4 GB of memory each:

```
:remote config alias g example_graph.a
==>g=example_graph.a

g.graph.configuration.setProperty("spark.cores.max", 10)
g.graph.configuration.setProperty("spark.executor.memory", "4g")
g.graph.configuration.setProperty("spark.executor.cores", "1")
```

The `spark.cores.max` property sets the maximum number of cores used by Spark. Setting this property lower than the total number of cores limits the number of nodes on which the queries will be run. The `spark.executor.memory` property sets the amount of memory used for each executor. The `spark.executor.cores` property sets the number of cores used for each executor.

Before you configure Spark properties from Gremlin kill the currently-running Spark context from the Spark web UI (page 209). This will kill all currently running Gremlin OLAP queries. From the Spark web UI, find the application named Apache TinkerPop's Spark-Gremlin and click `kill` next to the Application ID.

OLAP traversals create many intermediate objects during execution. These objects are garbage-collected by the JVM, so we recommend configuring a larger pool of executors each with smaller memory and CPU resources, compared to non-graph Spark jobs which typically perform better with fewer executors with higher memory and CPU resources.

We recommend allocating executors with no more then 8 cores (1 should work in most cases) to reduce garbage collection pauses and improve OLAP traversal performance. The memory available to Spark should be equally spread among the cores. For example, if you have 3 nodes and each has 24 cores and 96 GB dedicated to Spark you have 24 * 3 = 72 cores and 96 GB * 3 = 188 GB memory. To allocate all resources you should request 72 single core executors with 4 GB of memory each:

```
:remote config alias g example_graph.a
==>g=example_graph.a

g.graph.configuration.setProperty("spark.cores.max", 72)
g.graph.configuration.setProperty("spark.executor.memory", "4g")
g.graph.configuration.setProperty("spark.executor.cores", "1")
```
Some OLAP queries and most DseGraphFrame queries use Spark SQL joins for traversals. Spark has a predefined number of partitions to perform merge joins, by default set to 200. This can create huge Spark partitions for very large graphs, which slows query execution.

To reduce the size of single partitions and improve performance increase the number of shuffle partitions by setting the `spark.sql.shuffle.partitions` property to a larger number. We recommend the `spark.sql.shuffle.partitions` is set to 2-4 times the number of Spark cluster cores. So if you have a 200 core cluster, set `spark.sql.shuffle.partitions` to 400 or 800.

```java
:g=example_graph.a

g.graph.configuration.setProperty("spark.sql.shuffle.partitions", 500)
```

When to use analytic OLAP queries

On large graphs, OLAP queries typically perform better for deep queries. However, executing deep queries as part of an OLTP load may make sense if they are rarely performed. For example, on online payment provider will favor OLTP queries to process payments quickly, but may require a deep query if there are indications of fraud in the transaction. While the deep query may take much longer as an OLTP workload, on the whole the performance of the application will be faster than segmenting the application into OLTP and OLAP queries.

Long running and periodic processes like recommendation engines and search engines that analyze an entire graph are the ideal use cases for OLAP queries. However, one-off data analysis operations that involve deep queries or that scan the entire database also can benefit from being run as OLAP queries. See DSE Graph, OLTP, and OLAP (page 416) for detailed information on performance differences between OLTP and OLAP queries.

Best practices for deleting large numbers of edges and vertices

When deleting large numbers of edges or vertices from a graph, you may end up getting error messages in subsequent queries due the large number of tombstones left in the database before they are automatically removed.

The log entries for such errors resemble the following:

```java
- Scanned over 100001 tombstones during query 'SELECT * FROM
t33215.PhoneNumber_p WHERE token(community_id) > -7331398285705078207
AND token(community_id) <= -6858404847917653807 LIMIT 100' (last
scanned row partion key was ((216134144), 1250272)); query aborted
```

To avoid these errors, reduce the number of tombstones per request by setting the `spark.cassandra.input.split.size_in_mb` property to a smaller size than the default of 64 MB. The `spark.cassandra.input.split.size_in_mb` property sets the approximate size of data the Spark Cassandra Connector will request with each individual CQL query.
The following example shows how to set the `spark.cassandra.input.split.size_in_mb` property to 1 MB and then to drop all phone number vertices from a graph.

```java
:remote config alias g example_graph.a

g.graph.configuration.setProperty("spark.cassandra.input.split.size_in_mb", "1")
g.V().hasLabel("PhoneNumber").drop().iterate()
```

DSE authentication and OLAP queries

If DSE authentication is enabled, the internal user `dse_inproc_user` runs the application, not the user who submitted the Graph OLAP query.

**Using the DseGraphFrame framework for graph analytics queries**

The `DseGraphFrame` framework allows you to create applications that use the Spark API for analytics operations on DSE Graph. It is inspired by the Databricks `GraphFrame` library and supports a subset of the Gremlin graph traversal language. You can read DSE Graph data into a `GraphFrame` and write `GraphFrame` objects from any format supported by Spark into DSE Graph. You can also query `DseGraphFrame` vertices and edges in Spark SQL (page 246).

Choosing when to use DseGraphFrame or DSE Graph OLAP queries

DSE Graph OLAP (page 620) has broader support for Gremlin than the `DseGraphFrame` API. While Graph OLAP is the best choice for deep queries, simple filtering and counts are much faster using the `DseGraphFrame` API.

**Overview of DseGraphFrame**

`DseGraphFrame` represents a graph as two virtual tables: a vertex and an edge `DataFrame`. The `V()` method returns the vertex `DataFrame` of a graph. The `E()` method returns the edge `DataFrame` of a graph.

```scala
val g = spark.dseGraph("test")
g.V.show
g.E.show
```

`DseGraphFrame` uses a `GraphFrame`-compatible format. This format requires the vertex `DataFrame` to have only one `id` column and the edge `DataFrame` to have hard coded `src` and `dst` columns. Since DSE Graph allows users to define any arbitrary set of columns as the vertex `id` and since there is no concept of labels in `GraphFrame`, `DseGraphFrame` will serialize the entire DSE Graph `id` into one `id` column. The label is represented as part of the `id` and also as the `~label` property column.

**Using DseGraphFrame**

The starting point for all operations is the `DseGraphFrame` object. In Scala, there's an implicit conversion between `DseGraphFrame` objects and `GraphFrame` objects.
Using DataStax Enterprise advanced functionality

```scala
// load a graph
val graph = spark.dseGraph("my_graph")
// use the TinkerPop API
graph.V().has("edge", gt(100)).count().next()
// use the GraphFrame API
graph.find("(a)-[e]->(b); (b)-[e2]->(c)").filter("e2.`~label` = 'includedIn'").select("a.name", "e.`~label`", "b.name", "e2.`~label`", "c.name").distinct.show
// Use both the TinkerPop and GraphFrame APIs:
graph.V().out().hasLabel("label").df.show
```

In Java, use the `gf()` method, or use the `DseGraphFrameBuilder.dseGraph(String graphName, GraphFrame gf)` method to return a `GraphFrame` instance.

```java
// load a graph
GraphFrame graph = DseGraphFrameBuilder.dseGraph("my_graph", spark);
// use the TinkerPop API
graph.V().has("edge", gt(100)).count().next()
// use the GraphFrame API
graph.find("(a)-[e]->(b); (b)-[e2]->(c)").filter("e2.label = 'includedIn'").select("a.name", "e.`~label`", "b.name", "e2.`~label`", "c.name").distinct.show()
// Use both the TinkerPop and GraphFrame APIs:
graph.V().out().hasLabel("label").df.show()
```

Before doing complex queries, it is strongly recommended you cache the graph. You can do so using the `cache()` or `persist(level)` methods.

```scala
g.graph()
```

The `persist()` method requires one of the Spark persist levels as a parameter.

```java
g.persist(MEMORY_AND_DISK_SER)
```

### Table 15: DseGraphFrame method list

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gf()</code></td>
<td>Returns a <code>GraphFrame</code> object.</td>
</tr>
<tr>
<td><code>V()</code></td>
<td>Returns a <code>DseGraphTraversal[Vertex]</code> object to start a vertex traversal.</td>
</tr>
<tr>
<td><code>E()</code></td>
<td>Returns a <code>DseGraphTraversal[Edge]</code> object to start an edge traversal.</td>
</tr>
<tr>
<td><code>cache()</code></td>
<td>Cache the graph data with Spark.</td>
</tr>
<tr>
<td><code>persist(level)</code></td>
<td>Cache the graph data with one of the Spark persist levels.</td>
</tr>
<tr>
<td><code>deleteVertices()</code></td>
<td>Delete vertices.</td>
</tr>
</tbody>
</table>
# Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteVertices(label: String)</td>
<td>Delete all vertices with the specified label.</td>
</tr>
<tr>
<td>deleteEdges()</td>
<td>Delete edges.</td>
</tr>
<tr>
<td>deleteVertexProperties()</td>
<td>Delete vertex properties.</td>
</tr>
<tr>
<td>deleteEdgeProperties()</td>
<td>Delete edge properties.</td>
</tr>
<tr>
<td>updateVertices(df: DataFrame, labels: Seq[String] = Seq.empty)</td>
<td>Change the properties of an existing vertex or insert a new vertex. The optional parameter <code>labels</code> improves the performance of the update by iterating over the provided labels rather than deriving the vertices to update from the DataFrame.</td>
</tr>
<tr>
<td>updateEdges(df: DataFrame, labels: Seq[String] = Seq.empty)</td>
<td>Change the properties of an existing edge or insert a new edge. The optional parameter <code>labels</code> improves the performance of the update by iterating over the provided labels rather than deriving the edges to update from the DataFrame.</td>
</tr>
</tbody>
</table>

## Configuring DSE Smart Analytics query routing

By default DSE Graph uses the `DseGraphFrameInterceptorStrategy` which will automatically intercept `count`, `groupBy`, and `drop` queries and route them to `DseGraphFrame` to improve performance. These simpler queries skip StarGraph RDD creation, allowing for faster execution times.

If a `count` or `groupBy` query is against a snapshot or has a path length longer than 2, DSE will not intercept the query.

To disable `DseGraphFrameInterceptorStrategy`, call the `withoutStrategies` method on the graph.

```java
g.withoutStrategies(com.datastax.bdp.graph.impl.tinkerpop.optimizer.DseGraphFrameInterceptorStrategy.class)
.V()
.count()
```

## Using authorization with DseGraphFrame

If you have enabled authorization on DSE, *grant execute permissions* to the `DseGraphRpc` object.

## TinkerPop API support in DseGraphFrame

DseGraphFrame supports a subset of the Apache TinkerPop traversal API.
DseGraphFrame does not support org.apache.tinkerpop.gremlin.process.traversal.Traverser or org.apache.tinkerpop.gremlin.process.traversal.TraversalSideEffects.

Supported methods

DseGraphFrame mimics the TinkerPop graph traversal source by defining two methods: E() and V(). These methods return a GraphTraversal that has all methods defined below. Only a limit set of TinkerPop’s Step classes are supported. Steps other than the ones in the following table will throw an UnsupportedException.

Table 16: TinkerPop read methods

<table>
<thead>
<tr>
<th>Steps</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>CountGlobalStep</td>
<td>count()</td>
</tr>
<tr>
<td>GroupCountStep</td>
<td>groupCount()</td>
</tr>
<tr>
<td>IdStep</td>
<td>id()</td>
</tr>
<tr>
<td>PropertyValuesStep</td>
<td>values()</td>
</tr>
<tr>
<td>PropertyMapStep</td>
<td>propertyMap()</td>
</tr>
<tr>
<td>HasStep</td>
<td>has(), hasLabel()</td>
</tr>
<tr>
<td>IsStep</td>
<td>is()</td>
</tr>
<tr>
<td>VertexStep</td>
<td>to(), out(), in(), both(), toE(), outE(), inE(), bothE()</td>
</tr>
<tr>
<td>EdgeVertexStep</td>
<td>toV(), inV(), outV(), bothV()</td>
</tr>
<tr>
<td>NotStep</td>
<td>not()</td>
</tr>
<tr>
<td>TraversalFilterStep</td>
<td>where()</td>
</tr>
<tr>
<td>AndStep</td>
<td>and(A, B)</td>
</tr>
<tr>
<td>PageRankVertexProgramStep</td>
<td>pageRank()</td>
</tr>
<tr>
<td>DedupGlobalStep</td>
<td>dedup()</td>
</tr>
<tr>
<td>OrderGlobalStep</td>
<td>order()</td>
</tr>
<tr>
<td>LimitGlobalStep</td>
<td>limit()</td>
</tr>
<tr>
<td>SelectStep</td>
<td>as() and select()</td>
</tr>
<tr>
<td>OrStep</td>
<td>or()</td>
</tr>
</tbody>
</table>

This query finds people who know each other and demonstrates the as() and select() methods:
Using DataStax Enterprise advanced functionality

```java
g.V().as("a").out("knows").as("b").out("knows")
    .where(P.eq("a")).select("a", "b").by("name").show
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>Bob</td>
<td>Alice</td>
</tr>
</tbody>
</table>

Table 17: TinkerPop update steps and methods

<table>
<thead>
<tr>
<th>Steps</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>DropStep</td>
<td>V().drop(), E().drop(), properties().drop()</td>
</tr>
<tr>
<td>AddPropertyStep</td>
<td>property(name, value, ...)</td>
</tr>
</tbody>
</table>

DseGraphFrame can be used to drop millions of vertices or edges at once, and is much faster for bulk property updates than Gremlin OLAP or OLTP.

For example this query drops all person vertices and their associated edges:

```java
g.V().hasLabel("person").drop().iterate()
```

Using DseGraphFrame in Scala

GraphTraversal is a Java interface, and extends the Java Iterator interface. To iterate through the results of a traversal as a DataFrame use the df() method. DseGraphFrame supports implicit conversion to DataFrame.

The following example will traverse the vertices of a graph using TinkerPop and then show the result as a DataFrame.

```java
g.V().out().show
```

In some cases you may need to use the TinkerPop Java API to get the correct TinkerPop objects.

For example, to extract the DSE Graph Id object theTraversal Java iterator can be converted to a Scala iterator which allows direct access to the TinkerPop representation of the Id. This method allows you to use the original Id instead of the DataFrame methods which return the DataFrame String representation of the Id, you can also use the toList() and toSet() methods to set the appropriate ID.

```scala
import scala.collection.JavaConverters._
for(i <-g.V().id().asScala) println (i)
```

{~label=vertex, community_id=748226688, member_id=0}
Using DataStax Enterprise advanced functionality

```java
(~label=custom, name=Name, value=1)
g.V.id.toSet

res18: java.util.Set[Object] = [{~label=demigod,
  community_id=224391936, member_id=0}, ...
```

The TinkerPop P (predicate) and T (constant) classes are imported by the Spark shell automatically.

```java
g.E().groupCount().by(T.label)
g.V().has("age", P.gt(30)).show
```

For standalone applications, import theses classes.

```java
import org.apache.tinkerpop.gremlin.structure.T
import org.apache.tinkerpop.gremlin.process.traversal.P
import org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__
```

Scala is not always able to infer the return type, especially in the Spark shell. The property values of the type should be provided explicitly.

```java
g.V().values[Any]("name").next()
```

Or similarly:

```java
val n: String = g.V().values("name").next()
```

Explicitly set the type when dropping properties.

```java
g.V().properties[Any]("age", "name").drop().iterate()
```

In this case, using the DataFrame API is easier as you do not need to specify the type.

```java
g.V().properties("age", "name").drop().show()
```

```
+-----+
| age  |
+-----+
| 10000|
```
Using DataStax Enterprise advanced functionality

Table 18: Using Java methods in DseGraphFrame Scala applications

<table>
<thead>
<tr>
<th>Method</th>
<th>Use case</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasNext()</td>
<td>You want to know if there’s a result, but you don’t care about the value.</td>
<td>Did Alice create any other vertices</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g.V().has(&quot;name&quot;, &quot;Alice&quot;).outE(&quot;created&quot;).hasNext()</td>
</tr>
<tr>
<td>next()</td>
<td>You know that there is at least 1 result and you want to get the first one (or the second if you call it twice, and so on).</td>
<td>Get the vertex label distribution. Group steps will always return exactly 1 result.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g.V().groupCount().by(label).next()</td>
</tr>
<tr>
<td>iterate()</td>
<td>You just want to execute the traversal, but don’t care about the result and whether it did anything at all.</td>
<td>Set all person’s ages to 10.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g.V().property(&quot;age&quot;, 10).iterate()</td>
</tr>
<tr>
<td>toList(), toSet()</td>
<td>You expect the result to contain an arbitrary number of items and you want to get all of them.</td>
<td>Get all the people Alice knows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g.V().has(&quot;name&quot;, &quot;Alice&quot;).out(&quot;knows&quot;).toList()</td>
</tr>
</tbody>
</table>

Mapping rules for DseGraphFrame

DseGraphFrame uses mapping rules for column names and types.

Column mapping rules

DataFrame column names are the same as graph property names except in the following cases.

- Conflict with column names reserved by GraphFrame will result in an underscore (_) added to the property name. For example, the id column will result in a property named _id.

Table 19: Reserved column names in GraphFrame

<table>
<thead>
<tr>
<th>Reserved column name</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>src</td>
</tr>
<tr>
<td>dst</td>
</tr>
<tr>
<td>new_id</td>
</tr>
<tr>
<td>new_src</td>
</tr>
</tbody>
</table>
**Reserved column name**

<table>
<thead>
<tr>
<th>Reserved column name</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_dst</td>
</tr>
<tr>
<td>graphx_attr</td>
</tr>
</tbody>
</table>

- DseGraphFrame and Spark SQL are case insensitive by default. Column names that differ only in case will result in conflicts. Set the Spark property `spark.sql.caseSensitive=true` to avoid case conflicts.

```
$ dse spark --conf spark.sql.caseSensitive=true
```

**Type mapping rules**

DseGraphFrame and Spark SQL have a limited set of supported types. A vertex is represented by a `Row` instance.

If the vertex has multiple properties, each property will be represented as a Spark SQL array with property values. If a property has meta-properties it will be represented as `StructType`. The `value` field of the struct contains the property value. All other fields will represent the meta-properties.

**Table 20: DSE Graph to Spark SQL and DseGraphFrame type mapping**

<table>
<thead>
<tr>
<th>DSE Graph type</th>
<th>Spark SQL type</th>
<th>Conversion rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>BooleanType</td>
<td></td>
</tr>
<tr>
<td>smallint</td>
<td>ShortType</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>IntegerType</td>
<td></td>
</tr>
<tr>
<td>bigint</td>
<td>LongType</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>FloatType</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>DoubleType</td>
<td></td>
</tr>
<tr>
<td>decimal</td>
<td>DecimalType(38, 18)</td>
<td></td>
</tr>
<tr>
<td>varint</td>
<td>DecimalType(38, 0)</td>
<td></td>
</tr>
<tr>
<td>timestamp</td>
<td>TimestampType</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>DateType</td>
<td></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>DSE Graph type</th>
<th>Spark SQL type</th>
<th>Conversion rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>LongType</td>
<td>The number of nanoseconds from the beginning of the day.</td>
</tr>
<tr>
<td>text</td>
<td>StringType</td>
<td></td>
</tr>
<tr>
<td>uuid</td>
<td>StringType</td>
<td>The UUID.toString() and UUID.fromString() methods are used to convert the value.</td>
</tr>
<tr>
<td>inet</td>
<td>StringType</td>
<td>The toString and InetAddress.getByName() methods are used to convert the value.</td>
</tr>
<tr>
<td>blob</td>
<td>BinaryType</td>
<td></td>
</tr>
<tr>
<td>'PointType'</td>
<td>StringType</td>
<td>The toWellKnownText() and fromWellKnownText() methods are used to convert the value.</td>
</tr>
<tr>
<td>'LineStringType'</td>
<td>StringType</td>
<td>The toWellKnownText() and fromWellKnownText() methods are used to convert the value.</td>
</tr>
<tr>
<td>'PolygonType'</td>
<td>StringType</td>
<td>The toWellKnownText() and fromWellKnownText() methods are used to convert the value.</td>
</tr>
</tbody>
</table>

**Exporting graphs using DseGraphFrame**

Use `DseGraphFrame` to export the graph to any format supported by Spark.

1. Export the vertices and edges to a Spark-supported using the `write` method.

   Export the graph to a JSON file in the DSEFS file system.

   ```
g.V.write.json("/tmp/v_json")
g.E.write.json("/tmp/e_json")
```

   That will create two directories in the DSEFS file system with vertex and edge JSON files. You can get data locally if they are not too large for the local file system:

   ```
   $ dse fs "cat /tmp/v_json/* > file:/home/username/local_vertices.json" &&
   dse fs "cat /tmp/e_json/* > file:/home/username/local_edges.json"
   ```

**Importing graphs using DseGraphFrame**

Use `DseGraphFrame` to import a graph to DataStax Enterprise.
Prerequisites:

The graph schema should be created manually in the Gremlin console or DataStax Studio before importing the graph. Import only works with custom ID mapping.

1. Start the Spark shell.
   
   ```
   $ dse spark
   ```

2. If you exported the graph to JSON using DseGraphFrame, import it in the Spark shell.
   
   ```
   val g = spark.dseGraph("gods_import")
   g.updateVertices(spark.read.json("/tmp/v.json"))
   g.updateEdges(spark.read.json("/tmp/e.json"))
   ```

3. If you have a custom graph:

   a. Examine the schema of the graph and note how to map it to the expected schema of a DSE Graph schema.

   This example will use the `friends` graph from the GraphFrame project.

   ```
   import org.graphframes._
   val g: GraphFrame = examples.Graphs.friends
   g.vertices.printSchema
   ```

   ```
   root
   |-- id: string (nullable = true)
   |-- name: string (nullable = true)
   |-- age: integer (nullable = false)
   ```

   ```
   g.edges.printSchema
   ```

   ```
   root
   |-- src: string (nullable = true)
   |-- dst: string (nullable = true)
   |-- relationship: string (nullable = true)
   ```

   b. In the Gremlin console or DataStax Studio create the schema.

   ```
   system.graph('friends').create()
   :remote config alias g friends.g
   schema.propertyKey("age").Int().create()
   schema.propertyKey("name").Text().create()
   ```
Using DataStax Enterprise advanced functionality

```scala
schema.propertyKey("id").Text().single().create()

schema.vertexLabel('people').partitionKey("id").properties("name", "age").create();
    schema.edgeLabel("friend").create()
    schema.edgeLabel("follow").create()

c. In the Spark shell create an empty `DseGraphFrame` graph and check the target schemas.

```scala
val d = spark.dseGraph("friends")
d.V.printSchema

```
root
|-- id: string (nullable = false)
|-- ~label: string (nullable = false)
|-- _id: string (nullable = true)
|-- name: string (nullable = true)
|-- age: integer (nullable = true)
``` 

d. Convert the edges and vertices to the target format.

```scala
val v = g.vertices.select ($"id" as "_id", lit("people") as "~label", $"name", $"age")
val e = g.edges.select (d.idColumn(lit("people"), $"src") as "src", d.idColumn(lit("people"), $"dst") as "dst", $"relationship" as "~label")
```

e. Append the converted vertices and edges to the target graph.

```scala
d.updateVertices (v)
d.updateEdges (e)
```

Using the Northwind demo graph with Spark OLAP jobs

The Northwind demo included with the DSE demos has a script for creating a graph of the data for a fictional trading company.
In this task, you'll use the Gremlin console to create the Northwind graph, snapshot part of the graph, and run a count operation on the subgraph using the SparkGraphComputer.

**Prerequisites:**
- Enable DSE Graph, DSE Search, and DSE Analytics modes ([page 1437](#)) in your datacenter.
- Install the DSE Graph Loader ([page 485](#)).
- Clone the graph-examples Git repository to the machine on which you are running the Gremlin console.

```
$ git clone https://github.com/datastax/graph-examples.git
```

1. Load the Northwind graph and supplemental data using the `graphloader` tool:

```
graphloader -graph northwind -address localhost graph-examples/northwind/northwind-mapping.groovy -inputpath graph-examples/northwind/data &&
graphloader -graph northwind -address localhost graph-examples/northwind/supplemental-data-mapping.groovy -inputpath graph-examples/northwind/data/
```

2. Start the Gremlin console using the `dse gremlin-console` command:

```
$ dse gremlin-console
```

3. Alias the traversal to Northwind graph using the default OLTP traversal source:

```
:remote config alias g northwind.g
```

4. Set the schema mode to Development.

To allow modifying the schema for the connected graph database, you must set the mode to Development each session. The default schema mode for DSE Graph is Production, which doesn't allow you to modify the graph's schema.

```
schema.config().option('graph.schema_mode').set('Development')
```

5. Enable the use of scans and lambdas.

```
schema.config().option('graph.allow_scan').set('true')
graph.schema().config().option('graph.traversal_sources.g.restrict_lambda').set(false)
```

6. Look at the schema of the northwind graph:
Using DataStax Enterprise advanced functionality

7. Alias the traversal to the Northwind analytics OLAP traversal source \( a \). Alias \( g \) to the OLAP traversal source for one-off analytic queries:

\[
\text{:remote config alias g northwind.a}
\]

\[
\Rightarrow g = \text{northwind.a}
\]

8. Count the number of vertices using the OLAP traversal source:

\[
g.V().count()
\]

\[
\Rightarrow 3294
\]

When you alias \( g \) to the OLAP traversal source \text{database name.a}, DSE Analytics is the workload back-end.

9. Store subgraphs into snapshots using \text{graph.snapshot()}. When you need to run multiple OLAP queries on a graph in one session, use snapshots of the graph as the traversal source.

\[
\text{employees} = \text{graph.snapshot().vertices('employee').create()}
\]

\[
\Rightarrow \text{graphtraversalsource[hadoopgraph[persistedinputrdd->persistedoutputrdd], sparkgraphcomputer]}
\]

\[
\text{categories} = \text{graph.snapshot().vertices('category').create()}
\]

\[
\Rightarrow \text{graphtraversalsource[hadoopgraph[persistedinputrdd->persistedoutputrdd], sparkgraphcomputer]}
\]

The \text{snapshot()} method returns an OLAP traversal source using the \text{SparkGraphComputer}.

10. Run an operation on the snapshot graphs.

Count the number of employee vertices in the snapshot graph:

\[
\text{employees.V().count()}
\]

\[
\Rightarrow 9
\]

Count the number of category vertices in the snapshot graph:
Using DataStax Enterprise advanced functionality

categories.V().count()

==> 8

DSE Graph Tools

In addition to the Gremlin console, other tools are available for working with DSE Graph:

**DataStax Studio**
Web-based notebook-style visualization tool. Currently supports Markdown and Gremlin. Includes a variety of list and graph functions.

![DataStax Studio Screenshot](image)

**DSE OpsCenter**
Visual management and monitoring tool.

![DSE OpsCenter Screenshot](image)

**DSE Lifecycle Manager**
Powerful provisioning and configuration management tool.
Using DataStax Enterprise advanced functionality

Starting the Gremlin console

Gremlin is the query language used to interact with DSE Graph. One method of inputting Gremlin code is to use the Gremlin console. The Gremlin console is a useful interactive environment for directly inputting Gremlin to create graph schema, load data, administer graph, and retrieve traversal results. The Gremlin Console is an interface to the Gremlin Server that can interact with DSE Graph.

- Start the Gremlin console using the `dse` command and passing the additional command `gremlin-console`:

```
$ bin/dse gremlin-console
```

```
\,\,/  
( o o )  
-----o000o-(3)-o000o-----  
plugin activated: tinkerpop.server  
plugin activated: tinkerpop.utilities  
plugin activated: tinkerpop.tinkergraph  
gremlin>
```

Three plugins are activated by default, as shown. The Gremlin Server, `tinkerpop.server`, is started so that commands can be issued to DSE Graph. The `utilities` plugin, `tinkerpop.utilities`, provides various functions, helper methods and imports of external classes that are useful in Gremlin console. TinkerGraph, an in-memory graph that is used as an intermediary for some graph operations is started with `tinkerpop.tinkergraph`. The Gremlin console automatically connects to the remote Gremlin Server.

**Note:** The Gremlin console packaged with DataStax Enterprise does not allow plugin installation like the Gremlin console packaged with Apache TinkerPop.
• Gremlin console help can be displayed with the -h flag:

$ bin/dse gremlin-console -h

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-C, --color</td>
<td>Disable use of ANSI colors</td>
</tr>
<tr>
<td>-D, --debug</td>
<td>Enabled debug Console output</td>
</tr>
<tr>
<td>-Q, --quiet</td>
<td>Suppress superfluous Console output</td>
</tr>
<tr>
<td>-V, --verbose</td>
<td>Enable verbose Console output</td>
</tr>
<tr>
<td>-e, --execute=SCRIPT ARG1 ARG2 ...</td>
<td>Execute the specified script (SCRIPT ARG1 ARG2 ...) and close the console on completion</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Display this help message</td>
</tr>
<tr>
<td>-i, --interactive=SCRIPT ARG1 ARG2 ...</td>
<td>Execute the specified script and leave the console open on completion</td>
</tr>
<tr>
<td>-l</td>
<td>Set the logging level of components that use standard logging output</td>
</tr>
<tr>
<td>-v, --version</td>
<td>Display the version of the Console</td>
</tr>
</tbody>
</table>

**Tip:** Use -v to display all lines when loading a file, to discover which line of code causes an error.

• Run the Gremlin console with the **host:port** option to specify a specific host and port:

$ bin/dse gremlin-console 127.0.0.1:8182

Any hostname or IP address will work to specify the **host**.

• Run Gremlin console with the **-e** flag to execute one or more scripts:

$ bin/dse gremlin-console -e test1.groovy -e test2.groovy

If the scripts run successfully, the command will return with the prompt after execution. If errors occur, the standard output will show the errors.
Using DataStax Enterprise advanced functionality

• If you prefer to have Gremlin console open at the script completion, run Gremlin console with the `-i` flag instead of the `-e` flag:

```bash
$ bin/dse gremlin-console -i test1.groovy -i test2.groovy
```

If the scripts run successfully, the command will return with the Gremlin console prompt after execution. If errors occur, the console will show the errors.

• Discover all Gremlin console commands with help. Console commands are not Gremlin language commands, but rather commands issued to the Gremlin console for shell functionality. The Gremlin console is based on the Groovy shell.

```text
:help
```

For information about Groovy, visit:
http://groovy-lang.org

Available commands:
:help (:h ) Display this help message
? (:? ) Alias to: :help
:exit (:x ) Exit the shell
:quit (:q ) Alias to: :exit
import (:i ) Import a class into the namespace
;display (:d ) Display the current buffer
:clear (:c ) Clear the buffer and reset the prompt counter.
:show (:S ) Show variables, classes or imports
:inspect (:n ) Inspect a variable or the last result with the
GUI object browser
:purge (:p ) Purge variables, classes, imports or preferences
:edit (:e ) Edit the current buffer
:load (:l ) Load a file or URL into the buffer
.: (: ) Alias to: :load
:save (:s ) Save the current buffer to a file
:record (:r ) Record the current session to a file
:history (:H ) Display, manage and recall edit-line history
:alias (:a ) Create an alias
:register (:rc ) Registers a new command with the shell
:doc (:D ) Opens a browser window displaying the doc for the argument
:set (:= ) Set (or list) preferences
:uninstall (:u ) Uninstall a Maven library and its dependencies from the Gremlin Console
:install (:i ) Install a Maven library and its dependencies into the Gremlin Console
:plugin (:pin) Manage plugins for the Console
:remote (:rem) Define a remote connection
:submit (:> ) Send a Gremlin script to Gremlin Server

For help on a specific command type:
:help command
Using DataStax Enterprise advanced functionality

The Gremlin Console provides code help via auto-complete functionality, using the `<TAB>` key to trigger a list of possible options.

**Note:** `:install` and `:plugin` should not be used with DSE Graph. These commands will result in gremlin console errors.

## DSE Graph Reference

### The graph API

graph commands add data to an existing graph.

#### addEdge

**Synopsis**

```java
vertex1.addEdge('edgeLabel', vertex2, [T.id, 'edge_id'], ['key', 'value'] [...])
```

**Description**

Edge data is inserted using addEdge. A previously created edge label (page 652) must be specified. An edge_id may be specified, to upsert data for a multiple cardinality edge to prevent creation of a new edge. Property key-value pairs may be optionally specified.

**Examples**

Create an edge with an *edge label* rated between the vertices johnDoe and beefBourguignon with the *properties* timestamp, stars, and comment.

```
johnDoe.addEdge('rated', beefBourguignon, 'timestamp', '2014-01-01T00:00:00.00Z', 'stars', 5, 'comment', 'Pretty tasty!')
```

Update an edge with an *edge label* created between the vertices juliaChild and beefBourguignon, specifying the edge with an edge id of 2c85fabd-7c49-4b28-91a7-ca72ae53fd39, and a *property* createDate of 2017-08-22:

```
johnDoe.addVertex('label', 'label_name', 'key', 'value', 'key', 'value')
```

Note that a conversion function must be used to convert a string to the UUID. T.id is a literal that must be included in the statement.
Using DataStax Enterprise advanced functionality

Description

Vertex data is inserted using addVertex. A previously created vertex label (page 659) must be specified.

Examples

Create a vertex with a vertex label reviewer with the properties location and status.

```java
graph.addVertex(label, 'reviewer', 'location', 'Santa Cruz, CA', 'status', 'Rock Star')
```

io

Synopsis

```java
io( [gryo() | graphson() | graphml()]).[readGraph | writeGraph]

(file_name)
```

Description

Graph data is written to a file or read from a file using io. The file to read must be located on a DSE cluster node, and the written file will be created on the DSE cluster node on which the command is run.

Examples

Write the graph data to a file using the Gryo format:

```java
graph.io(gryo()).writeGraph('/tmp/test.gryo')
```

Read the graph data from a file using the Gryo format:

```java
graph.io(gryo()).readGraph('/tmp/test.gryo')
```

**Restriction:** This method of reading a graph is not recommended, and will not work with graphs larger than 10,000 vertices or elements. DSE Graph Loader (page 484) is a better choice in production. Additionally, a schema setting may need modification for this method to work:

```java
schema.config().option("tx_autostart").set(true)
```

property

Synopsis

```java
vertex1.property( [['key', 'value'] [...] | [T.id, 'property_id']])
```
Description

Property data is inserted using `property`. Property key-value pairs are specified. A `property_id` may be specified, to upsert data for a multiple cardinality property to prevent creation of a new property.

Examples

Create a property with values for `gender` and `nickname`.

```
jamieOliver.property('gender', 'M', 'nickname', 'jimmy')
```

Update the property `gender` for the vertex juliaChild specifying a property with a property id of 2c85fadb-7c49-4b28-91a7-ca72ae53fd39:

```
uuid = java.util.UUID.fromString('2c85fadb-7c49-4b28-91a7-ca72ae53fd39')
juliaChild.property('gender', 'F', T.id, uuid)
```

Note that a conversion function must be used to convert a string to the UUID. `T.id` is a literal that must be included in the statement.

**tx().config().option()**

Synopsis

```
tx().config().option(option).open()
```

Description

Examples

Change the value of `allow_scan` for a transaction. The effect of this change is to allow all commands executed in the gremlin-console on a particular node to do full graph scans, even if the consistency level for the cluster is not `QUORUM`, the value required to change this option in the appropriate system table.

```
graph.tx().config().option("allow_scan", true).open()
```
Using DataStax Enterprise advanced functionality

Note that the previous transaction (automatically opened in gremlin-console or Studio) must be committed before the new configuration option value is set.

**The schema API**

`schema` commands are used to create schema such as vertex labels, edge labels, property keys and indexes.

**drop**

**Synopsis**

```java
schema.drop()
```

**Description**

Drop either all schema information or a particular element of the schema for a particular graph using this command. All data will also be dropped if all schema is dropped.

**Examples**

If using the Gremlin console, an alias must be created to bind the graph to a graph traversal before running this command. If using Studio, no prerequisite is required. To drop all schema and data for a particular graph:

```java
schema.drop()
```

The result if the drop command is successful:

```java
==>null
```

**Danger:** If this command is used, the graph will no longer have any schema or data!

To drop a single schema element, such as a property key, specify the schema to drop:

```java
schema.propertyKey('gender').drop()
```

All schema elements (properties, edge labels, vertex labels, and indexes) can be removed with this method.

**connection**

**Synopsis**

```java
connection('outV', 'inV')
```

**Description**

An adjacency between two vertices is created using an edge label and the vertex labels of the outgoing and incoming vertices. This step is used in conjunction with `edgeLabel()`.
Examples

Create an edge label `{isA}` specifying that the outgoing vertex label is `{ingredient}` and the incoming vertex label is `{FridgeItem}`.

```java
schema.edgeLabel('isA').connection('ingredient', 'FridgeItem').create()
```

An adjacency between the vertexLabel `{author}` and `{author}` specifying the edgeLabel `{knows}`.

```java
schema.edgeLabel('knows').connection('author', 'author').add()
```

### config

**Synopsis**

```java
schema.config().option(arg).[ set(value) | unset(value) | get() | exists() | describe() ]
```

Schema can be configured per graph using the `config()` command. An option and value can be `set()` or `unset()`. An option's value can be retrieved with the `get()` command. Whether or not the option is configured can be discovered with the `exists()` command. The `describe()` command returns a value if the option has been set manually.

#### Table 21: Graph-specific options

[Graph-specific options are preceded by `graph`. For example, `graph.schema_mode`]

<table>
<thead>
<tr>
<th>Option argument</th>
<th>Setting Example</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>allow_scan</code></td>
<td>true</td>
<td>Setting to allow costly graph scan queries.</td>
<td>true</td>
</tr>
<tr>
<td><code>schema_mode</code></td>
<td>Production</td>
<td>Set mode to Production or Development.</td>
<td>Development</td>
</tr>
<tr>
<td><code>default_property_key_cardinality</code></td>
<td>single</td>
<td>Set the cardinality that will be used by default unless otherwise specified.</td>
<td></td>
</tr>
<tr>
<td><code>tx_autostart</code></td>
<td>true</td>
<td>Set whether transactions are started automatically or must be manually opened.</td>
<td>false</td>
</tr>
</tbody>
</table>

#### Table 22: TraversalSource-specific options

[TraversalSource-specific options are preceded by `graph.traversal_sources.*` where `*` must be a specified traversal source such as the graph traversal `g`. For example, `graph.traversal_sources.g.type`. The most common TraversalSource is the graph traversal `g`.]
### Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Option argument</th>
<th>Setting Example</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation_timeout</td>
<td>PT10S (10 seconds) or &quot;1500 ms&quot;</td>
<td>Maximum time to wait for a traversal to evaluate - this will override other system level settings for the current TraversalSource.</td>
<td>0 days</td>
</tr>
<tr>
<td>restrict_lambda</td>
<td>false</td>
<td>Prevent the use of lambdas with this TraversalSource. A particular traversal source can be identified.</td>
<td>true</td>
</tr>
<tr>
<td>type</td>
<td>read-only</td>
<td>Specify type of TraversalSource. A particular traversal source can be identified.</td>
<td>default</td>
</tr>
</tbody>
</table>

**Important:** Setting a timeout value greater than 1095 days (maximum integer) can exceed the limit of a graph session. Starting a new session and setting the timeout to a lower value can recover access to a hung session. This caution is applicable for all timeouts: evaluation_timeout, system_evaluation_timeout, analytic_evaluation_timeout, and realtime_evaluation_timeout.

**Table 23: Transaction-specific options**

[Transaction-specific options are preceded by `graph.tx_groups.*` where * must be specified as a transaction group or default. For example, `graph.tx_groups.default.read_only` will make all transactions which aren't explicitly named read_only, whereas `graph.tx_groups.myTxGroup.read_only` would apply only to transactions which are given the group name myTxGroup.]

<table>
<thead>
<tr>
<th>Option</th>
<th>Setting Example</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>authenticated_user</td>
<td>test_user</td>
<td>The username to use as the current user for a transaction.</td>
<td>ANONYMOUS_USER</td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td><strong>Setting Example</strong></td>
<td><strong>Description</strong></td>
<td><strong>Default</strong></td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>cache</td>
<td>false</td>
<td>Cache retrievals and data store calls within a transaction in transaction-level caches. This setting provides a restricted type of isolation within a transaction (concurrent modifications in other transactions aren't visible and result sets remain consistent between calls) and can improve performance at the expense of additional memory consumption.</td>
<td>true</td>
</tr>
<tr>
<td>deep_profiling</td>
<td>true</td>
<td>Enable CQL tracing for <code>profile()</code> in queries. Very costly profiling.</td>
<td>false</td>
</tr>
<tr>
<td>internal_vertex_verify</td>
<td>true</td>
<td>Set whether a transaction should verify that vertices for internally provided auto-generated vertex ids actually exist.</td>
<td>false</td>
</tr>
<tr>
<td>external_vertex_verify</td>
<td>false</td>
<td>Set whether a transaction should verify that vertices for externally provided user-defined vertex ids actually exist.</td>
<td>true</td>
</tr>
<tr>
<td>logged_batch</td>
<td>true</td>
<td>Use a logged batch when committing changes. This guarantees that all mutations will eventually occur at the expense of performance.</td>
<td>false</td>
</tr>
<tr>
<td>Option</td>
<td>Setting Example</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>max_mutations</td>
<td>5000</td>
<td>The maximum number of vertices, properties and edges (cumulatively) that may be added or removed in a single transaction.</td>
<td>10000</td>
</tr>
<tr>
<td>max_profile_events</td>
<td>5</td>
<td>The maximum number of profiling events to report for an individual traversal step. Restricting the number of reported events makes output manageable, but can hide important information.</td>
<td>10</td>
</tr>
<tr>
<td>prefetch</td>
<td>true</td>
<td>Sets whether the query executor should asynchronously pre-fetch data based on its expected execution of the traversal prior to the data being requested. This can reduce transaction latency but can cause throughput to worse.</td>
<td>true</td>
</tr>
<tr>
<td>read_only</td>
<td>true</td>
<td>Set whether a transaction is read-only.</td>
<td>false</td>
</tr>
<tr>
<td>read_consistency</td>
<td>ALL</td>
<td>Specify the consistency level for read operations of a transaction.</td>
<td>ONE</td>
</tr>
<tr>
<td>single_thread</td>
<td>true</td>
<td>Set whether a transaction is only accessed by a single thread.</td>
<td>false</td>
</tr>
<tr>
<td>thread_bound</td>
<td>true</td>
<td>Set whether a transaction is bound to a particular thread.</td>
<td>false</td>
</tr>
<tr>
<td>Option</td>
<td>Setting Example</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>transaction_timestamp</td>
<td></td>
<td>The timestamp at which all mutations of this transaction are persisted.</td>
<td>Instant.EPOCH</td>
</tr>
<tr>
<td>verify_unique</td>
<td>false</td>
<td>Set whether transactions should ensure that uniqueness constraints are enforced.</td>
<td>true</td>
</tr>
<tr>
<td>vertex_cache_size</td>
<td>4000</td>
<td>Maximum size of the transaction-level cache of recently-used vertices</td>
<td>20000l</td>
</tr>
<tr>
<td>vertex_dirty_size</td>
<td></td>
<td>This is a performance hint for write-heavy, performance-sensitive transactional workloads. If set, it should roughly match the median vertices modified per transaction.</td>
<td>32</td>
</tr>
<tr>
<td>write_consistency</td>
<td>ANY</td>
<td>Specify the consistency level for write operations of a transaction</td>
<td>QUORUM</td>
</tr>
</tbody>
</table>

Description

Configure a graph. Options can be set, unset, or get (retrieve the value).

Examples

Set the current graph to use the Development schema_mode:

```
schema.config().option('graph.schema_mode').set('Development')
```

Set the current graph to allow full graph scans in the currently aliased graph:

```
schema.config().option('graph.allow_scan').set('true')
```

To set restrict_lambda to FALSE in order to test lambda functions (only appropriate for non-production systems):
Using DataStax Enterprise advanced functionality

To retrieve all traversal sources that have been set, use the `get()` command with the traversal source type option:

```java
schema.config().option('graph.traversal_sources.*.type').get()
```

resulting in a list of values for the option that have been manually set:

```java
REAL_TIME
```

indicating that a real-time evaluation timeout is set.

To verify that user-defined vertex ids exist during data loading, set the `external_vertex_verify()` option:

```java
schema.config().option('graph.tx_groups.default.external_vertex_verify').set('TRUE')
```

If this setting is true, then a vertex will not be returned if it doesn’t exist. However, if `external_vertex_verify()` is set to false, then a vertex will be returned even if the vertex does not exist given an id. Applications should ensure that vertices exist using the `exists()` method for expected behavior. The `internal_vertex_verify()` setting is similarly used for auto-generated vertex ids.

Set the default write consistency for all transactions to `ALL` in the currently aliased graph:

```java
schema.config().option('graph.tx_groups.default.write_consistency').set('ALL')
```

Get the current write consistency for all transactions in the currently aliased graph:

```java
schema.config().option('graph.tx_groups.default.write_consistency').get()
```

To confirm that an option setting has been set manually, use the `exists()` command:

```java
schema.config().option('graph.tx_groups.default.write_consistency').exists()
```

This command will return:

```java
true
```

if the setting has been set to a value, otherwise it returns `false`.

To enable CQL tracing during traversal query profiling, set the `deep_profiling()` option:

```java
schema.config().option('graph.tx_groups.default.deep_profiling').set('TRUE')
```

To set schema settings (page 645) at the time a graph is created, chain the options into the `system.graph().create()` command:

```java
system.graph('food2').
    replication("{"class" : 'NetworkTopologyStrategy', 'dc1' : 3 }").
    systemReplication("{"class" : 'NetworkTopologyStrategy', 'dc1' : 3 }").
```
option("graph.default_property_key_cardinality").set("multiple").
option("graph.schema_mode").set("Development").
option("graph.allow_scan").set("true").
option("graph.tx_groups.*.write_consistency").set("ALL").
create()

Note the use of a wildcard * to set the write consistency for all transaction groups. If any option is modified after graph creation, a schema.config() command must be used.

To retrieve a list of configuration options that have been set, use the describe() command:

```java
:remote config alias g food2.g
schema.config().describe()
```

resulting in a list of all options that have been manually set:

```java
==>
schema_mode: Development
allow_scan: true

graph.tx_groups.*.write_consistency: ALL
default_property_key_cardinality: Multiple
```

To retrieve the currently set values of schema_mode or allow_scan, set manually or by default, see the schema.getEffective* commands.

There are some configuration options for which the default (for example, values are not explicitly set) is determined by using the value of other configuration options. For instance, if allow_scan is not explicitly set, the default value is true if schema_mode is set to Development, but false if the schema_mode is set to Production. These configuration options are not linked to the default settings, leading to potentially misleading information when using schema.config().get() to discover the setting value because the default value is displayed rather than a set value.

Full graph scan settings are as follows:

<table>
<thead>
<tr>
<th>setting</th>
<th>schema_mode</th>
<th>scans allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>dse.yaml</td>
<td>Production</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>schema_mode:Production</td>
<td></td>
</tr>
<tr>
<td>dse.yaml</td>
<td>Development</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>schema_mode:Development</td>
<td></td>
</tr>
<tr>
<td>graph.schema_mode</td>
<td>Production</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>:Production</td>
<td></td>
</tr>
<tr>
<td>graph.schema_mode</td>
<td>Development</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>:Development</td>
<td></td>
</tr>
<tr>
<td>graph.allow_scan</td>
<td>Production</td>
<td>yes</td>
</tr>
<tr>
<td>true</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>setting</th>
<th>schema mode</th>
<th>scans allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph.allow_scan:true</td>
<td>Development</td>
<td>yes</td>
</tr>
</tbody>
</table>

**describe**

**Synopsis**

`schema.describe()`

**Description**

List schema information about a particular graph using this command. An alias must be created to bind the graph to a graph traversal before running this command.

**Examples**

Discover if a particular graph exists. The return value is a boolean value.

```
gremlin> schema.describe()
```

The resulting list:

```
==>schema.vertexLabel("FridgeItem").create()
```

**edgeLabel**

**Synopsis**

`schema.edgeLabel('edgeLabel').
  [ single() | multiple() ].
  [ properties(property[, property]).[ add() | drop() ] ].
  [ connection(outVertex, inVertex) ].
  [ ttl ].
  [ ifNotExists() ].
  [ create() | add() | drop() | describe() | exists() ]`

**Description**

An edge label specifies a type of edge that can be stored in DSE Graph. An edge label can have cardinality specified (default is multiple), properties that an edge has defined, the connections that are defined between two types of vertices, and a time-to-live (TTL) to determine the lifecycle of an edge. The order that the options are added to the schema statement matter: cardinality, properties associated with the edge label, then connection.

**Examples**

Create an edgeLabel created:

```
schema.edgeLabel('created').create()
```
**Note:** Naming convention used to allow nearly unrestricted Unicode. Now the only allowed characters are [a-zA-Z0-9], underscore, hyphen, and period.

Create an edgeLabel **includedIn** if the edge label doesn't already exist:

```java
schema.edgeLabel('includedIn').ifNotExists().create()
```

Create an edgeLabel with multiple cardinality:

```java
schema.edgeLabel('reviewed').multiple().create()
```

Add properties to an edgeLabel:

```java
schema.edgeLabel('reviewed').properties('rating','last_date').add()
```

Create an edgeLabel with both a property and a connection:

```java
schema.edgeLabel('reviewed').properties('rating').connection('recipe', 'reviewer').create()
```

Create a time-to-live (TTL) for an edgeLabel of 60 seconds. Setting a TTL will expire all edges inserted with the edgeLabel at the set TTL value:

```java
schema.edgeLabel('createDate').ttl(60).create()
```

**Note:** DSE Graph sets TTL differently from the DSE database. The DSE database sets TTL per mutation (insertion or update) or can inherit a default value from the table schema. DSE Graph sets TTL per vertex label or edge label, and all vertices or edges will be affected by the TTL setting. DSE Graph cannot set TTL for an individual vertex or edge.

Check if an edgeLabel exists:

```java
schema.edgeLabel('reviewed').exists()
```

Get the schema creation command for an edgeLabel using the `describe()` command:

```java
schema.edgeLabel('createDate').describe()
```

Remove a edge label with the `drop()` command:

```java
schema.edgeLabel('reviewed').drop()
```

Remove a property **rating** from an edge label:
Using DataStax Enterprise advanced functionality

```java
schema.edgeLabel('reviewed').properties('rating').drop()
```

**exists**

**Synopsis**

```java
exists
```

**Description**

Discover if a particular schema element exists using this command. This command can be used with vertexLabel, edgeLabel, or propertyKey.

**Examples**

Discover if a particular vertex label exists. The return value is a boolean value.

```javascript
gremlin> schema.vertexLabel('author').exists()
```

The resulting list:

```javascript
==>true
```

**index - edge index**

**Synopsis**

```javascript
index('index_name').[outE('edgeLabel') | inE('edgeLabel') | bothE('edgeLabel')].by('propertykey_name').add()
```

**Description**

An edge index specifies an index that is built using an edge property key in DSE Graph. A vertex label must be specified, and edge indexes are only defined in relationship to a vertex label. The index name must be unique.

An edge index can be created using either outgoing edges (outE()) from a vertex label, incoming edges (inE()) from a vertex label, or both outgoing and incoming (bothE()). The last type, bothE(), is rarely used, but could be used in a situation where the index must track both the incoming and outgoing edges from a particular vertex label. An example would be a graph storing reviewers who can both be liked and like other reviewers. To search for reviewers who are liked and who like a particular reviewer, both incoming and outgoing edges would be searched.

**Examples**

Create an index ratedByStars with an outE edge label using the property key stars. The vertex label is specified as reviewer.

```javascript
schema.vertexLabel('reviewer').index('ratedByStars').outE('rated').by('stars').add()
```
Create an *index* `ratedByStars2Way` with a `bothE` edge label using the *property key* `year`. The edge index allows queries that find both recipes with a certain year and reviewers who gave a review in a certain year.

```java
schema.vertexLabel('recipe').index('byAuthOrRecipe').bothE('created').by('year').ifNotExists().add()
```

It can replace two indexes:

```java
schema.vertexLabel('recipe').index('toRecipesRated').inE('rated').by('year').add()
schema.vertexLabel('reviewer').index('toReviewersWhoRated').outE('rated').by('year').add()
```

### index - property index

**Synopsis**

```java
index('index_name').property('propertykey_name').by('meta-propertykey_name').add()
```

**Description**

A property index specifies an index that is built using the *meta-property* *(page 459)* of a vertex property key in DSE Graph. A vertex label must be specified. The index name must be unique. The property key specified must have multiple cardinality.

**Examples**

Create an *index* `byLocation` index using the *property key* `country` and *meta-property key* `livedIn`. The *vertex label* is specified as `author`.

```java
schema().vertexLabel('author').index('byLocation').property('country').by('livedIn').add()
```

### index - vertex index

**Synopsis**

```java
index('index_name').[secondary() | materialized() | search()].by('propertykey_name').[ asText() | asString() ].add()
```

**Description**

A vertex index specifies an index that is built using a vertex property key in DSE Graph. A vertex label must be specified. Vertex indexes can be specified as *secondary*, *materialized*, or *search*. The index name must be unique.

A search vertex index must be named `search`; only one search index can exist. Multiple property keys can be specified in a single search index definition. The options `asText()` and `asString()` must be specified for a search index.

**Examples**

Create an *index* `byRecipe` as a secondary index using the property key `name`. The *vertex label* is specified as `recipe`. 
Using DataStax Enterprise advanced functionality

```java
schema.vertexLabel('recipe').index('byRecipe').secondary().by('name').add()
```

Create an **index** byMeal as a materialized index using the property key name. The **vertex label** is specified as meal.

```java
schema.vertexLabel('meal').index('byMeal').materialized().by('name').add()
```

Create an **index** search as a search index using the property key instructions and specify that the index is a `asText()` . The **vertex label** is specified as recipe.

```java
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().add()
```

Create an **index** search as a search index using multiple property keys instructions with `asText()` and category with `asString()`. The **vertex label** is specified as recipe.

```java
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('category').asString().add()
```

**partitionKey - clusteringKey**

**Synopsis**

```java
partitionKey('id_name').[ clusteringKey('id_name') ]
```

**Description**

`partitionKey` and `clusteringKey` are used to specify a user-defined vertex id ([page 462](#)) in conjunction with `vertexLabel`. The `partitionKey` sets a partition key. A composite partition key can also be set by chaining `partitionKey` items. The `clusteringKey` sets a clustering key. The property keys used must be created prior to use.

**Examples**

Create a **propertyKey** city_id.

```java
schema.propertyKey('city_id').Int().create()
```

Create a **vertexLabel** using `sensor_id` as a partitioning key.

```java
schema().vertexLabel('FridgeSensor').partitionKey('sensor_id').create()
```

Create a vertex label with a custom partitioning key city_id and clustering key sensor_id.

```java
schema().vertexLabel('FridgeSensor').partitionKey('city_id').clusteringKey('sensor_id').create()
```

Create a **vertex** using city_id as a partitioning key and sensor_id as a clustering key. The property key sensor_id must already exist and be an UUID.

```java
graph.addVertex(label, 'FridgeSensor', 'city_id', 100, 'sensor_id', '60bcae02-f6e5-11e5-9ce9-5e5517507c66')
```
Using DataStax Enterprise advanced functionality

Create a **vertexLabel** using **city_id** and **sensor_id** as a composite partitioning key.

```java
schema().vertexLabel('FridgeSensor').partitionKey('city_id', 'sensor_id').create()
```

**properties**

**Synopsis**

```java
properties('name').add()
```

**Description**

Properties can be added to vertices and edges. A **property key (page 657)** must be created prior to adding it to either type of element. Allowed characters for the name are alphabetical or underscore.

**Examples**

Add a property key to a vertex label. The property key **nationality** must exist prior to adding it to the vertex label.

```java
schema.vertexLabel('author').properties('nationality').add()
```

Add more than one property to a vertex label.

```java
schema.vertexLabel('author').properties('nationality', 'age', 'assocRestaurants').add()
```

**propertyKey**

**Synopsis**

```java
propertyKey('name').
  type().
  [ single() | multiple() ].
  [ properties(metadata_property).[ add() | drop() ] ].
  [ ttl ].
  [ ifNotExists() ].
  [ create() | add() | | drop() | describe() | exists() ]
```

**Description**

Property keys are created for vertices and edges. A property key must be created prior to adding it to either type of element. The **data type (page 754)** must be included.

**Note:** Naming convention used to allow nearly unrestricted Unicode. Now the only allowed characters are [a-zA-Z0-9], underscore, hyphen, and period.

A property key can have cardinality specified, **single**(default) or **multiple**, **properties** (**meta-properties**), and a time-to-live (**TTL**) to determine the lifecycle of a property.
Using DataStax Enterprise advanced functionality

**Caution:** Multiple cardinality (multi-properties) will be retrieved in graph traversals more slowly than single cardinality properties, because vertices with multi-properties will default to requesting properties individually.

Examples

Create a property key with the *name* name of Text type:

```java
schema.propertyKey('name').Text().create()
```

**Note:** Naming convention used to allow nearly unrestricted Unicode. Now the only allowed characters are [a-zA-Z0-9], underscore, hyphen, and period.

Create a property key with the *name* num_items of Integer type if the property key doesn't already exist:

```java
schema.propertyKey('num_items').Int().ifNotExists().create()
```

Create a property key with the *name* createDate of Timestamp type with multiple property cardinality:

```java
schema.propertyKey('createDate').Timestamp().multiple().create()
```

Add a meta-property for a property. The meta-property, first_publication, must first be created as a property key.

```java
schema.propertyKey('createDate').properties('first_publication').add()
```

Create a time-to-live (TTL) for a property key of 60 seconds. Setting a TTL will expire all properties inserted with the propertyKey at the set TTL value:

```java
schema.propertyKey('createDate').ttl(60).create()
```

**Note:** DSE Graph sets TTL differently from the DSE database. The DSE database sets TTL per mutation (insertion or update) or can inherit a default value from the table schema. DSE Graph sets TTL per vertex label or edge label, and all vertices or edges will be affected by the TTL setting. DSE Graph cannot set TTL for an individual vertex or edge.

Check if a property key exists:

```java
schema.propertyKey('name').exists()
```

Get the schema creation command for a property key using the describe() command:

```java
schema.propertyKey('name').describe()
```

Remove a property key with the drop() command:
Using DataStax Enterprise advanced functionality

```java
schema.propertyKey('gender').drop()
```

Remove a meta-property `first_publication`:

```java
schema.propertyKey('createDate').properties('first_publication').drop()
```

### vertexLabel

**Synopsis**

```java
schema.vertexLabel('vertexLabel').
    [ partitionKey(propertyKey, [ partitionKey(propertyKey) ] ).
    [ clusteringKey(propertyKey) ].
    [ ttl ].
    [ ifNotExists() ].
    [ index ].
    [ properties(property, property).[ add() | drop() ]].
    [ partition() ].
    [ cache() ].
    [create() | drop() | describe() | exists() ]
```

**Description**

A vertex label specifies a type of vertex that can be stored in DSE Graph. A vertex label can have properties defined, a partition key, clustering key, indexes, cache, and a time-to-live (TTL) to determine the life cycle of an vertex.

DSE Graph limits the number of vertex labels to 200 per graph.

**Examples**

Create a vertexLabel `author`:

```java
schema.vertexLabel('author').create()
```

Create a vertexLabel `ingredient` if the vertex label doesn't already exist:

```java
schema.vertexLabel('ingredient').ifNotExists().create()
```

For partition and clustering keys, see `partitionKey-clusteringKey (page 656)`.

Add properties to a vertexLabel:

```java
schema.vertexLabel('author').properties('location','restaurant').add()
```

For indexes, see each index entry (`edge index (page 654), property index (page 655), vertex index (page 655)`) in the Schema API.

Cache all properties for `author` vertices up to an hour (3600 seconds):

```java
schema.vertexLabel('author').cache().properties().ttl(3600).add()
```
Using DataStax Enterprise advanced functionality

Enabling property cache causes index queries to use IndexCache for the specified vertex label.

Cache both incoming and outgoing created edges for author vertices up to a minute (60 seconds):

```java
schema.vertexLabel('author').cache().bothE('created').ttl(60).add()
```

Partition a vertexLabel based on a particular edgeLabel:

```java
schema.vertexLabel('author').partition().inE('created').add()
```

Create a time-to-live (TTL) for an vertexLabel of 60 seconds. Setting a TTL will expire all vertices inserted with the vertexLabel at the set TTL value:

```java
schema.vertexLabel('author').ttl(60).create()
```

**Note:** DSE Graph sets TTL differently from the DSE database. The DSE database sets TTL per mutation (insertion or update) or can inherit a default value from the table schema. DSE Graph sets TTL per vertex label or edge label, and all vertices or edges will be affected by the TTL setting. DSE Graph cannot set TTL for an individual vertex or edge.

Check if a vertexLabel exists:

```java
schema.vertexLabel('author').exists()
```

Get the schema creation command for a vertexLabel using the `describe()` command:

```java
schema.vertexLabel('author').describe()
```

Remove a vertex label with the `drop()` command:

```java
schema.vertexLabel('author').drop()
```

Remove a property `gender` from a vertex label:

```java
schema.vertexLabel('author').properties('gender').drop()
```

**The system API**

The `system` commands create, drop, and describe graphs, as well as list existing graphs and check for existence. Graph and system configuration can also be set and unset with `system` commands.

**create**

**Synopsis**

```java
system.graph('graph_name').create()
```
Description

Create a new graph. The graph_name specified is used to create two DSE database keyspaces, graph_name and graph_name_system, and can only contain alphanumeric and underscore characters.

Examples

Create a new graph.

```
gremlin> system.graph('FridgeItem').create()
```

The resulting list:

```
==>FridgeItem
```

Create a new graph if it doesn't currently exist by modifying with ifNotExists().

```
gremlin> system.graph('FridgeItem').ifNotExists().create()
```

The resulting list:

```
==>FridgeItem
```

Creating a graph should include setting the replication factor for the graph (page 664) and the graph_system (page 665). It can also include other options (page 662).

**drop**

Synopsis

```
system.graph('graph_name').[ifExists()].drop()
```

Description

Drop an existing graph using this command. All data and schema will be lost.

Examples

Drop a graph.

```
gremlin> system.graph('FridgeItem').drop()
```

The resulting list:

```
==>null
```

Drop an existing graph if it exists.

```
gremlin> system.graph('FridgeSensors').ifExists().drop()
```

The resulting list:
exists

Synopsis

system.graph('graph_name').exists()

Description

Discover if a particular graph exists using this command.

Examples

Discover if a particular graph exists. The return value is a boolean value.

```merlin
> system.graph('FridgeItem').exists()
```

The resulting list:

```merlin
===>true
```

graphs

Synopsis

system.graphs()

Description

Discover what graphs currently exist using this command.

Examples

Discover all graphs that exist in a DSE cluster.

```merlin
> system.graphs()
```

The resulting list:

```merlin
===>quickstart
===>test
```

option

Synopsis

option(arg).set( value )

Graphs can be configured per graph using the following options. The Gremlin console must be used to set system commands.
**Table 24: Graph-Specific Options**

[Graph-specific options are preceded by graph. For example, graph.replication_config.]

<table>
<thead>
<tr>
<th>Option argument</th>
<th>Setting Example</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>replication_config</td>
<td><code>{ 'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }</code></td>
<td>Set replication configuration for a single graph.</td>
<td><code>{ 'class' : 'SimpleStrategy', 'replication_factor' : 1 }</code></td>
</tr>
<tr>
<td>(replaced by replication (page 664) in DSE 5.1.3 and later)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>system_replication_config</td>
<td><code>{ 'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }</code></td>
<td>Set replication configuration for a single graph_system (page 755) data.</td>
<td><code>{ 'class' : 'SimpleStrategy', 'replication_factor' : 1 }</code></td>
</tr>
<tr>
<td>(replaced by systemReplication (page 665) in DSE 5.1.3 and later)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>default_property_key_cardinality</td>
<td><strong>Multiplicity</strong></td>
<td>The default cardinality for automatically defined properties</td>
<td>Single</td>
</tr>
</tbody>
</table>

**Description**

Configure a graph. Options can be set.

**Restriction:** The replication factor and system replication factor cannot be altered once set for the graph_name and graph_name_system keyspaces.

**Examples**

Create a new graph and set the graph replication configuration and the graph_system replication configuration to the DSE database settings shown.

```java
system.graph('food').
    option("graph.replication_config").set("{'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }").
    option("graph.system_replication_config").set("{'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }").
    ifNotExists().create()
```

The resulting list:

```java
==>null
```

The replication settings can be verified using the cqlsh tool, running the CQL command `DESCRIBE keyspace food;`.

**Note:** The options shown (graph.replication_config and graph.system_replication_config) have been replaced in DSE 5.1.3 and later; see the table above.
Other schema settings *(page 645)* can be set at graph creation, but must be changed using `schema.config()` if modified later.

```java
system.graph('food2')
  .option("graph.replication_config").set("{'class' : 'SimpleStrategy', 'replication_factor' : 1 }").
  option("graph.system_replication_config").set("{'class' : 'SimpleStrategy', 'replication_factor' : 1 }")
  option("graph.schema_mode").set("Development").
  option("graph.allow_scan").set("false").
  option("graph.default_property_key_cardinality").set("multiple").
  option("graph.tx_groups.*.write_consistency").set("ALL").
create()
```

To check the schema settings:

```java
:remote config alias g food2.g

schema.config().describe()
```

to get the results:

```java

graph.schema_mode: Development
graph.allow_scan: False
graph.tx_groups.*.write_consistency: ALL
graph.default_property_key_cardinality: Multiple
gremlin> schema.config().option("graph.allow_scan").set("true")
```

Note the use of a wildcard `*` to set the write consistency for all transaction groups.

**replication**

**Synopsis**

```java
system.graph('graph_name').replication("{'class' : 'NetworkTopologyStrategy', 'dc1' : 3}"
```

**Description**

Create a new graph. The `graph_name` specified is used to create two DSE database keyspaces, `graph_name` and `graph_name_system`, and can only contain alphanumeric and underscore characters.

**Examples**

Create a new graph and set the `graph_name` replication configuration using `replication()` as well as the `graph_name_system` configuration using `systemReplication()`. DSE database settings for replication factor are used, either `SimpleStrategy` for single nodes or `NetworkTopologyStrategy` for multiple nodes or multiple datacenters.

The default replication strategy for a multi-node or multi-datacenter graph is `NetworkTopologyStrategy`, whereas for a single node, the replication strategy will default to `SimpleStrategy`. The number of nodes will determine the default replication factor:
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>number of nodes per datacenter</th>
<th>graph_name replication factor</th>
<th>graph_name_system replication factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>number of nodes per datacenter</td>
<td>number of nodes per datacenter</td>
</tr>
<tr>
<td>greater than 3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

system.graph('food').

replication({"class" : 'NetworkTopologyStrategy', 'dcl' : 3 }).

systemReplication({"class" : 'NetworkTopologyStrategy', 'dcl' : 3 }).

ifNotExists().create()

The resulting list:

```
==>null
```

The replication settings can be verified using the cqlsh tool, running the CQL command `DESCRIBE keyspace food;`.

In addition to setting the replication factor for the `graph_name` keyspace, the replication factor for the `graph_name_system` (page 665) must also be set.

**Restriction:** The replication factor and system replication factor cannot be altered once set for the `graph_name` and `graph_name_system` keyspaces.

**systemReplication**

**Synopsis**

```
system.graph('graph_name').systemReplication({"class" : 'NetworkTopologyStrategy', 'dcl' : 3 })
```

**Description**

Create a new graph. The `graph_name` specified is used to create two DSE database keyspaces, `graph_name` and `graph_name_system`, and can only contain alphanumeric and underscore characters.

**Examples**

Create a new graph and set the `graph_name` replication configuration using `replication()` as well as the `graph_name_system` configuration using `systemReplication()`. DSE database settings for replication factor are used, either `SimpleStrategy` for single nodes or `NetworkTopologyStrategy` for multiple nodes.

The default replication strategy for a multi-node or multi-datacenter graph is `NetworkTopologyStrategy`, whereas for a single node, the replication strategy will default to `SimpleStrategy`. The number of nodes will determine the default replication factor:
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>number of nodes per datacenter</th>
<th>graph_name replication factor</th>
<th>graph_name_system replication factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>number of nodes per datacenter</td>
<td>number of nodes per datacenter</td>
</tr>
<tr>
<td>greater than 3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

```cql
system.graph('food').
    replication("{'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }")
    .systemReplication("{'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }")
    .ifNotExists().create()
```

The resulting list:

```cql
==>null
```

The system replication settings can be verified using the `cqlsh` tool, running the CQL command `DESCRIBE keyspace food_system;`.

**Important:** Because the graph’s schema is stored in `graph_name_system`, it is extremely important that the replication factor is set consistent with the table values above. If the graph’s schema is lost, it renders the entire graph inoperable.

In addition to setting the replication factor for the `graph_name_system` keyspace, the replication factor for the `graph_name` (page 664) must also be set.

**The traversal API**

**Apache TinkerPop™ graph computing framework**

Apache TinkerPop™ is a graph abstraction layer that works with numerous different graph databases and graph processors. TinkerPop is composed of two elements: a structure API and a process API.

The primary components of the TinkerPop structure API are:

**Graph**

Maintains a set of vertices and edges.

**Vertex**

Extends a general class `Element` and maintains a set of incoming and outgoing edges as well as a collection of properties and a vertex type. DSE Graph schema stores `VertexLabel` - ID, name, Time-To-Live (TTL).

**Edge**

Extends `Element` and maintains an incoming and outgoing vertex as well as a collection of properties and an edge type. DSE Graph schema stores `EdgeLabel` - ID, name, TTL, multiplicity (multi, simple), unidirected, visible, sort-key.

**Property**
A string key associated with a value. DSE Graph schema stores PropertyKey - ID, name, TTL, datatype, cardinality (single, list).

**VertexProperty**
A string key associated with a value as well as a collection of metadata properties (vertices only).

The primary components of the TinkerPop process API are:

**TraversalSource**
A generator of traversals for a particular graph, domain specific language (DSL), and execution engine.

**Traversal<\textit{S,E}>**
A functional data flow process transforming objects of type \textit{s} into object of type \textit{e}.

**GraphTraversal**
A traversal domain-specific language (DSL) that is oriented towards the semantics of the raw graph (i.e. vertices, edges, etc.).

**GraphComputer**
A system that processes the graph in parallel and potentially, distributed over a multi-machine cluster.

**VertexProgram**
Code executed at all vertices in a logically parallel manner with intercommunication via message passing.

**MapReduce**
Computations that analyzes all vertices in the graph in parallel and yields a single reduced result.

A key feature of TinkerPop is Gremlin, a graph traversal language and virtual machine. TinkerPop and Gremlin are to graph databases what JDBC and SQL are to relational databases. Gremlin variants are available for many languages: Java, Groovy, Python, and others.

**General steps, step-modulators, and predicates**

Apache TinkerPop has five general step types:

- **map(Traversal<\textit{S,E}> or map(Function<Traverser<\textit{S}>, \textit{E}>)**
  Maps the traverser to some object of type \textit{e} for the next step to process.

- **flatMap(Traversal<\textit{S,E}> or flatMap(Function<Traverser<\textit{S}>, Iterator<\textit{E}>})**
  Maps the traverser to an iterator of \textit{e} objects that are streamed to the next step.

- **filter(Traversal<\textit{?}, ?> or filter(Predicate<Traverser<\textit{S}>>)**
  Maps the traverser to either true or false, and where false will not pass the traverser to the next step.

- **sideEffect(Traversal<\textit{S}, \textit{S}>) or sideEffect(Consumer<Traverser<\textit{S}>>)**
  Performs some operation on the traverser and passes it to the next step.

- **branch(Traversal<\textit{S}, \textit{M}>) or branch(Function<Traverser<\textit{S}>, \textit{M}>)**
  Splits the traverser to all the traversals indexed by the \textit{M} token.
All other steps in the reference can be categorized as one of these steps or a variant, such as a terminal step that completes a traversal. In addition, steps have step-modulators, a helper step that assists a step:

**as()**
Provide a label to the step that can later be accessed by steps and data structures that make use of such labels.

**by()**
If a step is able to accept traversals, functions, or comparators, then by() is the means by which they are added.

**emit()**
If emit() is placed after repeat(), it is evaluated on the traversers leaving the repeat-traversal. If emit() is placed before repeat(), it is evaluated on the traversers prior to entering the repeat-traversal.

**from()**
Adds a string or traversal to a traversal to point the traversal FROM the next supplied step.

**option()**
Provide a option to a branch() or choose() step.

**to()**
Adds a string or traversal to a traversal to point the traversal TO the next supplied step.

**until()**
If until() comes after repeat() it is do/while looping. If until() comes before repeat() it is while/do looping.

Within steps, predicates are used to determine relationships between data:

**eq(object)**
Check if an incoming object is equal to the provided object.

**neq(object)**
Check if an incoming object is not equal to the provided object.

**lt(number)**
Check if an incoming number is less than the provided number.

**lte(number)**
Check if an incoming number is less than or equal the provided number.

**gt(number)**
Check if an incoming number is greater than the provided number.

**gte(number)**
Check if an incoming number is greater than or equal the provided number.

**between(number, number)**
Check if an incoming number is greater than or equal to the first provided number and less than the second provided number.

**within(objects...)**
Check if an incoming object is within an array of provided objects.
without(objects...)
Check if an incoming object is not within an array of provided objects.

**TinkerPop Predicates**

**eq**

**Synopsis**

```text
eq(object)
```

**Table 25: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `eq()` predicate answers the question: Is the incoming object equal to the provided object?

**Examples**

Find which recipes are main entrees:

```text
g.V().hasLabel('recipe').has('cuisine', eq('main entree')).values('name')
```

**neq**

**Synopsis**

```text
neq(object)
```
Table 26: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `neq()` predicate answers the question: Is the incoming object not equal to the provided object?

Examples

Find who the coauthors are of each book:

```java
g.V().hasLabel('person').as('oneAuthor').out('authored').in('authored').
  where(neq('oneAuthor')).as('anotherAuthor').
  dedup().
  select('oneAuthor','anotherAuthor').
  by('name')
```

Synopsis

```
lt(integer_value)
```

Table 27: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or. A vertical bar (</td>
</tr>
</tbody>
</table>

| Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required. |

Description

The `lt()` predicate answers the question: Is the incoming number less than the provided number?

Examples

Find all meal items with less than 800 calories:

```javascript
g.V().has('meal_item', 'calories', lt(800)).valueMap('name', 'calories')
```

**lte**

Synopsis

`lte(integer_value)`

Table 28: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ( ).</td>
</tr>
</tbody>
</table>

**Italics**

Variable value. Replace with a user-defined value.

| [ ] | Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets. |

| {} | Group. Braces ( {}) identify a group to choose from. Do not type the braces. |

| | Or. A vertical bar ( | ) separates alternative elements. Type any one of the elements. Do not type the vertical bar. |

| ... | Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required. |

Description

The `lte()` predicate answers the question: Is the incoming number less than or equal to the provided number?
Using DataStax Enterprise advanced functionality

Examples

Find all meal items with less than or equal to 1230 calories:

\[
g.V().has('meal_item', 'calories', lte(1230)).valueMap('name', 'calories')
\]

**gt**

Synopsis

\[
gt(integer_value)
\]

Table 29: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ( ).</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The **gt()** predicate answers the question: Is the incoming number greater than the provided number?

Examples

Find all recipes reviewed with a rating of greater than 3 stars:

\[
g.E().hasLabel('reviewed').has('stars', gt(3)).valueMap()
\]

**gte**

Synopsis

\[
gte(integer_value)
\]
Table 30: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ( ).</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `gte()` predicate answers the question: Is the incoming number greater than or equal to the provided number?

Examples

Find all recipes reviewed with a rating of greater than or equal to 3 stars:

```java
g.E().hasLabel('reviewed').has('stars', gte(3)).valueMap()
```

inside

Synopsis

```java
inside(integer_value, integer_value)
```

Table 31: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ( ).</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `inside()` predicate answers the question: Is the incoming number greater than the first provided number and less than the second?

**Examples**

Find all books published between 1960 and 1969 (from 1961 to 1968):

```java
g.V().hasLabel('book').has('publishYear', inside(1960,1969)).values('name', 'publishYear')
```

**outside**

**Synopsis**

`outside(integer_value, integer_value)`

**Table 32: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ( ).</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `outside()` predicate answers the question: Is the incoming number less than the first provided number or greater than the second?

**Examples**

Find all books published outside of 1960 and 1969 (before 1960 or after 1969):
Using DataStax Enterprise advanced functionality

```plaintext
    g.V().hasLabel('book').has('publishYear', outside(1960,1969)).values('name', 'publishYear')
```

**between**

**Synopsis**

```plaintext
    between(integer_value, integer_value)
```

**Table 33: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `between()` predicate answers the question: Is the incoming number greater than or equal to the first provided number and less than the second?

**Examples**


```plaintext
    g.V().hasLabel('book').has('publishYear', between(1960,1969)).values('name', 'publishYear')
```

**within**

**Synopsis**

```plaintext
    within(object, ...)
```

**Table 34: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `within()` predicate answers the question: Is the incoming object in the array of provided objects?

Examples

Get the two people who have the specified user-defined vertex ids:

```g.V().has(id, within("{~label=person, personId=1}", "{~label=person, personId=2"]))```

without

Synopsis

`without(object, ...)`

Table 35: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `without()` predicate answers the question: Is the incoming object not in the array of provided objects?

**Examples**

Get all the vertices (including recipes, books, etc.) that do not have the specified user-defined vertex ids:

```g.V().has(id, without("(~label=person, personId=1)", "(~label=person, personId=2")")```

**Table 36: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**TinkerPop Step-modulators**

**as**

**Synopsis**

`as('variable_name')`

**Description**

The `as()` step is a step modulator (page 668), a helper step for another traversal step.
Examples

Label all returned person vertices as PERSON, and all created edges as RECIPE, and then select() both the vertices and edges using the assigned variable names:

```java
g.V().hasLabel('person').as('PERSON').
out('created').as('RECIPE').
select('PERSON','RECIPE').
by('name')
```

**Synopsis**

```
by([ { property | traversal } ])
```

### Table 37: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The by() step is a *step modulator (page 668)* that can be used to modify sort order from a previous step.

**Examples**

```java
g.V().group().by(inE().count()).by(count())
```

**emit**

**Synopsis**

```
emit( [ 'predicate' | 'traversal' ])
```
Table 38: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ( ).</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `emit()` step is a step modulator (page 668), a helper step for another traversal step. Its main use is to emit either incoming traversers before a `repeat()` (page 737) step, or emit outgoing traversers after a `repeat()` step. The emission sends a copy of the current objects to the next step in the query. A predicate (page 668) or traversal can be used in an `emit()` step to cause the emission only if the predicate or traversal is true.

Examples

```
g.V(1).emit().repeat(out()).times(2).path()  
==>[v[1]]  
==>[v[1],v[3]]  
==>[v[1],v[2]]  
==>[v[1],v[4]]  
==>[v[1],v[4],v[5]]  
==>[v[1],v[4],v[3]]  
gremlin> g.V(1).repeat(out()).times(2).emit().path()  
==>[v[1],v[3]]  
==>[v[1],v[2]]  
==>[v[1],v[4]]  
==>[v[1],v[4],v[5]]  
==>[v[1],v[4],v[3]]
```

from

Synopsis

```
from('vertex_designator')
```
### Table 39: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The `from()` step is a step modulator (page 668), a helper step for another traversal step. Its main use is to designate the outgoing vertex for an `addE()` (page 693) step. Generally, a `to()` (page 682) step is paired with a `from()` step.

### Examples

```java
g.V().has('person', 'name', 'Jim Walsh').as('jim').
  V().has('person', 'name', 'Sharon Smith').as('sharon').
  addE('knows').from('jim').to('sharon').property('since', 1980)
```

### option

#### Synopsis

```java
option(value, value_returned)
```

### Table 40: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Or. A vertical bar (</td>
<td>) separates alternative elements. Type any one of the elements. Do not type the vertical bar.</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `option()` step modulator is used in conjunction with `branch()` or `choose()`, and specifies the returned values based on values found at that point in a traversal.

**Examples**

Find all people and list whether they are female, male, or unknown:

```java
g.V().hasLabel('person').
.project('name','gender').
    by('name').
    by(choose(values('gender')).
        option('F', constant('female')).
        option('M', constant('male')).
        option(none, constant('unknown')))
```

**times**

**Synopsis**

`times( integer )`

---

**Table 41: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Description

The `times()` step is a step modulator (page 668), a helper step for another traversal step. Its main use is to repeat a `repeat()` (page 737) step for the specified number of times.

Examples

An example that repeats through all the outgoing `knows` adjacent vertices of `John Doe` three times:

```sql
(g.V().hasLabel('person','name','John Doe').
  repeat(out('knows')).times(3)
)
```

to

Synopsis

```sql
to('variable_name')
```

Table 42: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `to()` step is a step modulator (page 668), a helper step for another traversal step. Its main use is to designate the incoming vertex for an `addE()` (page 693) step. Generally, a `to()` step is paired with a `from()` (page 679) step.

Examples

```sql
// create a new user
//g.addV('person').property('personId', 26).property('name','Jim Walsh').property('gender', 'M')
```
// use a mid-traversal V() step to create a new edge between two people

```java
    g.V().has('person', 'name', 'Jim Walsh').as('jim').
        V().has('person', 'name', 'Sharon Smith').as('sharon').
        addE('knows').from('jim').to('sharon').property('since', 1980)
```

**Synopsis**

```
    until( [ 'predicate' | 'traversal' ])
```

**Table 43: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `until()` step is a step modulator (page 668), a helper step for another traversal step. Its main use is to turn a `repeat()` (page 737) step into a do-while loop (if used after the `repeat()` step) or a while-do loop (if used before the `repeat()` step). A `predicate` (page 668) or traversal can be used in an `until()` step to cause the loop to complete only if the predicate or traversal is true.

**Examples**

An example that repeats through all the outgoing `knows` adjacent vertices of *John Doe* until the simple paths are exhausted:

```java
    g.V().hasLabel('person','name','John Doe').
        repeat(out('knows').dedup()).
        aggregate('x').
            by(project('person','level').
                by('name').
                by('name').
                by(loops())
            )
    ).
```

DSE 6.7 Developer Guide (Latest version)  Page 683
Using DataStax Enterprise advanced functionality

```java
until(__.not(out('knows').simplePath())).cap('x').next()
```

**TinkerPop Vertex Steps**

All the vertex steps are flatMap (page 667) steps.

### out

**Synopsis**

```javascript
out( [edgeLabel] )
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `out()` step moves the traversal to the outgoing adjacent vertices, given the edge labels. Specifying no edge label will traverse all incident edges.

**Examples**

Get all outgoing adjacent vertices for all the vertices in the graph:

```sql
g.V().out().valueMap()
```

Get all outgoing adjacent vertices for all the vertices with incident edges `knows`:
Using DataStax Enterprise advanced functionality

```java
G.V().out('knows').valueMap()
```

### in

#### Synopsis

```java
in( [edgeLabel] )
```

#### Table 45: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ( ).</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

#### Description

The **in()** step moves the traversal to the incoming adjacent vertices, given the edge labels. Specifying no edge label will traverse all incident edges.

#### Examples

Get all incoming adjacent vertices for all the vertices in the graph:

```java
G.V().in().valueMap()
```

Get all incoming adjacent vertices for all the vertices with incident edges **knows**:

```java
G.V().in('knows').valueMap()
```

### both

#### Synopsis

```java
both( [edgeLabel] )
```
Table 46: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `both()` step moves the traversal to both the outgoing and the incoming adjacent vertices, given the edge labels. Specifying no edge label will traverse all incident edges.

Examples

Get all outgoing and incoming adjacent vertices for all the vertices in the graph:

```plaintext
g.V().both().valueMap()
```

Get all outgoing and incoming adjacent vertices for all the vertices with incident edges `knows`:

```plaintext
g.V().both('knows').valueMap()
```

**outE**

Synopsis

```plaintext
outE( [edgeLabel] )
```

Table 47: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>Group. Braces ({} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `outE()` step moves the traversal to the outgoing incident edges, given the edge labels. Specifying no edge label will traverse all incident edges.

Examples

Get all outgoing incident edges for all the vertices in the graph:

```
g.V().outE().valueMap()
```

Get all outgoing incident edges for all the vertices with incident edges `knows`:

```
g.V().outE('knows').valueMap()
```

**inE**

Synopsis

```
inE( [edgeLabel] )
```

**Table 48: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes {}.</td>
</tr>
<tr>
<td><em>Italicics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Description

The `inE()` step moves the traversal to the incoming incident edges, given the edge labels. Specifying no edge label will traverse all incident edges.

Examples

Get all incoming incident edges for all the vertices in the graph:

```cql
g.V().inE().valueMap()
```

Get all incoming incident edges for all the vertices with incident edges `knows`:

```cql
g.V().inE('knows').valueMap()
```

**bothE**

Synopsis

```cql
bothE( [edgeLabel] )
```

### Table 49: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td><code>[]</code></td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td><code>{}</code></td>
<td>Group. Braces ( {}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `bothE()` step moves the traversal to both the outgoing and the incoming incident edges, given the edge labels. Specifying no edge label will traverse all incident edges.

Examples

Get all outgoing and incoming incident edges for all the vertices in the graph:

```cql
g.V().bothE().valueMap()
```
Get all outgoing and incoming incident edges for all the vertices with incident edges knows:

```
g.V().bothE('knows').valueMap()
```

### outV

**Synopsis**

`outV()`

### Table 50: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `outV()` step moves the traversal to the outgoing vertices.

**Examples**

Get all outgoing adjacent vertices for all the edges in the graph:

```
g.E().outV().valueMap()
```

Get all outgoing incident edges for all vertices, then move to the incoming vertices that have those incident edges:

```
g.V().outE('knows').inV().valueMap()
```

Get all outgoing incident edges for all vertices, then move to the incoming vertices that have those incident edges and a vertex label `person`:
Using DataStax Enterprise advanced functionality

```java
g.V().outE('knows').inV().hasLabel('person')
```

**inV**

**Synopsis**

```java
inV()
```

**Table 51: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ( ).</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `inV()` step moves the traversal to the incoming vertices.

**Examples**

Move to all incoming adjacent vertices for all the edges in the graph:

```java
g.E().inV().valueMap()
```

Get all incoming incident edges for all vertices, then move to the incoming vertices that have those incident edges:

```java
g.V().inE('knows').inV().valueMap()
```

Get all incoming incident edges for all vertices, then move to the incoming vertices that have those incident edges and a vertex label person:
bothV

Synopsis

bothV()

Table 52: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The bothV() step moves the traversal to both the outgoing and the incoming vertices.

Examples

Move to all adjacent vertices for all the edges in the graph:

```java
g.E().bothV().valueMap()
```

Get all incoming incident edges for all vertices, then move to the all vertices that have those incident edges:

```java
g.V().inE('knows').bothV().valueMap()
```

Get all incoming incident edges for all vertices, then move to the all vertices that have those incident edges and a vertex label person:
Using DataStax Enterprise advanced functionality

```java
G V().inE('knows').bothV().hasLabel('person')
```

### otherV

#### Synopsis

otherV()

---

### Table 53: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

---

### Description

The otherV() step moves the traversal to the vertex that was not the vertex that was moved from.

### Examples

Get all the recipe vertices, then traverse to the edges labeled with the edge labels specified, and finally move to the vertices other than the recipe vertices where the traversal started:

```java
G V().hasLabel('recipe').bothE('reviewed','created','includedIn').otherV()
```

### addV

#### Synopsis

```java
G V().hasLabel('vertexLabel')

  .[ property ('property_key') ... ]
```
Using DataStax Enterprise advanced functionality

### Table 54: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The `addV()` step is a map (page 667)/sideEffect (page 667). A vertex is added from a traversal `g` using `addV`. A previously created vertex label (page 659) must be specified. Property key-value pairs may be optionally specified.

### Examples

Create a vertex with a **vertex label** `person` with the **properties** `personId`, `name`, and `gender`.

```javascript
g.addV('person').
  .property('personId', 25).
  .property('name', 'Stephen Smith').
  .property('gender', 'M')
```

### addE

**Synopsis**

```javascript
g.V(vertex)
  .addE('edgeLabel')
  .to(vertex)
```

### Table 55: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

## Syntax conventions

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>Group. Braces ({} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

## Description

The `addE()` step is a map (page 667)/sideEffect (page 667). An edge is added from a traversal `g` using `addE` between two existing vertices. A previously created edge label (page 652) must be specified.

## Examples

Create an edge with an `edge label` `knows` between two vertices, `stephenSmith` and `johnDoe`. The first two lines assign the already existing vertices to variable names for use in the `addE()` step.

```java
// Get two users to join with an edge
johnDoe = g.V().has('name','John Doe').next()
stephenSmith = g.V().has('name','Stephen Smith').next()
// Create the edge between Stephen and John
g.V(stephenSmith).addE('knows').to(johnDoe)
```

## property

### Synopsis

```
g.addV('vertexLabel')
  .[ property ('property_key') ... ]
```

### Table 56: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

Description

The `property()` step is a sideEffect (page 667). When a vertex is added from a traversal `g` using `addV`, properties can be added with `property()`. For a previously created vertex, properties can be added. A previously created vertex label (page 659) must be specified. Property key-value pairs are specified.

Examples

Create a vertex with a vertex label `person` with the properties `personId` with value 17 and `name` with value `Jamie Oliver`:

```java
g.addV('person').
  property('personId', 17).
  property('name', 'Jamie Oliver').next()
```

Add the properties `gender` with value `M` and `nickname` with value `jimmy` to a person vertex previously created:

```java
g.V().has('person', 'name', 'Jamie Oliver').
  property('gender', 'M').
  property('nickname', 'jimmy')
```

Add the property `withSuppliedId` with value `341f9950-997c-11e7-9579-7f50358d3f8d` to a person vertex:

```java
// user-supplied property ID
g.addV('person').
  property('withSuppliedId', 'propValue', T.id, UUID.fromString('341f9950-997c-11e7-9579-7f50358d3f8d'))
```

mid-traversal V()

Synopsis

```java
V().has('vertexLabel', 'propertyKey', 'propertyValue')
```

Table 57: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
</table>
| Lowercase and uppercase | Literal keyword. Includes ().
| Italics | Variable value. Replace with a user-defined value. |
| [] | Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets. |
Using DataStax Enterprise advanced functionality

### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{}</code></td>
<td>Group. Braces ( <code>{}</code> ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Repeatable. An ellipsis ( <code>...</code> ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

A mid-traversal V() step can be used to fetch an additional vertex in the middle of a graph traversal, to use for some additional operation.

### Examples

Use a mid-traversal V() to fetch the person named Sharon Smith, so that an edge can be created between Jim Walsh and Sharon Smith:

```java
// graph step V() in mid-traversal
// create a new user to use in the addE() step named Jim Walsh
//g.addV('person').property('personId', 26).property('name','Jim Walsh').property('gender', 'M')

// use a mid-traversal V() step to create a new edge between two people
//g.V().has('person', 'name', 'Jim Walsh').as('jim').
//    V().has('person', 'name', 'Sharon Smith').as('sharon').
//    addE('knows').from('jim').to('sharon').
//    property('since', 1980)
```

### aggregate

**Synopsis**

`aggregate('variable_name')`

### Table 58: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lowercase and uppercase</strong></td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td><code>[ ]</code></td>
<td>Optional. Square brackets ( <code>[ ]</code> ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td><code>{ }</code></td>
<td>Group. Braces ( <code>{ }</code> ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The `aggregate()` step is a sideEffect (page 667). A vertex is added from a traversal `g` using `addV`. A previously created vertex label (page 659) must be specified.

### Examples

Return a collection of all the people that John Doe knows as an aggregate:

```
g.V().has('person', 'name', 'John Doe').
  out('knows').aggregate('x')
```

Return all the friends of John Doe’s friends, including those that are friends of John Doe:

```
g.V().has('person', 'name', 'John Doe').
  out('knows').aggregate('x').
  in('knows').out('knows').valueMap('name', 'gender')
```

Find all the friends of John Doe’s friends that are not friends of John Doe:

```
g.V().has('person', 'name', 'John Doe').
  out('knows').aggregate('x').
  in('knows').out('knows').
  where(without('x'))
```

Note the use of the `aggregate('x')` in the later step `where(without('x'))` that is used to exclude John Doe’s friends using the aggregate assigned to the variable `x`.

### and

### Synopsis

```
and(traversal, traversal, ...)  
```

### Table 59: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `property()` step is a filter (page 667). The `and()` step can take an arbitrary number of traversals to filter the returned objects using a boolean AND function. The related boolean OR function can be done with the `or()` (page 729) step.

Examples

Find all the person vertices that have created recipes AND authored books:

```java
// Java example
Graph g = DataStaxGraphDatabaseFactory.build(...);
List<Person> personVertices = g.V().hasLabel('person').
  and(outE('created'), outE('authored')).values('name');
```

The `barrier()` step is a bulk optimizing step that can create a single traverser for a repeated step.

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Examples

Repeat this traversal with the `barrier()` step and then without, to see the effect the barrier step has on the operation. The basic query starts at a particular book, then traverses to the `includedIn` edges that point to that book to

```java
g.V().has('book', 'bookId', 1004).
   in('includedIn').
   in('includedIn').
   groupCount('x').
   by('name').
   barrier().
   project('a', 'b').
   by('name').
   by(select('x'))
```

### branch

**Synopsis**

```java
branch(traversal)
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `branch` step is a general step ([page 667](#)).

**Examples**

Find all ingredients in a particular fridge (based on the fridgeSensor id), count, and print an message depending on the count returned.

```java
g.V().hasLabel('ingredient').as('ingredient').
   branch(inE('contains').filter(outV().has('stateId', 31).has('cityId', 200)).count()).
```
option(0L, constant('unavailable')).
option(1L, constant('time to reorder')).
option(none, constant('available')).as('status').
select('ingredient','status').
by('name').
by()

cap

Synopsis

cap( variable_name', ... )

Table 62: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ( ).</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The cap() step is a barrier step that iterates the traversal up to itself and returns the referenced object by the provided key. If multiple keys are provided, then a map of objects is returned.

Examples

Get the count of each vertex label and return as a map of vertex label : count value key-value pairs:

g.V().groupCount('a').by(label).cap('a')

choose

Synopsis

choose(traversal, choice_1, choice_2, choice_3)
### Table 63: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The `choose()` step is a branch step *(page 667)* that can be used to route a traverser to a particular traversal branch, similar to if-then-else logic.

### Examples

```plaintext
g.V().hasLabel('store').as('store'),
  V().hasLabel('ingredient').as('ingredient'),
  choose(where(__.in('isStockedWith').as('store')),
    constant('Y'), constant('N')).as('in stock'),
  select('store','ingredient','in stock'),
  by('name'),
  by('name'),
  by()
```

### `coalesce`

#### Synopsis

`coalesce(traversal, traversal_1, ...)`

### Table 64: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>Group. Braces ({} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The `coalesce()` step evaluates the provided traversals in order and returns the first traversal that emits at least one element.

### Examples

Create a list of each person and their mean average of stars given for recipe reviews, using `coalesce()` to either print the mean or print a constant zero value if no reviews are found:

```
"g.V().hasLabel('person').as('person','starCount').
    select('person','starCount').
    by('name').
    by(coalesce(outE('reviewed').values('stars'),constant(0)).mean()).
    order().by(select('starCount'), decr)"
```

### Synopsis

Table 65: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Description

The `coin()` step is a filter step (page 667) that can be used to get a random coin toss with the inserted odds between 0.0 and 1.0.

Examples

Find a random set of ingredients that, for instance, must be used in a cooking contest:

```velocity
g.V().hasLabel('ingredient').coin(0.1)
```

**constant**

Synopsis

```
constant(value)
```

Table 66: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()．</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value．</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets．</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces．</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required．</td>
</tr>
</tbody>
</table>

Description

The `constant()` step is a map step (page 667) that is often useful with `choose()` (page 700) or `coalesce()` (page 701) steps．

Examples

Use `constant(0)` to designate zero as the value for any returns that are not a number, such as NaN (not a number):

```velocity
g.V().hasLabel('person').as('person','starCount').
   select('person','starCount').
   by('name')
   by(coalesce(outE('reviewed').values('stars'),constant(0)).mean())
```

Using DataStax Enterprise advanced functionality

```java
order().by(select('starCount'), decr)
```

count

Synopsis

count()

Table 67: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `count()` step is a map step (page 667) that counts whatever precedes it.

Examples

Count the number of reviews for each recipe, then print the recipe name, the review count, and the review average stars given

```java

g.V().hasLabel('recipe').as('recipe','numberOfReviews','meanRating').
    select('recipe','numberOfReviews','meanRating').
    by('name').
    by(inE('reviewed').count()).
    by(inE('reviewed').values('stars').mean())
```

cyclicPath

Synopsis

Table 68: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The cyclicPath() step is a filter step ([page 667](#)) that allows a traversal to filter out any repeats as the traversal proceeds.

### Examples

Find all people where a person both created and reviewed a recipe:

```cypher
g.V().hasLabel('person').as('a').
  out('created').as('b').
  filter(_.in('reviewed').cyclicPath()).
  select('a','b').
  by('name')
```

**dedup**

### Synopsis

```cypher
dedup()
```

### Table 69: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `dedup()` step is a filter step (page 667) that can be used to eliminate duplication of returned objects in a traversal query.

**Examples**

Find all the ingredients in Julia Child's recipes, and deduplicate the returned list:

```sql
g.V().has('person','name','Julia Child').
  out('created').
  in('includedIn').dedup().
  values('name')
```

**drop**

**Synopsis**

`drop()`

**Table 70: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `drop()` step is a map (page 667)/sideEffect (page 667). Appending `drop()` to any traversal will drop the preceding objects from the graph.

**Examples**

Drop all outgoing edges from all vertices:
Using DataStax Enterprise advanced functionality

```
g.V().outE().drop()
```
Drop the property key *name* from all vertices; the values stored for the property will also be dropped from the graph.

```
g.V().properties('name').drop()
```
Drop all vertices from a graph:

```
g.V().drop()
```

**explain**

**Synopsis**

```
explain()
```

**Table 71: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lowercase and uppercase</strong></td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `explain()` step is a terminal step that will return a traversal explanation detailing how the traversal is compiled, and the TinkerPop traversal strategies that are used.

**Examples**

```
g.V().not(hasLabel('fridgeSensor').or().hasLabel('meal')).
group().by(label).by('name').explain()
```

The return of `explain()` for the example:

<table>
<thead>
<tr>
<th>Traversal Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Traversal</td>
</tr>
<tr>
<td>[GraphStep(vertex, [])], NotStep([HasStep([~label.eq(fridgeSensor)])),</td>
</tr>
</tbody>
</table>
OrStep, HasStep([~label.eq(meal)]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])

ConnectiveStrategy                    [D]   [GraphStep(vertex, []), NotStep([OrStep([HasStep([~label.eq(fridgeSensor)])], [HasStep([~label.eq(meal)])])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

RepeatUnrollStrategy                  [O]   [GraphStep(vertex, []), NotStep([OrStep([HasStep([~label.eq(fridgeSensor)])], [HasStep([~label.eq(meal)])])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

MatchPredicateStrategy                [O]   [GraphStep(vertex, []), NotStep([OrStep([HasStep([~label.eq(fridgeSensor)])], [HasStep([~label.eq(meal)])])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

PathRetractionStrategy                [O]   [GraphStep(vertex, []), NotStep([OrStep([HasStep([~label.eq(fridgeSensor)])], [HasStep([~label.eq(meal)])])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

IncidentToAdjacentStrategy            [O]   [GraphStep(vertex, []), NotStep([OrStep([HasStep([~label.or(eq(fridgeSensor), eq(meal))])], [HasStep([~label.or(eq(fridgeSensor), eq(meal))])])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

FilterRankingStrategy                 [O]   [GraphStep(vertex, []), NotStep([OrStep([HasStep([~label.or(eq(fridgeSensor), eq(meal))])], [HasStep([~label.or(eq(fridgeSensor), eq(meal))])])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

InlineFilterStrategy                  [O]   [GraphStep(vertex, []), NotStep([OrStep([HasStep([~label.or(eq(fridgeSensor), eq(meal))])], [HasStep([~label.or(eq(fridgeSensor), eq(meal))])])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

AdjacentVertexFilterOptimizerStrategy [P]   [DsegGraphStep(vertex, [], true, Unordered), NotStep([DsegHasStep([~label.or(eq(fridgeSensor), eq(meal))])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

LazyBarrierStrategy                   [O]   [GraphStep(vertex, []), NotStep([HasStep([~label.or(eq(fridgeSensor), eq(meal))])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

DseIncidentToAdjacentStrategy         [O]   [GraphStep(vertex, []), NotStep([HasStep([~label.or(eq(fridgeSensor), eq(meal))])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

HasStepStrategy                       [P]   [GraphStep(vertex, []), NotStep([DsegHasStep([~label.or(eq(fridgeSensor), eq(meal))])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

QueryStrategy                         [P]   [DsegGraphStep(vertex, [], true, Unordered), NotStep([DsegHasStep([~label.or(eq(fridgeSensor), eq(meal))])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]

AdjacentVertexFilterOptimizerStrategy [P]   [DsegGraphStep(vertex, [], true, Unordered), NotStep([DsegHasStep([~label.or(eq(fridgeSensor), eq(meal))])]), GroupStep(label, [TraversalMapStep(value(name)), FoldStep])]
The codes for various steps:

[D]ecoration
There is an application-level feature that can be embedded into the traversal logic.

[O]ptimization
There is a more efficient way to express the traversal at the TinkerPop level.

[P]rovider optimization
There is a more efficient way to express the traversal at the graph system, language, or driver level.

[F]inalization
There are some final adjustments, cleanups, or analyses required before executing the traversal.

[V]erification
There are certain traversals that are not legal for the application or traversal engine.

**fill**

Synopsis

```java
fill(collection_name)
```

### Table 72: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `fill()` step is a terminal step that will put all the results of a traversal into the named collection, such as an array. The collection is returned when the filling is complete.

**Examples**

```
results = []
g.V().out('authored').fill(results)
```

**filter**

**Synopsis**

`filter(traversal)`

**Table 73: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italicics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Description

The `filter()` step is a general filter step (page 667). This step is used to eliminate some of the returned objects at a particular point in a traversal based on some criteria.

Examples

Find the books that include an edge from a recipe:

```g.V().hasLabel('book').filter(__.in('includedIn').hasLabel('recipe'))```

**flatMap**

Synopsis

`flatMap(traversal)`

Table 74: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `flatMap()` step is a general `flatMap` step (page 667). This step creates a map of the traversal objects and streams the map to the next traversal step.

Examples

Return a map of the recipe properties included in the cookbooks in the graph, but limit the return to 3 recipes:
Using DataStax Enterprise advanced functionality

```
g.V().hasLabel('book').flatMap(__.in('includedIn').hasLabel('recipe').limit(3))
```

**fold**

**Synopsis**

```
fold()
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The fold() step is a map step (page 667). It aggregates all the returned objects into a list, then emits the list into the next traversal step.

**Examples**

Get back a list of all the persons that each person knows:

```
g.V().hasLabel('person').out('knows').values('name').fold()
```

**group**

**Synopsis**

```
group()
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[ ]</code></td>
<td>Optional. Square brackets <code>[]</code> surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td><code>{ }</code></td>
<td>Group. Braces <code>{ }</code> identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Repeatable. An ellipsis <code>...</code> indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The `group()` step is a map (page 667)/sideEffect (page 667). The step organizes the objects according to some function of the object, and reduces the returned objects to a list.

### Examples

List recipes by name, grouping by cuisine:

```java
g.V().hasLabel('recipe').
group().
  by('cuisine').
  by('name')
```

#### groupCount

### Synopsis

`groupCount()`

### Table 77: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td><code>[ ]</code></td>
<td>Optional. Square brackets <code>[]</code> surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td><code>{ }</code></td>
<td>Group. Braces <code>{ }</code> identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Repeatable. An ellipsis <code>...</code> indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

Description

The `groupCount()` step is a map (page 667)/sideEffect (page 667). The step determines the number of objects specified.

Examples

Print the number of ingredients per recipe:

```java
g.V().hasLabel('recipe').
    groupCount().
    by(inE('includedIn').count())
```

`has`

Synopsis

```java
has( { 'vertexLabel' | 'edgeLabel' }, [ 'propertyKey_name', 'propertyKey_value' ] ... )
```

<table>
<thead>
<tr>
<th>Table 78: Legend</th>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
<td></td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
<td></td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
<td></td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
<td>) separates alternative elements. Type any one of the elements. Do not type the vertical bar.</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
<td></td>
</tr>
</tbody>
</table>

Description

The `has()` step is a filter (page 667) step. It is the most common step used for graph traversals, since this step narrows the query to find particular vertices or edges with certain property values.

Examples

Find the meal item that is called a taco:
g.V().has('meal_item', 'name', 'taco')

**hasNext**

Synopsis

`hasNext()`

<table>
<thead>
<tr>
<th>Table 79: Legend</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax conventions</strong></td>
</tr>
<tr>
<td>Lowercase and uppercase</td>
</tr>
<tr>
<td>italic</td>
</tr>
<tr>
<td>[]</td>
</tr>
<tr>
<td>{}</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

**Description**

The `hasNext()` step is a terminal step. It determines whether or not there are available results from a traversal, returning a Boolean value of true or false.

**Examples**

```java
// determines whether there are available results
g.V().hasLabel('book').hasNext()
```

**id**

Synopsis

`id()`

<table>
<thead>
<tr>
<th>Table 80: Legend</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax conventions</strong></td>
</tr>
<tr>
<td>Lowercase and uppercase</td>
</tr>
<tr>
<td>italic</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `id()` step is a map (page 667) step that retrieves the id of a graph object.

**Examples**

Fetch the id of the person named James Beard:

```plaintext
g.V().has('person', 'name', 'James Beard').id()
```

Fetch the id of the edge with an edgeLabel created:

```plaintext
g.E().hasLabel('created').id()
```

**inject**

**Synopsis**

```plaintext
inject('object_name')
```

**Table 81: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Description

The `inject()` step is a side effect (page 667). It inserts objects arbitrarily into a traversal stream, but does not change the graph permanently.

Examples

```g.V().has('person', 'name', 'James Beard').out().values('name').inject('Cream Biscuits')```

is

Synopsis

| Table 82: Legend |
|------------------|-------------------|
| Syntax conventions | Description |
| Lowercase and uppercase | Literal keyword. Includes (). |
| **Italics** | Variable value. Replace with a user-defined value. |
| [] | Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets. |
| {} | Group. Braces ({} ) identify a group to choose from. Do not type the braces. |
| | Or. A vertical bar ( | ) separates alternative elements. Type any one of the elements. Do not type the vertical bar. |
| ... | Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required. |

Description

The `is()` step is a filter (page 667) step that is used to filter for scalar values.

Examples

```g.V().where(__.hasLabel('meal_item').values('calories').is(lte(1200))).values('name', 'calories')```

Note that this example could be accomplished with a `has()` step.

But this example could not be accomplished with a `has()` step:
Using DataStax Enterprise advanced functionality

```sql
g.V().hasLabel('meal').filter(out('includes').values('calories').sum().is(gte(1000)))
```

### key

**Synopsis**

#### key()

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `key()` is a map (page 667) step that extracts the property keys for a specified object.

**Examples**

Get a deduplicated list of all the vertexLabel property keys:

```sql
g.V().properties().key().dedup()
```

### label

**Synopsis**

#### label()

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The `label()` step is a map (page 667) step that extracts the specified labels.

#### Examples

Get all the vertex and edge labels in a deduplicated list:

```cql
g.V().label().dedup()
```

### limit

#### Synopsis

`limit(integer_value)`

#### Table 85: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italicics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The `limit()` step allows a limit to be set for the number of returned items. It can also be used with the `local()` step to scope the effect within a traversal.
Using DataStax Enterprise advanced functionality

Examples

Find all vertices with a vertex label *ingredient*, and limit the return to 10 items starting with 0 (zero).

```javascript
g.V().hasLabel('ingredient').limit(10)
```

**local**

Synopsis

```javascript
local(traversal)
```

**Table 86: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `local()` step is a branch (page 667) step that allows the action within the step to be applied to the element within the `local()` step.

Examples

Get a list of all people and the countries they have lived in, ordered by the year that the person started living in the country:

```javascript
g.V().hasLabel('person').as('person').
  local(properties('country').order().by('startYear', incr).limit(2)).value().as('country').
  select('person','country').
  by('name').by()
```

Here, `local()` is used to restrict the sorting by the meta-property `startYear` for each person.
Using DataStax Enterprise advanced functionality

```
g.V().has('person','personId','1').local(outE('created').has('createDate',
gte('1962-03-03')).order().by('createDate'))
```

// explain
//g.V().hasLabel('person').outE('reviewed').has('stars',
gte(1)).order().by('stars').profile()
// Sorts the reviews by stars for each user - uses the edge index
//g.V().hasLabel('person').local(outE('reviewed').has('stars',
gte(1)).order().by('stars')).profile()
// Sorts the users by name, and then their reviews by stars
//g.V().hasLabel('person').order().by('name').local(outE('reviewed').has('stars',
gte(1)).order().by('stars')).profile()

**loops**

**Synopsis**

```
loops()
```

**Table 87: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `loops()` step is a map (page 667) step that extracts the number of times the traverser has gone through the current loop.

**Examples**

```
g.V().hasLabel('person','name','John Doe').
   repeat(out('knows').dedup().
       aggregate('x').
       by(project('person','level').
           by('name').
           by(loops()))
```
Using DataStax Enterprise advanced functionality

```java
)
  until(__.not(out('knows').simplePath())).
cap('x').next()
```

**map**

Synopsis

```java
map(traversal)
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><em>Italicics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `map()` step is a **general map step (page 667)**. This step creates a map of the traverser to some object for the next step to process.

**Examples**

List all recipes and a map of their ingredients:

```java
g.V().hasLabel('recipe').as('recipe').
  map(__.in('includedIn').values('name').fold()).as('ingredients').
  select('recipe','ingredients').
  by('name').
  by()
```

**match**

Synopsis

```java
match(traversal_fragment, ...)```
Table 89: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `match()` step is a map (page 667) step provides a more declarative form of graph querying based on pattern matching.

Examples

Find the ingredients in the recipe for *Beef Bourguignon*:

```java
  g.V().match(
    __.as('INGREDIENT').hasLabel('ingredient'),
    __.as('INGREDIENT').out('includedIn').has('name','Beef Bourguignon')).
  select('INGREDIENT').by('name')
```

**math**

Synopsis

`math('math_function')`

Table 90: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

### Syntax conventions

| Or. A vertical bar ( | ) separates alternative elements. Type any one of the elements. Do not type the vertical bar. |
| Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required. |

### Description

The `math()` step enables scientific calculator functionality. The basic operations ( +, -, *, /, ^, and %) are available, as well as:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>absolute value</td>
</tr>
<tr>
<td>acos</td>
<td>arc cosine</td>
</tr>
<tr>
<td>asin</td>
<td>arc sine</td>
</tr>
<tr>
<td>atan</td>
<td>arc tangent</td>
</tr>
<tr>
<td>cbrt</td>
<td>cubic root</td>
</tr>
<tr>
<td>ceil</td>
<td>nearest upper integer</td>
</tr>
<tr>
<td>cos</td>
<td>cosine</td>
</tr>
<tr>
<td>cosh</td>
<td>hyperbolic cosine</td>
</tr>
<tr>
<td>exp</td>
<td>Euler’s number raised to the power $e^x$</td>
</tr>
<tr>
<td>floor</td>
<td>nearest lower integer</td>
</tr>
<tr>
<td>log</td>
<td>natural logarithm (base e)</td>
</tr>
<tr>
<td>log10</td>
<td>logarithm (base 10)</td>
</tr>
<tr>
<td>log2</td>
<td>logarithm (base 2)</td>
</tr>
<tr>
<td>sin</td>
<td>sine</td>
</tr>
<tr>
<td>sinh</td>
<td>hyperbolic sine</td>
</tr>
<tr>
<td>sqrt</td>
<td>square root</td>
</tr>
<tr>
<td>tan</td>
<td>tangent</td>
</tr>
<tr>
<td>tanh</td>
<td>hyperbolic tangent</td>
</tr>
<tr>
<td>signum</td>
<td>signum function</td>
</tr>
</tbody>
</table>

### Examples

Find the calories per meal item, then multiply by the number of servings for a meal:

```java
g.V().hasLabel('meal_item').as('a').
```
Using DataStax Enterprise advanced functionality

```sql
inE('includes').as('b').
  math('a*b').
  by('calories').
  by('numServ')
```

Note that two `by()` statements are used to retrieve a particular property for each math variable used, calories for `a` and numServ for `b`.

**max**

Synopsis

```sql
max()
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `max()` step is a *map* (page 667) step that will discover the largest number generated from the previous step in the traversal.

Examples

Find the maximum number of reviews written by any person:

```sql
g.V().hasLabel('person').map(outE('reviewed').count()).max()
```

**mean**

Synopsis

```sql
mean()
```
Using DataStax Enterprise advanced functionality

### Table 92: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The `mean()` step is a map ([page 667](page_667)) step that will discover the mean average of the numbers generated from the previous step in the traversal.

### Examples

**Find the mean average of the stars for reviews for each reviewer:**

```plaintext
g.V().hasLabel('person').as('person','starCount').
    select('person','starCount').
    by('name').
    by(outE('reviewed').values('stars').mean()).
order().by(select('starCount'), decr)
```

**Find the mean average of the number of ingredients for each recipe:**

```plaintext
g.V().hasLabel('recipe').
    map(inE('includedIn').count()).
    mean()
```

### Synopsis

### Table 93: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax conventions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
<td>[]</td>
</tr>
<tr>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
<td>{}</td>
</tr>
<tr>
<td>Or. A vertical bar (</td>
<td>) separates alternative elements. Type any one of the elements. Do not type the vertical bar.</td>
</tr>
<tr>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
<td>...</td>
</tr>
</tbody>
</table>

Description

The \texttt{min()} step is a \texttt{map (page 667)} step that will discover the smallest number generated from the previous step in the traversal.

Examples

Find the minimum number of recipes created by any person:

\[ \texttt{g.V().hasLabel('person').map(outE('created').count()).min()} \]

next

Synopsis

next(integer)

Table 94: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

Description

The `next()` step is a terminal step. It returns the next number of steps, based on a supplied integer value.

Examples

Return the next two results found:

```
g.V().hasLabel('store').next(2)
```

not

Synopsis

```
not(traversal)
```

Table 95: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `not()` step is a filter (page 667) step that excludes the objects specified.

Examples

If a vertex is not a fridgeSensor or a meal, group each vertex type and list by name:

```
g.V().not(hasLabel('fridgeSensor').or().hasLabel('meal')).
group().by(label).by('name')
```

optional

Synopsis

```
optional(traversal_fragment)
```
Table 96: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeateable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `optional()` step is a branch (page 667)/flatMap (page 667) step that will either return the result of the specified traversal if it yields a result, or the return of the calling element.

Examples

Returns the friends of John Doe, since he has friends:

```java
g.V().has('person', 'name', 'John Doe').optional(out('knows'))
```

Returns Julia Child, since she has no linked friends:

```java
g.V().has('person', 'name', 'Julia Child').optional(out('knows'))
```

or

Synopsis

```java
or()
```

Table 97: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `or()` step is a filter (page 667) step that ensures that at least one of the provided traversals yield a result.

Examples

Finds all vertices that are not a fridgeSensor or a meal and groups them by name:

```java
g.V().not(hasLabel('fridgeSensor').or().hasLabel('meal')).
group().by(label).by('name')
```

order

Synopsis

`order()`

Table 98: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes { }.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `order()` step is a map step that sorts returned objects given a certain criteria.
Examples

```java
g.V().hasLabel('person').values('name').order().by(decr).dedup()
```

**pageRank**

Synopsis

```java
pageRank()
```

**Table 99: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [[] ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { {} } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `pageRank()` step is a map/sideEffect step that calculates PageRank. It only runs in Analytics (OLAP) mode.

Examples

```java
g = graph.traversal().withComputer()
g.V().pageRank().by('pageRank').values('pageRank')
```

**path**

Synopsis

```java
path()
```

**Table 100: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `path()` step is a map (page 667) step that examines the history of the traversal path.

Examples

Find which recipes are included in which books that include the ingredients of beef or carrots:

```javascript
g.V().has('ingredient', 'name', within('beef','carrots')).out('includedIn').as('Recipe').out().hasLabel('book').as('Book').select('Book','Recipe').by('name').by('name').path()
```

**peerPressure**

Executes a Peer Pressure community detection algorithm over the graph.

**Synopsis**

peerPressure()
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `peerPressure()` step is a map/sideEffect step that executes a Peer Pressure community detection algorithm over the graph. It only runs in Analytics (OLAP) mode.

**Examples**

```java
g = graph.traversal().withComputer()
g.V().peerPressure().by('cluster').values('cluster')
```

**profile**

**Synopsis**

`profile()`

**Table 102: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `profile()` step is a side effect (page 667) step that profiles a traversal to determine statistical information like step runtime, counts, and percent time spent in each traverser. See Anatomy of a Query (page 579).
Examples

```
g.V().out('created').repeat(both()).times(3).hasLabel('person').profile()
```

**project**

**Synopsis**

```
project('variable_name', ...)
```

### Table 103: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `project()` step is a map (page 667) step that projects the current object into a map keyed by provided labels. It is similar to `select()` step.

**Examples**

Create a map of outgoing and incoming edge counts for a recipe:

```
g.V().has('name','Beef Bourguignon').
   project('out','in').
   by(outE().count()).
   by(inE().count())
```

**properties**

**Synopsis**

```
properties([ 'property_name', ... ])
```
Table 104: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `properties()` step retrieves the properties of the specified element.

Examples

Find the properties of *Julia Child*:

```csharp
g.V().has('person','name','Julia Child').properties()
```

Find the countries that *Julia Child* has lived in:

```csharp
g.V().has('person','name','Julia Child').properties('country')
```

propertyMap

Synopsis

Table 105: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{ }</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The `propertyMap()` step is a map (page 667) step that yields a map representation of the properties of a specified element.

### Examples

Return a map of the properties of all stores:

```
g.V().hasLabel('store').propertyMap()
```

### range

#### Synopsis

```
range([ local ], low_integer, high_integer)
```

#### Table 106: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

### Description

The `range()` step is a filter (page 667) step that allows only the specified number of objects to pass onto the next step.

### Examples

Find the first two countries listed for each person:

```
g.V().valueMap().
  select('country').
```
Using DataStax Enterprise advanced functionality

range(local, 0, 2)

Find the first two countries that each person has lived in ordered by `startYear`:

```javascript
g.V().hasLabel('person').
    local(properties('country').order().by('startYear').range(0, 2).value().fold())
```

**repeat**

**Synopsis**

```javascript
repeat(traversal)
```

**Table 107: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `repeat()` step is a branch (page 667) step and is used for looping over a traversal to some given break predicate. Three step modulators can be used with the `repeat()` step, `emit()` (page 678), `times()` (page 681), or `until()` (page 683).

**Examples**

Repeatedly gets the outgoing vertex, three times, and prints the results:

```javascript
g.V().hasLabel('fridgeSensor').repeat(out()).times(3).valueMap()
```

A more complex example that repeats through all the outgoing knows adjacent vertices of *John Doe*:

```javascript
g.V().hasLabel('person','name','John Doe').
    repeat(out('knows').dedup().
        aggregate('x').
        by(project('person','level').
            by('name').)
```
Synopsis

`sack(traversal)`

### Table 108: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `sack()` step is a map (page 667)/sideEffect (page 667) step that is used to read or write sacks, local data structures that traversers use.

Examples

```java
  g.withSack(0).V().
      has("person", "name", "John Doe").
  outE("reviewed").
      sack(sum).by("stars").
  outV().
  sack()
```

sample

Synopsis

`sample(integer_value)`
Description

The `sample()` step is useful for sampling some number of traversers previous in the traversal.

Examples

Get three samples of the property `numServ` from the `includes` edges of a `meal_item`:

```plaintext
g.V().hasLabel('meal_item').inE().sample(3).by('numServ')
```

**select**

Synopsis

```plaintext
select('variable_name', ...) 
```

Table 109: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `select()` step is a **map** ([page 667](#)) step that selects labeled steps designated with `as()` steps. This step is typically used to select particular properties from objects earlier in the traversal.

Examples

List meal items and the meals that are linked:

```java
g.V().hasLabel('meal_item').as('item').in().as('meal').
    select('item','meal').
    by('name').by('mealId')
```

### sideEffect

**Synopsis**

`sideEffect(traversal)`

**Table 111: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `sideEffect()` step is a **general filter step** ([page 667](#)). This step performs some operation on the traverser and passes the result to the next step.

**Examples**

For each person, find the number of people that person knows, and the number of people known by the person and return the results for both counts:
Using DataStax Enterprise advanced functionality

```java
g.V().hasLabel('person').
    sideEffect(outE('knows').count().store('knows')).
    sideEffect(inE('knows').count().store('known by')).
    cap('knows','known by')
```

**simplePath**

Synopsis

`simplePath()`

**Table 112: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `simplePath()` step is a filter (page 667) step. When it is important that a traverser not repeat its path through the graph, this step should be used.

Examples

Find the expanded neighborhood of all the people who know each other, without repeating any edges:

```java
g.V().has('person', 'name', 'John Doe').
    repeat(both('knows').simplePath()).
    emit().
    path().
    by('name')
    *.objects()
    *.join(" > ")
```

**Note:** This query uses Groovy notation (*.objects, *.join) in a Gremlin query, to return pretty results.

The results:
Using DataStax Enterprise advanced functionality

"John Doe > John Smith",
"John Doe > Jane Doe",
"John Doe > Betsy Jones",
"John Doe > John Smith > Jane Doe",
"John Doe > Jane Doe > Sharon Smith",
"John Doe > Jane Doe > John Smith",
"John Doe > Betsy Jones > Sharon Smith",
"John Doe > John Smith > Jane Doe",
"John Doe > Jane Doe > Sharon Smith > Betsy Jones",
"John Doe > Jane Doe > Sharon Smith > Jim Walsh",
"John Doe > Betsy Jones > Sharon Smith > Jane Doe",
"John Doe > Betsy Jones > Sharon Smith > Jim Walsh",
"John Doe > John Smith > Jane Doe > Sharon Smith > Betsy Jones",
"John Doe > John Smith > Jane Doe > Sharon Smith > Jim Walsh",
"John Doe > Betsy Jones > Sharon Smith > Jane Doe > John Smith"

skip

Synopsis

skip(integer_value)

Table 113: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The skip() step is a filter (page 667) step that is analogous to the range() (page 736) step with an upper limit of -1.

Examples

Find all the people who lived in more than one country:

g.V().hasLabel('person').
Using DataStax Enterprise advanced functionality

```java
filter(values('country').skip(1))
```

**store**

**Synopsis**

```java
store('variable_name')
```

### Table 114: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The `store()` step is a sideEffect (page 667) step that stores information for later use in the traversal.

**Examples**

Find the number of servings that any includes edge has as a value:

```java
g.E().hasLabel('includes').store('x').by('numServ').cap('x')
```

**subGraph**

**Synopsis**

```java
subGraph('subGraph_name')
```

### Table 115: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The subGraph() step is a sideEffect (page 667) step that provides a way to produce an edge-induced subgraph from virtually any traversal. This step is useful for visualization or further analysis on a smaller subset of the full graph.

Examples

Get a subgraph of all the stores and their stocked items, excluding stores without stocked items and items that are not available in any store:

```plaintext
g.E().hasLabel('isStockedWith').subgraph('subGraph').cap('subGraph').next()
```

sum

Synopsis

sum()

Table 116: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({ }) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Description

The `sum()` step is a **map (page 667)** step that sums the previously returned objects in the traversal.

Examples

Get the sum of the calories for all meal items in each meal:

```java
g.V().hasLabel('meal').local(out('includes').values('calories').sum())
```

### tail

**Synopsis**

```java
tail(integer_value)
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( { } ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `tail()` step is analogous to **limit() (page 719)** step, except that it emits the last n objects instead of the first n objects.

Examples

Get the last 10 items that are ingredients:

```java
g.V().hasLabel('ingredient').tail(10)
```

### timeLimit

**Synopsis**

```java
timeLimit(integer_value)
```
Table 118: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ( ).</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `timeLimit()` step is a filter (page 667) step that limits the traversal to a specified number of milliseconds of execution time.

Examples

Limit the traversal to 2 milliseconds of execution time:

```cql
g.V().repeat(timeLimit(2).both().groupCount('m')).times(16).cap('m').order(local).by(values,decr).next()
```

toBulkSet

Synopsis

toBulkSet()

Table 119: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ( ).</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
</tbody>
</table>
Description

The `toBulkSet()` step is a terminal step that returns all results in a weighted set.

Examples

Will return all results in a weighted set, with duplicates preserved via weighting:

```java
g.V().out('reviewed').toBulkSet()
```

**toList**

Synopsis

`toList()`

### Table 120: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (....) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `toList()` step is a terminal step that will return all the results in a list.
Examples

\[ g.V().out('reviewed').toList() \]

**toSet**

Synopsis

**toSet()**

### Table 121: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([]) surround optional command arguments.</td>
</tr>
<tr>
<td>{}</td>
<td>Do not type the square brackets.</td>
</tr>
<tr>
<td>|</td>
<td>Group. Braces ({</td>
</tr>
<tr>
<td></td>
<td>braces.</td>
</tr>
<tr>
<td>...</td>
<td>Or. A vertical bar (|) separates alternative elements. Type any one of</td>
</tr>
<tr>
<td></td>
<td>the elements. Do not type the vertical bar.</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax</td>
</tr>
<tr>
<td></td>
<td>element as often as required.</td>
</tr>
</tbody>
</table>

Description

The **toSet()** step is a terminal step that will return all the results in a set with all duplicates removed.

Examples

\[ g.V().out('reviewed').toSet() \]

**tree**

Synopsis

**tree()**

### Table 122: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
</tbody>
</table>
**Syntax conventions** | **Description**
--- | ---
[] | Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.
{} | Group. Braces ({}) identify a group to choose from. Do not type the braces.
| | Or. A vertical bar ( | ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
... | Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

**Description**

The `tree()` step is a sideEffect *(page 667)* step that returns a tree of the traversal.

**Examples**

Find John Doe’s network as a tree:

```clojure
  g.V().has('person','name','John Doe').
  repeat(out('knows').simplePath()).
  until(__.not(out('knows').simplePath())).
  tree().
  by('name')
```

**unfold**

**Synopsis**

`unfold()`

**Table 123: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Description

The `unfold()` step is a `flatMap` (page 667) step. Unfold a returned object into a linear form, such as a map into each individual value.

Examples

Find the top three countries that the most people have lived in:

```plaintext
g.V().hasLabel('person').values('country').
  groupCount().
  order(local).
  by(values, decr).
  limit(local, 3).
  select(keys).unfold()
```

union

Synopsis

```plaintext
union(traversal, ... )
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `union()` step is a `branch` (page 667) step and merges the results of an arbitrary number of traversals that are specified.

Examples

Recipes reviewed by John Smith and the people he knows:
Using DataStax Enterprise advanced functionality

```java
package "person"."name"."John Smith"."union(out('reviewed'),
  both('knows').out('reviewed')).dedup()
```

### value

**Synopsis**

value()

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lowercase and uppercase</strong></td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ({}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

**Description**

The value() step is a **map** (page 667) step that gets just the value of a property, not the key/value pair for a property, given a specified property.

**Examples**

Get the value of a properties for all person vertices:

```java
g.V().hasLabel('person').properties().value()
```

### valueMap

**Synopsis**

valueMap()

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lowercase and uppercase</strong></td>
<td>Literal keyword. Includes ()</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `valueMap()` step yields a map representation of the properties of an element.

Examples

Get a map of the properties and their values for *Jane Doe*:

```java
g.V().has('person', 'name', 'Jane Doe').valueMap()
```

Get a map of the properties and their values for the outgoing edges from *Jane Doe*:

```java
g.V().has('person', 'name', 'Jane Doe').outE().valueMap()
```

values

Synopsis

```java
values([ 'property_name', ... ])
```

Table 127: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes ().</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {}) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Description

The `values()` step is a map step that extracts the values of either all the properties or specified properties for an element.

Examples

Get all the values of the properties for a meal item of `taco`:

```java
g.V().has('meal_item', 'name', 'taco').values()
```

Get only the value of the property `macro` for a meal item of `taco`:

```java
g.V().has('meal_item', 'name', 'taco').values('macro')
```

where

Synopsis

`where()`

Table 128: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase and uppercase</td>
<td>Literal keyword. Includes () .</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>{}</td>
<td>Group. Braces ( {} ) identify a group to choose from. Do not type the braces.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>

Description

The `where()` step is a filter (page 667) step that filters the returned objects based on a predicate (page 668), a sideEffect (page 667), or a traversal.

Examples

First find all the reviewers that rated `Beef Bourguignon` 5-stars. Then find what other recipes those reviewers rated 5-stars, and count how many reviewers did by recipe, without including `Beef Bourguignon` in the count. It illustrates the use of the `where()` step with `where(neq('a'))` to exclude `Beef Bourguignon`.

```java
g.V().has('recipe', 'name', 'Beef Bourguignon').as('a').
```
Using DataStax Enterprise advanced functionality

```cql
inE('reviewed').has('stars', 5).outV().outE('reviewed').has('stars', 5).
inV().where(neq('a')).groupCount().by('name')
```

DSE Graph data types

DSE Graph has many data types that are aligned with CQL data types. For search indexes, see the relationship between DSE Graph and Solr data types.

Table 129: DSE Graph Data Types

<table>
<thead>
<tr>
<th>DSE Graph Data Type</th>
<th>Description</th>
<th>Schema example</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigint</td>
<td>64-bit signed long</td>
<td><code>schema.propertyKey('big_number').BigInt().create()</code></td>
</tr>
<tr>
<td>blob</td>
<td>Arbitrary bytes (no validation), expressed as base64 strings</td>
<td><code>graph.addVertex(T.label, 'answer', 'blob', '42');</code> <code>schema.propertyKey('serial_string').Blob().create()</code></td>
</tr>
<tr>
<td>boolean</td>
<td>True or false</td>
<td><code>schema.propertyKey('alive').Boolean().create()</code></td>
</tr>
<tr>
<td>date</td>
<td>Date, in the format of '1940' or '1940-01-01.'</td>
<td><code>schema.propertyKey('review_date').Date().create()</code></td>
</tr>
<tr>
<td>decimal</td>
<td>Variable-precision decimal</td>
<td><code>schema.propertyKey('book_price').Decimal().create()</code></td>
</tr>
<tr>
<td>double</td>
<td>64-bit IEEE-754 floating point</td>
<td><code>schema.propertyKey('stars').Double().create()</code></td>
</tr>
<tr>
<td>duration</td>
<td>Time duration in milliseconds</td>
<td><code>schema.propertyKey('until').Duration().create()</code></td>
</tr>
<tr>
<td>float</td>
<td>32-bit IEEE-754 floating point</td>
<td><code>schema.propertyKey('precise').Float().create()</code></td>
</tr>
<tr>
<td>inet</td>
<td>IP address string in IPv4 or IPv6 format, used by the python-cql driver and CQL native protocols</td>
<td><code>schema.propertyKey('website_ip').Inet().create()</code></td>
</tr>
<tr>
<td>int</td>
<td>32-bit signed integer</td>
<td><code>schema.propertyKey('age').Int().create()</code></td>
</tr>
<tr>
<td>linestring</td>
<td>Used for geospatial and Cartesian linestrings (double .... points)</td>
<td><code>schema.propertyKey('road').Linestring().withGeoBounds().create()</code> (geospatial) <code>schema.propertyKey('road').Linestring().withBounds(-1, -1, 1, 1).create()</code> (Cartesian)`</td>
</tr>
</tbody>
</table>
### DSE Graph Data Type

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Schema example</th>
</tr>
</thead>
<tbody>
<tr>
<td>point</td>
<td>Used for geospatial and Cartesian points (double x, double y); note that this corresponds to longitude/latitude, in that order, for mapping geospatial points.</td>
<td>schema.propertyKey('coordinates').Point().withGeoBounds().create() (geospatial) schema.propertyKey('coordinates').Point().withBounds(-1, -1, 1, 1).create() (Cartesian)</td>
</tr>
<tr>
<td>polygon</td>
<td>Used for geospatial and Cartesian polygons (double .... points)</td>
<td>schema.propertyKey('block').Polygon().withGeoBounds().create() (geospatial) schema.propertyKey('block').Polygon().withBounds(-1, -1, 1, 1).create() (Cartesian)</td>
</tr>
<tr>
<td>smallint</td>
<td>2 byte integer</td>
<td>schema.propertyKey('age').Smallint().create()</td>
</tr>
<tr>
<td>text</td>
<td>String or UTF-8 encoded string</td>
<td>schema.propertyKey('name').Text().create()</td>
</tr>
<tr>
<td>time</td>
<td>Time in the format of '10:00:00' or '10:00'.</td>
<td>schema.propertyKey('time').Time().create()</td>
</tr>
<tr>
<td>timestamp</td>
<td>Date, or date plus time, encoded as 8 bytes since epoch. The <em>timestamp</em> data type must be specified as a valid DSE database <em>timestamp</em>.</td>
<td>schema.propertyKey('mealCreationDate').Timestamp().create()</td>
</tr>
<tr>
<td>uuid</td>
<td>A UUID in standard UUID format or timeuuid format</td>
<td>schema.propertyKey('authorID').Uuid().create()</td>
</tr>
<tr>
<td>varint</td>
<td>Arbitrary-precision integer</td>
<td>schema.propertyKey('number').Varint().create()</td>
</tr>
</tbody>
</table>

### Graph storage in the DSE database keyspace and tables

DSE Graph uses the DSE database to store schema and data. Two DSE database keyspaces are created for each graph, `<graphname>` and `<graphname_system>`. For example, for a graph called `food`, the two keyspaces created will be `food` and `food_system`. The first keyspace `food` will hold the data for the graph. The second keyspace `food_system` holds schema and other system data about the graph.

```java
johnDoe.addEdge('rated', beefBourguignon, 'timestamp', '2014-01-01T00:00:00.002', 'stars', 5, 'comment', 'Pretty tasty!')
```
In the `<graphname>` keyspace, two tables are created for each vertex label to store vertex and edge information, `vertexLabel_p` and `vertexLabel_e`, respectively. For example, for a vertex label `author`, two tables are created, `author_p` and `author_e`.

### NodeSync service

### DSE Advanced Replication

DSE Advanced Replication supports configurable distributed data replication from source clusters to destination clusters. It is designed to tolerate sporadic connectivity that can occur in constrained environments, such as retail, oil-and-gas remote sites, and cruise ships.

**Note:** To learn about replication, see About data distribution and replication.

### About DSE Advanced Replication

DSE Advanced Replication supports configurable distributed data replication from source clusters to destination clusters. It is designed to tolerate sporadic connectivity that can occur in constrained environments, such as retail, oil-and-gas remote sites, and cruise ships.

**Note:** To learn about replication, see About data distribution and replication.

### Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smartly replicates data from source clusters to destination clusters</td>
<td>Supports replicating data in a spoke and hub configuration from remote locations to central data hubs and repositories. Enterprise customers with remote clusters are able to establish a cluster presence in each location. In addition, mesh configuration can replicate data from any source cluster to another destination cluster within reasonable limits.</td>
</tr>
<tr>
<td>Prioritizes data streams</td>
<td>Allows higher priority data streams to be sent from the source cluster to a destination cluster ahead of lower priority data streams.</td>
</tr>
<tr>
<td>Supports ingestion and querying of data at every source</td>
<td>DSE Advanced Replication enables ingesting and querying data at any source and sent to any destination that collects and analyzes data from all of the sites.</td>
</tr>
<tr>
<td>Solves problem of periodic downtime</td>
<td>Useful for energy (oil and gas), transportation, telecommunications, retail (point-of-sale systems), and other vertical markets that might experience periods of network or internet downtime at the remote locations.</td>
</tr>
<tr>
<td>Satisfies data sovereignty regulations</td>
<td>Provides configurable streams of selected outbound data, while preventing data changes to inbound data.</td>
</tr>
</tbody>
</table>
**Satisfies data locality regulations** | Prevents data from leaving the current geography.

---

**DSE Advanced Replication architecture**

DSE Advanced Replication enables configurable replication between clusters, identifying source and destination clusters with replication channels. Topologies such as hub-and-spoke or mesh networks can differentially push or pull data depending on operational needs.

A common operational scenario for DSE Advanced Replication is a network of remote sensors with poor network connection to a centrally located storage and analytics network. The remote edge clusters collect data, but can experience disconnections from the network and periodically send one-way updates to the central hub clusters when a connection is available. Some sensors may be deemed more important than others, requiring prioritization of transmission. All sensors can continue to collect data, and to transmit in a specified manner, or have collection turned off as needed. Each remote sensor cluster would be designated as a source, while the central database cluster would be a destination.
Using DataStax Enterprise advanced functionality

This configuration would also be suitable to a network of microservices clusters that report data to a central analytics cluster.

Another scenario may include similar remote sites that mainly send data to a centralized location, but must periodically be updated with information from the centralized location. In this scenario, each remote cluster would be both a source and a destination, with two channels designated, one upstream and one downstream. A small Point of Sale (POS) system serves as a possible model for this scenario, with periodic updates to the remote systems.
A mesh network can also use advanced replication, with remote clusters receiving updates from either a central location or another remote cluster.
Although any cluster, remote or centralized, may serve as a source for an advanced replication channel, a limited number of destinations can be configured for any one source. In general, consider the flow of replication as many sources to few destinations, rather than few sources to many destinations.

**Traffic between the clusters**

Traffic between the source cluster and the destination cluster is managed with permits, priority, and configurable failover behavior for multi-datacenter operation.

**Permits**

Traffic between the source cluster and the destination cluster is managed with permits. When a permit cannot be acquired, the message is postponed and waits in the replication log until it is processed when a permit becomes available. Permits are global and not per destination.

To manage permits and set the maximum number of messages that can be replicated to all destinations simultaneously, use `dse advrep conf`:

```
$ dse advrep conf update --permits 1000
```

The default is 30,000.
Channel with a higher priority will take precedence in acquiring permits. Permits are required to transmit data from a source to a destination.

Priority and FIFO/LIFO enablement

The commit log is flushed from memory to disk, writing the data to the appropriate table. A Capture-Data-Change (CDC) collection agent additionally filters the data written and creates replication log files on disk. Each channel source table will have a separate data directory created on disk into which data is appended each time the commit log is flushed, storing all the messages that are to be replicated to a destination. Several replication log files may exist per source table at any given time. Each file stores a contiguous time-slice, configurable with `dse advrep conf update` command and the `--collection-time-slice-width` option (default: 60 seconds). A CDC transmission agent then sends the messages stored in the replication log files to the destination, where the data is processed and written to the appropriate database table. The order in which source table data is transmitted can be altered with the `priority` option when creating a channel, and the order in which a source table’s replication log files are read can be tuned with the `--fifo-enabled` and `--lifo-enabled` options.

The replication log files are processed according to the time and priority of the replication channel. Replication channel priorities are set per table, and determines how the transmission agent orders the transmission of replication log files from the source to the destination. The replication log files can be passed to the destination in either last in, first out (LIFO) or first in, first out (FIFO); FIFO is the default. If the newest messages should be read first, use LIFO; if the oldest messages should be read first, use FIFO. Once an individual replication log file is transmitted, the messages it contains are read FIFO. Both options, priority and read order, can be set during channel creation:

```
$ dse advrep --host 192.168.3.10 channel create --source-keyspace foo --source-table bar --source-id source1 --source-id-column source_id --destination mydest --destination-keyspace foo --destination-table bar --collection-enabled true --priority 1 --lifo-enabled
```

This example sets the channel for table `foo.bar` to the top priority of one, so that the table’s replication log files will be transmitted before other table’s replication log files. It also sets the replication log files to be read from newest to oldest.

Configure automatic failover for hub clusters with multiple datacenters

DSE Advanced Replication uses the DSE Java driver load balancing policy to communicate with the hub cluster. You can explicitly define the local datacenter for the datacenter-aware round robin policy (DCAwareRoundRobinPolicy) that is used by the DSE Java driver.

You can enable or disable failover from a local datacenter to a remote datacenter. When multiple datacenter failover is configured and a local datacenter fails, data replication from the edge to the hub continues using the remote datacenter. Tune the configuration with these parameters:

**driver-local-dc**

For destination clusters with multiple datacenters, you can explicitly define the name of the datacenter that you consider local. Typically, this is the datacenter that
Using DataStax Enterprise advanced functionality

is closest to the source cluster. This value is used only for clusters with multiple
data enters.

driver-used-hosts-per-remote-dc

To use automatic failover for destination clusters with multiple datacenters, you
must define the number of hosts per remote datacenter that the datacenter aware
round robin policy (DCAwareRoundRobinPolicy) considers available.

driver-allow-remote-dcs-for-local-cl

Set to true to enable automatic failover for destination clusters with multiple
datacenters. The value of the driver-consistency-level parameter must be
LOCAL_ONE or LOCAL_QUORUM.

To enable automatic failover with a consistency level of LOCAL_QUORUM, use dse advrep
destination update:

```
$ dse advrep destination update --name mydest --driver-allow-remote-dcs-for-local-cl true --driver-consistency-level LOCAL_QUORUM
Destination mydest updated
Updated driver_allow_remote_dcs_for_local_cl from null to true
Updated driver_consistency_level from ONE to LOCAL_QUORUM
```

DSE Advanced Replication terminology

This terminology is specific to DSE Advanced Replication that supports distributed data
replication from a DataStax Enterprise source cluster to a destination cluster.

collection agent

The process thread that runs on the source cluster that captures the incoming
changes and populates the replication log.

destination cluster

The cluster to which the data flow is going from the source cluster.

source cluster

A cluster that primarily sends data to one or more destination clusters. DSE
Advanced Replication must be enabled on the source cluster.

source datacenter

A datacenter of a source cluster.

destination cluster

A cluster that generally supports one or more source clusters that replicate data
to the destination cluster. DSE Advanced Replication is not required on the
destination cluster.

destination datacenter

A datacenter of a destination cluster.

isolated

The state of a cluster when there is not a live connection between the source
cluster and the destination cluster.

replication agent

The process thread that runs on the source cluster that reads data from the
replication log and transmits that data to the destination cluster.

replication channel
A defined channel of change data between source clusters and destination clusters. A replication channel is defined by the source cluster, source keyspace, source table name, destination cluster, destination keyspace, and destination table name.

**replication channel priority**
The priority order of which replication channel has precedence when limited bandwidth occurs between the source cluster and the destination cluster.

**replication log**
The replication log on the source cluster stores data in preparation for transmission to the destination cluster.

**tethered**
The state when there is a live connection between the source cluster and the destination cluster.

---

**Getting started with DSE Advanced Replication**

To test Advanced Replication, you must set up an source cluster and a destination cluster. These steps set up one node in each cluster.

Getting started overview:

1. Setting up the destination cluster node *(page 763)*
2. Setting up the source cluster *(page 764)*
3. Creating sample keyspace and table *(page 765)*
4. Configuring replication on the source node *(page 765)*
5. Creating the replication channel *(page 771)*
6. Starting replication from source to destination *(page 772)*
7. Inserting data on the source *(page 772)*
8. Testing loss of connectivity *(page 773)*
9. Testing replication start and stop *(page 774)*

**Note:** Due to Cassandra-11368, list inserts might not be idempotent (unchanged). Because DSE Advanced Replication might deliver the same message to the destination more than once, this Cassandra bug might lead to data inconsistency if lists are used in a column family schema. DataStax recommends using other collection types, like sets or frozen lists, when ordering is not important.

**Setting up the destination cluster node**

**Attention: Prerequisite:** If you are using Advanced Replication V1 from DSE 5.0, you must upgrade to DSE 5.1 and migrate to Advanced Replication V2.
Using DataStax Enterprise advanced functionality

On the destination node:

1. **Install DataStax Enterprise.**

2. Start DataStax Enterprise as a transactional node with the command that is appropriate for the installation method *(page 1437).*

3. Note the public IP address for the destination node.

Setting up the source cluster

Advanced replication can operate in a mixed-version environment. The source cluster requires DataStax Enterprise 5.1 or later. On the source node:

1. **Install DataStax Enterprise 5.1 or later.**

2. To enable replication, edit the `dse.yaml` file.

   At the end of the file, uncomment the `advanced_replication_options` setting and options, and set `enabled: true`.

   ```yaml
   # Advanced Replication configuration settings
   advanced_replication_options:
     enabled: true
   ``

3. **Enable Capture-Data-Change (CDC) *(page 64)* in the `cassandra.yaml` file on a per-node basis for each source:

   ```yaml
   cdc_enabled: true
   ``

   **Note:** Advanced Replication will not start if CDC is not enabled, since CDC logs are used to implement the feature.

4. Consider increasing the default CDC disk space, depending on the load (default: 4096 or 1/8 of the total space where `cdc_raw_directory` resides):

   ```yaml
   cdc_total_space_in_mb: 16384
   ``

5. The commitlog compression should be checked for the following value:

   ```yaml
   commitlog_compression:
     - class_name: LZ4Compressor
   ``

6. Start DataStax Enterprise as a transactional node with the command that is appropriate for the installation method *(page 1437).*
7. Once advanced replication is started on a cluster, the source node will create keyspaces and tables that need alteration. See Keyspaces (page 774) for information.

Creating the sample keyspace and table

These steps show you how to create the demonstration keyspace and table.

1. On the source node and the destination node, create the sample keyspace and table:

   ```
   CREATE KEYSPACE foo
   WITH REPLICATION = {
     'class': 'SimpleStrategy',
     'replication_factor':1};
   ```

   **Remember:** Remember to use escaped quotes around keyspace and table names as command line arguments to preserve casing: `dse advrep create --keyspace "keyspaceName" --table "tableName"`

2. On the source node:

   ```
   CREATE TABLE foo.bar (
     name TEXT,
     val TEXT,
     scalar INT,
     PRIMARY KEY (name));
   ```

3. On the destination node:

   ```
   CREATE TABLE foo.bar (
     name TEXT,
     val TEXT,
     scalar INT,
     source_id TEXT,
     PRIMARY KEY (name, source_id));
   ```

   **Note:** The `source_id` column is required on the destination node.

Configuring a replication destination on the source node

DSE Advanced Replication stores all of its settings in CQL tables. To configure replication, use the `dse advrep command line tool (page 808)`.

When you configure replication on the source node:

- The source node points to its destination using the public IP address that you saved earlier.
- The `source-id` value is a unique identifier for all data that comes from this particular source node.
Using DataStax Enterprise advanced functionality

- The `source-id` unique identifier is written to the `source-id-column` that was included when the `foo.bar` table was created on the destination node.

To configure a replication destination, run this command:

```bash
dse advrep --verbose destination create --name mydest --addresses 10.200.182.148 --transmission-enabled true
```

Destination `mydest` created

To verify the configuration, run this command:

```bash
dse advrep destination list-conf
```

<table>
<thead>
<tr>
<th>destination</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mydest</td>
<td>driver_ssl_enabled</td>
<td>false</td>
</tr>
<tr>
<td>mydest</td>
<td>addresses</td>
<td>10.200.182.148</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_read_timeout</td>
<td>15000</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_connections_max</td>
<td>8</td>
</tr>
<tr>
<td>mydest</td>
<td>source_id_column</td>
<td>source_id</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_connect_timeout</td>
<td>15000</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_ssl_protocol</td>
<td>TLS</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_consistency_level</td>
<td>QUORUM</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_used_hosts_per_remote_dc</td>
<td>0</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_allow_remote_dcs_for_local_cl</td>
<td>false</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_compression</td>
<td>lz4</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_connections</td>
<td>1</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

```plaintext
mydest     | driver_ssl_cipher_suites |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384, |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384, |
| TLS_RSA_WITH_AES_256_CBC_SHA256, |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384, |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384, |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA256, |
| TLS_DHE_DSS_WITH_AES_256_CBC_SHA256, |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA, |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, |
| TLS_RSA_WITH_AES_256_CBC_SHA, |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA, |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA, |
| TLS_DHE_DSS_WITH_AES_256_CBC_SHA, |
```
Using DataStax Enterprise advanced functionality

| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_DHE_DSS_WITH_AES_128_CBC_SHA256, |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA, |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, |
| TLS_RSA_WITH_AES_128_CBC_SHA, |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA, |
| TLS_ECDH_RSA_WITH_AES_128_CBC_SHA, |
| TLS_RSA_WITH_AES_128_CBC_SHA, |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, |
Using DataStax Enterprise advanced functionality

| TLS_DHE_DSS_WITH_AES_128_CBC_SHA, | , |
| TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, | , |
| TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, | , |
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, | , |
| TLS_RSA_WITH_AES_256_GCM_SHA384, | , |
| TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384, | , |
| TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384, | , |
| TLS_DHE_RSA_WITH_AES_256_GCM_SHA256, | , |
| TLS_DHE_DSS_WITH_AES_256_GCM_SHA384, | , |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, | , |
| TLS_RSA_WITH_AES_128_GCM_SHA256, | , |
| TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256, | , |
| TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256, | , |
Using DataStax Enterprise advanced functionality

| TLS_DHE_DSS_WITH_AES_128_GCM_SHA256 | , |
| TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA, | , |
| TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA, | , |
| SSL_RSA_WITH_3DES_EDE_CBC_SHA, | , |
| TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA, | , |
| TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA, | , |
| SSL_RSA_WITH_3DES_EDE_CBC_SHA, | , |
| TLS_ECDH_ECDSA_WITH_RC4_128_SHA, | , |
| TLS_ECDHE_RSA_WITH_RC4_128_SHA, | , |
| SSL_RSA_WITH_RC4_128_SHA, | , |
| TLS_ECDHE_ECDSA_WITH_RC4_128_SHA, | , |
| TLS_ECDH_RSA_WITH_RC4_128_SHA, | , |
Creating the replication channel

A replication channel is a defined channel of change data between source clusters and destination clusters. A replication channel is defined by the source cluster, source keyspace, source table name, destination cluster, destination keyspace, and destination table name. Source clusters can exist in multi-datacenter clusters, but a replication channel is configured with only one datacenter as the responsible party.

The keyspace and table names on the destination can be different than on the source, but in this example they are the same. You can also set the source-id and source-id-column differently from the global setting.

To create the replication channel for our keyspace and table:

```shell
dse advrep channel create --source-keyspace foo --source-table bar --source-id source1 --source-id-column source_id --destination mydest --destination-keyspace foo --destination-table bar --collection-enabled true --transmission-enabled true --priority 1
```

Created channel dc=Cassandra keyspace=foo table=bar to mydest

```shell
dse advrep channel status
```

<table>
<thead>
<tr>
<th>dc</th>
<th>keyspace</th>
<th>table</th>
<th>collecting</th>
<th>transmitting</th>
<th>replication order</th>
<th>priority</th>
<th>dest ks</th>
<th>dest table</th>
<th>src id</th>
<th>src id col</th>
<th>dest enabled</th>
<th>dest enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>foo</td>
<td>bar</td>
<td>true</td>
<td>true</td>
<td>FIFO</td>
<td>1</td>
<td>foo</td>
<td>bar</td>
<td>source1</td>
<td>source_id</td>
<td>mydest</td>
<td>true</td>
</tr>
</tbody>
</table>

**Warning:** The designated keyspace for a replication channel must have durable writes enabled. If durable_writes = false, then an error message will occur and the channel will not be created. If the durable writes setting is changed after the replication channel is created, the tables will not write to the commit log and CDC will...
not work. The data will not be ingested through the replication channel and a warning is logged, but the failure will be silent.

Starting replication from source to destination

At this point, the replication is configured and the replication channel is enabled and replication has been started.

1. On the destination, use cqlsh to verify that no data is present:

```sql
SELECT * FROM foo.bar;
```

<table>
<thead>
<tr>
<th>name</th>
<th>source_id</th>
<th>scalar</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(0 rows)

2. On the source, replication to the destination can be paused or resumed, the latter shown here:

```shell
dse advrep channel resume --source-keyspace foo --source-table bar --transmission
```

Channel dc=Cassandra keyspace=foo table=bar collection to mydest was resumed

Notice that either `--transmission` or `--collection` can be specified, to resume transmission from the source to the destination or to resume collection of data on the source.

3. Review the number of records that are in the replication log. Because no data is inserted yet, the record count in the replication log is 0:

```shell
dse advrep replog count --destination mydest --source-keyspace foo --source-table bar
```

0

Inserting data on the source

Insert data on the source for replication to the destination.

1. On the source, insert data using cqlsh:

```sql
INSERT INTO foo.bar (name, val, scalar) VALUES ('a', '1', 1);
INSERT INTO foo.bar (name, val, scalar) VALUES ('b', '2', 2);
```

2. On the destination, verify that the data was replicated:

```sql
SELECT * FROM foo.bar;
```
Checking data on the destination

Check data on the destination.

1. On the destination, verify that the data was replicated:

```sql
SELECT * FROM foo.bar;
```

<table>
<thead>
<tr>
<th>name</th>
<th>source_id</th>
<th>scalar</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>source1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>source1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>(2 rows)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Testing loss of connectivity

To test loss of connectivity to the destination, stop the DataStax Enterprise process on the destination, and insert more data on the source. The expected result is for data to be replicated quickly after the destination cluster resumes.

1. On the destination cluster, stop DataStax Enterprise:

   ```
dse cassandra-stop
```

2. On the source, insert more data:

   ```
INSERT INTO foo.bar (name, val, scalar) VALUES ('c', '3', 3);
INSERT INTO foo.bar (name, val, scalar) VALUES ('d', '4', 4);
```

3. Review the number of records that are in the replication log. The replication log should have 2 entries:

   ```
dse advrep replog count --destination mydest --source-keyspace foo --source-table bar
```

   2

4. On the destination, restart DataStax Enterprise.

   ```
dse cassandra
```
Wait a moment for communication and data replication to resume to replicate the new records from the source to destination.

```sql
SELECT * FROM foo.bar;
```

<table>
<thead>
<tr>
<th>name</th>
<th>source_id</th>
<th>scalar</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>source1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>source1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>source1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>source1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

4 rows(s)

5. On the source, the replication log count should be back to 0:

```bash
dse advrep replog count --destination mydest --source-keyspace foo --source-table bar
```

0

**Testing replication start and stop**

Similar to testing loss of connectivity, you can pause and resume individual replication channels by using the `advrep command line tool (page 808)`. The expected result is that newly inserted data is not saved to the replication log and will never be sent to the destination.

1. On the source, pause the replication channel:

   ```bash
dse advrep --verbose channel pause --keyspace foo --table bar --collection
   ```

2. Insert more data.

3. On the source, resume the replication channel:

   ```bash
dse advrep --verbose channel resume --keyspace foo --table bar --collection
   ```

**DSE Advanced Replication keyspace overview**

Keypaces and tables are automatically created on the source cluster when DSE Advanced Replication runs for the first time. Two keyspaces are used, `dse_system` and `dse_advrep`. Each keyspace is configured differently.

**Note:** System keyspaces on the source and destination are not supported for advanced replication.
The *dse_system* keyspace uses the EverywhereStrategy replication strategy by default; this setting must not be altered. The *dse_advrep* keyspace is configured to use the SimpleStrategy replication strategy by default and this setting must be updated in production environments to avoid data loss. After starting the cluster, alter the keyspace to use the NetworkTopologyStrategy replication strategy with an appropriate settings for the replication factor and datacenters. For example, use a CQL statement to configure a replication factor of 3 on the DC1 datacenter using NetworkTopologyStrategy:

```
ALTER KEYSPACE dse_advrep
WITH REPLICAATION = {
    'class': 'NetworkTopologyStrategy',
    'DC1': '3'};
```

For most environments using DSE Advanced Replication, a replication factor of 3 is suitable. The strategy must be configured for any datacenters which are serving as an advanced replication source.

`nodetool repair` must be run on each node of the affected datacenters. to repair the altered keyspace:

```
nodetool repair -full dse_advrep
```

For more information, see [Changing keyspace replication strategy](#).

**DSE Advanced Replication data types**

DSE data types are supported for most operations in DSE Advanced Replication. The following table shows the supported data types and operations:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Advanced Replication Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitive data types: int, ascii, bigint, blob, boolean, decimal, double, float, inet, text, timestamp, timeuuid, uuid, varchar, varint</td>
<td>All types are implemented for insert/update/delete.</td>
</tr>
<tr>
<td>Frozen collections: frozen-list&lt;data_type&gt;&gt;, frozen-set&lt;d data_type&gt;&gt;, frozen-map&lt;data_type, data_type&gt;&gt;</td>
<td>All frozen collections are implemented for insert/update/delete, as values are immutable blocks - entire column value is replicated.</td>
</tr>
<tr>
<td>Tuples: tuple&lt;data_type, data_type, data_type&gt;, frozen&lt;tuple&lt;data_type, data_type, data_type&gt;&gt;</td>
<td>All tuples are implemented for insert/update/delete, as values are immutable blocks - entire column value is replicated.</td>
</tr>
<tr>
<td>Frozen user-defined type (UDT): UDT type and frozen UDT type</td>
<td>All UDTs are implemented for insert/update/delete, as values are immutable blocks - entire column value is replicated.</td>
</tr>
<tr>
<td>Geometric types: Point, LineString, Polygon</td>
<td>All geometric types are implemented for insert/update/delete.</td>
</tr>
</tbody>
</table>

The following table shows the data type and operations that are not supported in DSE Advanced Replication:
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Advanced Replication Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unfrozen updatable collections:</td>
<td>All unfrozen updatable collections are implemented for insert/delete if the entire column value is replicated. Unfrozen collections cannot update values.</td>
</tr>
<tr>
<td>&lt;list&lt;data_type&gt;&gt;, &lt;set&lt;data_type&gt;&gt;,</td>
<td></td>
</tr>
<tr>
<td>&lt;map&lt;data_type, data_type&gt;&gt;</td>
<td></td>
</tr>
<tr>
<td>Unfrozen updatable user-defined type (UDT)</td>
<td>All unfrozen updatable UDTs are implemented for insert/delete if the entire column value is replicated. Unfrozen UDTs cannot update values.</td>
</tr>
</tbody>
</table>

### Using DSE Advanced Replication

Operations including starting, stopping, and configuring DSE Advanced Replication.

1. **Starting DSE Advanced Replication** *(page 776)*
2. **Stopping DSE Advanced Replication** *(page 777)*
3. **Configuring global configuration settings** *(page 778)*
4. **Configuring destination settings** *(page 779)*
5. **Configuring channel settings** *(page 792)*
6. **Security** *(page 793)*
7. **Data insert methods** *(page 795)*
8. **Monitoring operations** *(page 795)*

### Starting DSE Advanced Replication

**Attention: Prerequisite:** If you are using Advanced Replication V1 from DSE 5.0, you must upgrade to DSE 5.1 and migrate to Advanced Replication V2.

Before you can start and use DSE Advanced Replication, you must create the user **keyspaces** *(page 774)* and tables on the source cluster and the destination cluster.

On all nodes in the source cluster:

1. Enable replication in the dse.yaml file.

   Uncomment all advanced_replication_options entries, set enabled: true, and specify a directory to hold advanced replication log files with advanced_replication_directory:

   ```yaml
   # Advanced Replication configuration settings
   advanced_replication_options:
     enabled: true
   ```
2. **Enable Capture-Data-Change (CDC)** ([page 64](#)) in the cassandra.yaml file on a per-node basis for each source:

```yaml
cdc_enabled: true
cdc_raw_directory: /var/lib/cassandra/cdc_raw
```

**Note:** Advanced Replication will not start if CDC is not enabled. Either use the default directory or change it to a preferred location.

3. Consider increasing the default CDC disk space, depending on the load (default: 4096 MB or 1/8 of the total space where cdc_raw_directory resides):

```yaml
cdc_total_space_in_mb: 16384
```

4. Commitlog compression is turned off by default. To avoid problems with advanced replication, this option should NOT be used:

```yaml
# commitlog_compression:
#   - class_name: LZ4Compressor
```

5. Do a rolling restart: restart the nodes in the source cluster one at a time while the other nodes continue to operate online.

**Disabling DSE Advanced Replication**

When replication is not enabled, data is not written to the replication log. On all nodes in the source cluster:

1. To disable replication, edit the dse.yaml file.

   In the advanced_replication_options section, set enabled: false.

   ```yaml
   # Advanced Replication configuration settings
   advanced_replication_options:
     enabled: false
   ```

2. Do a rolling restart: restart the nodes in the source cluster one at a time while the other nodes continue to operate online.

3. To clean out the data that was used for DSE Advanced Replication, use cqlsh to remove these keyspaces ([page 774](#)):

   ```cql
   DROP TABLE dse_system.advrep_source_config;
   DROP TABLE dse_system.advrep_destination_config;
   DROP TABLE dse_system.advrep_repl_channel_config;
   ```
Configuring global configuration settings

Global settings apply to the entire source cluster. These global settings are stored in the CQL table `dse_system.advrep_source_config` that is automatically created.

Change global settings by using the `dse advrep command line tool (page 808)` with this syntax:

```
dse advrep conf ...
```

To view the source node configuration settings:

```
dse advrep conf list
```

The result is:

```
<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_log_file</td>
<td>/tmp/myaudit.gz</td>
</tr>
<tr>
<td>audit_log_enabled</td>
<td>true</td>
</tr>
</tbody>
</table>
```

The following table describes the configuration keys, their default values, and identifies when a restart of the source node is required for the change to be recognized.

The `dse advrep` command line tool uses these configuration keys as command arguments to the `dse advrep (page 808)` command line tool.

<table>
<thead>
<tr>
<th>Configuration key</th>
<th>Default value</th>
<th>Description</th>
<th>Restart required</th>
</tr>
</thead>
<tbody>
<tr>
<td>permits</td>
<td>30,000</td>
<td>Maximum number of messages that can be replicated in parallel over all destinations.</td>
<td>No</td>
</tr>
<tr>
<td>source-id</td>
<td>N/A</td>
<td>Identifies this source cluster and all inserts from this cluster. The source-id must also exist in the primary key on the destination for population of the source-id to occur.</td>
<td>No</td>
</tr>
<tr>
<td>collection-expire-after-write</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>collection-time-slice-count</td>
<td>5</td>
<td>The number of files which are open in the ingestor simultaneously.</td>
<td>Yes</td>
</tr>
<tr>
<td>collection-time-slice-width</td>
<td>60 seconds</td>
<td>The time period in seconds for each data block ingested. Smaller time widths =&gt; more files. Larger timer widths =&gt; larger files but more data to resend on CRC mismatches.</td>
<td>Yes</td>
</tr>
<tr>
<td>Configuration key</td>
<td>Default value</td>
<td>Description</td>
<td>Restart required</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>invalid-message-log</td>
<td>SYSTEM_LOG</td>
<td>Select one of these logging strategies to adopt when an invalid message is discarded:</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSTEM_LOG: Log the CQL query and the error message in the system log on the destination.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHANNEL_LOG: Store the CQL query and the error message in files in /var/lib/cassandra/advrep/invalid_queries on the destination.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NONE: Perform no logging.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Managing invalid messages (page 805). Requires node restart.</td>
<td></td>
</tr>
<tr>
<td>audit-log-enable</td>
<td>false</td>
<td>Specifies whether to store the audit log.</td>
<td>Yes</td>
</tr>
<tr>
<td>audit-log-file</td>
<td>/tmp/advrep_rl_audit.log</td>
<td>Specifies the file name prefix template for the audit log file. The file name is appended with .gz if compressed using gzip.</td>
<td>Yes</td>
</tr>
<tr>
<td>audit-log-max-life-span-mins</td>
<td>0</td>
<td>Specifies the maximum lifetime of audit log files. Periodically, when log files are rotated, audit log files are purged when they:</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Match the audit log file template</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• And they have not been written to for more than the specified maximum lifespan minutes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To disable purging, set to 0.</td>
<td></td>
</tr>
<tr>
<td>audit-log-rotate-time-mins</td>
<td>60</td>
<td>Specifies the time interval to rotate the audit log file. On rotation, the rotated file is appended with the log counter .[logcounter], incrementing from [0]. To disable rotation, set to 0.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Configuring destination settings**

A destination is a location to which source data will be written. Destinations are stored in the CQL table dse_system.advrep_destination_config that is automatically created.

Change destination settings by using the dse advrep command line tool (page 808) with this syntax:

```
dse advrep destination ...
```

You can verify the channel configuration before you change it. For example:

```
dse advrep destination list-conf
```

The result is:
<table>
<thead>
<tr>
<th>destination</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mydest</td>
<td>driver_ssl_enabled</td>
<td>false</td>
</tr>
<tr>
<td>mydest</td>
<td>addresses</td>
<td>10.200.182.251</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_read_timeout</td>
<td>15000</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_connections_max</td>
<td>8</td>
</tr>
<tr>
<td>mydest</td>
<td>source_id_column</td>
<td>source_id</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_connect_timeout</td>
<td>15000</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_ssl_protocol</td>
<td>TLS</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_consistency_level</td>
<td>QUORUM</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_used_hosts_per_remote_dc</td>
<td>0</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_allow_remote_dcs_for_local_cl</td>
<td>false</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_compression</td>
<td>lz4</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_connections</td>
<td>1</td>
</tr>
<tr>
<td>mydest</td>
<td>driver_ssl_cipher_suites</td>
<td>[TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>]</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

- TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
- TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_DSS_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
|                                     |                                     |,
|                                     |                                     |,
| TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256, |                                     |,
|                                     |                                     |,
| TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256, |                                     |,
|                                     |                                     |,
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, |                                     |,
|                                     |                                     |,
| TLS_DHE_DSS_WITH_AES_128_CBC_SHA256, |                                     |,
|                                     |                                     |,
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA, |                                     |,
|                                     |                                     |,
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, |                                     |,
|                                     |                                     |,
| TLS_RSA_WITH_AES_128_CBC_SHA,       |                                     |,
|                                     |                                     |,
| TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA, |                                     |,
|                                     |                                     |,
| TLS_ECDH_RSA_WITH_AES_128_CBC_SHA, |                                     |,
|                                     |                                     |,
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA, |                                     |,
|                                     |                                     |,
| TLS_DHE_DSS_WITH_AES_128_CBC_SHA, |                                     |,
|                                     |                                     |,
| TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, |                                     |,
|                                     |                                     |,
| TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, |                                     |,
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, | |
| TLS_RSA_WITH_AES_256_GCM_SHA384, | |
| TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384, | |
| TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384, | |
| TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, | |
| TLS_DHE_DSS_WITH_AES_256_GCM_SHA384, | |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, | |
| TLS_RSA_WITH_AES_128_GCM_SHA256, | |
| TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256, | |
| TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256, | |
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, | |
| TLS_DHE_DSS_WITH_AES_128_GCM_SHA256, | |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, | |
| TLS_RSA_WITH_AES_128_GCM_SHA256, | |
| TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256, | |
| TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256, | |
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, | |
| TLS_DHE_DSS_WITH_AES_128_GCM_SHA256, | |
| TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA, | |
| TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA, |
Using DataStax Enterprise advanced functionality

| SSL_RSA_WITH_3DES_EDE_CBC_SHA, |
| TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA, |
| TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA, |
| SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA, |
| SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA, |
| TLS_ECDHE_ECDSA_WITH_RC4_128_SHA, |
| TLS_ECDHE_RSA_WITH_RC4_128_SHA, |
| SSL_RSA_WITH_RC4_128_SHA, |
| TLS_ECDH_ECDSA_WITH_RC4_128_SHA, |
| TLS_ECDH_RSA_WITH_RC4_128_SHA, |
| SSL_RSA_WITH_RC4_128_MD5, |
| TLS_EMPTY_RENEGOTIATION_INFO_SCSV |

<table>
<thead>
<tr>
<th>mydest</th>
<th>source_id</th>
<th>source1</th>
</tr>
</thead>
<tbody>
<tr>
<td>mydest</td>
<td>transmission_enabled</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>driver_ssl_enabled</td>
<td>false</td>
</tr>
<tr>
<td>---</td>
<td>--------------------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>addresses</td>
<td>10.200.177.184</td>
</tr>
<tr>
<td></td>
<td>driver_read_timeout</td>
<td>15000</td>
</tr>
<tr>
<td></td>
<td>driver_connections_max</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>source_id_column</td>
<td>source_id</td>
</tr>
<tr>
<td></td>
<td>driver_connect_timeout</td>
<td>15000</td>
</tr>
<tr>
<td></td>
<td>driver_ssl_protocol</td>
<td>TLS</td>
</tr>
<tr>
<td></td>
<td>driver_consistency_level</td>
<td>ONE</td>
</tr>
<tr>
<td></td>
<td>driver_used_hosts_per_remote_dc</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>driver_allow_remote_dcs_for_local_cl</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>driver_compression</td>
<td>lz4</td>
</tr>
<tr>
<td></td>
<td>driver_connections</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>driver_ssl_cipher_suites</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>,</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_DSS_WITH_AES_256_CBC_SHA256,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_RSA_WITH_AES_256_CBC_SHA,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_RSA_WITH_AES_256_CBC_SHA,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_DSS_WITH_AES_256_CBC_SHA,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Curve</td>
<td>Cipher</td>
<td>Authentication</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------</td>
<td>-----------------</td>
</tr>
<tr>
<td>TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_RSA_WITH_AES_128_CBC_SHA256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_DSS_WITH_AES_128_CBC_SHA256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_RSA_WITH_AES_128_CBC_SHA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_RSA_WITH_AES_128_CBC_SHA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_DSS_WITH_AES_128_CBC_SHA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA256</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>l1pdest</th>
<th>source_id</th>
<th>source1</th>
</tr>
</thead>
<tbody>
<tr>
<td>llpdest</td>
<td>transmission_enabled</td>
<td>false</td>
</tr>
</tbody>
</table>
The following table describes the configuration keys, their default values, and identifies when a restart of the source node is required for the change to be recognized.

<table>
<thead>
<tr>
<th>Configuration key</th>
<th>Default value</th>
<th>Description</th>
<th>Restart required</th>
</tr>
</thead>
<tbody>
<tr>
<td>separator</td>
<td>N/A</td>
<td>Field separator.</td>
<td>No</td>
</tr>
<tr>
<td>name</td>
<td>N/A</td>
<td>Name for destination (required).</td>
<td>No</td>
</tr>
<tr>
<td>addresses</td>
<td>none</td>
<td>REQUIRED. A comma separated list of IP addresses that are used to connect to the destination cluster using the DataStax Java driver.</td>
<td>No</td>
</tr>
<tr>
<td>driver-allow-remote-dcs-for-local-cl</td>
<td>false</td>
<td>Set to true to enable automatic failover for destination clusters with multiple datacenters. The value of the <code>driver-consistency-level</code> parameter must be LOCAL_ONE or LOCAL_QUORUM.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver-compression</td>
<td>lz4</td>
<td>The compression algorithm the DataStax Java driver uses to send data from the source to the destination. Supported values are lz4 and snappy.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver-connect-timeout</td>
<td>15000</td>
<td>Time in milliseconds the DataStax Java driver waits to connect to a server.</td>
<td>No</td>
</tr>
<tr>
<td>driver-connections</td>
<td>32</td>
<td>The number of connections the DataStax Java driver will create.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver-connections-max</td>
<td>256</td>
<td>The maximum number of connections the DataStax Java driver will create.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver-max-requests-per-connection</td>
<td>1024</td>
<td>The maximum number of requests per connection the DataStax Java driver will create.</td>
<td></td>
</tr>
<tr>
<td>driver-consistency-level</td>
<td>ONE</td>
<td>The consistency level used by the DataStax Java driver when executing statements for replicating data to the destination. Specify a valid DSE CONSISTENCY level: ANY, ONE, TWO, THREE, QUORUM, ALL, LOCAL_QUORUM, EACH QUORUM, SERIAL, LOCAL_SERIAL, or LOCAL.ONE.</td>
<td>No</td>
</tr>
<tr>
<td>driver-local-dc</td>
<td>N/A</td>
<td>For destination clusters with multiple datacenters, you can explicitly define the name of the datacenter that you consider local. Typically, this is the datacenter that is closest to the source cluster. This value is used only for clusters with multiple data enters.</td>
<td>Yes</td>
</tr>
<tr>
<td>Configuration key</td>
<td>Default value</td>
<td>Description</td>
<td>Restart required</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>driver-pwd</td>
<td>none</td>
<td>Driver password if the destination requires a user and password to connect. Changing the driver-pwd value for connection to a destination will automatically connect, but with a slight delay. Note: By default, driver user names and passwords are plain text. DataStax recommends encrypting the driver passwords before you add them to the CQL table.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver-read-timeout</td>
<td>15000</td>
<td>Time in milliseconds the DataStax Java driver waits to read responses from a server.</td>
<td>No</td>
</tr>
<tr>
<td>driver-ssl-enabled</td>
<td>false</td>
<td>Whether SSL is enabled for connection to the destination.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver-ssl-disabled</td>
<td></td>
<td>Disable SSL for connection to the destination.</td>
<td></td>
</tr>
<tr>
<td>driver_ssl_keystore_path</td>
<td>none</td>
<td>The path to the keystore for connection to DSE when SSL client authentication is enabled.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver_ssl_keystore_password</td>
<td>none</td>
<td>The keystore password for connection to DSE when SSL client authentication is enabled.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver_ssl_keystore_type</td>
<td>none</td>
<td>The keystore type for connection to DSE when SSL client authentication is enabled.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver_ssl_truststore_path</td>
<td>none</td>
<td>The path to the truststore for connection to DSE when SSL is enabled.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver-ssl-truststore-password</td>
<td>none</td>
<td>The truststore password for connection to DSE when SSL is enabled.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver-ssl-truststore-type</td>
<td>none</td>
<td>The keystore type for connection to DSE when SSL client authentication is enabled.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver-ssl-protocol</td>
<td>TLS</td>
<td>The SSL protocol for connection to DSE when SSL is enabled.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver-ssl-cipher-suites</td>
<td>none</td>
<td>A comma-separated list of SSL cipher suites for connection to DSE when SSL is enabled. Cipher suites must be supported by the source machine.</td>
<td>Yes</td>
</tr>
<tr>
<td>driver-used-hosts-per-remote-dc</td>
<td>0</td>
<td>To use automatic failover for destination clusters with multiple datacenters, you must define the number of hosts per remote datacenter that the datacenter aware round robin policy (DCAwareRoundRobinPolicy) considers available.</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Configuration key</th>
<th>Default value</th>
<th>Description</th>
<th>Restart required</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver-user</td>
<td>none</td>
<td>Driver username if the destination requires a user and password to connect. Changing the driver-user value for connection to a destination will automatically connect, but with a slight delay.</td>
<td>Yes</td>
</tr>
<tr>
<td>source-id</td>
<td>N/A</td>
<td>Identifies this source cluster and all inserts from this cluster. The source-id must also exist in the primary key on the destination for population of the source-id to occur.</td>
<td>No</td>
</tr>
<tr>
<td>source-id-column</td>
<td>source-id</td>
<td>The column to use on remote tables to insert the source id as part of the update. If this column is not present on the table that is being updated, the source id value is ignored.</td>
<td>No</td>
</tr>
<tr>
<td>transmission-enabled</td>
<td>false</td>
<td>Specify if data collector for the table should be replicated to the destination using boolean value.</td>
<td>No</td>
</tr>
</tbody>
</table>

Configuring channel settings

A replication channel is a defined channel of change data between source clusters and destination clusters.

A replication channel is defined by the source cluster, source keyspace, source table name, destination cluster, destination keyspace, and destination table name. Replications for each channel (unique keyspace and table) are stored in the CQL table dse_system.advrep_repl_channel_config that is automatically created.

Change the settings using the dse advrep (page 808) command line tool with this syntax:

```
dse advrep channel ...
```

You can verify the channel configuration before you change it. For example:

```
dse advrep channel status
```

The result is:

```
-------------------------------------------------------------------------------------------------------------------------------------
dc|keyspace|table          |collecting|transmitting|replication
order|priority|dest ks|dest table     |src id |src id col|dest  |dest enabled|
-------------------------------------------------------------------------------------------------------------------------------------
Cassandra|foo    |bar             |true      |true        |FIFO
|2       |foo   |bar             |source1|source_id   |mydest|true
-------------------------------------------------------------------------------------------------------------------------------------
```

Properties are continuously read from the metadata, so a restart is not required after configuration changes are made. The following table describes the configuration settings.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>separator</td>
<td>Field separator.</td>
</tr>
<tr>
<td>keyspace</td>
<td>The keyspace on the source for the table to replicate.</td>
</tr>
<tr>
<td>table</td>
<td>The table name on the source to replicate.</td>
</tr>
<tr>
<td>source-id</td>
<td>Placeholder to override the source-id that is defined in the advrep_conf</td>
</tr>
<tr>
<td></td>
<td>metadata</td>
</tr>
<tr>
<td>source-id-column</td>
<td>Placeholder to override the source-id-column that is defined in advrep_conf</td>
</tr>
<tr>
<td></td>
<td>metadata.</td>
</tr>
<tr>
<td>enabled</td>
<td>If true, replication will start for this table. If false, no more messages</td>
</tr>
<tr>
<td></td>
<td>from this table will be saved to the replication log.</td>
</tr>
<tr>
<td>data-center-id</td>
<td>Datacenter this replication channel is meant for, if none specified the</td>
</tr>
<tr>
<td></td>
<td>replication will happen in all specified dc1.</td>
</tr>
<tr>
<td>destination</td>
<td>Destination to which data is written.</td>
</tr>
<tr>
<td>destination-</td>
<td>The keyspace on the destination for the replicated table.</td>
</tr>
<tr>
<td>keyspace</td>
<td></td>
</tr>
<tr>
<td>destination-table</td>
<td>The table name on the destination for the replicated table.</td>
</tr>
<tr>
<td>priority</td>
<td>Messages are marked by priority in descending order (DESC).</td>
</tr>
<tr>
<td>transmission-</td>
<td>Specify if the data collector for the table should be replicated to the</td>
</tr>
<tr>
<td>enabled</td>
<td>destination.</td>
</tr>
<tr>
<td>fifo-order</td>
<td>Specify if the channel should be replicated in FIFO order (default).</td>
</tr>
<tr>
<td>lifo-order</td>
<td>Specify if the channel should be replicated in LIFO order.</td>
</tr>
</tbody>
</table>

Security

Authentication credentials can be provided in several ways, see Providing credentials from DSE tools.

The user who is doing the replicating with DSE Advanced Replication requires table and keyspace level authorization. If the same user access is required, then ensure that the authorization is the same on the source and destination clusters.

Advanced Replication also supports setting row-level permissions on the destination cluster. The user which connects to the destination cluster must have permission to write to the specified destination table at the row level replicated from the source, according to the RLAC restrictions. The user is specified with the --driver-user destination (page 779) setting. Row-level access control (RLAC) on the source cluster does not impact Advanced Replication. Because Advanced Replication reads the source data at the raw CDC file layer, it essentially reads as a superuser and has access to all configured data tables.

Advanced Replication supports encrypting the driver passwords. Driver passwords are stored in a CQL table. By default, driver passwords are plain text. DataStax recommends
encrypting the driver passwords before you add them to the CQL table. Create a global encryption key, called a system_key for **SSTable encryption**. Each node in the source cluster must have the same system key. The destination does not require this key.

1. In the dse.yaml file:
   - Verify that the `config_encryption_active` property is false:
     ```yaml
     config_encryption_active: false
     ```
   - Enable driver password encryption with the `conf_driver_password_encryption_enabled` property:
     ```yaml
     conf_driver_password_encryption_enabled: true
     ```
   - Define where system keys are stored on disk. The location of the key is specified on the command line with the `-d (page 1321)` option or with `system_key_directory (page 114)` in dse.yaml. The default filepath is `/etc/dse/conf`.
   - To configure the filename of the generated encryption key, set the `config_encryption_key_name (page 114)` option in dse.yaml. The default name is `system_key`.

2. Generate a system key:
   - **On-server**:
     ```bash
     dsetool createsystemkey cipher_algorithm strength system_key_file
     ```
   - **Off-server**:
     ```bash
     dsetool createsystemkey cipher_algorithm strength system_key_file -kmip=kmip_groupname
     ```
     For example:
     ```bash
     dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 system_key_file
     ```
     where `system_key_file` is a unique file name for the generated system key file. See `createsystemkey (page 1320)`.

     Result: Configure transparent data encryption (TDE) on a per table basis. You can configure encryption with or without compression. You can create a global encryption key in the location that is specified by `system_key_directory (page 114)` in the dse.yaml file. This default global encryption key is used when the `system_key_file` subproperty is not specified.

3. Copy the returned value.

4. On any node in the source cluster, use the `dse` command to set the encrypted password in the DSE Advanced Replication environment:
Using DataStax Enterprise advanced functionality

5. Start dse (page 1437).

Data insert methods

There are several ways to get data into a DataStax Enterprise cluster. Any normal paths used will result in data replication using DSE Advanced Replication.

Supported data insert methods:

- CQL insert, including cqlsh and applications that use the standard DSE drivers
- COPY from a CSV file
- Solr HTTP or CQL
- Spark saveToCassandra

Unsupported data insert methods:

- sstableloader (Cassandra bulk loader (page 1408))
- OpsCenter restore from backup
- Spark bulkSaveToCassandra

Monitoring operations

Advanced replication can be monitored with JMX metrics. The outgoing replication queue size is a key factor to watch. See Metrics (page 797) for more details.

CQL queries in DSE Advanced Replication

This overview of supported CQL queries and replication concepts for DSE Advanced Replication provide details on supported CQL queries and best practices guidelines.

DSE Advanced Replication replicates data from source clusters to destination clusters. Replication takes the CQL query on the source and then recreates a modified version of the query and runs it on the destination. DataStax Enterprise supports a restricted list of valid CQL queries to manipulate data. In DSE Advanced Replication, the same restrictions apply to the generated CQL queries that are used to replicate data into the destination.

Restrictions apply to the primary key. The primary key consists of two parts: the partition key and the clustering key. The primary key parts plus the optional field values comprise the database row.

If differences exist between the primary key on the source table and the primary key on the destination table, restrictions apply for which CQL queries are supported.

Best practices

DataStax recommends the following best practices to ensure seamless replication.

Schema structure on the source table and the destination table
Using DataStax Enterprise advanced functionality

- Maintain an identical primary key (partition keys and clustering keys) format in the same order, with the same columns.
- Add the optional `source_id` as the first clustering column.
- Maintain all, or a subset of, the field values.

**Note:** Although the `source_id` column can be present in the source table schema, values that are inserted into that column are ignored. When records are replicated, the configured `source-id` (page 811) value is used.

**Partition key columns**

The following list details support and restrictions for partition keys:

- In the destination table, only an additional optional `source_id` column is supported in the partition key. Additional destination table partition key columns are not supported. The `source_id` can be either a clustering column or a partition key, but not both.
- Using a subset of source table partition key columns in the destination table might result in overwriting. There is a many-to-one mapping for row entries.
- Order is irrelevant for replication. All permutations are supported.
- CQL `UPDATE` queries require that all of the partition key columns are fully restricted. Restrict partition key columns using `=` or `IN` (single column) restrictions.
- CQL `DELETE` queries require that all of the partition key columns are fully restricted. Restrict partition key columns using `=` or `IN` (single column) restrictions.

**Clustering columns**

The following list details support and restrictions for clustering columns:

- In the destination table, only an additional optional `source_id` column is supported in the clustering column. Additional destination table partition key columns are not supported. The `source_id` can be either a clustering column or a partition key, but not both.
- Using a subset of source table clustering columns in the destination table might result in overwriting. There is a many-to-one mapping for row entries.
- Order is irrelevant for replication when using CQL `INSERT` and `UPDATE` queries. All permutations are supported.
- Order is relevant for replication when using CQL `DELETE` queries. There are limits to permutation support, all permutations are not supported.
- CQL `UPDATE` queries require that all of the clustering columns are fully restricted. Restrict partition key columns using `=` or `IN` (single column) restrictions.
- CQL `DELETE` queries require that the last-specified clustering column be restricted using `=</>/>=/<<=` (single or multiple column) or `IN` (single or multiple column). All of the clustering columns that precede the last-specified clustering column must also be restricted using `=` or `IN`. 
• Restricting clustering columns is optional. However, if you do restrict clustering columns, then all of the clustering columns that you restrict between the first and last (in order) clustering columns must be restricted.

Field values

The following list details support and requirements for field values:
• A subset, or all, of the field values on the source are supported for replication to the destination.
• Fields that are present on the source, but absent on the destination, are not replicated.
• Fields that are present on the destination, but absent on the source, are not populated.

Source ID (source_id)

The source_id identifies the source cluster and all inserts from the source cluster. The following list details support and requirements for the source_id:
• The source_id configuration key (page 778) must be present and correct in the metadata.
• The source_id must be the first position in the clustering column, or any of the partition keys.

If not, then the CQL INSERT and UPDATE queries should work, but the CQL DELETE queries with partially restricted clustering columns might fail.

• The source_id is always restricted in CQL DELETE and UPDATE queries. Certain delete statements are not supported where the clustering key is not fully restricted, and the source_id is not the first clustering column.

DSE Advanced Replication metrics

Collect metrics on each source node to review the current status of that node in the source cluster. A working source and destination configuration is required to use the metrics feature. See Getting started (page 763).

Ensure JMX access

Metrics are stored in the DataStax Enterprise JMX system. JMX access is required.
• For production, DataStax recommends authenticating JMX users, see Configuring JMX authentication.
• Use these steps to enable local JMX access. Localhost access is useful for test and development.

1. On the source node, edit cassandra-env.sh and enable local JMX:

   ```
   JVM_OPTS="$JVM_OPTS -Djava.rmi.server.hostname=localhost"
   LOCAL_JMX=yes
   ```
2. On the source node, stop and restart (page 1437) DataStax Enterprise to recognize the local JMX change.

Display metrics on the command line

Use the `dse advrep` command line tool to display metrics on the command line. Ensure that the source node meets the command line prerequisites.

1. On the source node:

```bash
$ dse advrep --jmx-port 7199 metrics list
```

<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>MessagesDelivered</td>
<td>1002</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>CommitLogsToConsume</td>
<td>1</td>
</tr>
<tr>
<td>Tables</td>
<td>MessagesReceived</td>
<td>1002</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessageAddErrors</td>
<td>0</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>CommitLogsDeleted</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Count</th>
<th>RateUnit</th>
<th>MeanRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReplicationLog</td>
<td>MessagesAdded</td>
<td>1002</td>
<td>events/second</td>
<td>0.012688461014514603</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesDeleted</td>
<td>0</td>
<td>events/second</td>
<td>0.0</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesAcknowledged</td>
<td>1002</td>
<td>events/second</td>
<td>0.012688456391385135</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>CommitLogMessagesRead</td>
<td>16873</td>
<td>events/second</td>
<td>2.323039468969963E-114</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission</td>
<td>AvailablePermits</td>
<td>30000</td>
</tr>
</tbody>
</table>

Accessing the metrics

Use JMX to access the metrics. Any JMX tool, such as jconsole, can access the MBeans for advanced replication. The port listed above, 7199, is used with the hostname or IP address:

Choose the MBeans tab and find com.datastax.bdp.advrep.v2.metrics in the left-hand navigation frame:
Using DataStax Enterprise advanced functionality

The example shown here displays the attributes for
com.datastax.bdp.advrep.v2.metrics:type=ReplicationLog,name=MessagesAdded.

Performance metrics

Metrics are exposed as JMX MBeans under the com.datastax.bdp.advrep.v2.metrics path and are logically divided into main groups. Each group refers to an architecture component. Metrics types are:

**Counter**
A simple incrementing and decremented 64-bit integer.

**Meter**
Measures the rate at which a set of events occur.

**Histogram**
Measures the distribution of values in a stream of data.

**Timer**
A histogram of the duration of a type of event and a meter of the rate of its occurrence.

**Gauge**
A gauge is an instantaneous measurement of a value.

Metrics are available for the following groups:
Using DataStax Enterprise advanced functionality

- ReplicationLog (page 801)
- Transmission (page 802)
- AdvancedReplicationHub-[destinationId]-metrics (page 802)

Metrics are also available per table:
- Performance metrics per table (page 804)

Descriptions of each metric is provided.

**Note:** Metrics for DSE 5.0 (V1) are still present; see the DSE 5.0 documentation for those metrics.

### ReplicationLog

Metrics for the ReplicationLog group:

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Description</th>
<th>Metric type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessagesAdded</td>
<td>The number of messages that were added to the replication log, and the rate that the messages were added, per replica.</td>
<td>Meter</td>
</tr>
<tr>
<td>MessagesAcknowledged</td>
<td>The number of messages that were acknowledged (and removed) from the replication log. Acknowledgement can be 1 or 1+n if errors occur.</td>
<td>Meter</td>
</tr>
<tr>
<td>MessagesDeleted</td>
<td>The number of messages that were deleted from the replication log, including invalid messages and messages that were removed after a channel truncate operation.</td>
<td>Meter</td>
</tr>
<tr>
<td>MessageAddErrors</td>
<td>The number of errors that occurred when adding a message to the replication log.</td>
<td>Counter</td>
</tr>
<tr>
<td>CommitLogsToConsume</td>
<td>The number of commit logs that need to be consumed that have advanced replication messages.</td>
<td>Counter</td>
</tr>
<tr>
<td>CommitLogMessagesRead</td>
<td>The number of commit log messages added to the replication log. The commit log messages are read if a message pertains to a source table that has collection enabled.</td>
<td>Meter</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Description</th>
<th>Metric type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommitLogMessagesDeleted</td>
<td>The number of commit log messages deleted from the commit log after adding to the replication log. Like CommitLogMessagesRead, this metric only pertains to messages in tables that are enabled for advanced replication.</td>
<td>Meter</td>
</tr>
</tbody>
</table>

Transmission

Metrics for the Transmission group:

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Description</th>
<th>Metric type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailablePermits</td>
<td>The current number of available global permits for transmission.</td>
<td>Gauge</td>
</tr>
</tbody>
</table>

AdvancedReplicationHub-[destinationName]-metrics

Metrics for the AdvancedReplicationHub-[destinationName]-metrics group are provided automatically by the DSE Java driver.
In complete examples of per-destination-metrics are:

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Metric type</th>
</tr>
</thead>
<tbody>
<tr>
<td>known-hosts</td>
<td>Counter</td>
</tr>
<tr>
<td>connected-to</td>
<td>Counter</td>
</tr>
<tr>
<td>open-connections</td>
<td>Counter</td>
</tr>
<tr>
<td>requests-timer</td>
<td>Timer</td>
</tr>
<tr>
<td>connection-errors</td>
<td>Counter</td>
</tr>
<tr>
<td>write-timeouts</td>
<td>Counter</td>
</tr>
<tr>
<td>read-timeouts</td>
<td>Counter</td>
</tr>
<tr>
<td>unavailables</td>
<td>Counter</td>
</tr>
<tr>
<td>other-errors</td>
<td>Counter</td>
</tr>
<tr>
<td>retries</td>
<td>Counter</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Metric type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ignores</td>
<td>Counter</td>
</tr>
</tbody>
</table>

For details, see the DSE Java driver documentation.

Performance metrics per table

Use JMX to find performance metrics per table, look under the com.datastax.bdp.advrep.v2.metrics tab in the left-hand navigation frame for Tables, select a table and inspect the metrics:

For example, to access the MessagesReceived metric for the table sensor_readings in the keyspace demo look at the following path:

```
com.datastax.bdp.advrep.v2.metrics:type=Tables,scope=demo.sensor_readings,name=MessagesReceived
```

The following metrics are provided per table:

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Description</th>
<th>Metric type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessagesReceived</td>
<td>The number of messages received from the source cluster for this table.</td>
<td>Counter</td>
</tr>
<tr>
<td>Metric name</td>
<td>Description</td>
<td>Metric type</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MessagesDelivered</td>
<td>The number of messages for the source table that were replicated to the destination.</td>
<td>Counter</td>
</tr>
<tr>
<td>MessagesDeleted</td>
<td>The number of messages that were deleted from the replication log, including invalid messages and messages that were removed after a channel truncate operation.</td>
<td>Counter</td>
</tr>
</tbody>
</table>

**Managing invalid messages**

During message replication, DSE Advanced Replicates attempts to manipulate the message to ensure successful replication. In some cases, replication might occur with only a subset of the data.

In other cases, replication fails when there are too many differences between the schema on the source cluster and the schema on the destination cluster. For example, schema incompatibilities occur when a column in the destination has a different type than the same column in the source, or a table in the source doesn’t contain all the columns that form the primary key of the same table in the destination.

If a message cannot be replicated, a second transmission will be tried. If replication still fails after that the second try, the message is discarded and removed from the replication log. The replication log on the source cluster stores data in preparation for transmission to the destination cluster.

When a message is discarded, the CQL query string and the related error message are logged on the destination cluster. To define where to store the CQL strings and the error messages that are relevant to the failed message replication, use one of the following logging strategies:

- **SYSTEM_LOG**: Log the CQL query and the error message in the system log on the destination.
- **CHANNEL_LOG**: Store the CQL query and the error message in files in `/var/lib/cassandra/advrep/invalid_queries` on the destination. This is the default value.
- **NONE**: Perform no logging.

For the channel logging strategy, a file is created in the channel log directory on the source node, following the pattern `/var/lib/cassandra/advrep/invalid_queries/<keyspace>/<table>/<destination>/invalid_queries.log` where `keyspace`, `table` and `destination` are:

- **keyspace**: keyspace name of the invalid query
- **table**: table name of the invalid query
- **destination**: destination cluster of the channel

The log file stores the following data that is relevant to the failed message replication:
Using DataStax Enterprise advanced functionality

- **time_bucket**: an hourly time bucket to prevent the database partition from getting too wide
- **id**: a time based id (timeuuid)
- **cql_string**: the CQL query string, explicitly specifies the original timestamp by including the `USING TIMESTAMP` option.
- **error_msg**: the error message

Invalid messages are inserted by time in the log table.

**Manage invalid messages using channel logging:**

1. To store the CQL query string and error message using a channel log, instead of the default system log location, specify the `invalid_message_log` configuration key as `CHANNEL_LOG`:

   ```
   $ dse advrep conf update --invalid_message_log CHANNEL_LOG
   ```

**Manage invalid messages using system logging:**

2. To store the CQL query string and error message using a system log, instead of the default channel log location, specify the `invalid_message_log` configuration key as `SYSTEM_LOG`:

   ```
   $ dse advrep conf update --invalid_message_log SYSTEM_LOG
   ```

3. To identify the problem, examine the error messages, the CQL query strings, and the schemas of the data on the source and the destination.

4. Take appropriate actions to resolve the incompatibility issues.

**Managing audit logs**

DSE Advanced Replication provides replication audit logging and commands to manage the audit logs with metadata configuration. Audit logs are stored on the source cluster and are handled by the audit log analyzer (AuditLogAnalyzer). The audit log analyzer reads the log files, including audit log files in GZIP (.gz) format, that might be incomplete because they are still being written or they were improperly closed. The audit log analyzer identifies the list of files which match the template that is defined with the `audit_log_file` configuration key and that have exceeded the maximum time interval since they were written to. Purging is based on these criteria.

Global settings apply to the entire source cluster. These global settings are stored in the CQL table `dse_system.advrep_source_config` that is automatically created. To define configuration keys to change global settings (page 778), use the `dse advrep conf update` command. The audit log files are read/write (RW) only for the file owner, with no permissions for other users.
**Note:** The time stamp for all writes is UTC (Universal Time Coordinated).

1. Enable replication audit logging:
   ```bash
   $ dse advrep conf update --audit-log-enabled true
   ```

2. The default base audit log directory is `/var/lib/cassandra/advrep/auditlog`. To define a different directory for storing audit log files:
   ```bash
   $ dse advrep conf update --audit-log-file /tmp/auditAdvRep
   ```
   If the configured audit log file is a relative path, then the log files be placed in the default base directory. If the configured audit log file is an absolute path, then that path is used.

3. To compress the audit log output using the gzip file format:
   ```bash
   $ dse advrep conf update --audit-log-compression GZIP --audit-log-file /tmp/auditAdvRep/myaudit.gz
   ```
   The default value is NONE for compression. If .gz is not appended to the audit log filename in the command, it will be appended to the created files. Compressed audit log files will remain locked until rotated out; the active file cannot be opened.

4. Specify the time interval to rotate the audit log file. On rotation, the rotated file is appended with the log counter `[logcounter]`, incrementing from `[0]`. To disable rotation, set to 0.
   ```bash
   $ dse advrep conf update --audit-log-rotate-mins 120
   ```
   For example, the compressed file from the last step can be uncompressed after rotating out to `/tmp/auditAdvRep/myaudit.[0].gz`.

5. Specify the maximum lifetime of audit log files.
   After audit log files are rotated, they are periodically purged when the log files:
   - Match the audit log file
   - And have not been written to for more than the specified maximum lifespan minutes
   To disable purging, set to 0.
   ```bash
   $ dse advrep conf update --audit-log-max-life-span-mins 120
   ```

6. Restart the node to enable the changes.
When logging is enabled, log files that would be overwritten are moved to a subdirectory in the log directory. The subdirectory is named `archive_x`, where `x` increments from 0 until an unused directory is identified and created.

**dse advrep commands**

A list of commands for DSE Advanced Replication.

**About the dse advrep command**

The command line tool provides commands and options for configuring and using DSE Advanced Replication.

**Synopsis**

```
$ dse advrep [connection_options] [command] [sub_command] [sub_command_options]
```

The default port for DSE Advanced Replication is 9042.

**Table 130: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (( )) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Using dse advrep command line help

To view a listing of dse advrep commands:

```bash
$ dse advrep help
```

To view help for a specific command:

```bash
$ dse advrep help command [ sub_command ]
```

### Connection options

JMX authentication is supported by some dse commands. Other dse commands authenticate with the user name and password of the configured user. The connection option short form and long form are comma separated.

**Note:** You can provide authentication credentials in several ways, see [Credentials for authentication](#).

### General connection options:

- **--separator field_separator**
  The field separator for use with the **--no-pretty-print** command.

- **--verbose**
  Print verbose messages for command.

- **--verbose**
  Displays which arguments are recognized as Spark configuration options and which arguments are forwarded to the Spark shell.

- **--no-pretty-print**
  If not specified, data is printed using tabular output. If specified, data is printed as a comma separated list unless a separator is specified.

- **--cipher-suites ssl_cipherSuites**
  Specify comma-separated list of SSL cipher suites for connection to DSE when SSL is enabled. For example, --cipher-suites c1,c2,c3.

- **--host hostname**
  The DSE node hostname or IP address.

- **--jmx-port jmx_port**
The remote JMX agent port number. Default: 7199.

--jmx-pwd jmx_password
The password for authenticating with secure local JMX. If you do not provide a password, you are prompted to enter one.

--jmx-user jmx_username
The user name for authenticating with secure local JMX.

--kerberos-enabled true | false
Whether Kerberos authentication is enabled for connections to DSE. For example, --kerberos-enabled true.

--keystore-password keystore_password
Keystore password for connection to DSE when SSL client authentication is enabled.

--keystore-path ssl_keystore_path
Path to the keystore for connection to DSE when SSL client authentication is enabled.

--keystore-type ssl_keystore_type
Keystore type for connection to DSE when SSL client authentication is enabled. JKS is the type for keys generated by the Java keytool binary, but other types are possible, depending on user environment.

-p password
The password to authenticate for database access. Can use the DSE_PASSWORD environment variable.

--ssl
Whether SSL is enabled for connection to DSE. --ssl-enabled true is the same as --ssl.

--ssl-protocol ssl_protocol
SSL protocol for connection to DSE when SSL is enabled. For example, --ssl-protocol ssl4.

-t token
Specify delegation token which can be used to login, or alternatively, DSE_TOKEN environment variable can be used.

--truststore_password ssl_truststore_password
Truststore password to use for connection to DSE when SSL is enabled.

--truststore_path ssl_truststore_path
Path to the truststore to use for connection to DSE when SSL is enabled. For example, --truststore-path /path/to/ts.

--truststore-type ssl_truststore_type
Truststore type for connection to DSE when SSL is enabled. JKS is the type for keys generated by the Java keytool binary, but other types are possible, depending on user environment. For example, --truststore-type jks2.

-u username
User name of a DSE authentication account. Can use the DSE_USERNAME environment variable.

Examples
This connection example specifies that Kerberos is enabled and lists the replication channels:
Using DataStax Enterprise advanced functionality

```sh
$ dse advrep --host ip-10-200-300-138.example.lan --kerberos-enabled=true conf list
```

To use the server YAML files:

```sh
$ dse advrep --use-server-config conf list
```

To list output without pretty-print with a specified separator:

```
dse advrep --no.pretty-print --separator "|" destination list-conf
```

This output will result:

```
<table>
<thead>
<tr>
<th>destination</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mydest</td>
<td>addresses</td>
<td>192.168.200.100</td>
</tr>
<tr>
<td>mydest</td>
<td>transmission-enabled</td>
<td>true</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-ssl-cipher-suites</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA,SSL_RSA_WITH_RC4_128_MD5,TLS_EMPTY_RENEGOTIATION_INFO_SCSV</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-ssl-enabled</td>
<td>false</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-ssl-protocol</td>
<td>TLS</td>
</tr>
<tr>
<td>mydest</td>
<td>name</td>
<td>mydest</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-connect-timeout</td>
<td>15000</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-max-requests-per-connection</td>
<td>1024</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-connections-max</td>
<td>8</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-connections</td>
<td>1</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-compression</td>
<td>lz4</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-consistency-level</td>
<td>ONE</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-allow-remote-dcs-for-local-cl</td>
<td>false</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-used-hosts-per-remote-dc</td>
<td>0</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-read-timeout</td>
<td>15000</td>
</tr>
</tbody>
</table>
```

**dse advrep channel create**

Creates a replication channel for change data to flow between source clusters and destination clusters.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```sh
$ dse advrep channel create
   --source-keyspace  keyspace_name
   --source-table     source_table_name
   --source-id        source_id_name
   --source-id-column source_id_column_name
   --destination      destination
   --destination-keyspace destination_keyspace_name
   --destination-table destination_table_name
```
Using DataStax Enterprise advanced functionality

Table 131: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( {} ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

---

--source-keyspace keyspace_name (required)
The source cluster keyspace to replicate.

--source-table source_table_name (required)
The source table to replicate.

--source-id id
A unique identifier for all data that comes from a particular source node.
--source-id-column source_id
  The column that identifies the source id in the destination table.

--destination destination (required)
  The destination where the replication will be sent; the user names the destination.

--destination-keyspace keyspace_name
  The destination keyspace to which replication will be sent.

--destination-table table_name
  The destination table to which replication will be sent.

--fifo-order
  First in, first out channel (FIFO) replication order. Default.

--lifo-order
  Last in, last out (LIFO) channel replication order.

--collection-enabled (true | false)
  Whether to enable the source table for replication collection on creation.

--transmission-enabled (true | false)
  Whether to replicate data collector for the table to the destination.

--priority channel_priority
  The order in which the source table log files are transmitted.

Examples

To create a replication source channel:

```
$ dse advrep channel create --source-keyspace foo --source-table bar --source-id source1 --source-id-column source_id --destination mydest --destination-keyspace foo --destination-table bar --collection-enabled true --priority 1
```

with a result:

```
$ Created channel dc=Cassandra keyspace=foo table=bar to mydest
```

The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same. You can also set the source-id and source-id-column differently from the global setting.

**dse advrep channel update**

Updates a replication channel configuration.

A replication channel is a defined channel of change data between source clusters and destination clusters.

To update a channel, specify a new value for one or more options.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.


**Synopsis**

```
$ dse advrep channel update
    --source-keyspace  keyspace_name
    --source-table    source_table_name
    --source-id       source_id_name
    --source-id-column source_id_column_name
    --destination destination
    --destination-keyspace destination_keyspace_name
    --destination-table destination_table_name
[ --fifo-order | --lifo-order ]
[ --collection-enabled (true|false) ] [ --transmission-enabled (true|false) ]
[ --priority channel_priority ]
```

**Table 132: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks ( <code> </code> ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--source-keyspace **keyspace_name** (required)
The source cluster keyspace to replicate.

--source-table **source_table_name** (required)
The source table to replicate.

--source-id **id**
A unique identifier for all data that comes from a particular source node.

--source-id-column **source_id**
The column that identifies the source id in the destination table.

--destination **destination (required)**
The destination where the replication will be sent; the user names the destination.

--destination-keyspace **keyspace_name**
The destination keyspace to which replication will be sent.

--destination-table **table_name**
The destination table to which replication will be sent.

--fifo-order
First in, first out channel (FIFO) replication order. Default.

--lifo-order
Last in, last out (LIFO) channel replication order.

--collection-enabled ( true | false )
Whether to enable the source table for replication collection on creation.

--transmission-enabled ( true | false )
Whether to replicate data collector for the table to the destination.

--priority **channel_priority**
The order in which the source table log files are transmitted.

Examples

To update a replication source channel configuration:

```bash
$ dse advrep --verbose channel update --source-keyspace demo --source-table sensor_readings --destination mydest --lifo-order
```

with a result as seen using `dse advrep channel status`:

```bash
$ dse advrep channel status
```

<table>
<thead>
<tr>
<th>dc</th>
<th>keyspace</th>
<th>table</th>
<th>collecting</th>
<th>transmitting</th>
<th>replication order</th>
<th>priority</th>
<th>dest ks</th>
<th>dest table</th>
<th>src id</th>
<th>src id col</th>
<th>dest enabled</th>
<th>dest enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>demo</td>
<td>sensor_readings</td>
<td>true</td>
<td>true</td>
<td>LIFO</td>
<td>2</td>
<td>demo</td>
<td>sensor_readings</td>
<td>source1</td>
<td>source_id</td>
<td>mydest</td>
<td>true</td>
</tr>
</tbody>
</table>
```
Using DataStax Enterprise advanced functionality

The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same. You can also set the source-id and source-id-column differently from the global setting.

**dse advrep channel delete**

Deletes a replication channel.

A replication channel is a defined channel of change data between source clusters and destination clusters.

To delete a channel, you must specify source information and the destination and datacenter for the channel.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep channel delete
   --source-keyspace keyspace_name
   --source-table source_table_name
   --destination destination
   --data-center-id data_center_id
```

**Table 133: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ( {} ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt;&gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; ' </code></td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--source-keyspace keyspace_name (required)
   The source cluster keyspace to replicate.
--source-table source_table_name (required)
   The source table to replicate.
--destination destination (required)
   The destination where the replication will be sent; the user names the destination.
--data-center-id data_center_id
   The datacenter for this channel.

Examples

To create a replication source channel:

```
$ dse advrep channel delete --source-keyspace foo --source-table bar --destination mydest --data-center-id Cassandra
```

with a result:

```
Deleted channel dc=Cassandra keyspace=foo table=bar to mydest
```

The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same.

dse advrep channel pause

Pauses replication for a channel for change data to flow from a source cluster to a destination cluster.
A replication channel is a defined channel of change data between source clusters and destination clusters.

Pause collection of data or transmission of data between a source cluster and destination cluster.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep channel pause
   --source-keysapce keyspace_name
   --source-table source_table_name
   --destinations destination [ , destination ]
   --data-center-ids data_center_id [ , data_center_id ]
   --collection
   --transmission
```

**Table 134: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (( )) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--source-keyspace keyspace_name
The source cluster keyspace to replicate.

--source-table source_table_name
The source table to replicate.

--destinations destination [ , destination ]
The destinations where the replication are sent.

--data-center-ids data_center_id [ , data_center_id ]
The datacenters for this channel, which must exist.

--collection
No data for the source table is collected.

--transmission
No data for the source table is sent to the configured destinations.

Examples

To pause a replication source channel:

```
$ dse advrep channel pause --source-keyspace foo --source-table bar --destinations mydest --data-center-ids Cassandra
```

with a result:

```
Channel dc=Cassandra keyspace=foo table=bar collection to mydest was paused
```

The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same.

**dse advrep channel resume**

Resumes replication for a channel.

A replication channel is a defined channel of change data between source clusters and destination clusters.

A channel can resume either the collection or transmission of replication between a source cluster and destination cluster.
**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep channel resume
  --source-keyspace keyspace_name
  --source-table source_table_name
  --destinations destination [ , destination ]
  --data-center-ids data_center_id [ , data_center_id ]
  --collection
  --transmission
```

**Table 135: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>’Literal string’</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>’ &lt;schema&gt; ... &lt;/schema&gt; ’</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**--source-keyspace keyspace_name**  
The source cluster keyspace to replicate.

**--source-table source_table_name**  
The source table to replicate.

**--destinations destination [ , destination ]**  
The destinations where the replication are sent.

**--data-center-ids data_center_id [ , data_center_id ]**  
The datacenters for this channel, which must exist.

**--collection**  
No data for the source table is collected.

**--transmission**  
No data for the source table is sent to the configured destinations.

**Examples**

**To resume a replication source channel:**

```bash
$ dse advrep channel resume --source-keyspace foo --source-table bar --destinations mydest --data-center-ids Cassandra
```

with a result:

```
Channel dc=Cassandra keyspace=foo table=bar  collection to mydest was resumed
```

The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same.

**dse advrep channel status**

Prints status of a replication channel.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```bash
$ dse advrep channel status
   --data-center-id data_center_id
   --source-keyspace keyspace_name
   --source-table source_table_name
```
Using DataStax Enterprise advanced functionality

Table 136: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--source-keyspace keyspace_name
The source cluster keyspace to replicate.

--source-table source_table_name
The source table to replicate.

--destination destination
The destination where the replication will be sent; the user names the destination.

--data-center-id data_center_id
The datacenter for this channel.

Examples

To print the status of a replication channel:

```bash
$ dse advrep channel status --source-keyspace foo --source-table bar --destination mydest --data-center-id Cassandra
```

with a result:

<table>
<thead>
<tr>
<th>dc</th>
<th>keyspace</th>
<th>table</th>
<th>collecting</th>
<th>transmitting</th>
<th>replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>foo</td>
<td>bar</td>
<td>true</td>
<td>true</td>
<td>FIFO</td>
</tr>
<tr>
<td>2</td>
<td>foo</td>
<td>bar</td>
<td>source1</td>
<td>source_id</td>
<td>mydest</td>
</tr>
</tbody>
</table>

The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same.

**dse advrep channel truncate**

Truncates a channel to prevent replicating all messages that are currently in the replication log.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

Synopsis

```bash
$ dse advrep channel truncate
    --source-keyspace keyspace_name
    --source-table source_table_name
    --destinations destination [ , destination ]
    --data-center-ids data_center_id [ , data_center_id ]
```

**Table 137: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key: value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
| '
<schema> ... </schema>' | Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration. |
| @xml_entity='xml_entity_type' | Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files. |

---

**--source-keyspace keyspace_name**
- The source cluster keyspace to replicate.

**--source-table source_table_name**
- The source table to replicate.

**--destinations destination [ , destination ]**
- The destinations where the replication are sent.

**--data-center-ids data_center_id [ , data_center_id ]**
- The datacenters for this channel, which must exist.

### Examples

To truncate a replication channel to prevent replicating all messages that are currently in the replication log:
Using DataStax Enterprise advanced functionality

$ dse advrep channel status --source-keyspace foo --source-table bar --destinations mydest --data-center-ids Cassandra

with a result:

Channel dc=Cassandra keyspace=foo table=bar to mydest was truncated

The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same.

**dse advrep conf list**

Lists configuration settings for advanced replication.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

$ dse advrep conf list

**Table 138: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( {} ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity</code>=<code>xml_entity_type</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Examples

To list configuration settings:

```
$ dse advrep conf list
```

The result:

```
<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_log_file</td>
<td>auditLog</td>
</tr>
<tr>
<td>permits</td>
<td>8</td>
</tr>
<tr>
<td>audit_log_enabled</td>
<td>true</td>
</tr>
</tbody>
</table>
```

The number of permits is 8, audit logging is enabled, and the audit log file name is auditLog.

**dse advrep conf remove**

Removes configuration settings for advanced replication.

A replication channel is a defined channel of change data between source clusters and destination clusters.

Synopsis

```
$ dse advrep conf remove
   --separator  field_separator
   --audit-log-enabled true|false
   --audit-log-compression none|gzip
```
Using DataStax Enterprise advanced functionality

```
--audit-log-file log_file_name
--audit-log-max-life-span-mins number_of_minutes
--audit-log-rotate-mins number_of_minutes
--permits number_of_permits
--collection-max-open-files number_of_files
--collection-time-slice-count number_of_files
--collection-time-slice-width time_period_in_seconds
--collection-expire-after-write
--invalid-message-log
```

Table 139: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>

**--audit-log-compression true|false**
Enable or disable audit logging.

**--audit-log-compression none|gzip**
Enable audit log compression. Default: none

**--audit-log-file log_file_name**
Define the audit log filename.

**--audit-log-rotate-max number_of_minutes**
Define the maximum number of minutes for the audit log lifespan.

**--audit-log-rotate-mins number_of_minutes**
Define the number of minutes before the audit log will rotate.

**--permits number_of_permits**
Maximum number of messages that can be replicated in parallel over all destinations. Default: 1024

**--collection-max-open-files number_of_files**
Number of open files kept.

**--collection-time-slice-count number_of_files**
Specify the number of files which are open in the ingestor simultaneously.

**--collection-time-slice-width time_period_in_seconds**
Specify the time period in seconds for each data block ingested. Smaller time widths mean more files, whereas larger time widths mean larger files, but more data to resend on CRC mismatches.

**--collection-expire-after-write**
Specify if the collection expires after the write occurs.
Using DataStax Enterprise advanced functionality

**--invalid-message-log none|system_log|channel_log**

Specify where error information is stored for messages that could not be replicated. Default: channel_log

**Examples**

**To remove advanced replication configuration:**

```
$ dse advrep conf remove --permits 8
```

with a result:

```
Removed config permits
```

**dse advrep conf update**

Updates configuration settings for advanced replication.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep conf update
   --audit-log-enabled true|false
   --audit-log-compression none|gzip
   --audit-log-file log_file_name
   --audit-log-max-life-span-mins number_of_minutes
   --audit-log-rotate-mins number_of_minutes
   --permits number_of_permits
   --collection-max-open-files number_of_files
   --collection-time-slice-count number_of_files
   --collection-time-slice-width time_period_in_seconds
   --collection-expire-after-write
   --invalid-message-log
```

**Table 140: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>’ &lt;schema&gt; ... &lt;/schema&gt; ’</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--audit-log-compression true|false
Enable or disable audit logging.

--audit-log-compression none|gzip
Enable audit log compression. Default: none

--audit-log-file log_file_name
Define the audit log filename.

--audit-log-rotate-max number_of_minutes
Define the maximum number of minutes for the audit log lifespan.

--audit-log-rotate-mins number_of_minutes
Define the number of minutes before the audit log will rotate.

--permits number_of_permits
Maximum number of messages that can be replicated in parallel over all destinations. Default: 1024

--collection-max-open-files number_of_files
Number of open files kept.

--collection-time-slice-count number_of_files
Specify the number of files which are open in the ingestor simultaneously.

--collection-time-slice-width time_period_in_seconds
Using DataStax Enterprise advanced functionality

Specify the time period in seconds for each data block ingested. Smaller time widths mean more files, whereas larger timer widths mean larger files, but more data to resend on CRC mismatches.

--collection-expire-after-write
Specify if the collection expires after the write occurs.

--invalid-message-log none|system_log|channel_log
Specify where error information is stored for messages that could not be replicated. Default: channel_log

Examples

To update configuration settings:

```
$ dse advrep conf update --permits 8 --audit-log-enabled true --audit-log-file auditLog
```

with a result:

```
Updated audit_log_file from null to auditLog
Updated permits from null to 8
Updated audit_log_enabled from null to true
```

dse advrep destination create

Creates a replication destination.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

Synopsis

```
$ dse advrep destination create
  --name  destination_name
  --addresses address_name [ , address_name ]
  [ --transmission-enabled (true|false) ]
  --driver-user user_name
  --driver-pwd password
  --driver-used-hosts-per-remote-dc number_of_hosts
  --driver-connections number_of_connections
  --driver-connections-max number_of_connections
  --driver-local-dc data_center_name
  --driver-allow-remote-dcs-for-local-cl true|false
  --driver-consistency-level [ ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL_QUORUM|EACH_QUORUM|SERIAL|LOCAL_SERIAL|LOCAL_ONE ]
  --driver-compression [ snappy|lz4 ]
  --driver-connect-timeout timeout_in_milliseconds
  --driver-read-timeout timeout_in_milliseconds
  --driver-max-requests-per-connection number_of_requests
```
Using DataStax Enterprise advanced functionality

--driver-ssl-enabled true|false  
--driver-ssl-cipher-suites  
--driver-ssl-protocol  
--driver-ssl-keystore-path  
--driver-ssl-keystore-password  
--driver-ssl-keystore-type  
--driver-ssl-truststore-path  
--driver-ssl-truststore-password  
--driver-ssl-truststore-type

Table 141: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
--name **destination_name** *(required)*
   Designate the name of the destination.
--addresses **address_name** [, **address_name** ] *(required)*
   Identify the IP addresses of the destinations.
--transmission-enabled true | false
   Specify if data collector for the table should be replicated to the destination.
--driver-user **user_name**
   Specify the username for the destination.
--driver-pwd **password**
   Specify the password for the destination.
--driver-used-hosts-per-remote-dc **number_of_hosts**
   Define the number of hosts per remote datacenter that the datacenter-aware round robin policy considers available for use.
--driver-connections **number_of_connections**
   Specify the number of connections that the driver creates.
--driver-connections-max **number_of_connections**
   Specify the maximum number of connections that the driver creates.
--driver-local-dc **data_center_name**
   Specify the name of the datacenter that is considered local.
--driver-consistency-level ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL_QUORUM|EACH_QUORUM|SERIAL|LOCAL_SERIAL|LOCAL_ONE
   Specify the consistency level for the destination.
--driver-compression snappy|lz4
   Specify a compression algorithm for data files.
--driver-connect-timeout **timeout_in_milliseconds**
   Specify the timeout for the driver connection.
--driver-read-timeout **timeout_in_milliseconds**
   Specify the timeout for the driver reads.
--driver-max-requests-per-connection **number_of_requests**
   Specify the maximum number of requests per connection.
--driver-ssl-enabled true|false
   Enable SSL connection for the destination.
--driver-ssl-cipher-suites
   Specify the SSL cipher suites to use for driver connections.
--driver-ssl-protocol
   Specify the SSL protocol to use for driver connections.
--driver-keystore-path
   Specify the SSL keystore path to use for driver connections.
--driver-keystore-password
   Specify the SSL keystore password to use for driver connections.
--driver-keystore-type
   Specify the SSL keystore type to use for driver connections.
--driver-truststore-path
   Specify the SSL truststore path to use for driver connections.
--driver-truststore-password
   Specify the SSL truststore password to use for driver connections.
--driver-truststore-type
Specify the SSL truststore type to use for driver connections.

Examples

To update a replication destination:

```bash
$ dse advrep --verbose destination update --name mydest --addresses 10.200.182.148 --transmission-enabled true
```

with a result:

```
Destination mydest created
```

**dse advrep destination create**

Creates a replication destination.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```bash
$ dse advrep destination update
   --name  destination_name
   --addresses address_name [ , address_name ]
   [ --transmission-enabled (true|false) ]
   --driver-user user_name
   --driver-pwd password
   --driver-used-hosts-per-remote-dc
   --driver-connections
   --driver-connections-max
   --driver-local-dc
   --driver-allow-remote-dcs-for-local-cl true|false
   --driver-consistency-level [ ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL_QUORUM|EACH_QUORUM|SERIAL|LOCAL_SERIAL|LOCAL_ONE ]
   --driver-compression [ snappy|lz4 ]
   --driver-connect-timeout timeout_in_milliseconds
   --driver-read-timeout timeout_in_milliseconds
   --driver-max-requests-per-connection number_of_requests
   --driver-ssl-enabled true|false
   --driver-ssl-cipher-suites
   --driver-ssl-protocol
   --driver-ssl-keystore-path
   --driver-ssl-keystore-password
   --driver-ssl-keystore-type
   --driver-ssl-truststore-path
   --driver-ssl-truststore-password
   --driver-ssl-truststore-type
```
<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italicics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td><code>[ ]</code></td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td><code>( )</code></td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td><code>'Literal string'</code></td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>'&lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**--name destination_name (required)**
- Designate the name of the destination.

**--addresses address_name [ , address_name ] (required)**
- Identify the IP addresses of the destinations.

**--transmission-enabled true | false**
- Specify if data collector for the table should be replicated to the destination.

**--driver-user user_name**
- Specify the username for the destination.

**--driver-pwd password**
Specify the password for the destination.

```
--driver-used-hosts-per-remote-dc number_of_hosts
```

Define the number of hosts per remote datacenter that the datacenter-aware round robin policy considers available for use.

```
--driver-connections number_of_connections
```

Specify the number of connections that the driver creates.

```
--driver-connections-max number_of_connections
```

Specify the maximum number of connections that the driver creates.

```
--driver-local-dc data_center_name
```

Specify the name of the datacenter that is considered local.

```
--driver-consistency-level ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL_QUORUM|EACH_QUORUM|SERIAL|LOCAL_SERIAL|LOCAL_ONE
```

Specify the consistency level for the destination.

```
--driver-compression snappy|lz4
```

Specify a compression algorithm for data files.

```
--driver-connect-timeout timeout_in_milliseconds
```

Specify the timeout for the driver connection.

```
--driver-read-timeout timeout_in_milliseconds
```

Specify the timeout for the driver reads.

```
--driver-max-requests-per-connection number_of_requests
```

Specify the maximum number of requests per connection.

```
--driver-ssl-enabled true|false
```

Enable SSL connection for the destination.

```
--driver-ssl-cipher-suites
```

Specify the SSL cipher suites to use for driver connections.

```
--driver-ssl-protocol
```

Specify the SSL protocol to use for driver connections.

```
--driver-keystore-path
```

Specify the SSL keystore path to use for driver connections.

```
--driver-keystore-password
```

Specify the SSL keystore password to use for driver connections.

```
--driver-keystore-type
```

Specify the SSL keystore type to use for driver connections.

```
--driver-truststore-path
```

Specify the SSL truststore path to use for driver connections.

```
--driver-truststore-password
```

Specify the SSL truststore password to use for driver connections.

```
--driver-truststore-type
```

Specify the SSL truststore type to use for driver connections.

Examples

To create a replication destination:

```
$ dse advrep --verbose destination update --name mydest --addresses 10.200.182.148 --driver-consistency-level ANY
```

with a result:
Destination mydest updated
Updated addresses from 10.200.182.148 to 10.200.182.1648
Updated driver_consistency_level from ONE to ANY
Updated name from mydest to mydest

Notice that any option included causes a change to occur.

**dse advrep destination delete**

Deletes a given replication destination.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep destination delete
   --name  destination_name
```

**Table 143: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity=’xml_entity_type‘</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--name destination_name (required)

Designate the name of the destination.

**Examples**

**To delete a replication destination:**

```
$ dse advrep destination delete --name mydest
```

with a result:

```
Destination mydest removed
```

**dse advrep destination list**

Lists all replication destinations.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep destination list
```

**Table 144: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Examples

#### To list all replication destinations:

$ dse advrep destination list

with a result:

<table>
<thead>
<tr>
<th>name</th>
<th>enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>mydest</td>
<td>true</td>
</tr>
</tbody>
</table>
**Using DataStax Enterprise advanced functionality**

---

**dse advrep destination list-conf**

Lists all configuration for a given replication destination.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep destination list-conf
   --separator field_separator
   --name destination_name
```

**Table 145: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>Literal string</td>
<td>Single quotation (   ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>&lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**--name destination_name (required)**

Designate the name of the destination.

**Examples**

**To list the configuration for a replication destination:**

```
$ dse advrep destination list-conf --name mydest
```

with a result:

```
<table>
<thead>
<tr>
<th>destination</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mydest</td>
<td>addresses</td>
<td>10.200.180.162</td>
</tr>
<tr>
<td>mydest</td>
<td>transmission-enabled</td>
<td>true</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-ssl-cipher-suites</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_DHE_DSS_WITH_AES_256_CBC_SHA256,</td>
<td></td>
</tr>
</tbody>
</table>
```
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA, |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, |
| TLS_RSA_WITH_AES_256_CBC_SHA, |
| TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA, |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA, |
| TLS_DHE_DSS_WITH_AES_256_CBC_SHA, |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256, |
| TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_DHE_DSS_WITH_AES_128_CBC_SHA256, |
| TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, |
| TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, |
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, |
| TLS_RSA_WITH_AES_256_GCM_SHA384, |
| TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384, |
| TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384, |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA, |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA, |
| TLS_DHE_DSS_WITH_AES_128_CBC_SHA, |
| TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, |
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, |
| TLS_RSA_WITH_AES_256_GCM_SHA384, |
| TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384, |
| TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384, |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA, |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA, |
| TLS_DHE_DSS_WITH_AES_128_CBC_SHA, |
Using DataStax Enterprise advanced functionality

| TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, |
| TLS_DHE_DSS_WITH_AES_256_GCM_SHA384, |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, |
| TLS_RSA_WITH_AES_128_GCM_SHA256, |
| TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256, |
| TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256, |
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, |
| TLS_DHE_DSS_WITH_AES_128_GCM_SHA256, |
| TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA, |
| TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA, |
| SSL_RSA_WITH_3DES_EDE_CBC_SHA, |
| TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA, |
| TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA, |
| SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA, |
| SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA, |
| TLS_ECDHE_ECDSA_WITH_RC4_128_SHA, |
| TLS_ECDHE_RSA_WITH_RC4_128_SHA, |
| SSL_RSA_WITH_RC4_128_SHA, |
| TLS_ECDH_ECDSA_WITH_RC4_128_SHA, |
| TLS_ECDH_RSA_WITH_RC4_128_SHA, |
| SSL_RSA_WITH_RC4_128_MD5, |
| TLS_EMPTY_RENEGOTIATION_INFO_SCSV |

<table>
<thead>
<tr>
<th>mydest</th>
<th>driver-ssl-enabled</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>mydest</td>
<td>driver-ssl-protocol</td>
<td>TLS</td>
</tr>
<tr>
<td>mydest</td>
<td>name</td>
<td>mydest</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>mydest</th>
<th>driver-connect-timeout</th>
<th>15000</th>
</tr>
</thead>
<tbody>
<tr>
<td>mydest</td>
<td>driver-max-requests-per-connection</td>
<td>1024</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-connections-max</td>
<td>8</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-connections</td>
<td>1</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-compression</td>
<td>lz4</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-consistency-level</td>
<td>ONE</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-allow-remote-dcs-for-local-cl</td>
<td>false</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-used-hosts-per-remote-dc</td>
<td>0</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-read-timeout</td>
<td>15000</td>
</tr>
</tbody>
</table>

**dse advrep destination remove-conf**

Removes configuration for a destination.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep destination remove-conf
   --name destination_name
   --addresses address_name [ , address_name ]
   [ --transmission-enabled (true|false) ]
   --driver-user user_name
   --driver-pwd password
   --driver-used-hosts-per-remote-dc
   --driver-connections
   --driver-connections-max
   --driver-local-dc
   --driver-allow-remote-dcs-for-local-cl true|false
   --driver-consistency-level [ ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL_QUORUM|EACH_QUORUM|SERIAL|LOCAL_SERIAL|LOCAL_ONE ]
```
Using DataStax Enterprise advanced functionality

```
--driver-compression [ snappy|lz4 ]
--driver-connect-timeout timeout_in_milliseconds
--driver-read-timeout timeout_in_milliseconds
--driver-max-requests-per-connection number_of_requests
--driver-ssl-enabled true|false
--driver-ssl-cipher-suites
--driver-ssl-protocol
--driver-ssl-keystore-path
--driver-ssl-keystore-password
--driver-ssl-keystore-type
--driver-ssl-truststore-path
--driver-ssl-truststore-password
--driver-ssl-truststore-type
```

Table 146: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

```
--name destination_name (required)
  Designate the name of the destination.
--addresses address_name [, address_name] (required)
  Identify the IP addresses of the destinations.
--transmission-enabled true | false
  Specify if data collector for the table should be replicated to the destination.
--driver-user user_name
  Specify the username for the destination.
--driver-pwd password
  Specify the password for the destination.
--driver-connections number_of_connections
  Specify the number of connections that the driver creates.
--driver-connections-max number_of_connections
  Specify the maximum number of connections that the driver creates.
--driver-local-dc data_center_name
  Specify the name of the datacenter that is considered local.
--driver-consistency-level ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL_QUORUM|EACH_QUORUM|SERIAL|LOCAL_SERIAL|LOCAL_ONE
  Specify the consistency level for the destination.
--driver-compression snappy|lz4
  Specify a compression algorithm for data files.
--driver-connect-timeout timeout_in_milliseconds
  Specify the timeout for the driver connection.
--driver-read-timeout timeout_in_milliseconds
  Specify the timeout for the driver reads.
--driver-max-requests-per-connection number_of_requests
  Specify the maximum number of requests per connection.
--driver-ssl-enabled true|false
  Enable SSL connection for the destination.
--driver-ssl-cipher-suites
  Specify the SSL cipher suites to use for driver connections.
--driver-ssl-protocol
  Specify the SSL protocol to use for driver connections.
--driver-keystore-path
  Specify the SSL keystore path to use for driver connections.
--driver-keystore-password
  Specify the SSL keystore password to use for driver connections.
--driver-keystore-type
  Specify the SSL keystore type to use for driver connections.
--driver-truststore-path
```
Using DataStax Enterprise advanced functionality

Specify the SSL truststore path to use for driver connections.

--driver-truststore-password
Specify the SSL truststore password to use for driver connections.

--driver-truststore-type
Specify the SSL truststore type to use for driver connections.

Examples

To remove configuration for a replication destination:

```
$ dse advrep --verbose destination remove-conf --transmission-enabled true
```

with a result:

```
Removed config transmission-enabled
```

**dse advrep metrics list**

Lists advanced replication JMX metrics.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep metrics list
     --metric group metric_group
     --metric-type metric_type
```

**Table 147: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**--metric group metric_group**

The source cluster keyspace for which to show count.

**--metric-type metric_type**

The source table for which to show count.

Examples

To display the JMX metrics:

```bash
$ dse advrep --host localhost --port 7199 metrics list
```

with a result:

<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>MessagesDelivered</td>
<td>3000</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>CommitLogsToConsume</td>
<td>1</td>
</tr>
<tr>
<td>Tables</td>
<td>MessagesReceived</td>
<td>3000</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessageAddErrors</td>
<td>0</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>CommitLogsDeleted</td>
<td>0</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Count</th>
<th>RateUnit</th>
<th>MeanRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>FifteenMinuteRate</td>
<td>OneMinuteRate</td>
<td>FiveMinuteRate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesAdded</td>
<td>3000</td>
<td>events/second</td>
<td>0.02079053277209345E-28</td>
</tr>
<tr>
<td></td>
<td>MessagesDeleted</td>
<td>0</td>
<td>events/second</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>MessagesAcknowledged</td>
<td>3000</td>
<td>events/second</td>
<td>0.02079053277209345E-28</td>
</tr>
<tr>
<td></td>
<td>CommitLogMessagesRead</td>
<td>30740</td>
<td>events/second</td>
<td>0.21303361656215317</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission</td>
<td>AvailablePermits</td>
<td>30000</td>
</tr>
</tbody>
</table>

To display JMX metrics for a particular metric group:

```
$ dse advrep --host localhost --port 7199 metrics list --metric-group Tables
```

with a result:

<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>MessagesDelivered</td>
<td>3000</td>
</tr>
<tr>
<td>Tables</td>
<td>MessagesReceived</td>
<td>3000</td>
</tr>
</tbody>
</table>

To display JMX metrics for a particular metric type:

```
$ dse advrep --host localhost --port 7199 metrics list --metric-type MessagesAdded
```

with a result:

<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Count</th>
<th>RateUnit</th>
<th>MeanRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>FifteenMinuteRate</td>
<td>OneMinuteRate</td>
<td>FiveMinuteRate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Page 848
Using DataStax Enterprise advanced functionality

| ReplicationLog | MessagesAdded | 3000 | events/second | 0.020827685267120057 |
| 6.100068258619765E-28 | 2.964393875E-314 | 5.515866021410421E-82 |

---

dse advrep replog count

Returns the messages that have not been replicated.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep replog count
   --source-keyspace keyspace_name
   --source-table source_table_name
   --destination destination
   --data-center-id data_center_id
```

**Table 148: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (( )) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td><em>Literal string</em></td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (<code>&lt; &gt;</code>) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (<code>;</code>) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens (<code>--</code>). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; ' </code></td>
<td>Search CQL only: Single quotation marks (<code>'</code>) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

```
--source-keyspace keyspace_name (required)
  Define the source cluster keyspace for which to show count..
--source-table source_table_name (required)
  Define the source table for which to show count.
--destination destination (required)
  Define the destination for which to show count.
--data-center-id data_center_id
  Define the data center for which to show the count.
```

Examples

To verify the record count held in a replication log:

```
$ dse advrep replog count --destination mydest --source-keyspace foo --source-table bar
```

with a result:

```
2
```

**dse advrep replog analyze-audit-log**

Reads the audit log and prints a summary.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.
Synopsis

$ dse advrep replog analyze-audit-log
   --file audit_log_filename

Table 149: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--file audit_log_filename
The audit log file to create.
Using DataStax Enterprise advanced functionality

Examples

To analyze the data in a replication log:

```bash
$ dse advrep replog analyze-audit-log --file auditLog
```

with a result:

```
foo, bar : inserts = 1000, insertErrors = 0
foo, bar : reads = 1000, sent = 0, deletes = 1000, readingErrors = 0,
        deletingErrors = 0
```
Developing applications with DataStax drivers

The DataStax drivers are the primary resource for application developers creating solutions using DataStax Enterprise.

Prerequisites

Before building an application, learn about the DSE architecture and review the DataStax Academy tutorials. It’s especially important to understand data modeling (page 434) for DSE. This foundation helps ensure the application performs to its fullest potential.

Programming language options

There are DataStax drivers for a range of programming languages. The drivers have a set of core functionalities common for all the drivers. Please consult the individual driver documentation pages for detailed information on the driver’s features and API.

Table 150: Development status of DataStax drivers

<table>
<thead>
<tr>
<th>Actively developed drivers</th>
<th>Maintenance mode drivers$^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++ driver</td>
<td>PHP driver</td>
</tr>
<tr>
<td>C# driver</td>
<td>C# DSE Graph Extension</td>
</tr>
<tr>
<td>Java driver (DSE Graph Extension included)</td>
<td></td>
</tr>
<tr>
<td>Node.js driver</td>
<td>Node.js DSE Graph Extension</td>
</tr>
<tr>
<td>Python driver</td>
<td>Python Graph Extension</td>
</tr>
</tbody>
</table>

DSE and open source drivers

DataStax develops the drivers for DataStax Enterprise (DSE drivers) and for Apache Cassandra™ (OSS drivers). Use the DSE drivers when developing applications for DataStax Enterprise.

The DSE drivers have all of the functionality of the OSS drivers as well as additional features for DSE-specific features such as Unified Authentication and DSE Graph. By starting with the DSE drivers, it is much easier to add DSE optimizations and features to applications.

1. Supported by DataStax, but only critical bug fixes will be included in new versions.

---

1. Supported by DataStax, but only critical bug fixes will be included in new versions.
Developing applications with DataStax drivers

A list of common driver features are available to all driver languages:

- CQL support
- Synchronous and Asynchronous (page 874)
- Address translation
- Load balancing policies (page 861)
- Retry policies (page 863)
- Reconnection policies (page 865)
- Connection pooling (page 863)
- Auto node discovery
- SSL (page 860)
- Compression
- Query builder
- Object mapper (page 882)

DSE workloads

DataStax Enterprise supports several workload types:

- Transactional (Cassandra-only)
- Search
- Analytics
- Graph

The DataStax drivers support most of these workloads natively, simplifying applications that run Transactional, Search, and Graph queries alongside one another.

The DataStax drivers in this guide do not support DSE Analytics queries. The access patterns in analytics use cases are different than the access patterns used by the other workloads. The ODBC and JDBC drivers for DSE Analytics and the DSE Spark Cassandra Connector are typically used with the DSE drivers for analytics applications.
For more about workloads, see Working with multi-workload clusters *(page 868).*

Getting started

Each DataStax driver is hosted on the common distribution channel for the particular language. Visit the DataStax Downloads for direct links to download locations and the DataStax Documentation for installation information.

<table>
<thead>
<tr>
<th>Table 151: Drivers for each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
<tr>
<td>Binaries hosted on datastax.com</td>
</tr>
</tbody>
</table>

Best practices for DataStax drivers

These rules and recommendations improve performance and minimize resource utilization in applications that use DataStax drivers.

Use a single session object per application

Create and reuse a single session for the entire lifetime of an application. Sessions are expensive to create because they initialize and maintain connection pools to every node in a cluster. A single driver session can handle thousands of queries concurrently. Use a single driver session to execute all the queries in an application. Using a single session per cluster allows the drivers to coalesce queries destined for the same node, which can significantly reduce system call overhead.

**Note:** It is not recommended that a session be created per keyspace. Applications should use fully qualified keyspaces in their query strings or explicitly set the keyspace on statement objects. Applications should use fully qualified keyspaces in their query strings or explicitly set the keyspace on statement objects.

<table>
<thead>
<tr>
<th>Table 152: API references for session</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

Use a single cluster object per physical cluster

Create a single cluster object per physical cluster. Cluster objects are relatively expensive to create because they maintain a control connection to a given cluster. Creating more than one cluster object per physical cluster duplicates these resources unnecessarily.

**Important:** This rule doesn’t apply to the C/C++ and OSS Java 4.X / DSE Java 2.x drivers because each session maintains its own cluster state. However, the single session per application rule still applies.
Note: The Node.js driver combines the concepts of session and cluster into a single Client interface.

<table>
<thead>
<tr>
<th>C/C++</th>
<th>C#</th>
<th>Java</th>
<th>Node.js</th>
<th>PHP</th>
<th>Python</th>
<th>Ruby</th>
</tr>
</thead>
</table>

Run queries asynchronously for higher throughput

Use the driver’s asynchronous APIs to achieve maximum throughput. The asynchronous APIs provide execution methods that return immediately without blocking the application’s progress, allowing a single application thread to run many queries concurrently. Asynchronous execution methods return future objects that can be used by the application to obtain query results and errors if they occur. Running many queries concurrently allows applications to optimize their query processing, improves the driver’s ability to coalesce query requests, and maximizes use of server-side resources.

Figure 83: Asynchronous queries

Use prepared statements for frequently run queries

Prepare queries that are used more than once. Preparing queries allows the server and driver to reduce the amount of processing and network data required to run a query. For prepared statements, the server parses the query once and it is then cached for the lifetime of an
Developing applications with DataStax drivers

application. The server also avoids sending response metadata after the initial prepare step, which reduces the data sent over the network and the corresponding client side processing.

<table>
<thead>
<tr>
<th>Table 154: API references for prepared statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

Explicitly set the local datacenter when using a datacenter-aware load balancing

When using a datacenter-aware load balancing policy, your application should explicitly set the local datacenter instead of allowing the drivers to infer the local datacenter from the contact points. If the driver chose the wrong local datacenter, it increases cross-datacenter traffic, which is often higher latency and monetarily expensive than inter-datacenter traffic. Setting the local datacenter explicitly eliminates the chance that the driver will choose the wrong local datacenter.

When configuring a driver connection, it is easy to include contact points in remote datacenters or invalid datacenters. For example, an application might include contact points for an internal datacenter used during testing. Explicitly setting the local datacenter avoids these types of errors.

<table>
<thead>
<tr>
<th>Table 155: API references for load balancing</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

Avoid delete workloads and writing nulls

In DSE and Cassandra, a tombstone is a marker that indicates that table data is logically deleted. DSE and Cassandra store updates to tables in immutable SSTable files to maintain throughput and avoid reading stale data. Deleted data, time-to-live (TTL) data, and null values will create tombstones, which allows the database to reconcile the logically deleted data with new queries across the cluster. While tombstones are a necessary byproduct of a distributed database, limiting the number of tombstones and avoiding tombstone creation increases database and application performance.

Deletes can often be avoided through data modeling techniques. Nulls can be avoided with proper query construction. For more details on tombstones, see this article on DataStax Academy.

Heavy deletes and nulls use extra disk space and decrease performance on reads. Tombstones can cause warnings and log errors.

For example, in the following schema:

```sql
CREATE TABLE test_ks.my_table_compound_key (  
    primary_key text,  
    clustering_key text,  
    regular_col text,  
    PRIMARY KEY (primary_key, clustering_key)  
)```
Developing applications with DataStax drivers

This query results in no tombstones for `regular_col`:

```
INSERT INTO my_table_compound_key (primary_key, clustering_key)
VALUES ('pk1', 'ck1');
```

However this query results in a tombstone for `regular_col`:

```
INSERT INTO my_table_compound_key (primary_key, clustering_key,
   regular_col)
VALUES ('pk1', 'ck1', null);
```

Connecting to DSE clusters using DSE drivers

All the DSE drivers share common features for connecting to DSE clusters.

**Authentication in DSE drivers**

The DSE drivers support DSE Unified Authentication.

To simplify and standardize DataStax Enterprise security features, Unified Authentication was introduced in DSE 5.0 to give operators and developers a single, flexible security model. A single DSE server can accept multiple forms of authentication:

- internal username and password authentication
- LDAP or Active Directory authentication
- Kerberos authentication

Clients with different levels of access can use varying authentication schemes to connect to the same DSE server.

DSE’s Unified Authentication provides the ability to assign users to roles and to tie access to database resources based on that role. DSE also allows users to login and execute using proxy roles. All of these features are enabled directly in the DataStax drivers, with built-in classes to enable the desired security configuration.

The DataStax drivers ship with built-in authentication providers that provide the necessary utilities to connect to a secured DSE cluster.

By default, DSE has no authentication service enabled. This makes it easy to get started but is not intended for production deployments. Before configuring the driver to use authentication, enable the desired security schemes within DSE and create users and roles in the database.

Authenticating with internal or LDAP usernames and passwords

The drivers use a plain text authentication provider to perform both DSE’s internal and LDAP or Active Directory authentication. The driver will send a plain text username and password to the server, which will authenticate to the underlying configured scheme.
Because these mechanisms transmit credentials in clear text in the native protocol, they should always be used in conjunction with client-server transport encryption (page 860).

<table>
<thead>
<tr>
<th>Table 156: References for plain text authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

Authenticating with Kerberos

Each driver extends authentication providers to perform DSE’s Kerberos authentication.

Kerberos uses a krb5.conf file for configuration. Credentials can reside in a ticket cache or a keytab.

**Important:** The C# driver uses the Microsoft security framework called SSPI, which supports Kerberos. When using the C# driver, a Kerberos ticket is obtained during system login and the driver uses that ticket to authenticate. The rest of this section does not apply to the DSE C# driver.

The krb5.conf configuration file

DSE driver authentication against a Kerberos-enabled DSE cluster requires a krb5.conf file containing the Kerberos configuration settings. This file may be in the node’s /etc directory. If it is not, contact your Kerberos system administrator to locate the file.

To reference a krb5.conf file in a non-default location, set the KR5_CONFIG environment variable to the location of krb5.conf. Kerberos command line tools such as kinit, klist, and kdestroy respect this variable.

All drivers except Java and C# respect this environment variable. Java clients must set the java.security.krb5.conf system property to the path to the krb5.conf file at startup. The C# driver uses SSPI, which doesn’t use krb5.conf.

The Kerberos ticket cache

To use the Kerberos ticket cache, use the kinit command to authenticate with the Kerberos server and obtain a ticket. Verify the ticket cache contains a ticket for the successful authentication with the klist command. Once you verify there is a ticket in the ticket cache, an application that has been configured to use the Kerberos authentication provider is ready to run. If multiple principals have valid tickets in the ticket cache and no principal was specified in the application, the driver will arbitrarily choose one and use that ticket.

Kerberos keytabs

A keytab can be used to authenticate with Kerberos without requiring any additional credentials or a password. Keytab files must have their permissions set properly to restrict access. The permissions should be set to allow the application user to access the keytab.
Proxy login and execution

Proxy login allows users to authenticate using a fixed set of authentication credentials but allow authorization of resources based on another user role. To use proxy login, see the documentation for the individual drivers.

<table>
<thead>
<tr>
<th>Table 157: References for proxy authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

Like proxy login, proxy execute allows users to authenticate using a fixed set of authentication credentials but execute requests based on another user role. To use proxy execute, see the documentation for the individual drivers.

<table>
<thead>
<tr>
<th>Table 158: References for proxy execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

Using SSL in DSE drivers

DSE drivers support SSL-encrypted connections between the driver and server.

By default DataStax Enterprise (DSE) is configured to communicate with clients using an unencrypted binary protocol. This is convenient for getting started with DSE but is usually not suitable for production environments, especially those with clients communicating with DSE nodes over the public internet.

SSL in DSE can be configured different ways depending on the security requirements of the deployment. All configurations result in encrypted communication between the client and server.

SSL allows for different levels of identity verification:

- No identity verification (page 861) between the client and server.
- The client verifies the identity of the server (page 861).
- The server verifies the identity of the client (page 861).

A typical SSL workflow consists of the following stages:

1. The client opens a TCP connection to the server on the configured SSL port.
2. An SSL handshake is initialized by the server, sending its public key (or certificate) to the client.
3. The client uses that public key certificate to generate an encrypted session key and sends it back to the server.
4. The server decrypts the message using its private key and retrieves the session key.
5. All communication from that point on is encrypted using that session key.

### Table 159: SSL certificate instructions

<table>
<thead>
<tr>
<th>C/C++</th>
<th>C#</th>
<th>Java</th>
<th>Node.js*</th>
<th>PHP</th>
<th>Python</th>
<th>Ruby</th>
</tr>
</thead>
</table>

#### No identity verification

Clients with DSE drivers communicating securely with DSE servers should validate the server's identity. While most drivers support creating SSL connections to the server without identity verification, it is not recommended for production deployments.

When a secure browser contacts a web server, the browser verifies the identity of the server before sending it requests in case an attacker is masquerading as the web server. Secure communication to a bad actor defeats the purpose of configuring secure communication between the browser and web server in the first place.

#### Client verifies server

To verify the identity of a server, the driver must be configured with a list of trusted certificate authorities (CAs). When the driver receives the server's SSL certificate during the SSL handshake, it checks that the certificate was signed by one of the registered CAs. If the certificate was not signed by a registered CA, the client checks that the signer was signed by one of the registered CAs. It continues through the signers until it finds one that is in the client's list of trusted CAs.

If the client doesn't find a registered CA, identity verification fails.

#### Server verifies client

A server is configured to verify the identity of a client by setting the `require_client_auth` to true under `client_encryption_options` in `cassandra.yaml`. This scenario requires clients be configured with their own certificates to send to the server upon request during the SSL handshake.

### Load balancing with DSE drivers

The DataStax drivers control the distribution of the incoming load across the DSE cluster. The load balancing policy determines the node in the cluster to be the coordinator for executing a given query. Determine the load balancing policy during application development because the policy determines the nodes that will have connection pools created and maintained by the driver. For most deployments, use the default load balancing policy.

#### Table 160: Load balancing policy configuration

<table>
<thead>
<tr>
<th>C/C++</th>
<th>C#</th>
<th>Java</th>
<th>Node.js</th>
<th>PHP</th>
<th>Python</th>
<th>Ruby</th>
</tr>
</thead>
</table>
Coordinator selection

Each time a query is executed, the load balancing policy returns a query plan that determines which hosts are eligible to receive the query. The driver uses the first host on the list to execute the request, leaving the successive hosts for retry (page 863) and speculative execution (page 875).

Token awareness

Token awareness is common across all drivers. Token awareness uses the primary key information for a given query and parameters to retrieve the replica nodes. By selecting replicas, this policy guarantees that the selected coordinator for the query owns the data that will be written or retrieved, thereby avoiding an extra network connection on the server side.

The key is automatically calculated for prepared statement (page 856) executions to obtain accurate query routing.

Datacenter awareness

In some use cases, application requests should be limited to a given datacenter to ensure the data is returned to the user as efficiently as possible.

In a global application, users in North America should have their requests directed to a datacenter in North America. Users in Europe should have their requests routed to a datacenter in Europe. To accomplish this, specify a local datacenter in the load balancing policy so that the driver routes this query more efficiently.

If the requests to the local datacenter do not succeed, many of the drivers support using remote datacenter hosts for queries. Though this may appear to be a way to enact datacenter failover, this feature often leads to unexpected latencies and behaviors in the application. For a detailed explanation, see this “Cassandra: Local_Quorum Should Stay Local” blog post.

Default load balancing policy

The DataStax drivers integrate the best practices of token awareness and datacenter awareness into the default load balancing policy. Specifically, the default policy will retrieve the replicas for a given token and return a list of hosts containing the replicas in the local datacenter first, followed by the rest of nodes in the specified local datacenter. Using a load distributing algorithm, the default load balancing policy fairly distributes the load across the replica nodes.

Customizing load balancing

If custom routing and load balancing are required in an application, the existing load balancing interface can be extended. Custom load balancing is provided through whitelist and blacklist load balancing policies. Refer to the individual driver documentation for information on whitelist and blacklist load balancing policies. Customizing the load balancing
policy is an advanced topic. Study the existing policies before implementing a custom load balancing policy.

**Connection pooling**

The DataStax drivers maintain a pool of connections to each of the DataStax nodes selected by the load balancing policy (page 861). By default, the driver instance creates one connection to each of the local datacenter hosts in the default load balancing policy.

Connection pooling is separate from the initial contact points. Initial contact points are supplied to the driver instance. Those contact points are used only to establish the control connection to discover the DSE cluster topology.

Connection pools are accessed asynchronously. Multiple requests can be submitted on a single connection simultaneously. For most workloads, it is recommended to use one long-lived connection from the driver to each DSE server. If the default connection pool settings are not adequate, the number of connections per host and the maximum number of simultaneous requests per connection are configurable. The DSE binary protocol allows up to 32768 concurrent requests per connection.

<table>
<thead>
<tr>
<th>Table 161: Connection pools configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

**Retry policies**

Retry policies allow the DataStax drivers to automatically retry a request upon encountering specific types of server errors:

- read timeouts (page 889)
- write timeouts (page 888)
- unavailable exceptions (page 887)

In these scenarios, a node is designated as a coordinator for the request by the load balancing policy (page 861). The coordinator routes the request to the replicas and returns the response to the driver.

Each DataStax driver implements a default retry policy.

<table>
<thead>
<tr>
<th>Table 162: Retry policies for drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

Along with the driver-included retry policies, some drivers allow extended retry policies to implement custom behavior based on read timeout, write timeout, unavailable exceptions, and request errors.

2. Does not have ability to customize connections per host, single thread/GIL is limiting factor.
3. Search page for connections_per_local_node and requests_per_connection.
Developing applications with DataStax drivers

Default retry policy

The default retry policy retries a request when it is safe to do so while preserving the consistency level of the original request. Use this policy for most deployments.

Read timeout

If the number of replicas that reply is greater than or equal to the number of required responses per the consistency level, the default retry policy retries the request. In all other cases, it returns an error.

Figure 84: Read timeout

Write timeout

If the request is a logged batch request and fails to write to the batch log, the default retry policy retries the request. In all other cases, it returns an error.

Figure 85: Write timeout

Unavailable errors

If the request encounters an unavailable error, the default retry policy retries the request using the next host in the load balancing policy.

Figure 86: Unavailable errors
Fall-through retry policy

The fall-through retry policy never retries or ignores a failed request. In all cases, the fall-through retry policy returns an error. Use this policy for applications that need to implement their own business logic to handle retrying a request.

Logging retry policy

The logging retry policy is intended to be a parent policy for another retry policy implementation and only logs the retry decision made by its child policy. This policy is typically used to debug driver retry behavior.

Reconnection policies

Reconnection policies allow the DataStax drivers to automatically reestablish a connection to a node that was previously marked as down. A node can be marked down by the server's gossip process or as a result of an idle connection timeout. Status changes are passed along the control connection back to the driver.

All of the drivers offer a standard reconnection policy. Some drivers offer additional reconnection policies:

- **Constant**: The driver waits a constant amount of time between each reconnection attempt.
- **Exponential**: The driver waits exponentially longer between each reconnection attempt.
- **Fixed**: The driver waits a different amount of time between each reconnection attempt.

**Note**: Drivers that offer the exponential reconnection policy use that policy as their default. For other drivers, the constant reconnection policy is the default policy.

<table>
<thead>
<tr>
<th>Table 163: Reconnection policies for drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

DSE 6.7 Developer Guide (Latest version)
Gossip reconnection

The control connection for the drivers listens to push notifications from the DSE server cluster. When a node is marked up, all scheduled reconnections are canceled and a new connection to that node is established.

Figure 89: Gossip reconnection policy example

Idle disconnect and reconnection

To prevent intermediate network devices like routers and firewalls from disconnecting the drivers from a node, an OPTIONS request is sent to a connection at a constant interval, also known as a heartbeat. If the connection becomes idle and the node does not respond to the heartbeat in a given amount of time, the node is marked down. Once this occurs, the driver waits a specified amount of time based on the reconnection policy before attempting to reconnect to the node.
Execution profiles

Execution profiles allows a single session to run different types of query workloads, each with its own settings. An execution profile encapsulates a group of settings that can then be associated with individual queries. This provides a convenient way to group queries based on the following settings:

- Request timeout
- Consistency level
- Load balancing policy
- Retry policy
- Speculative execution policy

**Note:** This is not an exhaustive list and can vary by driver implementation.

<table>
<thead>
<tr>
<th>Table 164: Execution policies for drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

Figure 90: Idle disconnect and reconnection policy example
Using execution profiles

Create, configure, and register an execution profile object by name using the cluster or session objects, depending on the driver. Associate the named execution profile object with a query by providing the name to the query execution method or setting it on a statement object.

The method of associating the profile name with the query execution can vary by driver implementation. See the language-specific documentation for additional details on using execution profiles.

Submitting queries with DataStax drivers

The ability to submit queries and receive results is the core functionality of DataStax drivers.

Working with multi-workload clusters

There are several different ways to query the data stored in DataStax Enterprise. These access patterns are enabled by what are known as workloads in DSE. The DSE drivers can query clusters with different workload types. Below are the different workloads that DSE offers to extend a multi-model experience to the developer.

- DSE Core: transactional, typically through standard Cassandra CQL queries
- DSE Search: filtering, typically through Lucene queries
- DSE Analytics: computation, typically through Spark jobs
- DSE Graph: relationships, typically through TinkerPop traversals

When developing an application, each workload type requires different techniques to effectively leverage the use case covered by the workload.

Before creating applications, study and understand the DSE deployment architecture. Developers need to know which datacenters make up a DSE cluster and the supported workloads to direct the different types of queries to the appropriate datacenter.

For example, to use a solr_query in the application, the target datacenter must have DSE Search enabled. To execute graph traversals from the application, the connected datacenter must have DSE Graph enabled. Transactional queries can typically be made against any datacenter, as this core functionality is present for all workloads.

The DataStax drivers expose load balancing policies (page 861) as a means to steer from the application. The load balancing policy can be supplied in the execution profiles (page 867) or while configuring the driver cluster and session objects. Use the datacenter-aware policy to restrict queries to a specified datacenter.

Execution profiles

For the drivers that support execution profiles (page 867), define separate profiles for the different workloads used in the application. For example, if there are two datacenters, one with DSE Search enabled and one with DSE Core only, use a SearchExecutionProfile to direct the DSE Search queries to the DSE Search datacenter.
Developing applications with DataStax drivers

For the `SearchExecutionProfile`, pass the DSE Search datacenter as the local datacenter in the `DCAware` load balancing policy. This profile can then be passed to all execution methods that use DSE Search indexes in the queries. These queries will be directed to the datacenter that supports DSE Search workloads.

Driver instance per workload

For drivers that do not support execution profiles, use separate driver instances for the different workloads used in the application. This is similar to the execution profile mechanics except that the local datacenter is passed to the load balancing policy when creating the driver instance. For example, create a `SearchSession` with the DSE Search datacenter configured as the local datacenter in the load balancing policy. Use this `SearchSession` in the application for all queries that use DSE Search indexes.

DSE Core and DSE Search

Plain CQL queries and CQL queries that use the `solr_query` syntax for DSE Search (page 869) are natively supported in the DataStax drivers through the synchronous and asynchronous execute methods.

<table>
<thead>
<tr>
<th>Table 165: CQL queries for drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

DSE Graph

DSE Graph (page 871) queries are supported in the DataStax drivers through dedicated graph methods. We strongly recommend using the DSE Graph Fluent API (page 871) (similar to TinkerPop's ByteCode API) to execute graph queries. The String API (page 872) is also available (similar to TinkerPop's Script API) for the drivers that do not support the Fluent API.

<table>
<thead>
<tr>
<th>Table 166: DSE Graph queries for drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

DSE Analytics

DSE Analytics queries are supported by these drivers:

- Simba ODBC Driver for Apache Spark (page 265)
- Simba JDBC Driver for Apache Spark (page 264)
- Spark DSE Connector

Using DSE Search with the DataStax drivers

The DataStax drivers allow developers to perform DSE Search queries in their applications.

DataStax Enterprise Search enables applications to query data. Queries may use these features:
Developing applications with DataStax drivers

- general
- indexing
- full-text
- faceted (categorization)
- hit prioritization
- spatial and temporal filtering
- social media matchups

The DataStax drivers allow applications to access these features using solr_query syntax in the `WHERE` clause or normal CQL semantics.

Before using DSE Search capabilities in an application, enable the target DSE nodes for DSE Search. Create search indexes for the columns that will be accessed in the queries.

**Note:** It is a best practice to plan ahead for columns and types that are indexed because adding search indexes has a resource and performance cost.

Load balancing

DSE Search queries must be directed to a datacenter with DSE Search enabled by using the load balancing policy (page 861) in the driver. Use a datacenter-aware load balancing policy with the DSE Search local datacenter. For more, see Working with multi-workload clusters (page 868).

Paging

DSE Search paging is integrated in the DataStax driver execution implementation. The drivers use cursors for deep pagination. Enable paging through the `cql_solr_query_paging` option in `dse.yaml` on the server, or dynamically in the application in the `solr_query` parameters.

Geospatial data types

Location-based search is a key feature for a personalized user experience. DataStax Enterprise enables this through special geospatial data types:

- Point
- LineString
- Polygon

See Geospatial queries for Point and LineString. There are basic and advanced examples of geospatial data type queries.
Date ranges

Filtering by date and time is a common use case in search queries. DSE Search delivers powerful filtering on single point-in-time or open bound date ranges through the CQL DateRangeType.

**Submitting DSE Graph queries with the DataStax drivers**

The DataStax drivers expose the String API and Fluent API for executing DSE Graph traversals.

Graph use cases are characterized by highly connected data. Traversing these connections is essential for solving modern fraud detection and personalization use cases. To address the emerging demand of the Graph database, DataStax invests heavily into the Apache TinkerPop graph computing framework that leverages Gremlin as its property graph query language and core API. The DataStax drivers expose several interfaces for executing DSE Graph traversals:

- Fluent API (analogous to TinkerPop Bytecode API)
- String API (analogous to TinkerPop Script API)
- remote traversal sources for full compatibility with TinkerPop's execution model

DataStax recommendes Fluent API as the interface for graph traversals.

**Fluent API**

The DataStax drivers Graph Fluent API leverages TinkerPop's Gremlin Language Variants and allows developers to programatically construct Gremlin traversals and execute the compiled bytecode through a DSE session, similar to standard CQL queries. This interface is recommended for all new DSE Graph applications.

Figure 91: DSE Graph Fluent API
Developing applications with DataStax drivers

Table 167: Fluent API for drivers

<table>
<thead>
<tr>
<th>C/C++ (not supported)</th>
<th>C#</th>
<th>Java</th>
<th>Node.js</th>
<th>PHP (not supported)</th>
<th>Python</th>
<th>Ruby (not supported)</th>
</tr>
</thead>
</table>

String API

The String API is a more limited interface than the Fluent API. The String API simply passes Gremlin Groovy strings through the DataStax Driver to the DSE Graph server.

Table 168: String API for drivers

<table>
<thead>
<tr>
<th>C/C++</th>
<th>C#</th>
<th>Java</th>
<th>Node.js</th>
<th>PHP</th>
<th>Python</th>
<th>Ruby</th>
</tr>
</thead>
</table>

Remote traversal source

The DataStax drivers allow a TinkerPop `GraphTraversalSource` to be remotely connected to DSE Graph. This source lends full compatibility with TinkerPop types and uses an implicit execution model through the TinkerPop `terminal steps`.

**Note:** The results for a `GraphTraversalSource` are detached from the server. Modifications to the remote elements do not directly affect the data stored in DSE Graph.

Table 169: Remote traversal source for drivers

<table>
<thead>
<tr>
<th>C/C++ (not supported)</th>
<th>C#</th>
<th>Java</th>
<th>Node.js</th>
<th>PHP (not supported)</th>
<th>Python</th>
<th>Ruby (not supported)</th>
</tr>
</thead>
</table>

Domain Specific Languages

Domain Specific Languages (DSLs) simplify code and provide concise APIs for DSE Graph applications. DSLs allow the developer to abstract the underlying Gremlin code that is traversing the DSE property graph into usable methods that are tailored to the application.

Table 170: Domain Specific Languages for drivers

| C/C++ (not supported) | C#  | Java | Node.js* (not supported) | PHP (not supported) | Python | Ruby (not supported) |
User-defined IDs

Partition and clustering keys in DSE Core extend to DSE Graph. Use partition and clustering keys when creating vertex labels (page 656). Vertex labels more effectively distribute the data throughout the cluster and gives the user control over where the data is distributed.

Result paging with DataStax drivers

Large result sets can be divided into multiple pages that the client will fetch in separate network requests.

The page size specifies how many rows will be returned at a single time from the server. With each response, the server returns a paging state which is a binary token for the next request to indicate where to restart from.

Figure 92: Result paging example

While the paging API is specific to each driver, they all share a common set of features:

- Paging is enabled by default with options that can be configured.
- The page size can be overridden per query, or paging can be disabled for individual queries.
- Result objects provide a way to fetch the next page directly without manipulating the paging state or re-executing the query explicitly.
- The paging state can be extracted from a given result object and reinjected in a query later. This option is useful if you need to store the state across executions. For example, in a stateless REST web service, the paging state can be encoded in the link to the next page to seamlessly navigate to where the user left off.
Note: The paging state can be forged to access different partitions, so it should not be exposed in plain text in unsafe environments.

Some drivers (C#, Java, Node.js and Python) provide a way to traverse the whole result set transparently by triggering background fetches as the iteration crosses page boundaries.

### Table 171: Result paging for drivers

<table>
<thead>
<tr>
<th>C/C++</th>
<th>C#</th>
<th>Java</th>
<th>Node.js</th>
<th>PHP</th>
<th>Python</th>
<th>Ruby</th>
</tr>
</thead>
</table>

### Synchronous and asynchronous query execution

Queries can be executed against the database synchronously or asynchronously. The correct execution paradigm to use depends on the application.

**Synchronous execution**

Synchronous query execution is *blocking*, meaning nothing else in the application proceeds until the result from the query is returned. The application blocks for the entire round trip, from when the query is first sent to the database until the results are retrieved and returned to the application.

The advantage of synchronous queries is that it is simple to tell when a query completes, so the execution logic of the application is easy to follow. However, synchronous queries cause poor application throughput.

### Table 172: Synchronous query execution for drivers

<table>
<thead>
<tr>
<th>C/C++</th>
<th>C#</th>
<th>Java</th>
<th>Node.js</th>
<th>PHP</th>
<th>Python</th>
<th>Ruby</th>
</tr>
</thead>
</table>

**Asynchronous execution**

Asynchronous query execution is more complex. An asynchronous query execute call does not block for results. Instead, a *future* is immediately returned from the asynchronous execute call. A future is a placeholder object that stands in for the result until the result is returned from the database. Depending on the driver and feature set of the language, this future can facilitate asynchronous processing of results. This typically allows high throughput.
Speculative query execution

Speculative queries are used to preemptively start a second execution of a query against another node, before the first node has replied or returned an error. Sometimes a node may be slow to respond. Queries sent to that node will experience increased latency.

To mitigate this situation, use speculative execution to preemptively start a second execution of the query against another node, before the first node has replied or returned an error. If the second node replies faster, that response is returned to the client and the first execution is cancelled. If the first execution is cancelled, the driver ignores the response, but the request still interacts with the server.
Developing applications with DataStax drivers

Figure 94: Speculative execution with the second node responding first

The first node might reply just after the second execution call was started. In this case, the second preemptive execution is cancelled. For applications that use speculative execution, the data from whichever node replies faster is returned to the client.

Figure 95: Speculative execution with the first node responding first

Configuration

Speculative execution is disabled by default in the DSE drivers. See the driver-specific links for how to enable and configure speculative execution.

Table 174: Speculative execution for drivers

<table>
<thead>
<tr>
<th>C/C++</th>
<th>C#</th>
<th>Java</th>
<th>Node.js</th>
<th>PHP (not supported)</th>
<th>Python</th>
<th>Ruby (not supported)</th>
</tr>
</thead>
</table>
Tuning and practical details

The goal of speculative execution is to improve the overall latency of an application. However, too many speculative executions increase the load on the cluster. If speculative executions are used to avoid sending queries to unhealthy nodes, a healthy node should rarely reach resource limits. Drivers provide a configurable delay threshold at which speculative executions will be sent. In order to determine an appropriate threshold for your application, benchmark the healthy platform state (all nodes are up and under a normal load), monitoring the response latencies. Based on these results, use the latency at a high percentile (p99.9) as the speculative executions threshold.

Alternatively, when low latency is the highest priority and the cluster can handle the increased throughput, set the threshold to 0, which effectively always performs speculative executions.

Most drivers surface metrics (page 881) for speculative executions that can be used to observe the speculative executions frequency.

Query idempotence

If a query is not idempotent (page 879), the driver will never schedule speculative executions for the query, because there is no way to guarantee that only one coordinator will apply the mutation.

Retries

Turning on speculative executions doesn’t change the driver’s retry behavior (page 863). Each parallel execution triggers retries independently. The only impact is that all executions of the same query always share the same query plan, so each node will be used by no more than one execution.
Developing applications with DataStax drivers

Stream ID exhaustion

One effect of speculative executions is that many requests get cancelled, which can lead to a phenomenon called stream ID exhaustion. Each TCP connection can handle multiple simultaneous requests, identified by a unique number called a stream ID. See Connection pooling (page 863). When a request gets cancelled, the stream ID for that request cannot be used immediately because the response for that request may be returned later from the server once it fulfills the request. If this happens often, the number of available stream IDs diminishes over time. When the available stream IDs goes below a given threshold, the connection is closed and a new connection is created. If requests are often cancelled, connections will be recycled at a high rate.

In practice, exhausting all stream IDs on a connection should not occur. Each connection can reference 32768 stream IDs. Most driver implementations are configured by default to only send 1000 requests per connection.

Most drivers provide a metric for observing the number of inflight requests for a node. Some drivers provide a metric for observing the number of orphaned stream IDs. Monitor these metrics to ensure that stream ID exhaustion is not occurring.

If stream ID exhaustion is occurring, the typical solution is to add more capacity to the cluster, or adjust the system settings of the nodes to avoid resource limits.
Request ordering

Ordering issues are only a problem when using server-side timestamps. All recent versions of the DataStax drivers use client-side timestamps with exception of the C++, PHP, and Ruby drivers. Unless the driver is explicitly configured to use server-side timestamps, this section does not apply. See Query timestamps (page 884) for details.

For example, the following query is run with speculative execution and server-side timestamps enabled.

```
INSERT INTO my_table (k, v) VALUES (1, 1);
```

When the first execution is slow, a second execution is triggered. Finally, the first execution completes, so the second execution is cancelled. However, cancelling an execution only means that the driver stops waiting for the server’s response. The request could still be active.

Suppose that while the second request is still active after the driver canceled the execution, the following query is run and completes successfully.

```
DELETE from my_table where k = 1;
```

The second request of the `INSERT` query finally reaches its target node, which applies the `INSERT`. The row that was successfully deleted is back, despite the driver canceling the second request.

**Important:** To avoid this scenario, use client-side timestamps.

Query idempotence

A CQL query is *idempotent* if it can be applied multiple times without changing the result of the initial application.

```
UPDATE my_table SET list_col = [1] WHERE pk = 1
```

This query is idempotent because no matter how many times it is executed, `list_col` will always end up with the value `[1]`.

```
UPDATE my_table SET list_col = [1] + list_col WHERE pk = 1
```

This query is not idempotent because if `list_col` was initially empty, it will contain `[1]` after the first execution, `[1, 1]` after the second.

By default, all DataStax Drivers consider queries to be non-idempotent. It is the user’s responsibility to mark queries as idempotent to leverage features such as retry (page 863) and speculative execution (page 875).
Non-idempotent examples

Queries that insert the result of a non-deterministic functional call (for example `now()` and `uuid()`) are not idempotent.

The following query is not idempotent because `now()` produces a value based on the current time on the DSE coordinator responsible for handling the request, so successive invocations will produce different values if done at different times.

```
UPDATE my_table SET v = now() WHERE pk = 1
```

Counter updates are not idempotent. Each application of a counter update changes the accumulated value of the counter.

```
UPDATE my_table SET counter_value = counter_value + 1 WHERE pk = 1;
```

Prepend, append, or deletion operations on lists are not idempotent.

Prepend and append add elements to the list on each invocation. Delete removes values at a position which may vary depending on the state of the list when the query is invoked. Note that update, insert, and delete operations on set, map, tuples and user defined types are idempotent.

```
UPDATE my_table SET list_col = [1] + list_col WHERE pk = 1
```

Lightweight transactions should be considered non-idempotent if linearizability is a concern. For example:

```
UPDATE my_table SET v = 4 WHERE k = 1 IF v = 1
```

If this statement is executed twice, the IF condition will fail on the second execution. In this case, the second execution will do nothing and v will still have the value 4. The problem appears when multiple clients execute the query with retries enabled.

1. v has the value 1.
2. Client 1 executes the query above, performing a compare and set operation from 1 to 4.
3. Client 1’s connection drops, but the query completes successfully. v now has the value 4.
4. Client 2 executes a compare and set operation on v from 4 to 2.
5. Client 2’s transaction succeeds. v now has the value 2.
6. Since Client 1 lost its connection, it considers the query as failed, and transparently retries the compare and set operation on \( v \) from 1 to 4. Because \( v \) now has a value of 2, it receives a “not applied” response.

One important aspect of lightweight transactions is linearizability. Given a set of concurrent operations on a column from different clients, there must be a way to reorder them to yield a sequential history that is correct. In the above example, from the client’s point of view there were two operations.

- Client 1 executed a compare and set operation on \( v \) from 1 to 4 that was not applied.
- Client 2 executed a compare and set operation on \( v \) from 4 to 2 that was applied.

Overall the column changed from 1 to 2. There is no ordering of the two operations that can explain the change, and linearizability was broken by the transparent retry at step 6.

**Driver metrics**

DataStax drivers expose metrics through different libraries and APIs depending on the language.

Whether in the process of developing an application or deploying the solution in production, it is critical to have systems in place that provide insights into the performance of the application. Many modern applications are critical to maintaining a business's value. It is vitally important to effectively monitor and alert operators when systems are degrading. A framework for the application monitoring allows for greater ease when tracing the source of performance issues.

For example, an organization uses DataStax OpsCenter to manage a DataStax Enterprise deployment. The operators receive an alert that there is a spike in latency on the server side. If application monitoring is also in place, operators investigating the issue could narrowed the latency to a single DataStax driver instance, and then evaluate how to fix the latency problem.

<table>
<thead>
<tr>
<th>Table 175: Driver metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

The C/C++ driver tracks its metrics through an internal object called CassMetrics. This object contains information about requests (latency and throughput), stats (connections), and errors (timeouts). The DSE C/C++ Driver also exposes information for speculative executions through a CassSpeculativeExecutionMetrics object.

Java

The Java driver delivers its internal measurements through the Dropwizard Metrics library. For all versions of the Java Driver, metrics are exposed through a MetricRegistry. The
reporter options include JMX, JSON (via a servlet), stdout, CSV files, SLF4J logs, and Graphite. See the Dropwizard Documentation for more details.

Node.js

The Node.js driver exposes several internal driver metrics in the form of counters in 2 different ways:

- A default implementation which leverages the Node.js events API to expose different counter increments and push it in your existing application metrics toolkit.
- A ClientMetrics interface that can be used by metrics libraries, service providers and the community to implement support for existing toolkits like metrics, datadog, prometheus, and measured.

Python

The Python driver uses the scales library for its metrics. Metrics collection is not enabled by default in the Python Driver. To use these metrics, create the Cluster object with metrics_enabled set to True. To view the reported statistics, use a simple HTTP server for spot checking the metrics. A more robust solution for collecting and reporting to Graphite is also supported via a GraphitePusher.

Object mappers in DSE drivers

The C#, Java, Node.js and Python drivers each provide an object mapper. These mappers are tools for generating, executing, and consuming the results of queries.

The object mapper APIs intentionally do not implement all the CQL features. The mapper APIs differ for each language, as each language requires a different set of patterns. Examine the API documentation for each language to make sure it fits with your application.

The following concepts are common across the mapper APIs:

- A model class is a class that represents a DSE table. These classes have member variables that map to columns in that table.
- Instances of model classes are data objects.

<table>
<thead>
<tr>
<th>Table 176: Object mapper API documentation for drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++ (not supported)</td>
</tr>
</tbody>
</table>

C# object mapper

In the C# object mapper, model classes are normal classes with setters and getters, also known as plain old CLR objects (POCOs). By default, the mapper automatically discovers the object member to DSE column name mapping. The mapping can also be explicitly configured by using the Define method.

Because model classes are POCOs, all queries must be mediated by the mapper object. Mappers wrap Session instances and provide methods, such as First and Fetch, that execute statements and define the types into which the mapper reads data. Similarly,
Developing applications with DataStax drivers

mappers provide methods, such as Insert and Update, that take in a data object and use it to generate a write statement. All of these methods are generic and can take a parameterized query and arguments or a data object.

By default, writing null values deletes or unsets the corresponding value in the database. Writing null values creates tombstones that can impact query performance and the overall health of the database. Writing nulls can be controlled on a per-operation basis by setting insertNulls when using the Insert or InsertAsync methods.

The C# driver also includes a C# LINQ API available that will not be covered in detail in this guide.

Java object mapper

For the Java object mapper, model classes are created by annotating classes with @Table. By default, the mapper automatically discovers the object member to DSE column name mapping. The mapping can also be explicitly configured using the @Column annotation.

To execute queries with the mapper, create a data object that defines the query and pass it using CRUD operations on the mapper. For read queries, the data object arguments are used as filters on the DSE columns. If custom queries are needed, the mapper extends functionality via Accessors. It is also possible to map regular ResultSets to data objects. For write requests, the data object values are used as values in the insert query.

By default, writing null values deletes or unsets the corresponding value in the database. Writing null values creates tombstones that can impact query performance and the overall health of the database. Writing nulls can be controlled by default or on a per-operation basis by setting the saveNullFields option.

Node.js object mapper

For details on the Node.js object mapper, see this DataStax Academy blog post.

Python object mapper (cqlengine)

For the Python object mapper, model classes are created by subclassing cassandra.cqlengine.model.Model. By default, the mapper automatically discovers the mapping of object attributes created from cassandra.cqlengine.columns.Column to DSE column names. This mapping can also be explicitly defined using the db_field kwarg to Column subclass initializers.

It is safest to create the tables for model objects outside the scope of the mapper, though the Python driver does allow for making schema changes with the object mapper. Creating tables within the mapper can result in concurrent schema modifications, which are not recommended.

The Python mapper provides class methods for reading and writing data objects. In addition, queries can be executed by directly calling methods on the model class. The mapper read query methods return collections of data objects that have instance methods for CRUD operations.
Passing None corresponds to a DELETE operation on the value in the corresponding row. This creates tombstones that can impact query performance and the overall health of the database.

Mapper connections are maintained in a connection registry that can be used to access the sessions that connect to the database.

**Query timestamps**

Timestamps determine the order of precedence for operations on the same column value from different queries. In DataStax Enterprise and Cassandra, each mutation—update, insert, delete—is assigned a microsecond-precision timestamp to order operations relative to each other. The order of precedence for operations on the same column value is:

1. Data with the latest timestamp.
2. If the operations have the same timestamp, deletes have priority over inserts and updates.
3. Otherwise, the lexically larger value of data has priority. For example, 2 is chosen over 1.

Timestamps can be assigned by the driver client or the server-side node coordinating the request. All recent versions of the DataStax drivers use client-generated timestamps by default for DataStax Enterprise versions 4.7 and later. Older versions of DSE (and versions of Apache Cassandra older than 2.1) do not support client timestamps, as they were introduced in the CQL native protocol version 3.

Client-side timestamp generation is the default to keep order of operations predictable from the perspective of a single client. Through monotonically increasing client-side timestamps, the driver ensures that all operations are written in the sequential order that they were executed within the scope of that instance.

Without client timestamps, the client is at the whim of timestamps assigned by coordinating nodes. Coordinating nodes assign timestamps based on their internal system clock. It is difficult to keep the different nodes system clock synchronized in a distributed system. Each node is subject to clock drifts ranging from tens of milliseconds to seconds, even when the nodes use NTP or other clock synchronization software.

For example, consider the following scenario where server timestamps are used.

1. A client executes the following query:

   ```sql
   DELETE FROM tbl_a WHERE key = 0
   ```

   The query is sent to Node A, which creates a delete mutation with timestamp 10.

2. The client then executes:

   ```sql
   UPDATE tbl_a SET x = 'hello' where key = 0
   ```
The query is sent to Node B, which creates an update mutation with timestamp 9.

3. The client executes:

```
SELECT x from tbl_a where key = 0
```

and receives a result set with 0 rows.

It should be surprising that no rows were returned from the `SELECT` query in step 3. Even though the `DELETE` operation in step 1 was executed before the `UPDATE` operation in step 2, it takes precedence because the largest timestamp (10) was assigned to it. This scenario is avoided completely by using client timestamps.

Configuring timestamp generation in the drivers

Client timestamp generation can be configured or disabled in each of the drivers. See the individual driver documentation for more information on each driver:

<table>
<thead>
<tr>
<th>Table 177: Client timestamp generation for drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

When to use server timestamps

One possible downside to using client timestamps is that the number of client application servers often outnumber DataStax Enterprise nodes in production environments. It’s not unusual for different applications using the same DSE cluster to be managed by different teams. In these cases, it may be operationally challenging to keep the clocks synchronized between many different client application servers.

Out-of-sync client application server clocks is an issue only when there are clients making updates to the same partition values as other clients within a window that would be smaller than the expected clock drift between client nodes. Even in this case, it may not be important that updates made in this window be properly ordered in the sequence in which they were executed. It is possible that these updates were made by different parties who are not aware of one another. If it is important, consider using lightweight transactions.

Lightweight transactions and client timestamps

When executing lightweight transactions (LWTs), any client timestamp assigned to those operations is discarded. This is because DSE maintains a separate timestamp generator that ensures the timestamp assigned is monotonically increased across all LWTs.

One common mistake users make is mixing the use of LWTs and other mutation operations on a single table. This is not recommended, especially since the timestamp mechanism used for normal operations is different than the one used by LWTs, even when using server timestamps.

Keeping clocks in sync across servers

No matter the timestamp strategy, DataStax strongly recommends using a service like NTP to keep the system clocks synchronized across all machines in the data ecosystem.
DataStax also recommends organizations measure and understand the degree of clock drift among all the servers in their production environment to understand the time windows that may exist between nodes. Use utilities and commands, such as `clockdiff`, `ntpd`e `ntpq`, and `ntp -q`, to measure clock differences between servers.

**Error handling**

When using the drivers with a DataStax Enterprise or Cassandra cluster, various errors and exceptions may be encountered. The correct way to handle these error conditions often depends on the requirements of the application utilizing the driver. In this section, types of errors are covered as well as causes and common remediation of each.

At the broadest level, there are two types of errors:

- **Server-originated errors**: Server errors are returned directly from the coordinator to the driver and are identical across all of the drivers.
- **Client-side errors**: Client-side errors are specific to issues that occur in the driver itself and vary from driver to driver.

For specifics, refer to the individual driver error documentation.

<table>
<thead>
<tr>
<th>Table 178: Driver error documentations</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
</tr>
</tbody>
</table>

**Server errors**

Server errors originate at the server and are sent back to the client. Additional information about all of these errors is available in the Apache Cassandra native_protocol document, section 9. Error Codes.

**Authentication errors**

**Description**

Authentication was required by the server and failed. The possible reason for failing depends on the authenticator in use and the error message may or may not contain more detail about the failure.

**Remediation**

Investigate the authentication mechanisms used by DSE and the application. Double check username and passwords. See Authentication in DSE drivers (page 858) and review About DSE Unified Authentication for more information.
Unavailable exceptions

Figure 97: Unavailable exception

Description

This error indicates that the consistency level of the query is higher than the number of available replicas to serve that query. This exception contains 3 parts.

- **CL**: The consistency level of the query that triggered the exception.
- **Required**: An integer representing the number of nodes that must be alive to honor the CL.
- **Alive**: An integer representing the number of replicas that were known to be alive when the request had been processed.

Remediation

Ensure that a sufficient number of replicas are available for your consistency level. This error often signals that nodes are down or lacking connectivity to the coordinator. Another possible cause is that the DSE cluster is in the middle of a rolling restart or upgrade. When operators are performing a rolling upgrade or restart, ensure that the previous node is fully up and ready to receive query requests before restarting the next node in the procedure.

Overloaded exceptions

Description

The request cannot be processed because the coordinator node is overloaded by requests.

Remediation

Overloaded exceptions signal that the cluster can not handle the incoming traffic from clients. This can be triggered during spikes in traffic or due to expensive queries exhausting the node's resources. This typically indicates an under provisioned cluster.
Write timeouts

Figure 98: Write timeout exceptions

Description

Write timeouts signal that a server side timeout exception occurred during the write request. This error contains 4 parts.

- **CL**: The consistency level of the query that triggered the write timeout.
- **received**: The number of nodes that acknowledged the request.
- **blockfor**: An integer that represents the number of replicas required to satisfy the consistency level.
- **writetype**: A string that describes the type of write that timed out. Below are the different types of writes.

  # SIMPLE
  # BATCH
  # BATCH_LOG
  # UNLOGGED_BATCH
  # COUNTER
  # CAS
  # VIEW (MV)
  # CDC

Remediation

When this happens on a non-idempotent (page 879) write, such as incrementing a counter, caution must be exercised by the client, as the data may or may not have been written to the table by the node. With an idempotent write, the write can simply be retried. Only batchlog writes are retried by the driver’s default retry policy. A query’s idempotence can be defined in the application. See the individual driver documentation for the API specifics.

Depending on the SLAs and application requirements, the default server side write timeout may not be adequate and this value can be adjusted (page 85) in the cassandra.yaml. One common case when write timeouts surface is when batches are large or span multiple
Developing applications with DataStax drivers

partitions. To address this, consider decreasing the batch size and limiting batch writes to a single partition. See this blog post for more details on correctly handling this error.

Read timeouts

Description

Read timeouts signal that a server side timeout exception occurred during the read request. This error contains 4 parts.

- **CL**: The consistency level of the query that triggered the read timeout.
- **received**: The number of nodes that acknowledged the request.
- **blockfor**: An integer that represents the number of replicas required to satisfy the consistency level.
- **data present**: If this value is 0 it means the replica that was asked for the data did not respond. Otherwise the value is not 0. The coordinator will only ask a single node for the data and uses a checksum from the other nodes to determine if the data is consistent.

Remediation

Read timeouts can occur for a variety of reasons. Some possible causes are if the query is requesting a very large amount of data at all once or if there are long server side garbage collection events occurring. This typically indicates issues with the data model or query patterns that are causing poor performance on the server. To debug, first verify in the server logs that garbage collection times are acceptable and then examine the data model and access patterns. The server side read timeout (page 85) can be altered in cassandra.yaml if no other underlying cause of the timeouts can be diagnosed.

Read failures

Description

A read failure is a non-timeout exception encountered during a read request. This error contains 5 parts.

- **CL**: The consistency level of the query that triggered the error.
- **received**: The number of nodes that acknowledged the request.
- **blockfor**: An integer that represents the number of replicas required to satisfy the consistency level.
- **reasonmap**: A map of endpoint to failure reason codes. This maps the endpoints of the replica nodes that failed executing the request to the code representing the reason for the failure.
- **data present**: If this value is 0 it means the replica that was asked for the data did not respond. Otherwise the value is not 0. The coordinator will only ask a single node for the data and uses a checksum from the other nodes to determine if the data is consistent.

Remediation
Developing applications with DataStax drivers

This error is rarely encountered. Investigate the reason map to find the root cause. The most common cause for this type of error is when too many tombstones are read during the request.

Write failures

Description

A write failure is a non-timeout exception encountered during a write request. This error contains 5 parts.

- **cl**: The consistency level of the query that triggered the error.
- **received**: The number of nodes that acknowledged the request.
- **blockfor**: An integer that represents the number of replicas required to satisfy the consistency level.
- **reasonmap**: A map of endpoint to failure reason codes. This maps the endpoints of the replica nodes that failed executing the request to the code representing the reason for the failure.
- **writeType**: A string that describes the type of write that failed. The value of the string will describe the type of write that failed. Below are the different types of writes.

```plaintext
# SIMPLE
# BATCH
# BATCH_LOG
# UNLOGGED_BATCH
# COUNTER
# CAS
# VIEW (MV)
# CDC
```

Remediation

This error is rarely encountered. Examine the reason map to find to the root cause. The most common cause for this type of error is when batch sizes are too large.

Function failures

Description

A user defined function (UDF) failed during execution. The error message contains the following information.

- **keyspace**: The keyspace of the failed function.
- **function**: The name of the failed function.
- **arg_types**: A list of argument types of the failed function.
Remediation

It is likely that something is logically wrong with the user defined function, such as an infinite loop or syntax error. Scrutinize the UDF definition to find the issue.

Syntax errors

Description

The submitted query contains invalid syntax.

Remediation

Ensure the CQL has correct syntax.

Invalid errors

Description

The submitted query is syntactically correct, but is not a valid query.

Remediation

Ensure the query is valid. Examples of syntactically correct but invalid queries include trying to set the keyspace to a nonexistent keyspace or querying a table that does not exist.

Already exists errors

Description

The query attempted to create a keyspace or table that already exists. This error contains 2 parts.

- **ks**: The keyspace associated with the keyspace or table that already exists.
- **table**: The name of the table that already exists. If no table is involved this is empty.

Remediation

Make sure the keyspace or table does not exist before trying to create it or use the `IF NOT EXISTS` CQL syntax.

Unprepared errors

Description

The execution of a prepared statement was attempted when the statement was not prepared in advance.

Remediation
Prepare the statement before executing it.

**Client errors**

Client errors originate from problems with the driver itself. These vary from driver to driver depending on implementation, execution model, and ecosystem. All drivers however have the concept of a driver timeout.

**Driver timeout**

**Description**

This error name varies from driver to driver but indicates that there was a timeout on the client side. This means that the client side timeout was hit before any response was received from the server side coordinator.

**Remediation**

The client side timeout is commonly configured as either part of the execution profile or on the query execution method. The value for the client side timeout should be set higher than that of the server side write and read timeouts. This error is typically encountered when a read query is requesting a large amount of data or when batch write sizes are large or span multiple partitions.
DataStax Enterprise tools

DataStax Enterprise Metrics Collector

DSE Metrics Collector aggregates DataStax Enterprise (DSE) metrics and integrates with existing monitoring solutions to facilitate problem resolution and remediation.

DSE Metrics Collector is built on collectd, a popular, well-supported, open source metric collection agent. With over 90 plugins, you can tailor the solution to collect metrics most important to your organization.

When DSE starts, it automatically begins sending metrics and other structured events to DSE Metrics Collector. Use dsetool insights_config (page 1333) to configure the frequency and type of metrics that are sent to DSE Metrics Collector. After setting the configuration properties, you can export the aggregated metrics to tools like Prometheus, Graphite, and Splunk, which can then be visualized in a dashboard such as Grafana.

Enabling and disabling DSE Metrics Collector

Enable DSE Metrics Collector to aggregate and collect metrics in a meaningful way to provide fast, accurate problem resolution that system administrators and DataStax Support can use to troubleshoot problems.

Enabling DSE Metrics Collector

DSE Metrics Collector must be enabled to aggregate and collect metrics.

DSE Metrics Collector is enabled by default in DSE 6.7.

1. To view the current configuration:

   ```
s $ dsetool insights_config --show_config
   
   The results of the default configuration show that the DSE Metrics Collector is enabled with local storage:
   
   {
    "mode" : "ENABLED_WITH_LOCAL_STORAGE",
    "config_refresh_interval_in_seconds" : 30,
    "metric_sampling_interval_in_seconds" : 30,
    "data_dir_max_size_in_mb" : 1024,
    "node_system_info_report_period" : "PT1H"
   }
   ```

2. If the current configuration is disabled, use the dsetool insights_config (page 1333) command to change the mode.
   - To enable metrics collection with local storage:
DSE Metrics Collector is enabled.

**Disabling DSE Metrics Collector**

1. To view the current configuration:

   ```
   $ dsetool insights_config --show_config
   ```

   When enabled, the results show that DSE Metrics Collector is enabled with local storage:

   ```json
   {
     "mode" : "ENABLED_WITH_LOCAL_STORAGE",
     "config_refresh_interval_in_seconds" : 30,
     "metric_sampling_interval_in_seconds" : 30,
     "data_dir_max_size_in_mb" : 1024,
     "node_system_info_report_period" : "PT1H"
   }
   ```

2. To disable the DSE Metrics Collector, use the `dsetool insights_config` (page 1333) command to change the mode.

   ```
   $ dsetool insights_config --mode DISABLED
   ```

3. To view the current configuration:

   ```
   $ dsetool insights_config --show_config
   ```

   When disabled, the results show that DSE Metrics Collector is enabled with local storage:

   ```json
   {
     "mode" : "DISABLED",
     "config_refresh_interval_in_seconds" : 30,
     "metric_sampling_interval_in_seconds" : 30,
     "data_dir_max_size_in_mb" : 1024,
     "node_system_info_report_period" : "PT1H"
   }
   ```
DSE Metrics Collector is disabled.

**Configuring DSE Metrics Collector**

DataStax Enterprise (DSE) Metrics Collector configuration includes tuning:

- Time interval of metrics collection
- Time interval to refresh configuration changes
- Time interval of node system information collection
- Maximum size of the local data directory
- The directories where DSE Metrics Collector stores metrics and metrics logs

1. Ensure that the DSE Metrics Collector is enabled *(page 893).*

2. To configure metrics collection, use the `dsetool insights_config` *(page 1333)* command with the applicable option to adjust:
   - Interval of metrics collection
   - Interval to refresh configuration changes
   - Maximum size of the local data directory
   - Node system information reporting period

   See *configuration examples* *(page 896)* for instructions to modify specific configurations.

3. Define the directories where DSE Metrics Collector stores metrics and metrics logs.

   **Important:** DataStax recommends explicitly setting the location of the DSE Metrics Collector data directory. The maximum size of the local data directory must not exceed 2 GB.

   a. To make changes, uncomment the `insights_options` *(page 140)* section in `dse.yaml` and the options under that heading:

   ```yaml
   insights_options:
       data_dir: /var/lib/cassandra/insights_data
       log_dir: /var/log/cassandra/
   ```

   When `data_dir` is not set, the default location of the `/insights_data` directory is the same location as the `/commitlog` directory, as defined with the `commitlog_directory` property in `cassandra.yaml`. The default location of the `commitlog` directory is `/var/lib/cassandra/commitlog`. 
b. Restart (page 1437) the node for DSE to recognize the directory changes.

DSE Metrics Collector configuration examples

These configuration examples show how to:

• View the current configuration
• Enable and disable metrics collection
• Adjust the interval of metrics collection
• Adjust the interval to refresh configuration changes
• Set the maximum size of the local data directory (2 GB limit)
• Adjust the node system information reporting period

Examples

To view the current DSE Metrics Collector configuration

```
$ dsetool insights_config --show_config
```

The results of the default configuration:

```
{
  "mode" : "DISABLED",
  "config_refresh_interval_in_seconds" : 30,
  "metric_sampling_interval_in_seconds" : 30,
  "data_dir_max_size_in_mb" : 1024,
  "node_system_info_report_period" : "PT1H"
}
```

To enable metrics collection when collectd is configured to report to a real-time monitoring system

```
$ dsetool insights_config --mode ENABLED_NO_STORAGE
```

To enable metrics collection with local storage

```
$ dsetool insights_config --mode ENABLED_WITH_LOCAL_STORAGE
```

To configure 1500 MB for the DSE Metrics Collector local data directory

```
$ dsetool insights_config --data_dir_max_size_in_mb 1500
```

Note: The maximum size of the local data directory must not exceed 2 GB.

To change the node system reporting duration to 1 week
Use a ISO-8601 time duration string.

```
$ dsetool insights_config --node_system_info_report_period P1W
```

To disable metrics collection

```
$ dsetool insights_config --mode DISABLED
```

To configure the metric sampling interval for 60 seconds

```
$ dsetool insights_config --metric_sampling_interval_in_seconds 60
```

To configure 120 seconds for the configuration refresh interval

Push configuration changes to all nodes in the cluster every 2 minutes:

```
$ dsetool insights_config --config_refresh_interval_in_seconds 120
```

## Filtering metrics

Use the `dsetool insights_filters` ([page 1336](#)) command to filter metrics deemed unimportant or sensitive, and focus on specific metrics important to your organization.

Apply filters with one of these options:

- **--global** Filters metrics reported locally and insights data files.
- **--insights_only** Filters insights data files only (for diagnostic purposes).

Use a regular expression (regex) to specify which metrics to include or exclude from the filter. The regex is not anchored, so a substring match like `Keyspace` blocks anything with keyspace in the metric name. You can also specify a full regex such as `org.apache.cassandra.metrics.Keyspace.+` to filter metrics from a specific keyspace. See example filters ([page 898](#)) for details.

The following tables describe whitelist and blacklist combinations to grant or deny access to a metric using a regex. When evaluating a regex, the blacklist always overrides the whitelist.

### Table 179: Blacklist and whitelist combinations to grant access

<table>
<thead>
<tr>
<th>Regex</th>
<th>Blacklist</th>
<th>Whitelist</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>No metric indicated.</td>
<td>-</td>
<td>-</td>
<td>#</td>
</tr>
<tr>
<td>Metric included in whitelist without any blacklist entries.</td>
<td>-</td>
<td>Match</td>
<td>#</td>
</tr>
</tbody>
</table>
### Table 180: Blacklist and whitelist combinations to deny access

<table>
<thead>
<tr>
<th>Regex</th>
<th>Blacklist</th>
<th>Whitelist</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric not included in blacklist without any whitelist entries.</td>
<td>No match</td>
<td>-</td>
<td>#</td>
</tr>
<tr>
<td>Metric included in whitelist without any matches in blacklist.</td>
<td>No match</td>
<td>Match</td>
<td>#</td>
</tr>
</tbody>
</table>

Example: Example filters

#### Show all active filters

```bash
$ dsetool insights_filters --show_filters
```

#### Remove all active filters

```bash
$ dsetool insights_filters --remove_all_filters
```

#### Add a global filter to deny all metrics matching KeyspaceMetrics

```bash
```

#### Remove a global filter to deny metrics for a specific keyspace

```bash
$ dsetool insights_filters --remove --global --deny
```
Add a filter to insights data files that deny grace period metrics

```
$ dsetool insights_filters --add --insights_only --deny .+gc.+
```

**Exporting and visualizing metrics with Prometheus and Docker**

To quickly get started with DSE Metrics Collector, download and install Docker, which provides the ability to run DataStax preconfigured dashboards. If you already have a monitoring and visualization solution, see Manually exporting and visualizing metrics with Prometheus (page 902).

**Warning:** To avoid the loss of metrics data, before using the Docker examples provided in this repository, make sure you have a good understanding of Prometheus’s default data retention period and how to adjust it. By default, Prometheus is configured to only retain 15 days’ worth of data. If you want to keep a longer history of your metrics data, revise `docker-compose.yml` to add the `--storage.tsdb.retention` flag onto the Prometheus runtime command line. Specify an appropriate value for your environment.

**Prerequisites:**

1. If necessary, download and install DataStax Enterprise (DSE) 6.7 or later.
2. Download and install Docker.
3. Download and install Docker Compose.
4. Clone the DSE Metrics Collector Dashboards repository from GitHub.

**Note:** You must install Docker, Docker Compose, and the DSE Metrics Collector Dashboards repository on a node where DSE is not installed.

1. On your DSE cluster, check the configuration for DSE Metrics Collector:

```
$ dsetool insights_config --show_config
```

```
{
  "mode" : "DISABLED",
  "config_refresh_interval_in_seconds" : 30,
  "metric_sampling_interval_in_seconds" : 30,
  "data_dir_max_size_in_mb" : 1024,
  "node_system_info_report_period" : "PT1H"
}
```
2. To enable the Prometheus server, create a configuration file on a DataStax Enterprise (DSE) cluster that DSE Metrics Collector reads from.

For clusters managed by Lifecycle Manager (LCM), you can use LCM to export metrics collection to Prometheus and other supported monitoring tools.

a. Create a prometheus.conf file in one of the following directories:

- **Tarball:** `installation_location/resources/dse/collectd/etc/collectd`
- **Package:** `/etc/dse/collectd`

```bash
$ sudo touch prometheus.conf
```

**Tip:** If the `/collectd` directory does not exist, create it, and then create the `prometheus.conf` file.

b. Add the following code to the `prometheus.conf` configuration file and save it:

```bash
$ vi prometheus.conf
```

```
LoadPlugin write_prometheus

<Plugin write_prometheus>
  Port "9103"
</Plugin>
```

c. On any node in the cluster, disable and re-enable DSE Metrics Collector to propagate the changes:

```bash
$ dsetool insights_config --mode DISABLED
```

**Note:** Disabling DSE Metrics Collector can take up to 30 seconds to propagate across the cluster. Wait at least 30 seconds before re-enabling DSE Metrics Collector.

```bash
$ dsetool insights_config --mode ENABLED_WITH_LOCAL_STORAGE
```

3. Ensure that the node is listening on port 9103, in addition to the typical DSE ports:

```bash
$ netstat -lnt
```

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address       State
```
Note: If you configured Prometheus but port 9103 is not listening, read Error messages when libaio not installed (DSE 6.0 and 6.7) and ensure that libaio is installed and Epoll is working. The collectd daemon will not start if Epoll is not working.

4. On the node where you cloned the DSE Metrics Collector Dashboard repository, navigate to the /prometheus directory and modify the tg_dse.json file to include the IP address of each node in your cluster under “targets”. The list of IP addresses is comma-delimited, and each IP address must be enclosed in double quotes.

   $ vi tg_dse.json

   ```json
   [
     {
       "targets": [
         "10.100.110.96:9103",
         "10.100.100.95:9103",
         "10.100.100.97:9103"
       ]
     }
   ]
   ```

5. From the cloned directory containing the DSE Metrics Collector repository, start and attach the Docker containers. This command also starts the included Prometheus and Grafana servers.

   $ docker-compose up

6. View the Prometheus and Grafana dashboards where DSE Metrics Collector is reporting metrics:
**a.** To view the Prometheus dashboard, which displays each DSE node as a target, open a browser and navigate to [http://localhost:9090/targets](http://localhost:9090/targets).

**b.** To view the Grafana dashboards that are linked to local Prometheus data source, open a browser and navigate to [http://localhost:3000/dashboards](http://localhost:3000/dashboards).

7. List the name of all Docker containers to obtain the container IDs for the Prometheus and Grafana images:

```
$ docker ps -a
```

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>IMAGE</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>78d3324982ba</td>
<td>grafana/grafana</td>
<td>&quot;/run.sh&quot;</td>
</tr>
<tr>
<td>aadc3d4ad1d1</td>
<td>prom/prometheus</td>
<td>&quot;/bin/prometheus --c...&quot;</td>
</tr>
</tbody>
</table>

The Grafana dashboards display metrics for the defined clusters.

**Manually exporting and visualizing metrics with Prometheus**

After configuring DSE Metrics Collector, you can export metrics and visualize them in a dashboard. In the following examples, Prometheus is the monitoring solution and Grafana is used as the dashboard. However, other monitoring and visualization tools can be used with DSE Metrics Collector.

If you already have a Prometheus server running, this method of exporting and visualizing metrics enables use of the existing server.

**Prerequisites:**

1. If necessary, **Download and install DSE 6.7** or later.

2. **Download and install Prometheus**, which is used as the monitoring solution, or use a Prometheus server that is already running.

3. **Download and install Grafana**, which is used as the visualization dashboard.

4. **Download the prometheus.yaml file** from GitHub and save it in a directory on the server where Prometheus is running. The prometheus.yaml file is preconfigured to scrape metrics reported by DSE Metrics Collector. Each metric uses a specific regex for the metric type.

5. **Download the tg_dse.json file** and save it in the same directory as the prometheus.yaml file. The prometheus.yaml file references this file to obtain the IP addresses for each cluster node.
6. Download the JSON files for the preconfigured Grafana dashboards provided by DataStax. Place these files in a local directory that is accessible.

Quick check

On the server where Prometheus is running, the prometheus.yaml file and the tg_dse.json file are collocated in the same directory.

1. To enable the Prometheus server, create a configuration file on a DataStax Enterprise (DSE) cluster that DSE Metrics Collector reads from.

For clusters managed by Lifecycle Manager (LCM), you can use LCM to export metrics collection to Prometheus and other supported monitoring tools.

a. Create a prometheus.conf file in one of the following directories:

- **Tarball**: installation_location/resources/dse/collectd/etc/collectd
- **Package**: /etc/dse/collectd

$ sudo touch prometheus.conf

**Tip:** If the /collectd directory does not exist, create it, and then create the prometheus.conf file.

b. Add the following code to the prometheus.conf configuration file and save it:

```
$ vi prometheus.conf

LoadPlugin write_prometheus

<Plugin write_prometheus>
Port "9103"
</Plugin>
```

c. On any node in the cluster, disable and re-enable DSE Metrics Collector to propagate the changes:

```
$ dsetool insights_config --mode DISABLED

Note: Disabling DSE Metrics Collector can take up to 30 seconds to propagate across the cluster. Wait at least 30 seconds before re-enabling DSE Metrics Collector.

$ dsetool insights_config --mode ENABLED_WITH_LOCAL_STORAGE
```
2. Modify the downloaded `tg_dse.json` file to include the IP address of each node in the cluster under “targets”. The list of IP addresses is comma-delimited, and each IP address must be enclosed in double quotes.

```
$ vi tg_dse.json

[
  {
    "targets": [
      "10.100.110.96:9103",
      "10.100.100.95:9103",
      "10.100.100.97:9103"
    ]
  }
]
```

3. Start Prometheus with the `--config` option to specify the `prometheus.yaml` file you downloaded from the DSE Metrics Collector Dashboards repository. For example, if you saved the `prometheus.yaml` file in the `/etc/dse/tmp` directory, the command would look like this:

```
$ ./prometheus --config.file=/etc/dse/tmp/prometheus.yaml
```

4. Verify the Prometheus and Grafana targets:

   a. Open a browser and navigate to http://prometheus_server_IP_address:9090, where `prometheus_server_IP_address` is the IP address where the Prometheus server is running, and 9090 is the default port where Prometheus runs.

   b. Navigate to http://prometheus_server_IP_address:9090/targets to view the targets that Prometheus is monitoring. Each of the nodes specified in the `tg_dse.json` file displays as an endpoint, with the current status and last scrape time.

5. After verifying that Prometheus is scraping the endpoints where DSE Metrics Collector is running, start Grafana and enable the Prometheus plugin, which is included with Grafana by default (separate installation not required).

   Note: You must have Admin privileges for your organization to add data sources in Grafana.

   a. Log in to your Grafana instance at http://grafana_server_IP_address:3000, where `grafana_server_IP_address` is the IP address where your Grafana server is running.
b. Open the side menu by clicking the Grafana icon in the upper left.

c. In the side menu, select **Configuration#Data Sources**.

d. On the Configuration page, click **Add data source**.

e. Enter a name for your data source.

f. Select **Prometheus** as the data source Type.

g. Enter the URL where your Prometheus server is running.

h. Select **Server** or **Browser**, depending on where your Prometheus instance is running.

i. Click **Save & Test**.

6. Create a Grafana dashboard using the Prometheus data source.

a. Log in to your Grafana instance.

b. In the side menu, click **Create#Import**.

c. Click **Upload .json file** and select the one of the JSON files you downloaded. For example, dse-cluster-condensed.json.

d. Enter a name for the dashboard and click **Import**.

The dashboard displays in your Grafana instance.
nodetool

About the nodetool utility

The nodetool utility is a command-line interface for monitoring a cluster and performing routine database operations. It is typically run from an operational node.

The nodetool utility supports the most important JMX metrics and operations, and includes other useful commands for cluster administration. Use nodetool commands to view detailed metrics for tables, server metrics, and compaction statistics.

nodetool abortrebuild

Aborts a currently running rebuild operation. Completes processing of active streams, but no new streams are started.

Synopsis

```
$ nodetool [connection_options] abortrebuild [-r reason]
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname
    The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
-p, --port jmx_port
    The JMX port number.
-pw, --password jmxpassword
    The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
-pwf, --password-file jmx_password_filepath
    The filepath to the file that stores JMX authentication credentials.
-u, --username jmx_username
The user name for authenticating with secure JMX.

**Command arguments**

- `-r, --reason reason`
  
  Comment to add to the log file.

**Examples**

**Stop a build operation with reason comment**

```
$ nodetool abortrebuild -r 'stopping for quarterly maintenance'
```

**nodetool assassinate**

Forcefully removes a dead node without re-replicating any data. Use as a last resort when you cannot successfully use `nodetool removenode` (page 1052).

**Synopsis**

```
$ nodetool [connection_options] assassinate ip_address
```

**Table 182: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td><code>[ ]</code></td>
<td>Optional. Square brackets (<code>[ ]</code>) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td><code>( )</code></td>
<td>Group. Parentheses (<code>( )</code>) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Repeatable. An ellipsis (<code>...</code>) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td><code>'Literal string'</code></td>
<td>Single quotation (<code>'</code>) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces (<code>{ }</code>) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (<code>&lt; &gt;</code>) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th><strong>cql_statement;</strong></th>
<th>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  
The JMX port number.

- **-pw, --password jmxpassword**
  
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  
The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  
The user name for authenticating with secure JMX.

### Command arguments

- **ip_address**
  
  IP address of the node.

### Examples

#### Forcefully remove a node

```bash
$ nodetool -u user1 -pw password1 assassinate 192.168.100.2
```

The node at IP address 192.168.100.2 is forcefully removed. Data is not re-replicated.

### nodetool bootstrap

Monitors and manages the bootstrap process on one or more nodes.
**Synopsis**

```
$ nodetool [connection_options] bootstrap [resume]
```

**Table 183: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (()) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (....) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({} ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (-- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.
Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

**resume**

Restart the operation.

Examples

**Resume the local bootstrap operation**

```
$ nodetool -u user1 -pw pwd1 bootstrap resume
```

**nodetool cfhistograms**

This tool has been renamed to *nodetool tablehistograms (page 1117)*

**nodetool cfstats**

This tool has been renamed to *nodetool tablestats (page 1118)*

**nodetool cleanup**

Triggers immediate cleanup of keyspaces that no longer belong to a node.

OpsCenter provides a Cleanup option in the Nodes UI for Running cleanup.

DataStax Enterprise does not automatically remove data from nodes that lose part of their partition range to a newly added node. Run *nodetool cleanup* on the source node and on neighboring nodes that shared the same subrange after the new node is up and running. After adding a new node, run this command to prevent the database from including the old data to rebalance the load on that node. This command temporary increases disk space use proportional to the size of the largest SSTable and causes Disk I/O to occur.
Synopsis

$ nodetool [connection_options] cleanup
[-j num_jobs] [--] [keyspace_name table_name [table_name ...]]

Table 184: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>' Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.
Connection options

-h, --host hostname
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port
The JMX port number.

-pw, --password jmxpassword
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-pwf, --password-file jmx_password_filepath
The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username
The user name for authenticating with secure JMX.

Command arguments

--
Separates an option from an argument that could be mistaken for a option.

-j, --jobs num_jobs
• num_jobs - Number of SSTables affected simultaneously.
• 0 - Use all available compaction threads.

keyspace_name
Keyspace name. By default, all keyspaces.

table_name
The table name.

nodetool clearsnapshot

Removes one or all snapshots.

Warning: This command deletes the backup (snapshot) copy of your node.

Synopsis

$ nodetool [connection_options] clearsnapshot
[--all | -t snapshotname] [--] [keyspace_name]

Table 185: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (() ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (‘) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (-- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (‘) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
The user name for authenticating with secure JMX.

Command arguments

--
Separates an option from an argument that could be mistaken for a option.

--all
Removes all snapshots.

keyspace_name
Keyspace name. By default, all keyspaces.

-t snapshotname, --tag snapshotname
The snapshot filepath. To remove all snapshots, omit the snapshot filepath.

Examples

To delete all snapshots for a node

```
$ nodetool -h localhost -p 7199 clearsnapshot
```

To delete snapshot1

```
$ nodetool clearsnapshot -t snapshot1
```

**nodetool compact**

Forces a major compaction on one or more tables or user-defined compaction on given SSTables.

OpsCenter provides a Compact option in the Nodes UI for Running compaction.

Major compactions may behave differently depending which compaction strategy is used for the affected tables:

- **SizeTieredCompactionStrategy (STCS)**: The default compaction strategy. This strategy triggers a minor compaction when there are a number of similar sized SSTables on disk as configured by the table subproperty, `min_threshold`. A minor compaction does not involve all the tables in a keyspace. Also see STCS compaction subproperties.

- **DateTieredCompactionStrategy (DTCS)** (deprecated).

- **TimeWindowCompactionStrategy (TWCS)** This strategy is an alternative for time series data. TWCS compacts SSTables using a series of time windows. While with a time window, TWCS compacts all SSTables flushed from memory into larger SSTables using STCS. At the end of the time window, all of these SSTables are compacted into a single SSTable. Then the next time window starts and the process repeats. The duration of the time window is the only setting required. See TWCS compaction subproperties. For more information about TWCS, see How is data maintained?.

- **LeveledCompactionStrategy (LCS)**: The leveled compaction strategy creates SSTables of a fixed, relatively small size (160 MB by default) that are grouped into
levels. Within each level, SSTables are guaranteed to be non-overlapping. Each level (L0, L1, L2 and so on) is 10 times as large as the previous. Disk I/O is more uniform and predictable on higher than on lower levels as SSTables are continuously being compacted into progressively larger levels. At each level, row keys are merged into non-overlapping SSTables in the next level. This process can improve performance for reads, because the database can determine which SSTables in each level to check for the existence of row key data. This compaction strategy is modeled after Google’s LevelDB implementation. Also see LCS compaction subproperties.

See How is data maintained? and Configuring compaction.

Note: A major compaction incurs considerably more disk I/O than minor compactions.

Synopsis

```bash
$ nodetool [connection_options] compact
    [-et end_token] [-s] [-st start_token] [--user-defined]
    [--] [keyspace tables [tables ...] | sstable_name ...]
```

Table 186: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
</tr>
<tr>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
| '
  <schema> ...
  </schema>'                                                              |
| Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration. |
| '@xml_entity='xml_entity_type'                                             |
| Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files. |

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**  
The JMX port number.

- **-pw, --password jmxpassword**  
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**  
The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**  
The user name for authenticating with secure JMX.

### Command arguments

- **--**  
  Separates an option from an argument that could be mistaken for a option.

- **-et, --end-token end_token**  
The token at which the range ends. Requires start token (-st).

- **keyspace_name**  
  Keyspace name. By default, all keyspaces.

- **-s, --split-output**  
  Do not create a single large file. Split output when using SizeTieredCompactionStrategy (STCS) to files that are 50%-25%-12.5% and so on of the total size. Ignored for DTCS.

- **sstable_name**  
  The name of the SSTable file. Specify sstable_name or sstable_directory.

- **-st, --start-token start_token**  
The token at which the range starts. Requires end token (-et).

- **table_name**  
  The table name.

- **--user-defined**
DataStax Enterprise tools

Submits listed files for user-defined compaction.

**nodetool compactionhistory**

Prints the history of compaction.

Synopsis

```
$ nodetool [connection_options] compactionhistory
  [-F (json | yaml)]
```

### Table 187: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key: value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**  
The JMX port number.

- **-pw, --password jmxpassword**  
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**  
The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**  
The user name for authenticating with secure JMX.

Command arguments

- **-F, --format json | yaml**  
The format for the output. The default is plain text.

Examples

To view the compaction history

```
$ nodetool compactionhistory
```

The output of compaction history is seven columns wide. The first three columns show the id, keyspace name, and table name of the compacted SSTable.

<table>
<thead>
<tr>
<th>Compaction History:</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>keyspace_name</td>
</tr>
<tr>
<td>table_name</td>
</tr>
<tr>
<td>schema_columnfamilies</td>
</tr>
<tr>
<td>d06f7080-07a5-11e4-9b36-abc3a0ec9088</td>
</tr>
<tr>
<td>d198ae40-07a5-11e4-9b36-abc3a0ec9088</td>
</tr>
<tr>
<td>users</td>
</tr>
<tr>
<td>0381bc30-07b0-11e4-9b36-abc3a0ec9088</td>
</tr>
<tr>
<td>Standard1</td>
</tr>
<tr>
<td>74eb69b0-0621-11e4-9b36-abc3a0ec9088</td>
</tr>
<tr>
<td>local</td>
</tr>
<tr>
<td>e35dd980-07ae-11e4-9b36-abc3a0ec9088</td>
</tr>
<tr>
<td>compactions_in_progress</td>
</tr>
<tr>
<td>8d5cf160-07ae-11e4-9b36-abc3a0ec9088</td>
</tr>
<tr>
<td>compactions_in_progress</td>
</tr>
<tr>
<td>ba376020-07af-11e4-9b36-abc3a0ec9088</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

libout d18cc760-07a5-11e4-9b36-abc3a0ec9088 libdata
libout 64009bf0-07a4-11e4-9b36-abc3a0ec9088 libdata
libout d04700f0-07a5-11e4-9b36-abc3a0ec9088 system
sstable_activity c2a97370-07a9-11e4-9b36-abc3a0ec9088 libdata
users cb928a80-07ae-11e4-9b36-abc3a0ec9088 Keyspace1
Standard1 cd8d1540-079e-11e4-9b36-abc3a0ec9088 system
schema_columns 62ced2b0-07a4-11e4-9b36-abc3a0ec9088 system
schema_keyspaces d19cccf0-07a5-11e4-9b36-abc3a0ec9088 system
compactions_in_progress 640bbf80-07a4-11e4-9b36-abc3a0ec9088 libdata
users 6cd54e60-07ae-11e4-9b36-abc3a0ec9088 Keyspace1
Standard1 c29241f0-07a9-11e4-9b36-abc3a0ec9088 libdata
libout c2a30ad0-07a9-11e4-9b36-abc3a0ec9088 system
compactions_in_progress e3a6d920-079d-11e4-9b36-abc3a0ec9088 system
schema_keyspaces 62c55cd0-07a4-11e4-9b36-abc3a0ec9088 system
schema_columnfamilies 62b07540-07a4-11e4-9b36-abc3a0ec9088 system
schema_columns cdd038c0-079e-11e4-9b36-abc3a0ec9088 system
schema_keyspaces b797af00-07af-11e4-9b36-abc3a0ec9088 Keyspace1
Standard1 8c918b10-07ae-11e4-9b36-abc3a0ec9088 Keyspace1
Standard1 377d73f0-07ae-11e4-9b36-abc3a0ec9088 system
compactions_in_progress 62b9c410-07a4-11e4-9b36-abc3a0ec9088 system
local d0566a40-07a5-11e4-9b36-abc3a0ec9088 system
schema_columns ba637930-07af-11e4-9b36-abc3a0ec9088 system
compactions_in_progress cdbc1480-079e-11e4-9b36-abc3a0ec9088 system
schema_columnfamilies e3456f80-07ae-11e4-9b36-abc3a0ec9088 Keyspace1
Standard1 d086f020-07a5-11e4-9b36-abc3a0ec9088 system
schema_keyspaces d06118a0-07a5-11e4-9b36-abc3a0ec9088 system
local cdaaf8d80-079e-11e4-9b36-abc3a0ec9088 system
local
The four columns to the right of the table name show the timestamp, size of the SSTable before and after compaction, and the number of partitions merged. The notation means \{tables:rows\}. For example: \{1:3, 3:1\} means 3 rows were taken from one SSTable (1:3) and 1 row taken from 3 SSTables (3:1) to make the one SSTable in that compaction operation.

<table>
<thead>
<tr>
<th>bytes_out</th>
<th>rows_merged</th>
<th>compacted_at</th>
<th>bytes_in</th>
</tr>
</thead>
<tbody>
<tr>
<td>1404936947592</td>
<td>8096</td>
<td>1404936947592</td>
<td>7211</td>
</tr>
<tr>
<td>{1:3, 3:1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1404936949540</td>
<td>144</td>
<td>1404936949540</td>
<td>144</td>
</tr>
<tr>
<td>{1:1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1404941328243</td>
<td>1305838191</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{1:4647111}</td>
<td>5864</td>
<td>1404770149323</td>
<td>5701</td>
</tr>
<tr>
<td>1305838191</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1404940844824</td>
<td>148</td>
<td>1404940700534</td>
<td>155</td>
</tr>
<tr>
<td>{1:1, 2:2}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1404941205282</td>
<td>766331398</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{1:1, 2:2}</td>
<td>766331398</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1404936949462</td>
<td>8901649</td>
<td>1404936949462</td>
<td>8901649</td>
</tr>
<tr>
<td>{1:9315}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1404936336175</td>
<td>8900821</td>
<td>140493636175</td>
<td>8900821</td>
</tr>
<tr>
<td>{1:9315}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1404936947327</td>
<td>108</td>
<td>1404936947327</td>
<td>108</td>
</tr>
<tr>
<td>{1:3, 2:1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1404938642471</td>
<td>144</td>
<td>1404938642471</td>
<td>144</td>
</tr>
<tr>
<td>{1:1}</td>
<td>1404940804904</td>
<td></td>
<td>383020422</td>
</tr>
<tr>
<td>1404940804904</td>
<td>383020422</td>
<td>1404940804904</td>
<td></td>
</tr>
<tr>
<td>{1:1363062}</td>
<td>1404933936276</td>
<td></td>
<td>4889</td>
</tr>
<tr>
<td>383020422</td>
<td></td>
<td>1404933936276</td>
<td>4177</td>
</tr>
<tr>
<td>{1:4}</td>
<td>1404936334171</td>
<td></td>
<td>441</td>
</tr>
<tr>
<td>1404936334171</td>
<td>281</td>
<td>1404936949567</td>
<td>79</td>
</tr>
<tr>
<td>{1:3, 2:1}</td>
<td>379</td>
<td>1404936336248</td>
<td>144</td>
</tr>
<tr>
<td>1404936947592</td>
<td>144</td>
<td>1404936947592</td>
<td>4177</td>
</tr>
<tr>
<td>{1:1}</td>
<td></td>
<td>1404940645958</td>
<td>144</td>
</tr>
<tr>
<td>1404936949540</td>
<td></td>
<td>1404936949540</td>
<td></td>
</tr>
<tr>
<td>{1:9315}</td>
<td></td>
<td>1404936949540</td>
<td></td>
</tr>
<tr>
<td>1404936947327</td>
<td></td>
<td>1404936947327</td>
<td></td>
</tr>
<tr>
<td>{1:3, 2:1}</td>
<td></td>
<td>1404936947327</td>
<td></td>
</tr>
<tr>
<td>1404940645958</td>
<td></td>
<td>1404940645958</td>
<td></td>
</tr>
<tr>
<td>{1:1094380}</td>
<td></td>
<td>1404940645958</td>
<td></td>
</tr>
<tr>
<td>1404938642319</td>
<td>8901649</td>
<td>1404938642319</td>
<td>8901649</td>
</tr>
<tr>
<td>{1:9315}</td>
<td></td>
<td>1404938642319</td>
<td></td>
</tr>
<tr>
<td>1404938642429</td>
<td>165</td>
<td>1404938642429</td>
<td>165</td>
</tr>
<tr>
<td>{1:3, 2:1}</td>
<td></td>
<td>1404938642429</td>
<td></td>
</tr>
</tbody>
</table>
### node tool compactionstats

Prints statistics about compactions.

**Synopsis**

```
$ node tool [connection_options] compactionstats
[-H]
```

#### Table 188: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**

---

DSE 6.7 Developer Guide (Latest version)  Page 923
The user name for authenticating with secure JMX.

**Command arguments**

- **-H, --human-readable**
  Display bytes in human readable form: KiB (kibibyte), MiB (mebibyte), GiB (gibibyte), TiB (tebibyte).

**Examples**

The total column shows the total number of uncompressed bytes of SSTables being compacted. The system log lists the names of the SSTables compacted.

```
$ nodetool compactionstats
pending tasks: 5

completed             compaction type  keyspace       table                  total      unit       progress
Compaction             Keyspace1       Standard1     282310680        bytes     93.43%
Compaction             Keyspace1       Standard1     58457931         bytes     19.01%

Active compaction remaining time :   0h00m16s
```

**nodetool decommission**

Causes a live node to decommission itself, streaming its data to the next node on the ring to replicate appropriately.

See Decommissioning a datacenter, Removing a node, and Adding a node and then decommissioning the old node.

**Tip:** Use `nodetool netstats` *(page 1013)* to monitor the progress.

OpsCenter provides an option to Decommission a node.

**Note:** Decommission does not shut down the node. Shut down the node after decommission is complete.

**Synopsis**

```
$ nodetool [connection_options] decommission [-f]
```

<table>
<thead>
<tr>
<th>Table 189: Legend</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax conventions</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  
  The JMX port number.

- **-pw, --password jmxpassword**
  
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
The filepath to the file that stores JMX authentication credentials.

\texttt{-u, \textendash username jmx\_username}

The user name for authenticating with secure JMX.

\textbf{Command arguments}

\texttt{-f, \textendash force}

Force decommission of the node even when it reduces the number of replicas to below configured RF.

\textbf{Examples}

\textbf{Decommission a remote node}

\begin{verbatim}
$ nodetool -h 10.46.123.12 decomission
\end{verbatim}

\textbf{nodetool describecluster}

Prints the name, snitch, partitioner and schema version of a cluster.

Typically used to validate the schema after upgrading. If a schema disagreement occurs, check for and resolve schema disagreements.

\textbf{Synopsis}

\begin{verbatim}
$ nodetool [connection\_options] describecluster
datacenter\_name
\end{verbatim}

\textbf{Table 190: Legend}

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italicites</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ( {} ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

- **datacenter_name**
  The datacenter name.

### Example

Get cluster name, snitch, partitioner and schema version

```
$ nodetool describecluster
```
Cluster Information:
Name: Test Cluster
Snitch: com.datastax.bdp.snitch.DseDelegateSnitch
DynamicEndPointSnitch: enabled
Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
Schema versions:
d4f18346-f81f-3786-aed4-40e03558b299: [127.0.0.1]

Get cluster name, snitch, partitioner and schema version

$ nodetool describecluster

When schema disagreement occurs, the last line of the output includes information about unreachable nodes:

Cluster Information:
Name: Production Cluster
Snitch:
org.apache.cassandra.locator.DynamicEndpointSnitch
Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
Schema versions:
UNREACHABLE: 1176b7ac-8993-395d-85fd-41b89ef49fbb:
[10.202.205.203]

nodetool describering

Shows the token ranges.

Synopsis

$ nodetool [connection_options] describering
[---] [keyspace_name [keyspace_name ...]]

Table 191: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

```
--
```

Separates an option from an argument that could be mistaken for a option.

**keyspace_name**
DataStax Enterprise tools

The keyspace name.

Examples

Get token range information on the cycling keyspace

$ nodetool describering cycling

<table>
<thead>
<tr>
<th>Schema Version:1b04bd14-0324-3fc8-8bcb-9256d1e15f82</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyspace: cycling</td>
</tr>
<tr>
<td>TokenRange:</td>
</tr>
<tr>
<td>TokenRange(start_token:3074457345618258602,</td>
</tr>
<tr>
<td>end_token:-9223372036854775808,</td>
</tr>
<tr>
<td>endpoints:[127.0.0.1, 127.0.0.2, 127.0.0.3],</td>
</tr>
<tr>
<td>rpc_endpoints:[127.0.0.1, 127.0.0.2, 127.0.0.3],</td>
</tr>
<tr>
<td>endpoint_details:[EndpointDetails(host:127.0.0.1,</td>
</tr>
<tr>
<td>datacenter:datcenter1, rack:rack1),</td>
</tr>
<tr>
<td>EndpointsDetails(host:127.0.0.2, datacenter:datcenter1,</td>
</tr>
<tr>
<td>rack:rack1),</td>
</tr>
<tr>
<td>EndpointsDetails(host:127.0.0.3, datacenter:datcenter1,</td>
</tr>
<tr>
<td>rack:rack1)],</td>
</tr>
<tr>
<td>TokenRange(start_token:-3074457345618258603,</td>
</tr>
<tr>
<td>end_token:3074457345618258602,</td>
</tr>
<tr>
<td>endpoints:[127.0.0.3, 127.0.0.1, 127.0.0.2],</td>
</tr>
<tr>
<td>rpc_endpoints:[127.0.0.3, 127.0.0.1, 127.0.0.2],</td>
</tr>
<tr>
<td>endpoint_details:[EndpointDetails(host:127.0.0.3,</td>
</tr>
<tr>
<td>datacenter:datcenter1, rack:rack1),</td>
</tr>
<tr>
<td>EndpointsDetails(host:127.0.0.1, datacenter:datcenter1,</td>
</tr>
<tr>
<td>rack:rack1),</td>
</tr>
<tr>
<td>EndpointsDetails(host:127.0.0.2, datacenter:datcenter1,</td>
</tr>
<tr>
<td>rack:rack1)],</td>
</tr>
<tr>
<td>TokenRange(start_token:-9223372036854775808,</td>
</tr>
<tr>
<td>end_token:-3074457345618258603,</td>
</tr>
<tr>
<td>endpoints:[127.0.0.2, 127.0.0.3, 127.0.0.1],</td>
</tr>
<tr>
<td>rpc_endpoints:[127.0.0.2, 127.0.0.3, 127.0.0.1],</td>
</tr>
<tr>
<td>endpoint_details:[EndpointDetails(host:127.0.0.2,</td>
</tr>
<tr>
<td>datacenter:datcenter1, rack:rack1),</td>
</tr>
<tr>
<td>EndpointsDetails(host:127.0.0.3, datacenter:datcenter1,</td>
</tr>
<tr>
<td>rack:rack1),</td>
</tr>
<tr>
<td>EndpointsDetails(host:127.0.0.1, datacenter:datcenter1,</td>
</tr>
<tr>
<td>rack:rack1)],</td>
</tr>
<tr>
<td>EndpointsDetails(host:127.0.0.2, datacenter:datcenter1,</td>
</tr>
<tr>
<td>rack:rack1)],</td>
</tr>
<tr>
<td>EndpointsDetails(host:127.0.0.3, datacenter:datcenter1,</td>
</tr>
<tr>
<td>rack:rack1)]</td>
</tr>
</tbody>
</table>

nodetool disableautocompaction

Disables autocompaction for a keyspace and one or more tables for the current node or the specified node.

Synopsis

$ nodetool [connection_options] disableautocompaction
[|--] [keyspace_name table_name [table_name ...] ]
Table 192: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
DataStax Enterprise tools

The JMX port number.

`-pw, --password jmxpassword`

The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

`-pwf, --password-file jmx_password_filepath`

The filepath to the file that stores JMX authentication credentials.

`-u, --username jmx_username`

The user name for authenticating with secure JMX.

Command arguments

`--`

Separates an option from an argument that could be mistaken for a option.

`keyspace_name`

The keyspace name.

`table_name`

One or more table names, separated by a space.

`table_name`

The table name.

`nodetool disablebackup`

Disables incremental backup.

Synopsis

```
$ nodetool [connection_options] disablebackup
```

Table 193: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>'&lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks (‘’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  
The JMX port number.

- **-pw, --password jmxpassword**
  
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  
The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  
The user name for authenticating with secure JMX.

### Command arguments

This command takes no arguments.

### Examples

**Disable incremental backup**
$ nodetool disablebackup

**nodetool disablebinary**

Disables the native transport that defines the format of the binary messages.

**Synopsis**

$ nodetool [connection_options] disablebinary

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Disable native transport

```bash
$ nodetool disablebinary
```

### nodetool disablegossip

Disables the gossip protocol, which effectively marks the node as down.

#### Synopsis

```bash
$ nodetool [connection_options] disablegossip
```

#### Table 195: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  
  The JMX port number.

- **-pw, --password jmxpassword**
  
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  
  The user name for authenticating with secure JMX.
Command arguments

This command takes no arguments.

Examples

Disable gossip

$ nodetool disablegossip

**nodetool disablehandoff**

Disables storing of future hints.

Synopsis

$ nodetool [connection_options] disablehandoff

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>*</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Disable storing future hints

```
$ nodetool disablehandoff
```

**nodetool disablehintsfordc**

Turns off hints for a datacenter, but continue hints on other datacenters.

Useful if there is a downed datacenter and during datacenter failover, when hints will put unnecessary pressure on the datacenter.

Synopsis

```
$ nodetool [connection_options] disablehintsfordc
```
Table 197: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { ) } enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname

The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
DataStax Enterprise tools

-p, --port jmx_port
   The JMX port number.
-pw, --password jmxpassword
   The JMX password for authenticating with secure JMX. If a password is not
   provided, you are prompted to enter one.
-pwf, --password-file jmx_password_filepath
   The filepath to the file that stores JMX authentication credentials.
-u, --username jmx_username
   The user name for authenticating with secure JMX.

Command arguments

--
   Separates an option from an argument that could be mistaken for a option.

datacenter_name
   The datacenter name.

Examples

Turn off hints for specific datacenter

$ nodetool -u joe -pw P@ssw0rd! disablehintsfordc DC2

nodetool drain

Flushes all memtables from the node to SSTables on disk. DSE stops listening for
connections from the client and other nodes. You need to restart DSE after running
nodetool drain. Typically, use this command before upgrading a node to a new version of
DSE.

To simply flush memtables to disk, use nodetool flush.

OpsCenter provides an option for Draining a node.

Synopsis

$ nodetool [connection_options] drain

Table 198: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>'@xml_entity=&quot;xml_entity_type&quot;'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname
   The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port
   The JMX port number.

-pw, --password jmxpassword
   The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-pwf, --password-file jmx_password_filepath
   The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username
   The user name for authenticating with secure JMX.
**Command arguments**

This command takes no arguments.

**Examples**

**Flush memtables from node to SSTables on disk**

```
$ nodetool drain
```

**nodetool enableautocompaction**

Enables autocompaction for a keyspace and one or more tables, or all tables.

**Synopsis**

```
$ nodetool [connection_options] enableautocompaction
[--] [keyspace_name table_name [table_name ...]]
```

**Table 199: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Literal keyword.</td>
</tr>
<tr>
<td><strong>Lowercase</strong></td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
**Syntax conventions**

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th><strong>Syntax conventions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
<td>[ -- ]</td>
</tr>
<tr>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
</tr>
<tr>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
<td>@xml_entity='xml_entity_type'</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

|--
| Separates an option from an argument that could be mistaken for a option.

- **keyspace_name**
  Keyspace name. By default, all keyspaces.

- **table_name**
  The table name.

**Examples**

Enable autocompaction on cyclist_name table in cycling keyspace

```
$ nodetool enableautocompaction cycling cyclist_name
```

**nodetool enablebackup**

Enables incremental backup.
Synopsis

$ nodetool [connection_options] enablebackup

Table 200: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>’&lt;schema&gt; ... &lt;/schema&gt; ‘</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options
-h, --host hostname
   The hostname or IP address of a remote node or nodes. When omitted, the default
   is the local machine.
-p, --port jmx_port
   The JMX port number.
-pw, --password jmxpassword
   The JMX password for authenticating with secure JMX. If a password is not
   provided, you are prompted to enter one.
-pwf, --password-file jmx_password_filepath
   The filepath to the file that stores JMX authentication credentials.
-u, --username jmx_username
   The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Enable incremental backup

$ nodetool enablebackup

nodetool enablebinary

Re-enables the native transport that defines the format of the binary messages.

Synopsis

$ nodetool [connection_options] enablebinary

Table 201: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>...</code></td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td><code>'Literal string'</code></td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  - The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  - The JMX port number.

- **-pw, --password jmxpassword**
  - The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  - The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  - The user name for authenticating with secure JMX.

### Command arguments

This command takes no arguments.
Examples

Re-enable native transport

$ nodetool enablebinary

**nodetool enablegossip**

Re-enables gossip.

**Synopsis**

$ nodetool [connection_options] enablegossip

---

### Table 202: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'&lt; &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (') surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity=xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

**Re-enable gossip**

```bash
$ nodetool enablegossip
```

**nodetool enablehandoff**

Reenables storing of future hints on the current node.

Synopsis

```
$ nodetool [connection_options] enablehandoff
```

**Table 203: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td><strong>Syntax conventions</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>‘ &lt;schema&gt; ... &lt;/schema&gt; ‘</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
DataStax Enterprise tools

-pwf, --password-file jmx_password_filepath
   The filepath to the file that stores JMX authentication credentials.
-u, --username jmx_username
   The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Reenable future hints storage on current node

```
$ nodetool enablehandoff
```

**nodetool enablehintsfordc**

Turns on hints for a datacenter that was previously disabled with nodetool disablehintsfordc.

Synopsis

```
$ nodetool [connection_options] enablehintsfordc
[--] datacenter_name
```

**Table 204: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (<code>&lt; &gt;</code>) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (<code>;</code>) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens (<code>--</code>). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; ' </code></td>
<td>Search CQL only: Single quotation marks (<code>'</code>) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  - The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  - The JMX port number.

- **-pw, --password jmxpassword**
  - The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  - The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  - The user name for authenticating with secure JMX.

### Command arguments

- **--**
  - Separates an option from an argument that could be mistaken for a option.

- **datacenter_name**
  - The datacenter name.

### Examples

**Turn on hints for DC2**
nodetool -u elsa -pw P@ssw0rd! enablehintsfordc DC2

**nodetool failedetector**

Shows the failure detector information for the cluster.

**Synopsis**

$ nodetool [connection_options] failedetector

<table>
<thead>
<tr>
<th><strong>Table 205: Legend</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax conventions</strong></td>
</tr>
<tr>
<td>UPPERCASE</td>
</tr>
<tr>
<td>Lowercase</td>
</tr>
<tr>
<td>Italics</td>
</tr>
<tr>
<td>[ ]</td>
</tr>
<tr>
<td>( )</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>'Literal string'</td>
</tr>
<tr>
<td>{key:value}</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
</tr>
<tr>
<td>cql_statement;</td>
</tr>
<tr>
<td>[ -- ]</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
</tr>
</tbody>
</table>
Definition
The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments
This command takes no arguments.

Examples

**Show failure detection information for cluster**

```
$ nodetool failuredetector
```

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Phi</th>
</tr>
</thead>
</table>

**nodetool flush**

Flushes one or more tables from the memtable to SSTables on disk.

OpsCenter provides a flush option in the Nodes UI for **flushing tables**.

Synopsis

```
$ nodetool [connection_options] flush
[--] [keyspace_name table_name [table_name ...]]
```

**Table 206: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**  
The JMX port number.
- **-pw, --password jmxpassword**  
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- **-pwf, --password-file jmx_password_filepath**
DataStax Enterprise tools

The filepath to the file that stores JMX authentication credentials.

-\texttt{u, --username jmx\_username}
  The user name for authenticating with secure JMX.

**Command arguments**

--
Separates an option from an argument that could be mistaken for a option.

\texttt{keyspace\_name}
Keyspace name. By default, all keyspaces.

\texttt{table\_name}
The table name.

**Examples**

**Flush cycling keyspace and cyclist\_name table**

```bash
$ nodetool flush cycling cyclist\_name
```

\textbf{nodetool garbagecollect}

Removes deleted data from one or more tables.

\textbf{Note:} The \texttt{nodetool garbagecollect} command is not the same as the \textbf{Perform GC} option in OpsCenter.

**Synopsis**

```bash
$ nodetool \[connection\_options\] garbagecollect \[-g ROW\|CELL\] \[-j job\_threads\] \[--\] \[keyspace\_name table\_name [table\_name ...]]
```

\textbf{Table 207: Legend}

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>\textit{Italics}</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

## Definition

The short form and long form parameters are comma-separated.

## Connection options

- **-h, --host hostname**  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**  
The JMX port number.

- **-pw, --password jmxpassword**  
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**  
The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**  
The user name for authenticating with secure JMX.

## Command arguments

- **--**  
Separates an option from an argument that could be mistaken for a option.

- **-g, --granularity ROW|CELL**  
ROW (default) removes deleted partitions and rows.
CELL also removes overwritten or deleted cells.

-\texttt{j, --jobs \texttt{num\_jobs}}
  - \texttt{num\_jobs} - Number of SSTables affected simultaneously.
  - \texttt{0} - Use all available compaction threads.

\texttt{keyspace\_name}
  The keyspace name.

\texttt{table\_name}
  One or more table names, separated by a space.

Examples

To remove deleted data from all tables and keyspaces at the default granularity

\begin{verbatim}
$ nodetool garbagecollect
\end{verbatim}

To remove deleted data from all tables and keyspaces, including overwritten or deleted cells

\begin{verbatim}
$ nodetool garbagecollect -g CELL
\end{verbatim}

\texttt{nodetool gcstats}

Prints garbage collection statistics that returns values based on all the garbage collection that has run since the last time this command was run. Statistics identify the interval, GC elapsed time (total and standard deviation), the disk space reclaimed in megabytes (MB), number of garbage collections, and direct memory bytes.

Synopsis

\begin{verbatim}
$ nodetool [connection\_options] gcstats
\end{verbatim}

Table 208: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  - The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
  - The JMX port number.
- **-pw, --password jmxpassword**
  - The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- **-pwf, --password-file jmx_password_filepath**
  - The filepath to the file that stores JMX authentication credentials.
- **-u, --username jmx_username**
  - The user name for authenticating with secure JMX.

**Command arguments**

This command takes no arguments.
Examples

To print garbage collection statistics

$ nodetool gcstats

Result: the garbage collection statistics since the last time the command was run are returned:

<table>
<thead>
<tr>
<th>Interval (ms)</th>
<th>Max GC Elapsed (ms)</th>
<th>Total GC Elapsed (ms)</th>
<th>Stdev GC Elapsed (ms)</th>
<th>GC Reclaimed (MB)</th>
<th>Collections</th>
<th>Direct Memory Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1890311355</td>
<td>113</td>
<td>89238</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5699826457288</td>
<td>2267</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**nodetool getbatchlogreplaythrottle**

Prints batchlog replay throttle in KBs. The batchlog replay throttle replays hints. The throttle is reduced proportionally to the number of nodes in the cluster.

Synopsis

$ nodetool [connection_options] getbatchlogreplaythrottle

Table 209: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (<code>&lt; &gt;</code>) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (<code>;</code>) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens (<code>--</code>). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks (<code>'</code>) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  
  The JMX port number.

- **-pw, --password jmxpassword**
  
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  
  The user name for authenticating with secure JMX.

### Command arguments

This command takes no arguments.

### Examples

**Print batchlog replay throttle in KBs**

```bash
$ nodetool getbatchlogreplaythrottle
```
nodetool getcompactionthreshold

Prints the minimum and maximum compaction thresholds in megabytes (MBs) for a given table.

Synopsis

$ nodetool [connection_options] getcompactionthreshold

[--] keyspace_name table_name

Table 210: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>‘ &lt;schema&gt; ... &lt;/schema&gt; ’</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
**DataStax Enterprise tools**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- `-h, --host hostname`
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- `-p, --port jmx_port`
  The JMX port number.
- `-pw, --password jmxpassword`
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- `-pwf, --password-file jmx_password_filepath`
  The filepath to the file that stores JMX authentication credentials.
- `-u, --username jmx_username`
  The user name for authenticating with secure JMX.

**Command arguments**

- `--`
  Separates an option from an argument that could be mistaken for a option.
- `keyspace_name`
  The keyspace name.
- `table_name`
  The table name.

**Examples**

**Print compaction thresholds**

```sh
$ nodetool getcompactionthreshold cycling birthday_list
```

Current compaction thresholds for cycling/birthday_list:
  min = 4,  max = 32

**nodetool getcompactionthroughput**

Prints current compaction throughput in megabytes (MBs) per second.

**Synopsis**

```sh
$ >nodetool [connection_options] getcompactionthroughput
```
### Table 211: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

#### Definition

The short form and long form parameters are comma-separated.

#### Connection options

**-h, --host hostname**  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

**-p, --port jmx_port**
DataStax Enterprise tools

The JMX port number.

-\texttt{pw, --password jmxpassword}
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-\texttt{pwf, --password-file jmx\_password\_filepath}
  The filepath to the file that stores JMX authentication credentials.

-\texttt{u, --username jmx\_username}
  The user name for authenticating with secure JMX.

\textbf{Command arguments}

This command takes no arguments.

\textbf{Examples}

\textbf{To print current compaction throughput for the system}

\begin{verbatim}
$ nodetool -u username -pw password getcompactionthroughput
Current compaction throughput: 16 MB/s
\end{verbatim}

\textbf{nodetool getconcurrentcompactors}

Gets the number of concurrent compactors in the system.

\textbf{Synopsis}

\begin{verbatim}
$ nodetool [connection_options] getconcurrentcompactors
\end{verbatim}

\textbf{Table 212: Legend}

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>\textit{Italics}</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>\textbar{}</td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

To get number of concurrent compactors
nodetool -u joe -pw P@ssw0rd! getconcurrentcompactors

Current concurrent compactors in the system is:
2

nodetool getconcurrentviewbuilders

Display the number of concurrent materialized view builders in the system.

Synopsis

$ nodetool [connection_options] getconcurrentviewbuilders

Table 213: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- `-h, --host hostname`
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- `-p, --port jmx_port`
  The JMX port number.
- `-pw, --password jmxpassword`
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- `-pwf, --password-file jmx_password_filepath`
  The filepath to the file that stores JMX authentication credentials.
- `-u, --username jmx_username`
  The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Display the number of concurrent materialized view builders

```
$ nodetool getconcurrentviewbuilders

Current number of concurrent view builders in the system is:
6
```

**nodetool getendpoints**

Prints the endpoints that own the partition key.

Synopsis

```
$ nodetool  [connection_options] getendpoints
[--]  keyspace_name  table_name  partition_key
```
Table 214: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
The JMX port number.

-`pw`, --password `jmxpassword`
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-`pwf`, --password-file `jmx_password_filepath`
  The filepath to the file that stores JMX authentication credentials.

-`u`, --username `jmx_username`
  The user name for authenticating with secure JMX.

**Command arguments**

`--`
Separates an option from an argument that could be mistaken for a option.

`key`
Partition key of the end points you want to get.

`keyspace_name`
The keyspace name.

`table_name`
The table name.

**Examples**

**Print endpoints that own partition key**

For example, which nodes own partition key_1, key_2, and key_3?

**Note:** The partitioner returns a token for the key. DSE will return an endpoint regardless of whether data exists on the identified node for that token.

```
$ nodetool -h 127.0.0.1 -p 7100 getendpoints myks mytable key_1
127.0.0.2

$ nodetool -h 127.0.0.1 -p 7100 getendpoints myks mytable key_2
127.0.0.2
```

For example, consider the following table, which uses a primary key of `race_year` and `race_name`. This table is created in the `cycling` keyspace.

```
CREATE TABLE cycling.rank_by_year_and_name (
  race_year int,
  race_name text,
  rank int,
  cyclist_name text,
  PRIMARY KEY ((race_year, race_name), rank)
) WITH CLUSTERING ORDER BY (rank ASC);
```
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank) VALUES (2015, 'Tour of Japan - Stage 4 - Minami > Shinshu', 'Benjamin PRADES', 1);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank) VALUES (2015, 'Tour of Japan - Stage 4 - Minami > Shinshu', 'Adam PHELAN', 2);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank) VALUES (2015, 'Tour of Japan - Stage 4 - Minami > Shinshu', 'Thomas LEBAS', 3);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank) VALUES (2015, 'Giro d''Italia - Stage 11 - Forli > Imola', 'Ilnur ZAKARIN', 1);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank) VALUES (2015, 'Giro d''Italia - Stage 11 - Forli > Imola', 'Carlos BETANCUR', 2);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank) VALUES (2014, '4th Tour of Beijing', 'Phillippe GILBERT', 1);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank) VALUES (2014, '4th Tour of Beijing', 'Daniel MARTIN', 2);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank) VALUES (2014, '4th Tour of Beijing', 'Johan Esteban CHAVES', 3);

Given the previous information that was inserted into the table, run `nodetool getendpoints` and enter a value from the partition key. For example:

```
$ nodetool getendpoints cycling rank_by_year_and_name "2014"
10.255.100.150
```

The resulting output is the IP address of the replica that owns the partition key.

**To specify values that comprise the full primary key**

```
$ nodetool getendpoints cycling rank_by_year_and_name "2014:4th Tour of Beijing"
10.255.100.150
```

**nodetool getinterdcstreamthroughput**

Prints the outbound throttle (throughput cap) for all streaming file transfers between datacenters.

**Synopsis**

```
nodetool [connection_options] getinterdcstreamthroughput
```
Table 215: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>''Literal string''</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; </code></td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port
The JMX port number.

`-pw, --password jmxpassword`
- The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

`-pwf, --password-file jmx_password_filepath`
- The filepath to the file that stores JMX authentication credentials.

`-u, --username jmx_username`
- The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Print the outbound throttle (throughput cap) for streaming file transfers between datacenters

```
$ nodetool getinterdcstreamthroughput
```

The result is the default 200 megabits (Mbps) per second

```
Current inter-datacenter stream throughput: 200 Mb/s
```

`nodetool getlogginglevels`

Gets the runtime logging levels.

**Tip:** To change logging levels, use `nodetool setlogginglevel` *(page 1088).* See Configuring logging.

Synopsis

```
$ nodetool [connection_options] getlogginglevels
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  - The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
  - The JMX port number.
- **-pw, --password jmxpassword**
  - The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- **-pwf, --password-file jmx_password_filepath**
  - The filepath to the file that stores JMX authentication credentials.
- **-u, --username jmx_username**
  - The user name for authenticating with secure JMX.

Command arguments
This command takes no arguments.

Examples

Get runtime logging levels

$ nodetool getlogginglevels

<table>
<thead>
<tr>
<th>Logger Name</th>
<th>Log Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOT</td>
<td>INFO</td>
</tr>
<tr>
<td>DroppedAuditEventLogger</td>
<td>INFO</td>
</tr>
<tr>
<td>SLF4JAuditWriter</td>
<td>INFO</td>
</tr>
<tr>
<td>com.cryptsoft</td>
<td>OFF</td>
</tr>
<tr>
<td>com.datastax.bdp.db</td>
<td>DEBUG</td>
</tr>
<tr>
<td>com.datastax.bdp.search.solr.metrics.SolrMetricsEventListener</td>
<td>DEBUG</td>
</tr>
<tr>
<td>com.datastax.bdp.util.process.InternalServiceRunner</td>
<td>DEBUG</td>
</tr>
<tr>
<td>com.datastax.bdp.util.process.ServiceRunner</td>
<td>DEBUG</td>
</tr>
<tr>
<td>com.datastax.driver.core.NettyUtil</td>
<td>ERROR</td>
</tr>
<tr>
<td>org.apache.cassandra</td>
<td>DEBUG</td>
</tr>
<tr>
<td>org.apache.lucene.index</td>
<td>INFO</td>
</tr>
<tr>
<td>org.apache.solr.core.CassandraSolrConfig</td>
<td>WARN</td>
</tr>
<tr>
<td>org.apache.solr.core.RequestHandlers</td>
<td>WARN</td>
</tr>
<tr>
<td>org.apache.solr.core.SolrCore</td>
<td>WARN</td>
</tr>
<tr>
<td>org.apache.solr.handler.component</td>
<td>WARN</td>
</tr>
<tr>
<td>org.apache.solr.search.SolrIndexSearcher</td>
<td>WARN</td>
</tr>
<tr>
<td>org.apache.solr.update</td>
<td>WARN</td>
</tr>
<tr>
<td>org.apache.spark.rpc</td>
<td>ERROR</td>
</tr>
<tr>
<td>org.apache.spark.util.logging.FileAppender</td>
<td>OFF</td>
</tr>
</tbody>
</table>

nodetool getmaxhintwindow

Prints the maximum time that the database generates hints for an unresponsive node.

Synopsis

$ nodetool [connection_options] getmaxhintwindow

Table 217: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

-h, --host hostname  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port  
The JMX port number.

-pw, --password jmxpassword  
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-pwf, --password-file jmx_password_filepath  
The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username  
The user name for authenticating with secure JMX.

### Command arguments
This command takes no arguments.

Examples

Get maximum hint window

```bash
$ nodetool getmaxhintwindow
```

Result: the maximum time that the database generates hints for an unresponsive node is 10800000 milliseconds (3 hours).

```plaintext
Current max hint window: 10800000 ms
```

`nodetool getseeds`

Gets the IP list of the current seed node.

Synopsis

```bash
$ nodetool [connection_options] getseeds
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (....) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (‘) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

This command takes no arguments.

### Examples

**To get IP list of current seed node**

```
$ nodetool getseeds
```

Current list of seed node IPs excluding the current node IP: /10.100.15.1

**nodetool getsstables**

Prints the SSTable that owns the partition key.
Synopsis

$ nodetool [connection_options] getsstables
[-hf] [--] keyspace_name table_name partition_key

Table 219: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.
Connection options

- **h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **p, --port jmx_port**
  The JMX port number.

- **pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

```
--
```
Separates an option from an argument that could be mistaken for a option.

- **hf, --hex-format**
  Specify the key in hexadecimal string format.

- **key**
  Partition key of the end points you want to get.

- **keyspace_name**
  The keyspace name.

- **table_name**
  The table name.

Examples

Get the SSTable that owns the given partition key

```
$ nodetool getsstables cycling cyclist_name fb372533-eb95-4bb4-8685-6ef61e994caa
```

The result is:

```
/var/lib/cassandra/data/cycling/comments-b6239e719c0411e8a6f11f56fd0aa24a/aa-3-bti-Data.db
```

The hex string representation of the partition key is useful to resolve errors. For example, find out which SSTable owns the faulty partition key for this exception:

```
java.lang.AssertionError: row DecoratedKey(2769066505137675224, 00040000002e000000800000153441a3ef000) received out of order wrt DecoratedKey(2774747040849866654, 000400000019b00000080000015348847eb200)
```

When the primary key of the given table is a blob, get the DecoratedKey from the hexadecimal representation of the partition key:
nodetool getsstables -hf cycling stats
00040000002e00000800000153441a3ef000
/var/lib/cassandra/data/cycling/comments-
b6239e719c0411e8a6f11f5cd5459987/aa-2-bti-Data.db

Get the SSTables by specifying the full primary key

$ nodetool getsstables cycling rank_by_year_and_name "2014:4th Tour of Beijing"
/var/lib/cassandra/data/cycling/comments-
b6239e719c0411e8a6f11f5cd5459987/aa-2-bti-Data.db

nodetool getstreamthroughput

Gets the throughput throttle for streaming file transfers.

Synopsis

$ nodetool [connection_options] getstreamthroughput

Table 220: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td><em>Literal string</em></td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( <code>&lt; &gt;</code> ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( <code>;</code> ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( <code>--</code> ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks ( <code>‘</code> ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity=’xml_entity_type’</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

This command takes no arguments.

### Examples

**To get the throughput throttle in megabits per second**

```
$ nodetool getstreamthroughput
Current stream throughput: 200 Mb/s
```
DataStax Enterprise tools

Current streaming connections per host: 1

nodetool gettimeout

Prints the current timeout values in milliseconds.

   Tip: To change the timeout, use nodetool settimeout (page 1094).

Synopsis

$ nodetool [connection_options] gettimeout
    [--] [timeout_type]

Table 221: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
Definition

The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port
The JMX port number.

-pw, --password jmxpassword
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-pwf, --password-file jmx_password_filepath
The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username
The user name for authenticating with secure JMX.

Command arguments

--
Separates an option from an argument that could be mistaken for a option.

timeout_type
The timeout type: read, range, write, counterwrite, cascontention, truncate, streamingsocket, or misc (general rpc_timeout_in_ms).

Examples

To get timeouts for all types

$ nodetool gettimeout

Current timeout for type read: 5000 ms
Current timeout for type range: 10000 ms
Current timeout for type write: 2000 ms
Current timeout for type counterwrite: 5000 ms
Current timeout for type cascontention: 1000 ms
Current timeout for type truncate: 60000 ms
Current timeout for type misc: 10000 ms

To get timeout for read requests
nodetool gettimeout read

Current timeout for type read: 5000 ms

nodetool gettraceprobability

Prints the current trace probability value.

**Tip:** To set the trace probability, see `nodetool settraceprobability (page 1096)`.

**Synopsis**

```bash
$ nodetool [connection_options] gettraceprobability
```

**Table 222: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>'&lt; &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port
The JMX port number.

-pw, --password jmxpassword
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-pwf, --password-file jmx_password_filepath
The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username
The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Print current trace probability value

```
$ nodetool gettraceprobability
Current trace probability: 0.10
```

**nodetool gossipinfo**

Shows the gossip information to discover broadcast protocol between nodes in a cluster.

Synopsis

```
$ nodetool [connection_options] gossipinfo
```
### Table 223: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
The JMX port number.
-\texttt{-pw, --password jmxpassword}
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
-\texttt{-pwf, --password-file jmx\_password\_filepath}
  The filepath to the file that stores JMX authentication credentials.
-\texttt{-u, --username jmx\_username}
  The user name for authenticating with secure JMX.

\textbf{Command arguments}

This command takes no arguments.

\textbf{Examples}

\textbf{Show gossip information for cluster}

\begin{verbatim}
$ nodetool gossipinfo

localhost/127.0.0.1
generation:1532896921
heartbeat:2038494
STATUS:1611484:NORMAL,8242717283351148695
LOAD:2038483:262546.0
SCHEMA:975284:d4f18346-f81f-3786-aed4-40e03558b299
DC:26:Search
RACK:18:rack1
RELEASE\_VERSION:4.4.0.0.602
STORAGE\_PORT:8:7000
STORAGE\_PORT\_SSL:9:7001
JMX\_PORT:10:7199
TOKENS:1611483:<hidden>
\end{verbatim}

\textbf{nodetool handoffwindow}

Prints current hinted handoff window.

\textbf{Synopsis}

\begin{verbatim}
$ nodetool [connection\_options] handoffwindow
\end{verbatim}
### Table 224: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Literal keyword.</td>
</tr>
<tr>
<td><strong>Lowercase</strong></td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( {} ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
The JMX port number.

-pw, --password jmxpassword
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-pwf, --password-file jmx_password_filepath
The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username
The user name for authenticating with secure JMX.

Command arguments
This command takes no arguments.

Examples
Print current hinted handoff window

$ nodetool handoffwindow

The maximum time that the database generates hints for an unresponsive node is 10800000 ms (3 hours).

Hinted handoff window is 10800000

nodetool help
Provides a synopsis and brief description of each nodetool command.

Synopsis

$ nodetool [connection_options] help [command_name]

Table 225: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;/&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**  
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**  
  The JMX port number.

- **-pw, --password jmxpassword**  
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**  
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**  
  The user name for authenticating with secure JMX.

### Command arguments

- **command_name**  
  Name of nodetool command.
Examples

Print list and brief description of all nodetool commands

$ nodetool help

The most commonly used nodetool commands are:

- **abortrebuild** - Abort a currently running rebuild operation. Currently active streams will finish but no new streams will be started.
- **assassinate** - Forcefully remove a dead node without re-replicating any data. Use as a last resort if you cannot remove node.
- **bootstrap** - Monitor/manage node's bootstrap process.
- **cleanup** - Triggers the immediate cleanup of keys no longer belonging to a node. By default, clean all keyspaces.
- **clearsnapshot** - Remove the snapshot with the given name from the given keyspaces. If no snapshotName is specified we will remove all snapshots.
- **compact** - Force a (major) compaction on one or more tables or user-defined compaction on given SSTables.
- **compactionhistory** - Print history of compaction.
- **compactionstats** - Print statistics on compactions.
- **decommission** - Decommission the *node I am connecting to*.
- **describecluster** - Print the name, snitch, partitioner and schema version of a cluster.
- **describereading** - Shows the token ranges info of a given keyspace.
- **disableautocompaction** - Disable autocompaction for the given keyspace and table.
- **disablebackup** - Disable incremental backup.
- **disablebinary** - Disable native transport (binary protocol).
- **disablegossip** - Disable gossip (effectively marking the node down).
- **disablehandoff** - Disable storing hinted handoffs.
- **disablehintsfordc** - Disable hints for a data center.
- **drain** - Drain the node (stop accepting writes and flush all tables).
- **enableautocompaction** - Enable autocompaction for the given keyspace and table.
- **enablebackup** - Enable incremental backup.
- **enablebinary** - Reenable native transport (binary protocol).
- **enablegossip** - Reenable gossip.
- **enablehandoff** - Reenable future hints storing on the current node.
- **enablehintsfordc** - Enable hints for a data center that was previously disabled.
- **failuredetector** - Shows the failure detector information.
- **flush** - Flush one or more tables.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>garbagecollect</td>
<td>Remove deleted data from one or more tables</td>
</tr>
<tr>
<td>gcstats</td>
<td>Print GC Statistics</td>
</tr>
<tr>
<td>getbatchlogreplaythrottle</td>
<td>Print batchlog replay throttle in KB/s.</td>
</tr>
<tr>
<td>This is reduced proportionally to the number of nodes in the cluster.</td>
<td></td>
</tr>
<tr>
<td>getcompactionthreshold</td>
<td>Print min and max compaction thresholds</td>
</tr>
<tr>
<td>for a given table</td>
<td></td>
</tr>
<tr>
<td>getcompactionthroughput</td>
<td>Get the MB/s throughput cap for compaction in the system.</td>
</tr>
<tr>
<td>compaction in the system</td>
<td></td>
</tr>
<tr>
<td>getconcurrentcompactors</td>
<td>Get the number of concurrent compactors</td>
</tr>
<tr>
<td>in the system.</td>
<td></td>
</tr>
<tr>
<td>getconcurrentviewbuilders</td>
<td>Print the number of concurrent view builders in the system.</td>
</tr>
<tr>
<td>getendpoints</td>
<td>Get the end points that owns the key</td>
</tr>
<tr>
<td>getinterdcstreamthroughput</td>
<td>Print the Mb/s throughput cap for inter-datacenter streaming in the system.</td>
</tr>
<tr>
<td>datacenter streaming in the system</td>
<td></td>
</tr>
<tr>
<td>getlogginglevels</td>
<td>Get the runtime logging levels</td>
</tr>
<tr>
<td>getmaxhintwindow</td>
<td>Print the max hint window in ms</td>
</tr>
<tr>
<td>getseeds</td>
<td>Get the currently in use seed node IP</td>
</tr>
<tr>
<td>list excluding the node IP</td>
<td>Get the sstable filenames that own the key</td>
</tr>
<tr>
<td>getsstables</td>
<td></td>
</tr>
<tr>
<td>getstreamthroughput</td>
<td>Print the Mb/s throughput cap for streaming in the system</td>
</tr>
<tr>
<td>streaming in the system</td>
<td>Print the timeout of the given type in ms</td>
</tr>
<tr>
<td>gettimeout</td>
<td></td>
</tr>
<tr>
<td>gettraceprobability</td>
<td>Print the current trace probability</td>
</tr>
<tr>
<td>gossipinfo</td>
<td>Shows the gossip information for the cluster</td>
</tr>
<tr>
<td>handoffwindow</td>
<td>Print current hinted handoff window</td>
</tr>
<tr>
<td>help</td>
<td>Display help information</td>
</tr>
<tr>
<td>info</td>
<td>Print node information (uptime, load, ...)</td>
</tr>
<tr>
<td>load, ...)</td>
<td></td>
</tr>
<tr>
<td>inmemorystatus</td>
<td>Returns a list of the in-memory tables for this node and the amount of memory each table is using, or information about a single table if the keyspace and columnfamily are given.</td>
</tr>
<tr>
<td>invalidatecountercache</td>
<td>Invalidate the counter cache</td>
</tr>
<tr>
<td>invalidatekeycache</td>
<td>Invalidate the key cache</td>
</tr>
<tr>
<td>invalidaterowcache</td>
<td>Invalidate the row cache</td>
</tr>
<tr>
<td>join</td>
<td>Join the ring</td>
</tr>
<tr>
<td>listsnapshots</td>
<td>Lists all the snapshots along with the size on disk and true size.</td>
</tr>
<tr>
<td>mark_unrepaired</td>
<td>Mark all SSTables of a table or keyspace as unrepaired. Use when no longer running incremental repair on a table or keyspace.</td>
</tr>
<tr>
<td>move</td>
<td>Move node on the token ring to a new</td>
</tr>
<tr>
<td>token</td>
<td></td>
</tr>
<tr>
<td>netstats</td>
<td>Print network information on provided</td>
</tr>
<tr>
<td>host (connecting node by default)</td>
<td>Manage the NodeSync service on the connected node</td>
</tr>
<tr>
<td>nodesyncservice</td>
<td></td>
</tr>
<tr>
<td>connected node</td>
<td></td>
</tr>
<tr>
<td>pausehandoff</td>
<td>Pause hints delivery process</td>
</tr>
</tbody>
</table>
proxyhistograms
operations
rangekeysample
debuild
node (similarly to bootstrap)
indexes for a given table
reload
without restart
refresh
sizeestimates
reloadlocalschema
tables
reloadseeds
node provider
reloctriggers
relocatesstables
removenode
force completion of pending removal or remove provided ID
repair
repair_admin
sessions
replaybatchlog
finish
resetlocalschema
resumehandoff
ring
cruba
sequence
tables
setcachekeystosave
for faster post-restart warmup. 0 to disable
setcompactionthreshold
for a given table
setcompactionthroughput
compaction in the system, or 0 to disable throttling
setconcurrentcompactors
in the system.
setconcurrentviewbuilders
sethintedhandoffthrottlekb
setinterdcstreamthroughput
datacenter streaming in the system, or 0 to disable throttling
setlogginglevel
component or class. Will reset to the initial configuration if called
with no parameters.
load newly placed SSTables to the system
Reload the seed node list from the seed
Reload trigger classes
Relocates sstables to the correct disk
Show status of current node removal, force completion of pending removal or remove provided ID
Repair one or more tables
list and fail incremental repair
Kick off batchlog replay and wait for finish
Reset node's local schema and resync
Resume hints delivery process
Print information about the token ring
Scrub (rebuild sstables for) one or more tables
Run multiple nodetool commands from a file, resource or stdin in sequence. Common options (host, port, username, password) are passed to child commands.
Set batchlog replay throttle in KB per second, or 0 to disable throttling. This will be reduced proportionally to the number of nodes in the cluster.
Set global key, row, and counter cache capacities (in MB units)
Set number of keys saved by each cache
for faster post-restart warmup. 0 to disable
Set min and max compaction thresholds
Set the MB/s throughput cap for compaction in the system, or 0 to disable throttling
Set number of concurrent compactors in the system.
Set the number of concurrent view builders in the system
Set hinted handoff throttle in kb per second, per delivery thread.
Set the Mb/s throughput cap for inter-datacenter streaming in the system, or 0 to disable throttling
Set the log level threshold for a given component or class. Will reset to the initial configuration if called
with no parameters.
Print statistic histograms for network operations
Shows the sampled keys held across all keyspaces
Rebuild data by streaming from other nodes (similarly to bootstrap)
A full rebuild of native secondary indexes for a given table
Load newly placed SSTables to the system
Refresh system.size_estimates
Reload local node schema from system tables
Reload the seed node list from the seed
Relocates trigger classes
Relocates sstables to the correct disk
Show status of current node removal, force completion of pending removal or remove provided ID
Repair one or more tables
list and fail incremental repair
Kick off batchlog replay and wait for finish
Reset node's local schema and resync
Resume hints delivery process
Print information about the token ring
Scrub (rebuild sstables for) one or more tables
Run multiple nodetool commands from a file, resource or stdin in sequence. Common options (host, port, username, password) are passed to child commands.
Set batchlog replay throttle in KB per second, or 0 to disable throttling. This will be reduced proportionally to the number of nodes in the cluster.
Set global key, row, and counter cache capacities (in MB units)
Set number of keys saved by each cache
for faster post-restart warmup. 0 to disable
Set min and max compaction thresholds
Set the MB/s throughput cap for compaction in the system, or 0 to disable throttling
Set number of concurrent compactors in the system.
Set the number of concurrent view builders in the system
Set hinted handoff throttle in kb per second, per delivery thread.
Set the Mb/s throughput cap for inter-datacenter streaming in the system, or 0 to disable throttling
Set the log level threshold for a given component or class. Will reset to the initial configuration if called
with no parameters.
**DataStax Enterprise tools**

- `setmaxhintwindow` Set the specified max hint window in ms
- `setstreamthroughput` Set the Mb/s throughput cap for streaming in the system, or 0 to disable throttling
- `settimeout` Set the specified timeout in ms, or 0 to disable timeout
- `settraceprobability` Sets the probability for tracing any given request to value. 0 disables, 1 enables for all requests, 0 is the default
- `sjk` Run commands of 'Swiss Java Knife'. Run 'nodetool sjk --help' for more information.
- `snapshot` Take a snapshot of specified keyspaces or a snapshot of the specified table
- `status` Print cluster information (state, load, IDs, ...)
- `statusautocompaction` Status of autocompaction of the given keyspace and table
- `statusbackup` Status of incremental backup
- `statusbinary` Status of native transport (binary protocol)
- `statusgossip` Status of gossip
- `statushandoff` Status of storing future hints on the current node
- `stop` Stop compaction
- `stopdaemon` Stop DSE daemon
- `tablehistograms` Print statistic histograms for a given table
- `tablestats` Print statistics on tables
- `toppartitions` Sample and print the most active partitions for a given column family
- `tpstats` Print usage statistics of thread pools
- `truncatehints` Truncate all hints on the local node, or truncate hints for the endpoint(s) specified.
- `upgradesstables` Rewrite sstables (for the requested tables) that are not on the current version (thus upgrading them to said current version)
- `verify` Verify (check data checksum for) one or more tables
- `version` Print DSE DB version
- `viewbuildstatus` Show progress of a materialized view build

---

**Get synopsis and brief description of nodetool netstats**

$ nodetool help netstats

**NAME**

nodetool netstats - Print network information on provided host (connecting node by default)

**SYNOPSIS**
nodetool info

Provides node information, including the token and on disk storage (load) information, times started (generation), uptime in seconds, and heap memory usage.

Synopsis

```
$ nodetool [connection_options] info
    [-T]
```

Table 226: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>'&lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>&lt;xml_entity=&quot;xml_entity_type</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

- **-T, --tokens**
  Show all tokens.

### Examples

**Print node information with token**

```
$ nodetool info -T
```
Result:

<table>
<thead>
<tr>
<th>ID</th>
<th>3b8e8192-c1d3-4b01-a792-9673b4e377c1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gossip active</td>
<td>true</td>
</tr>
<tr>
<td>Native Transport active</td>
<td>true</td>
</tr>
<tr>
<td>Load</td>
<td>255.29 KiB</td>
</tr>
<tr>
<td>Generation No</td>
<td>1532896921</td>
</tr>
<tr>
<td>Uptime (seconds)</td>
<td>1882997</td>
</tr>
<tr>
<td>Heap Memory (MB)</td>
<td>604.32 / 4012.00</td>
</tr>
<tr>
<td>Off Heap Memory (MB)</td>
<td>0.00</td>
</tr>
<tr>
<td>Data Center</td>
<td>Search</td>
</tr>
<tr>
<td>Rack</td>
<td>rack1</td>
</tr>
<tr>
<td>Exceptions</td>
<td>0</td>
</tr>
<tr>
<td>Key Cache</td>
<td>entries 0, size 0 bytes, capacity 100 MiB, 0 hits, 0 requests, NaN recent hit rate, 14400 save period in seconds</td>
</tr>
<tr>
<td>Row Cache</td>
<td>entries 0, size 0 bytes, capacity 0 bytes, 0 hits, 0 requests, NaN recent hit rate, 0 save period in seconds</td>
</tr>
<tr>
<td>Counter Cache</td>
<td>entries 0, size 0 bytes, capacity 50 MiB, 1 hits, 2 requests, 0.500 recent hit rate, 7200 save period in seconds</td>
</tr>
<tr>
<td>Chunk Cache</td>
<td>entries 7871, size 260.79 MiB, capacity 2.79 GiB, 7871 misses, 14839137 requests, 0.999 recent hit rate, 937.529 microseconds miss latency</td>
</tr>
<tr>
<td>Percent Repaired</td>
<td>100.0%</td>
</tr>
<tr>
<td>Token</td>
<td>8242717283351148695</td>
</tr>
</tbody>
</table>

**nodetool inmemorystatus**

Returns a list of the in-memory tables and the amount of memory each table is using.

**Synopsis**

```
$ nodetool [connection_options] inmemorystatus
[--] [keyspace_name table_name [table_name ...]]
```

**Table 227: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

- **--**
  Separates an option from an argument that could be mistaken for a option.

- **keyspace_name**
  The keyspace name.
table_name
One or more table names, separated by a space.

Examples

Print information on all in-memory tables

$ nodetool inmemorystatus

Result:

Max Memory to Lock: 3209MB
Current Total Memory Locked: 0MB
Current Total Memory Not Able To Lock: 0MB
No MemoryOnlyStrategy tables found.

Print information on in-memory tables in the cycling keyspace and popular_count table

$ nodetool inmemorystatus cycling popular_count

Result:

nodetool: Keyspace cycling Table birthday_list is not using MemoryOnlyStrategy.

nodetool invalidatecountercache

Resets global counter cache parameter to save all counter keys. Invalidates the counter_cache_keys_to_save setting in cassandra.yaml to enable the default behavior to save all keys.

Synopsis

$ nodetool [connection_options] invalidatecountercache

Table 228: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Repeatable. An ellipsis ( <code>...</code> ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td><code>'Literal string'</code></td>
<td>Single quotation ( <code>'</code> ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ( <code>{ }</code> ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( <code>&lt; &gt;</code> ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( <code>;</code> ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( <code>--</code> ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; </code></td>
<td>Search CQL only: Single quotation marks ( <code>'</code> ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

-h, --host hostname
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port
The JMX port number.

-pw, --password jmxpassword
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-pwf, --password-file jmx_password_filepath
The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username
The user name for authenticating with secure JMX.

**Command arguments**
This command takes no arguments.

Examples

**Reset counter_cache_keys_to_save to save all keys**

```
$ nodetool -u joe -pw P@ssw0rd! invalidatecountercache
```

**nodetool invalidatekeycache**

Clears the key cache. The key cache is present only until nodetool sstableupgrades is run.

**Synopsis**

```
$ nodetool [connection_options] invalidatekeycache
```

**Table 229: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt; &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

This command takes no arguments.

### Examples

#### Clears the key cache

```
$ nodetool invalidkeycache
```

**nodetool invalidaterowcache**

Invalidate the row_cache_keys_to_save setting in cassandra.yaml to enable the default behavior to save all keys.

### Synopsis

```
$ nodetool [connection_options] invalidaterowcache
```
### Table 230: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
DataStax Enterprise tools

The JMX port number.

-\texttt{pw, --password jmxpassword}
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-\texttt{pwf, --password-file jmx_password_filepath}
  The filepath to the file that stores JMX authentication credentials.

-\texttt{u, --username jmx_username}
  The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Resets row_cache_keys_to_save parameter to save all keys

```
$ nodetool invalidate_rowcache
```

\textbf{nodetool join}

Joins the node to the ring. Valid only when the node was initially \textit{not} started in the ring with the \texttt{-Djoin\_ring=false} (page 158) start-up parameter. The joining node must be properly configured with the required cassandra.yaml options for seed list, initial token, and auto-bootstrapping.

Synopsis

```
$ nodetool \langle connection_options \rangle join
```

\textbf{Table 231: Legend}

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>\textit{Italics}</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (‘’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (‘’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- `-h, --host hostname`
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- `-p, --port jmx_port`
  The JMX port number.
- `-pw, --password jmxpassword`
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- `-pf, --password-file jmx_password_filepath`
  The filepath to the file that stores JMX authentication credentials.
- `-u, --username jmx_username`
  The user name for authenticating with secure JMX.

**Command arguments**

This command takes no arguments.

**Examples**

**Join node to ring**
DataStax Enterprise tools

nodetool -u admin1 join

nodetool listendpointspendinghints

Prints information about hints that the node has for other nodes.

Hint information includes Host ID, Address, Rack, DC (datacenter), node status, total number of hints and files, and timestamp of newest and oldest hints.

Synopsis

$ nodetool [connection_options] -h hostname listendpointspendinghints

Table 232: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- `-h, --host hostname`
  - The hostname or IP address of the remote node to get information about hints that the node has for other nodes.
- `-p, --port jmx_port`
  - The JMX port number.
- `-pw, --password jmxpassword`
  - The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- `-pwf, --password-file jmx_password_filepath`
  - The filepath to the file that stores JMX authentication credentials.
- `-u, --username jmx_username`
  - The user name for authenticating with secure JMX.

**Command arguments**

This command takes no arguments.

**Examples**

**To print relevant hint information about the local node endpoints**

```bash
nodetool listendpointspendinghints
```

<table>
<thead>
<tr>
<th>Host ID</th>
<th>Total hints</th>
<th>Total files</th>
<th>Newest</th>
<th>Address</th>
<th>Rack</th>
<th>DC</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>5762b140-3fdf-4057-9ca7-05c070ccc9c3</td>
<td>25098</td>
<td>2</td>
<td>2018-09-18 14:05:18,835</td>
<td>127.0.0.2</td>
<td>rack1</td>
<td>datacenter1</td>
<td>DOWN</td>
</tr>
</tbody>
</table>

**nodetool listsnapshots**

Lists all the snapshots, along with the size on disk, and true size.

**Synopsis**

```bash
$ nodetool [connection_options] listsnapshots
```
Table 233: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>',&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
-p, --port jmx_port
The JMX port number.

**-pw, --password jmxpassword**
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

**-pf, --password-file jmx_password_filepath**
The filepath to the file that stores JMX authentication credentials.

**-u, --username jmx_username**
The user name for authenticating with secure JMX.

### Command arguments

This command takes no arguments.

### Examples

#### List snapshots

```
$ nodetool listsnapshots
```

<table>
<thead>
<tr>
<th>Snapshot name</th>
<th>Keyspace name</th>
<th>Column family name</th>
<th>True size</th>
<th>Size on disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1534456548264</td>
<td>cycling</td>
<td>popular_count</td>
<td>4.84 KiB</td>
<td>5.7 KiB</td>
</tr>
<tr>
<td>1534456548264</td>
<td>cycling</td>
<td>calendar</td>
<td>5.33 KiB</td>
<td>6.34 KiB</td>
</tr>
<tr>
<td>1534456548264</td>
<td>cycling</td>
<td>comments</td>
<td>6.83 KiB</td>
<td>7.79 KiB</td>
</tr>
<tr>
<td>1534456548264</td>
<td>cycling</td>
<td>birthday_list</td>
<td>5.22 KiB</td>
<td>6.09 KiB</td>
</tr>
</tbody>
</table>

Total TrueDiskSpaceUsed: 22.21 KiB

#### nodetool mark_unrepaired

Marks all SSTables of a table or keyspace as unrepaired.

**Warning:** This operation marks all targeted SSTables as unrepaired, potentially creating new compaction tasks. Use only if you are no longer running incremental repair on this node.

**Note:** When no table name is specified, marks all tables in the keyspace as unrepaired.

### Synopsis

```
$ nodetool [connection_options] mark_unrepaired
[-f] [--] keyspace_name [table_name ...]
```

### Table 234: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>{}</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>’Literal string’</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>’ &lt;schema&gt; ... &lt;/schema&gt; ‘</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity=’xml_entity_type’</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**  
The JMX port number.

- **-pw, --password jmxpassword**  
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
-pf, --password-file jmx_password_filepath
   The filepath to the file that stores JMX authentication credentials.
-u, --username jmx_username
   The user name for authenticating with secure JMX.

Command arguments

--
   Separates an option from an argument that could be mistaken for a option.
-f, --force
   Confirms the operation.

keyspace_name
   The keyspace name.

   table_name
   One or more table names, separated by a space.

Examples

Mark cycling keyspace unrepaired

$ nodetool mark_unrepaired cycling

Result:

nodetool: WARNING: This operation will mark all SSTables of keyspace
cycling as unrepaired, potentially creating new compaction tasks. Only
use this when no longer running incremental repair on this node. Use --
force option to confirm.

nodetool move

Moves the node on the token ring to a new token, generally used to shift tokens slightly.

Additional syntax is required to move a node to a negative tokens:

- Use the preferred double hyphen (--):

   $ nodetool move -- -9223372036854775808

- Escape the hyphen with a backslash (\):

   $ nodetool move \-9223372036854775808

OpsCenter provides an option in the Nodes UI for Moving a node.

Synopsis

$ nodetool [connection_options] move
   [--] new_token
Table 235: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port
The JMX port number.

-\texttt{pw, --password \textit{jmxpassword}}
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-\texttt{pf, --password-file \textit{jmx\_password\_filepath}}
  The filepath to the file that stores JMX authentication credentials.

-\texttt{u, --username \textit{jmx\_username}}
  The user name for authenticating with secure JMX.

\textbf{Command arguments}

\texttt{--}
  Separates an option from an argument that could be mistaken for a option.

\texttt{token}
  The new token. Number in partition range. For Murmur3Partitioner (default): \(-2^{63}\) to \(+2^{63}-1\).

\textbf{Examples}

\textbf{Move node to new token}

\begin{verbatim}
$ nodetool move 3074457345618258602
\end{verbatim}

\textbf{Move node to new negative token}

\begin{verbatim}
$ nodetool move \-9223372036854775808
\end{verbatim}

\textbf{nodetool netstats}

Prints network information about the host.

The output includes the following information:

- JVM settings
- Mode - The operational mode of the node: JOINING, LEAVING, NORMAL, DECOMMISSIONED, CLIENT
- Read repair statistics
- Attempted - The number of successfully completed read repair operations.
- Mismatch (blocking) - The number of read repair operations since server restart that blocked a query.
- Mismatch (background) - The number of read repair operations since server restart performed in the background.
- Pool name - Information about client read and write requests by thread pool size.
- Active, pending, and completed number of commands and responses

\textbf{Synopsis}

\begin{verbatim}
$ nodetool [connection\_options] netstats
\end{verbatim}
Table 236: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
-p, --port jmx_port
    The JMX port number.
-pw, --password jmxpassword
    The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
-pwf, --password-file jmx_password_filepath
    The filepath to the file that stores JMX authentication credentials.
-u, --username jmx_username
    The user name for authenticating with secure JMX.

Command arguments

-H, --human-readable
    Display bytes in human readable form: KiB (kibibyte), MiB (mebibyte), GiB (gibibyte), TiB (tebibyte).
-h, --host hostname
    The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

Examples

Get network information of local node

$ nodetool netstats

The output is:

<table>
<thead>
<tr>
<th>Pool Name</th>
<th>Active</th>
<th>Pending</th>
<th>Completed</th>
<th>Dropped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large messages</td>
<td>n/a</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Small messages</td>
<td>n/a</td>
<td>0</td>
<td>23295</td>
<td>0</td>
</tr>
<tr>
<td>Gossip messages</td>
<td>n/a</td>
<td>0</td>
<td>1853117</td>
<td>0</td>
</tr>
</tbody>
</table>

nodetool nodesyncservice

Use the following subcommands to manage the NodeSync service on the connected node.

The NodeSync service automatically starts when a DataStax Enterprise node is started.

**Tip:** The service runs continuous repair for tables that have nodesync set to true. By default, the table option is set to false (disabled). Use CQL ALTER TABLE to
change the NodeSync setting on a specific table or dse nodesync (page 1240) to change the setting on multiple tables.

**nodetool nodesyncservice enable**

Starts up the NodeSync service on the connected host. By default, NodeSync service automatically starts with DataStax Enterprise, but keyspaces and tables must be explicitly opted in.

Use Lifecycle Manager for Enabling keyspaces and tables for monitoring NodeSync in OpsCenter.

**Synopsis**

```bash
$ nodetool [connection_options] nodesyncservice enable
[-t timeoutSec]
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

- **-t seconds, --timeout seconds**
  Time to wait in seconds for the service to start.
  Default: 120 (2 minutes).

**Examples**

**Start up the NodeSync service on the local host**

1. Run the enable command:

   ```sh
   $ nodetool nodesyncservice enable
   ```

2. Check the status:

   ```sh
   $ nodetool nodesyncservice status --boolean-output
   ```
true

Start up the NodeSync service on the host northeast

1. Run the enable command:

   $ nodetool -h northeast nodesyncservice enable

2. Check the status:

   $ nodetool -h northeast nodesyncservice status

   The NodeSync service is running

`nodetool nodesyncservice disable`

Shuts down the NodeSync service on the connected host. Shut down occurs after the in-progress segment validations complete, or when the timeout period is reached.

Synopsis

$ nodetool [connection_options] nodesyncservice disable
[-f] [-t timeoutSec]

Table 238: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({} ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (-- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

- **-f, --force**
  Forces service to shutdown immediately without completing segment validations that are currently running.

- **-t seconds, --timeout seconds**
  Time to wait in seconds for the service to start.
  Default: 120 (2 minutes).
Examples

Shut down NodeSync service on local host without waiting for validations that in process to complete

1. Run the disable command:

   ```
   $ nodetool nodesyncservice disable -f
   ```

2. Check the status:

   ```
   $ nodetool nodesyncservice status --boolean-output
   false
   ```

Shut down NodeSync service on host northeast using a timeout period of five minutes

1. Run the disable command:

   ```
   $ nodetool -h northeast nodesyncservice disable -t 300
   ```

2. Check the status:

   ```
   $ nodetool -h northeast nodesyncservice status
   The NodeSync service is running
   ```

**nodetool nodesyncservice getrate**

Returns the configured synchronization rate-limit of the connected node.

**Tip:** Set the rate limit temporarily using `nodetool nodesyncservice setrate (page 1026)`. To persist the rate limit, use the `rate_in_kb (page 79)` setting in `cassandra.yaml`.

**Synopsis**

```
$ nodetool [connection_options] nodesyncservice getrate
```
Syntax conventions | Description
--- | ---
**Italics** | Variable value. Replace with a valid option or user-defined value.
[] | Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.
() | Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.
| | Or. A vertical bar ( | ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
... | Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' Literal string' | Single quotation (' ') marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ key:value } | Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.
<datatype1,datatype2> | Set, list, map, or tuple. Angle brackets (< >) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
cql_statement; | End CQL statement. A semicolon (;) terminates all CQL statements.
[ -- ] | Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' &lt;schema&gt; ... &lt;/schema&gt; ' | Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.
@xml_entity=’xml_entity_type’ | Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
DataStax Enterprise tools

- **pwf**, **--password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **u**, **--username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Get configured validation rate of local host

```
$ nodetool nodesyncservice getrate
Current rate limit=1024 KB/s
```

**nodetool nodesyncservice ratesimulator**

Simulates rates necessary to achieve the NodeSync deadline based on configurable assumptions. Rate simulations are useful, but in production simulations are not a viable substitute for monitoring NodeSync and adjusting the rate.

Restriction: Do not use this command on a keyspace with RF=1 or on a single node cluster.

Monitor NodeSync status using OpsCenter. See NodeSync metrics.

Synopsis

```
$ nodetool [connection_options] nodesyncservice ratesimulator
[--deadline-overrides keyspace_name.table_name:deadline_target_time, ...]
[-e keyspace_name.table_name, ...]
[help] [-i keyspace_name.table_name, ...]
[--ignore-replication-factor]
[simulate -ds factor_integer -rs factor_integer -sg factor_integer | recommended | recommended_minimum | theoretical_minimum]
[] [-v]
```

**Table 240: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
  The JMX port number.
- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- **-pwf, --password-file jmx_password_filepath**
  The file path to the file that stores JMX authentication credentials.
- **-u, --username jmx_username**
The user name for authenticating with secure JMX.

**Command arguments**

--**deadline-overrides**
Allows override on the configure deadline for some/all of the tables in the simulation.

-ds, --**deadline-safety-factor**
Specify factor (integer) to decrease table deadlines to account for imperfect conditions.
Only for `simulate` sub-command.

-e, --**excludes keyspace_name.table_name,** ...
A comma-separated list of tables to exclude from the simulation when NodeSync is enabled on the server-side; this simulates the impact on the rate of disabling NodeSync on those tables.

**help**
Displays options and usage instructions.

--**ignore-replication-factor**
Ignores the replication factor for the simulation. Without this option, the default assumes that NodeSync runs on every node of the cluster (which is highly recommended) and assumes that validation work is spread among replicas. When NodeSync runs on every node of the cluster, each node must validate the fraction 1/RF of the data the node owns. This option removes that assumption, and computes a rate that accounts for all the data the node stores.

-i, --**includes keyspace_name.table_name,** ...
A comma-separated list of tables to include in the simulation when NodeSync is not enabled server-side; simulates the impact on the rate of enabling NodeSync on those tables.

-rs, --**rate-safety-factor factor_integer**
Represents a factor of how much to increase the final rate to account for imperfect conditions. Applies only to the `simulate` sub-command.

-sg, --**size-growth-factor factor_integer**
Represents a factor of how much to increase data sizes to account for data growth. Applies only to the `simulate` sub-command.

-v, --**verbose**
Provides details on how the simulation is carried out. Displays all steps taken by the simulation. Although this option is useful for understanding the simulations, results can be large or may be excessive if many tables exist.

**Examples**

**Simulate rates for comments table**

```
$ nodetool nodesyncservice ratesimulator -i cycling.comments
Computed rate: 420kB/s.
```

Simulate rates with new target times for the comments table
$ nodetool nodesyncservice ratesimulator --deadline-overrides
cycling.comments:20h

Simulate example

1. In CQL, create tables within a keyspace of RF > 1 and NodeSync enabled. For example:

```
CREATE KEYSPACE cycling WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 2};
USE cycling;
CREATE TABLE comments (record_id timeuuid, id uuid, commenter text, comment text, created_at timestamp, PRIMARY KEY (id, created_at)) WITH nodesync={'enabled': 'true'};
CREATE TABLE comments2 (record_id timeuuid, id uuid, commenter text, comment text, created_at timestamp, PRIMARY KEY (id, created_at)) WITH nodesync={'enabled': 'true'};
```

2. Insert data into the tables. For example:

```
INSERT INTO cycling.comments (record_id, id, created_at, comment, commenter) values (now(), e7ae5cf3-d358-4d99-b900-85902fda9bb0, '2017-02-14 12:43:20-0800', 'Raining too hard should have postponed', 'Alex');
INSERT INTO cycling.comments (record_id, id, created_at, comment, commenter) values (now(), e7ae5cf3-d358-4d99-b900-85902fda9bb0, '2017-02-14 12:43:20.234-0800', 'Raining too hard should have postponed', 'Alex');
INSERT INTO cycling.comments (record_id, id, created_at, comment, commenter) values (now(), e7ae5cf3-d358-4d99-b900-85902fda9bb0, '2017-03-21 13:11:09.999-0800', 'Second rest stop was out of water', 'Alex');
INSERT INTO cycling.comments (record_id, id, created_at, comment, commenter) values (now(), e7ae5cf3-d358-4d99-b900-85902fda9bb0, '2017-04-01 06:33:02.16-0800', 'LATE RIDERS SHOULD NOT DELAY THE START', 'Alex');
```

3. Run the simulator:

```
$ nodetool nodesyncservice ratesimulator recommended
```
Computed rate: 16B/s.

As expected, the computed rate is rather small because very little data was inserted.

4. Run the simulator with the verbose flag to view insights on why that rate was calculated:

```bash
$ nodetool nodesyncservice ratesimulator recommended -v
```

Using parameters:
- Size growing factor: 1.00
- Deadline safety factor: 0.25
- Rate safety factor: 0.10

cycling.comments:
- Deadline target=7.5d, adjusted from 10d for safety.
- Size=1.1MB to validate (2.3MB total (adjusted from 1.1MB for future growth) but RF=2).
- Added to previous tables, 1.1MB to validate in 7.5d => 2B/s
  => New minimum rate: 2B/s

cycling.comments2:
- Deadline target=7.5d, adjusted from 10d for safety.
- Size=7.1MB to validate (14MB total (adjusted from 7.1MB for future growth) but RF=2).
- Added to previous tables, 8.3MB to validate in 7.5d => 14B/s
  => New minimum rate: 14B/s

Computed rate: 16B/s, adjusted from 14B/s for safety.

As expected, the computed rate is rather small because very little data was inserted.

**nodetool nodesyncservice setrate**

Temporarily sets the maximum data validation rate.

**Tip:** To persist the rate limit, use the `rate_in_kb` (page 79) setting in `cassandra.yaml`.

**Tip:** Use the `nodetool nodesyncservice ratesimulator` (page 1022) to review how the change may impact performance. For more details, see Setting the rate (page 171).

**Synopsis**

```bash
$ nodetool [connection_options] nodesyncservice setrate
```

**Table 241: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th><strong>Syntax conventions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable value. Replace with a valid option or user-defined value.</td>
<td><strong>Italics</strong></td>
</tr>
<tr>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
<td>[ ]</td>
</tr>
<tr>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
<td>( )</td>
</tr>
<tr>
<td>Or. A vertical bar (</td>
<td>) separates alternative elements. Type any one of the elements. Do not type the vertical bar.</td>
</tr>
<tr>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
<td>...</td>
</tr>
<tr>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
<td>'Literal string'</td>
</tr>
<tr>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
<td>{ key:value }</td>
</tr>
<tr>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
<td>&lt;datatype1,datatype2&gt;</td>
</tr>
<tr>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
<td>cql_statement;</td>
</tr>
<tr>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
<td>[ -- ]</td>
</tr>
<tr>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
</tr>
<tr>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
<td>@xml_entity='xml_entity_type'</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  - The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
  - The JMX port number.
- **-pw, --password jmxpassword**
  - The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
**-pwf, --password-file jmx_password_filepath**
The filepath to the file that stores JMX authentication credentials.

**-u, --username jmx_username**
The user name for authenticating with secure JMX.

**Command arguments**

This command takes no arguments.

**Examples**

**Configure the validation rate of the local host**

```
$ nodetool nodesyncservice setrate 2048
```

**Verify the setting change**

```
$ nodetool nodesyncservice getrate
```

Current rate limit=2048 KB/s

**nodetool nodesyncservice status**

Returns the status of the NodeSync on the connected node.

**Synopsis**

```
$ nodetool [connection_options] nodesyncservice status [-b]
```

**Table 242: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Literal string'</td>
<td>Single quotation (‘’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (‘’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host** hostname
  - The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port** jmx_port
  - The JMX port number.

- **-pw, --password** jmxpassword
  - The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file** jmx_password_filepath
  - The filepath to the file that stores JMX authentication credentials.

- **-u, --username** jmx_username
  - The user name for authenticating with secure JMX.

### Command arguments

- **-b, --boolean-output**
  - Output NodeSync service status as true or false.
    - True when service is running and false otherwise. Output is useful for scripts.
Examples

Show NodeSync status on the local host using the boolean option

```
$ nodetool nodessyncservice status -b
false
```

Show NodeSync status on a remote host

```
$ nodetool -h northeast nodessyncservice status
```

The NodeSync service is running

**nodetool pausehandoff**

Pauses the hints delivery process.

**Synopsis**

```
$ nodetool [connection_options] pausehandoff
```

**Table 243: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  - The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
  - The JMX port number.
- **-pw, --password jmxpassword**
  - The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- **-pf, --password-file jmx_password_filepath**
  - The filepath to the file that stores JMX authentication credentials.
- **-u, --username jmx_username**
  - The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Pause hint delivery process
nodetool proxyhistograms

Provides a histogram of network operation statistics at the time of the command.

The output of this command shows the full request latency recorded by the coordinator. The output includes the percentile rank of read and write latency values for inter-node communication. Typically, you use the command to see if requests encounter a slow node.

Synopsis

$ nodetool [connection_options] proxyhistograms

Table 244: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td><em>Literal string</em></td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>'&lt; &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (') surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

This command takes no arguments.

**Examples**

**Get network statistics histogram**

This example shows the output from nodetool proxyhistograms after running 4,500 insert statements and 45,000 select statements on a three ccm node-cluster on a local computer.

```
$ nodetool proxyhistograms
```

<table>
<thead>
<tr>
<th>Latency</th>
<th>Percentile</th>
<th>Read Latency</th>
<th>Write Latency</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>446.00</td>
<td>50%</td>
<td>1502.50</td>
<td>375.00</td>
<td></td>
</tr>
<tr>
<td>498.00</td>
<td>75%</td>
<td>1714.75</td>
<td>420.00</td>
<td></td>
</tr>
<tr>
<td>800.20</td>
<td>95%</td>
<td>31210.25</td>
<td>507.00</td>
<td></td>
</tr>
</tbody>
</table>
Useful metrics in the output include:

- CAS Read Latency
- CAS Write Latency
- View Write Latency

CAS Read and Write Latency provides data for compare-and-set operations, while View Write Latency provides data for materialized view write operations.

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Range Latency (micros)</th>
<th>Read Latency</th>
<th>Write Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CAS Read Latency (micros)</td>
<td>CAS Write Latency (micros)</td>
<td>View Write Latency (micros)</td>
</tr>
<tr>
<td>50%</td>
<td>454.83</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>75%</td>
<td>1358.10</td>
<td>943.13</td>
<td>0.00</td>
</tr>
<tr>
<td>95%</td>
<td>3379.39</td>
<td>12108.97</td>
<td>0.00</td>
</tr>
<tr>
<td>98%</td>
<td>7007.51</td>
<td>155469.30</td>
<td>0.00</td>
</tr>
<tr>
<td>99%</td>
<td>8409.01</td>
<td>155469.30</td>
<td>0.00</td>
</tr>
<tr>
<td>Min</td>
<td>73.46</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Max</td>
<td>14530.76</td>
<td>155469.30</td>
<td>0.00</td>
</tr>
</tbody>
</table>

nodetool rangekeysample

Shows the sampled keys held across all keyspaces.

Synopsis

$ nodetool [connection_options] rangekeysample

Table 245: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
DataStax Enterprise tools

The filepath to the file that stores JMX authentication credentials.

- \( u, --username \ jmx\_username \)
The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Show sampled keys across all keyspaces


c ode

```bash
$ nodetool rangekeysample
```

RangeKeySample:
-5883607023773259416
-5883607023773259416
-5189327806405140569
-5189327806405140569
2008715943680221220
3066791452337107542
3066791452337107542
4479264904256751477
4479264904256751477
6874493789974003618
6874493789974003618
8718305215016653338
8718305215016653338
-79752896362648430
-79752896362648430
1139519215559584928
1139519215559584928
1178565181744072132
1178565181744072132
-5883607023773259416
-5883607023773259416
-5189327806405140569
-5189327806405140569
2008715943680221220
3066791452337107542
```

**nodetool rebuild**

Rebuilds data by streaming from other nodes.
This command operates on multiple nodes in a cluster and streams data only from a single source replica when rebuilding a token range. Use this command to add a new datacenter to an existing cluster.

**Note:** If `nodetool rebuild` is interrupted before completion, restart it by re-entering the command. The process resumes from the point at which it was interrupted.

**Synopsis**

```bash
$ nodetool [connection_options] rebuild
[-c num_connections] [-dc src_dc_names] [-ks keyspace_name]
[-m mode] [-s source_ip_address]
[-ts (start_token_1,end_token_1),(start_token_2,end_token_2), ...]
[-x exclude_source_IPs] [-xdc exclude_dc_names] [--] src_dc-name
```

**Table 246: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (() ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
**Syntax conventions**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'&lt; &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (’ ’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

- **--**
  Separates an option from an argument that could be mistaken for a option.

- **-c, --connections-per-host num_connections**
  Maximum number of connections per host for streaming. Overrides value of streaming_connections_per_host in cassandra.yaml.

- **-dc src_dc_names, --dcs src_dc_names**
  Comma-separated list of datacenters from which to stream. Datacenter names are case sensitive. For example, dc-a,dc-b. To include a rack name, separate datacenter and rack name with a colon (:). For example, dc-a:rack1,dc-a:rack2.

- **-ks, --keyspace keyspace_name, ...**
  Comma-separated list of one or more keyspaces. List only the keyspaces to include in the rebuild.
  
  **Tip:** Do not include keyspaces that are not replicated across datacenters (for example, dsefs keyspaces, and keyspaces with local strategy).

- **-m, --m mode**
  - normal - conventional behavior, streams only ranges that are not already locally available
  - refetch - resets locally available ranges, streams all ranges but leaves current data untouched
- reset - resets the locally available ranges, removes all locally present data (like a TRUNCATE), streams all ranges
- reset-no-snapshot - (like reset) resets the locally available ranges, removes all locally present data (like a TRUNCATE), streams all ranges but prevents a snapshot even if auto_snapshot is enabled

When not specified, the default is normal.

-s, --sources source_ip_address
Comma-separated list of IP addresses from which to stream.

src-dc-name
- datacenter - name of datacenter from which to select sources for streaming
- when not set - the default is to pick any datacenter

-ts, --tokens (start_token_1,end_token_1], (start_token_2,end_token_2], ...
Comma-separated list of token ranges, in this format
(start_token_1,end_token_1], (start_token_2,end_token_2],
(start_token_n,end_token_n]

-x, --exclude-sources exclude_source_IPs
Comma-separated list of IP addresses to exclude from streaming.

-xdc, --exclude-dcs exclude_dc_name
Comma-separated list of datacenters to exclude from streaming. For example, dc-a, dc-b. To include a rack name in the list, separate datacenter and rack name with a colon (:). For example, dc-a:rack1, dc-a:rack2.

Examples

Rebuild DC2

$ nodetool rebuild DC2

nodetool rebuild_index

Performs a full rebuild of native secondary indexes for a given table.

Synopsis

$ nodetool [connection_options] rebuild_index
[--] keyspace_name table index_name [index_name ...]

Table 247: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
Syntax conventions | Description
---|---
( ) | Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
| | Or. A vertical bar ( | ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
... | Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.
'Literal string' | Single quotation ( ‘ ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ key:value } | Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
<datatype1,datatype2> | Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
cql_statement; | End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ] | Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' <schema> ... </schema> ' | Search CQL only: Single quotation marks ( ‘ ’ ) surround an entire XML schema declaration.
@xml_entity='xml_entity_type' | Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

Definition
The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
-p, --port jmx_port
The JMX port number.
-pw, --password jmxpassword
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
-pwf, --password-file jmx_password_filepath
The filepath to the file that stores JMX authentication credentials.
-u, --username jmx_username
The user name for authenticating with secure JMX.
Command arguments

--
Separates an option from an argument that could be mistaken for a option.

index_name
One or more index names, separated by a space.

keyspace_name
The keyspace name.

table_name
The table name.

Examples

Rebuild indexes Standard3.IdxName and Standard3.IdxName1 on cycling keyspace and cyclist_name table

$ nodetool rebuild_index cycling cyclist_name Standard3.IdxName Standard3.IdxName1

nodetool rebuild_view

Performs a rebuild of the specified materialized views for a particular base table on the node on which the command is run. Use this command to rebuild materialized views after restoring sstables or after restarting a materialized view build that was previously stopped. If no materialized views are specified, all materialized views based on the specified table are rebuilt.

Note: The rebuild_view command does not clear existing data in the materialized view.

Synopsis

$ nodetool [connection_options] rebuild_view
    [--] keyspace_name table [materialized_view_name]
    [materializeds_view_name ...]

Table 248: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port
The JMX port number.

-pw, --password jmxpassword
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-pwf, --password-file jmx_password_filepath
The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username
The user name for authenticating with secure JMX.
Command arguments

```
--
Separates an option from an argument that could be mistaken for an option.
```

```
materialized_view_name
One or more materialized view names, separated by a space. If not specified, all
materialized views in the table are rebuilt.
```

```
keyspace_name
The keyspace name.
```

```
table_name
The table name.
```

Examples

```
Rebuild materialized views cyclist_by_age and cyclist_by_birthday_and_age on
cycling keyspace and cyclist_base table
```

```
$ Rebuild materialized views cyclist_by_age and
    cyclist_by_birthday_and_age on cycling keyspace and cyclist_base table
```

nodetool refresh

Loads newly placed SSTables onto the system without a restart.

Synopsis

```
$ nodetool [connection_options] refresh
[--reset-levels] [--] keyspace_name table_name
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (()) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
**Syntax conventions**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( `` ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( <code>{ </code>) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( <code>&lt; </code>) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( <code>; </code>) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( <code>-- </code>). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>'&lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks ( `` ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

- **--**
  Separates an option from an argument that could be mistaken for a option.

- **keyspace_name**
  The keyspace name.

- **--reset-levels**
  Force all sstables to level 0.
table_name

The table name.

Examples

Load new SSTables

$ nodetool refresh cycling comments

nodetool refreshsizeestimates

Refreshes system.size_estimates table. Use when huge amounts of data are inserted or truncated, which can result in incorrect size estimates.

Synopsis

$ nodetool [connection_options] refreshsizeestimates

Table 250: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

## Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  - The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
  - The JMX port number.
- **-pw, --password jmxpassword**
  - The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- **-pwf, --password-file jmx_password_filepath**
  - The filepath to the file that stores JMX authentication credentials.
- **-u, --username jmx_username**
  - The user name for authenticating with secure JMX.

### Command arguments

This command takes no arguments.

### Examples

**Refresh system.size_estimates table**

```bash
$ nodetool refreshsizeestimates
```

**nodetool reloadseeds**

Reloads the seed node list from the seed node provider.

### Synopsis

```bash
$ nodetool [connection_options] reloadseeds
```
Table 251: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>&lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type''</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
The JMX port number.

-\texttt{-pw, --password jmxpassword}
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-\texttt{-pwf, --password-file jmx\_password\_filepath}
  The filepath to the file that stores JMX authentication credentials.

-\texttt{-u, --username jmx\_username}
  The user name for authenticating with secure JMX.

\textbf{Command arguments}

This command takes no arguments.

\textbf{Examples}

\textbf{Reload seed node list from seed node provider}

\begin{verbatim}
\texttt{nodetool reloadseeds}
\end{verbatim}

Updated seed node IP list excluding the current node IP: /10.100.15.1

\begin{verbatim}
\texttt{nodetool reloadtriggers}
\end{verbatim}

Reloads trigger classes.

\textbf{Synopsis}

\begin{verbatim}
\texttt{nodetool [connection\_options] reloadtriggers}
\end{verbatim}

\begin{table}[h]
\centering
\caption{Legend}
\begin{tabular}{|l|l|}
\hline
\textbf{Syntax conventions} & \textbf{Description} \\
\hline
UPPERCASE & Literal keyword. \\
Lowercase & Not literal. \\
\textit{Italics} & Variable value. Replace with a valid option or user-defined value. \\
\{ \} & Optional. Square brackets ( \{ \} ) surround optional command arguments. Do not type the square brackets. \\
\{ \} & Group. Parentheses ( \{ \} ) identify a group to choose from. Do not type the parentheses. \\
\textbar & Or. A vertical bar ( \textbar{} ) separates alternative elements. Type any one of the elements. Do not type the vertical bar. \\
\ldots & Repeatable. An ellipsis ( \ldots{} ) indicates that you can repeat the syntax element as often as required. \\
\hline
\end{tabular}
\end{table}
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Literal string'</td>
<td>Single quotation (’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**  
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**  
  The JMX port number.

- **-pw, --password jmxpassword**  
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**  
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**  
  The user name for authenticating with secure JMX.

### Command arguments

This command takes no arguments.

### Examples

**Reload trigger classes**
DataStax Enterprise tools

$ nodetool reloadtriggers

**nodetool relocatesstables**

Rewrites SSTables to the correct disk.

Use with JBOD disk storage to manually rewrite the location of SSTables on disk. Useful if you have changed the replication factor for the cluster or if you added a new disk.

**Synopsis**

```
$ nodetool [connection_options] relocatesstables
    [-j num_jobs] [--] keyspace_name table_name
```

**Table 253: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks (`) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  
  The JMX port number.

- **-pw, --password jmxpassword**
  
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  
  The user name for authenticating with secure JMX.

### Command arguments

- **--**
  
  Separates an option from an argument that could be mistaken for a option.

- **-j, --jobs num_jobs**
  
  - num_jobs - Number of SSTables affected simultaneously.
  - 0 - Use all available compaction threads.

### Examples

**Relocate SSTables after adding a new disk**

```bash
$ nodetool relocatesstables cycling birthday_list
```
If the SSTables are on the correct disk already, no action is taken.

**nodetool removenode**

Shows the status of current node removal; forces completion of pending removal, or removes identified node.

Use when the node is down and nodetool decommission cannot be used. If the cluster does not use vnodes, adjust the tokens before running this command.

**Warning:** Run this command only on nodes that are down. This command triggers cluster streaming. In large environments, the additional streaming activity causes more pending gossip tasks in the output of nodetool tpstats. Nodes can start to appear offline and might need to be restarted to clear up the backlog of pending gossip tasks.

### Synopsis

```
$ nodetool [connection_options] removenode
[--] status | force | host_ID
```

### Table 254: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>&quot;Literal string&quot;</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
</tbody>
</table>
### Syntax conventions

| cql_statement; | End CQL statement. A semicolon ( ; ) terminates all CQL statements. |
| [ -- ] | Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options. |
| ' <schema> ... </schema> ' | Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration. |
| @xml_entity='xml_entity_type' | Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files. |

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

- **force**
  Force completion of pending removal.

- **ID**
  Remove provided ID.

- **status**
  Show status of current node removal.

### Examples

#### Determine UUID of the node to remove

```bash
$ nodetool removenode status

Datacenter: DC1
            ===========
```
DataStax Enterprise tools

<table>
<thead>
<tr>
<th>Status=Up/Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ State=Normal/Leaving/Joining/Moving</td>
</tr>
<tr>
<td>-- Address Load Tokens Owns (effective)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host ID</th>
<th>Address</th>
<th>Load</th>
<th>Tokens</th>
<th>Owns (effective)</th>
<th>Rack</th>
</tr>
</thead>
<tbody>
<tr>
<td>UN 192.168.2.101</td>
<td>112.82 KB</td>
<td>256</td>
<td>31.7%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>420129fc-0d84-42b0-be41-ef7dd3a8ad06</td>
<td>RAC1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DN 192.168.2.103</td>
<td>91.11 KB</td>
<td>256</td>
<td>33.9%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d0844a21-3698-4883-ab66-9e2fd5150edd</td>
<td>RAC1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN 192.168.2.102</td>
<td>124.42 KB</td>
<td>256</td>
<td>32.6%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8d5ed9f4-7764-4dbd-bad8-43fddce94b7c</td>
<td>RAC1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Remove down node with UUID

$ nodetool removenode d0844a21-3698-4883-ab66-9e2fd5150edd

View status of operation to remove node

$ nodetool removenode status

RemovalStatus: No token removals in process.

Confirm node has been removed

$ nodetool status

The removed node no longer shows in the Host ID column.

<table>
<thead>
<tr>
<th>Status=Up/Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ State=Normal/Leaving/Joining/Moving</td>
</tr>
<tr>
<td>-- Address Load Tokens Owns (effective)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host ID</th>
<th>Address</th>
<th>Load</th>
<th>Tokens</th>
<th>Owns (effective)</th>
<th>Rack</th>
</tr>
</thead>
<tbody>
<tr>
<td>UN 192.168.2.101</td>
<td>112.82 KB</td>
<td>256</td>
<td>37.7%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>420129fc-0d84-42b0-be41-ef7dd3a8ad06</td>
<td>RAC1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN 192.168.2.102</td>
<td>124.42 KB</td>
<td>256</td>
<td>38.3%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8d5ed9f4-7764-4dbd-bad8-43fddce94b7c</td>
<td>RAC1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

nodetool repair

Repairs tables on one or more nodes in a cluster when all involved replicas are up and accessible.

Important: Tables with NodeSync enabled will be skipped for repair operations run against all or specific keyspaces. For individual tables, running the repair command will be rejected when NodeSync is enabled.
See Repairing nodes. Before using this command, be sure to have an understanding of how node repair works.

**Important:** If repair encounters a down replica, an error occurs and the repair process halts. Re-run repair after bringing all replicas back online.

OpsCenter provides a repair option in the Nodes UI for Running a manual repair.

**Synopsis**

```
$ nodetool [connection_options] repair
    [-dcpar | -seq]
    [-full | -inc]
    [-hosts ip_address [ip_address ...]]
    [-local | -dc datacenter_name[,datacenter_name,...]]
    [-pl] [-pr] [-prv]
    [-pull -hosts local_ip_address [remote_ip_address]
    [-j job_threads]
    [-st start_token -et end_token]
    [-tr] [--]
    [keyspace_name table_name [table_name ...]]
```

**Table 255: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Literal keyword.</td>
</tr>
<tr>
<td><strong>Lowercase</strong></td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  - The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  - The JMX port number.

- **-pw, --password jmxpassword**
  - The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  - The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  - The user name for authenticating with secure JMX.

### Command arguments

`--`
- Separates an option from an argument that could be mistaken for a option.

- **-dc datacenter_name, --in-dc datacenter_name**
  - Comma-separated list of datacenters to limit repairs to. Datacenter names are case sensitive. Decreases network traffic while repairing more nodes than the local option. When this option is not specified, repair is run cluster-wide on all nodes that contain replicas.

- **-dcpar, --dc-parallel**
  - Run repairs on all nodes with the same replica data at the same time, recommended for repairs across datacenters. A single node in each datacenter runs repair, one after another until the repair is complete. This option combines sequential and parallel repair by simultaneously running a sequential repair in all datacenters. Use with the -local option only when the datacenter nodes have all the data for all ranges.

- **-et, --end-token end_token**
  - The token at which the range ends. Requires start token (-st).
-force, --force
  Filter out down endpoints.

-full, --full
  Issue a full repair.

-hosts, --in-hosts host_name
  Repair specific hosts.

-inc, --inc
  Issue an incremental repair.

-j, --job-threads num_threads
  Number of threads to run repair jobs. Usually this means number of tables to repair concurrently. Default: 1. Max: 4.
  **Warning:** Increasing job threads puts more load on repairing nodes.

keyspace_name
  The keyspace name.

-local, --in-local-dc
  Repair only against nodes in the same datacenter.

-pl, --pull
  Runs a one-way repair directly from another node that has a replica in the same token range. This option minimizes performance impact when cross-datacenter repairs are required.

-pr, --partitioner-range
  Repair only the first range returned by the partitioner.

-prv, --preview
  Determine ranges and amount of data to be streamed, but doesn't perform repair.

-seq, --sequential
  Perform sequential repair.

-st, --start-token start_token
  The token at which the range starts. Requires end token (-et).

table_name
  One or more table names, separated by a space.

-tr, --trace
  Trace the repair. Traces are logged to system_traces.events.

-vd, --validate
  Checks that repaired data is in sync between nodes.

  **Note:** Out of sync repaired data indicates a full repair should be run.

Examples

All nodetool repair command options are optional. When optional command arguments are not specified, the defaults are:

- Full repair runs on all keyspaces and all tables.
- Repair runs in parallel on all nodes with the same replica data at the same time.
- The number of job threads is 1.
- No tracing. No validation.
Sequential repair of all keyspaces

$ nodetool repair -seq

Partitioner range repair

$ nodetool repair -pr

Start-point-to-end-point repair of all nodes between two nodes on the ring

$ nodetool repair -st -9223372036854775808 -et -3074457345618258603

Restrict repair to local datacenter

$ nodetool repair -dc DC1

Results in output:


The system.log shows repair runs only on IP addresses in DC1.

RepairSession.java:171
- [repair #16499ef0-1381-11e4-88e3-c972e09793ca]
Received merkle tree for sessions from /192.168.2.101
RepairJob.java:145
- [repair #16499ef0-1381-11e4-88e3-c972e09793ca]
requesting merkle trees for events (to [/192.168.2.103, /192.168.2.101])
nodetool replaybatchlog

Forces batchlog replay and blocks until batches have been replayed.

Synopsis

$ nodetool [connection_options] replaybatchlog

Table 256: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

Definition
The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments
This command takes no arguments.

Examples

**Force batchlog replay**

```
$ nodetool replaybatchlog
```

**nodetool resetlocalschema**

Fixes schema disagreements between nodes by dropping the schema information of the local node and resynchronizing the schema from another node. When schema information on the local node is dropped, the system schema tables are truncated. The node temporarily loses metadata about the tables on the node, but rewrites the information from another node.

Useful when:
- Table schema changes have generated too many tombstones (100,000s).
- One node is out of sync with the cluster.

Synopsis

```
$ nodetool [connection_options] resetlocalschema
```
### Table 257: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td><code>'Literal string'</code></td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  - The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
The JMX port number.

-**pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-**pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

-**u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

This command takes no arguments.

**nodetool resume**

**Synopsis**

Restart a node's bootstrap process.

```bash
$ nodetool [options] bootstrap resume
```

**Tarball path:**

`installation_location/resources/cassandra/bin`

**Table 258: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address.</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number.</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path.</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password.</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>Remote JMX agent username.</td>
</tr>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:**

- For tarball installations, execute the command from the `installation_location/bin` directory.
- If a username and password for RMI authentication are set explicitly in the `cassandra-env.sh` file for the host, then you must specify credentials.
- `nodetool bootstrap` operates on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command...
is the intended target, you do not need the -h option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using -h.

Description

The `nodetool bootstrap resume` command restarts bootstrap streaming.

Examples

```bash
$ nodetool -u username -pw password bootstrap resume
```

**nodetool resumehandoff**

Resumes hints delivery process.

Synopsis

```bash
$ nodetool [connection_options] resumehandoff
```

**Table 259: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity=&quot;xml_entity_type&quot;</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Resume hints delivery process

```
$ nodetool resumehandoff
```

**nodetool ring**

Provides node status and information about the ring as determined by the node being queried. This information provides an idea of the load balance and if any nodes are down. If the cluster is not properly configured, different nodes may show a different ring. Check that the node appears the same way in the ring. If you use virtual nodes (vnodes), use `nodetool status (page 1104)` for succinct output.

- Address
The node’s URL.

- **DC (datacenter)**
  The datacenter containing the node.

- **Rack**
  The rack or, in the case of Amazon EC2, the availability zone of the node.

- **Status - Up or Down**
  Indicates whether the node is functioning or not.

- **State - N (normal), L (leaving), J (joining), M (moving)**
  The state of the node in relation to the cluster.

- **Load - updates every 90 seconds**
  The amount of file system data under the cassandra data directory after excluding all
  content in the snapshots subdirectories. Because all SSTable data files are included,
  any data that is not cleaned up, such as TTL-expired cell or tombstoned data) is
  counted.

- **Token**
  The end of the token range up to and including the value listed. For an explanation of
  token ranges, see Data Distribution in the Ring.

- **Owns**
  The percentage of the data owned by the node per datacenter times the replication
  factor. For example, a node can own 33% of the ring, but show100% if the replication
  factor is 3.

- **Host ID**
  The network ID of the node.

**Synopsis**

```
$ nodetool [connection_options] ring  
[-r] [--] [keyspace]
```

**Table 260: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

- **-h**, **--host hostname**  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p**, **--port jmx_port**  
The JMX port number.
- **-pw, --password jmxpassword**  
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- **-pwf, --password-file jmx_password_filepath**  
The filepath to the file that stores JMX authentication credentials.
- **-u, --username jmx_username**
The user name for authenticating with secure JMX.

--
Separates an option from an argument that could be mistaken for a option.

**keyspace_name**
The keyspace name.

**-r, --resolve-ip**
Node domain names instead of IPs.

**nodetool scrub**

Creates a snapshot and then rebuilds SSTables on a node. If possible use nodetool **upgradesstables** *page 1140* instead of scrub.

Scrub automatically discards broken data and removes any tombstoned rows that have exceeded the grace period of the table. If partition key values do not match the column data type, the partition is considered corrupt and the process automatically stops.

**Attention:** For LeveledCompactionStrategy (LCS), resets all SSTables back to Level 0 and requires recompaction of all SSTables.

**Synopsis**

```
$ nodetool [connection_options] scrub
[-j num_jobs] [-n] [-ns] [-r] [-s]
[-] [keyspace_name table_name [table_name ...]]
```

**Table 261: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>’Literal string’</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax conventions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map collection. Braces ({} ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
<td>{key:value}</td>
</tr>
<tr>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
<td>&lt;datatype1,datatype2&gt;</td>
</tr>
<tr>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
<td>cql_statement;</td>
</tr>
<tr>
<td>Separate the command line options from the command arguments with two hyphens (-- ). This syntax is useful when arguments might be mistaken for command line options.</td>
<td>[ -- ]</td>
</tr>
<tr>
<td>Search CQL only: Single quotation marks (’ ) surround an entire XML schema declaration.</td>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
</tr>
<tr>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
<td>@xml_entity='xml_entity_type'</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

- **--**
  Separates an option from an argument that could be mistaken for a option.

- **-j, --jobs num_jobs**
  - num_jobs - Number of SSTables affected simultaneously.
  - 0 - Use all available compaction threads.

- **keyspace_name**
  The keyspace name.

- **-n, --no-validate**
Do not validate columns using column validator.

-ns, --no-snapshot
If disablesnapshot is false, scrubbed tables are snapshotted first. Default: false.

-r, --reinsert-overflowed-ttl
Rewrite rows with overflowed expiration date affected by CASSANDRA-14092 with the maximum supported expiration date of 2038-01-19T03:14:06+00:00. The rows are rewritten with the original timestamp incremented by one millisecond to override/supersede any potential tombstone that may have been generated during compaction of the affected rows.

-s, --skip-corrupted
Skip corrupted partitions even when scrubbing counter tables. Default is false.

table_name
One or more table names, separated by a space.

nodetool sequence

Sequentially run multiple nodetool commands from a file, resource, or standard input (StdIn) to reduce overhead. Faster than running nodetool commands individually from a shell script because the JVM doesn’t have to restart for each command.

Synopsis

```
$ nodetool [connection_options] sequence
[--failonerror] [-i input [input ...]] [--stoponerror] [--]
[command_name [command_name : ...]]
```

Table 262: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ({[ ]}) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (( )) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>’Literal string’</td>
<td>Single quotation (‘’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (<code>&lt; &gt;</code>) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (<code>;</code>) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens (<code>--</code>). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>'&lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks (<code>'</code>) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  - The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  - The JMX port number.

- **-pw, --password jmxpassword**
  - The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  - The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  - The user name for authenticating with secure JMX.

### Command arguments

- **--**
  - Separates an option from an argument that could be mistaken for a option.

- **command_name**
  - Commands to execute. Separate individual commands using a colon surrounded by whitespaces (`:`).

- **--failonerror**
  - Set this option to true to return an error exit code if a child command fails. By default, an error exit code is not returned if one or more child commands fail.

- **-i, --input input**
  - The input to run the command.

- **--stoponerror**
Set to true to stop command on error. Default is if one child command fails, the sequence command continues with remaining commands.

Examples

To run commands in a file

First, create a text file with one command per line.

```
$ nodetool sequence -i /my/file/commands
```

To run commands on the command line

```
$ nodetool sequence info : gettimeout read : gettimeout write : status
```

Each command in the file runs sequentially.

```
# Executing 4 commands:
# info
# gettimeout read
# gettimeout write
# status

# Network interface lo (lo): /0:0:0:0:0:0:0:1%lo/128 [null], /127.0.0.1/8 [null]

# Command: info
# Timestamp: August 31, 2018 8:24:46 PM UTC
# Timestamp (local): August 31, 2018 8:24:46 PM UTC
# Timestamp (millis since epoch): 1535747086687

ID : 3b8e8192-c1d3-4b01-a792-9673b4e377c1
Gossip active : true
Native Transport active: true
Load : 625.97 KiB
Generation No : 1532896921
Uptime (seconds) : 2850186
Heap Memory (MB) : 1903.08 / 4012.00
Off Heap Memory (MB) : 0.01
Data Center : SearchGraphAnalytics
Rack : rack1
Exceptions : 0
Key Cache : entries 0, size 0 bytes, capacity 100 MiB, 0 hits, 0 requests, NaN recent hit rate, 14400 save period in seconds
Row Cache : entries 0, size 0 bytes, capacity 0 bytes, 0 hits, 0 requests, NaN recent hit rate, 0 save period in seconds
Counter Cache : entries 0, size 0 bytes, capacity 50 MiB, 1 hits, 2 requests, 0.500 recent hit rate, 7200 save period in seconds
```
Chunk Cache: entries 15972, size 595.42 MiB, capacity 2.79 GiB, 15972 misses, 25462774 requests, 0.999 recent hit rate, 606.208 microseconds miss latency
Percent Repaired: 0.0%
Token: 8242717283351148695

# Command 'info' completed successfully in 331 ms

# Command: gettimeout read
# Timestamp: August 31, 2018 8:24:47 PM UTC
# Timestamp (local): August 31, 2018 8:24:47 PM UTC
# Timestamp (millis since epoch): 1535747087024
Current timeout for type read: 5000 ms
# Command 'gettimeout read' completed successfully in 0 ms

# Command: gettimeout write
# Timestamp: August 31, 2018 8:24:47 PM UTC
# Timestamp (local): August 31, 2018 8:24:47 PM UTC
# Timestamp (millis since epoch): 1535747087025
Current timeout for type write: 2000 ms
# Command 'gettimeout write' completed successfully in 0 ms

# Command: status
# Timestamp: August 31, 2018 8:24:47 PM UTC
# Timestamp (local): August 31, 2018 8:24:47 PM UTC
# Timestamp (millis since epoch): 1535747087026
Datacenter: SearchGraphAnalytics

---
Status=Up/Down
| State=Normal/Leaving/Joining/Moving
-- Address Load Owns Host ID
Token
UN 127.0.0.1  625.97 KiB  ? 3b8e8192-c1d3-4b01-a792-9673b4e377c1
   8242717283351148695
rack1

Note: Non-system keyspaces don't have the same replication settings, effective ownership information is meaningless
# Command 'status' completed successfully in 29 ms
# Total duration: 374ms
# Out of 4 commands, 4 completed successfully, 0 failed.

----------
tool setbatchlogreplaythrottle

Sets batchlog replay throttle in KB per second, or 0 to disable throttling. This will be reduced proportionally to the number of nodes in the cluster.

Synopsis

$ nodetool [connection_options] setbatchlogreplaythrottle

---

### Table 263: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (()) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td><code>Literal string</code></td>
<td>Single quotation (’’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code> &lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks (‘’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

**-h, --host hostname**

The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
DataStax Enterprise tools

-p, --port jmx_port
   The JMX port number.

-pw, --password jmxpassword
   The JMX password for authenticating with secure JMX. If a password is not
   provided, you are prompted to enter one.

-pwf, --password-file jmx_password_filepath
   The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username
   The user name for authenticating with secure JMX.

Command arguments

--
   Separates an option from an argument that could be mistaken for a option.

value_in_kb_per_sec
   • value - the number of milliseconds that the database generates hints for an
     unresponsive node
   • 0 - disables throttling

Examples

Set batchlog replay throttle at 60 KB per second

$ nodetool setbatchlogreplaythrottle 60

Disable batchlog replay throttle

$ nodetool setbatchlogreplaythrottle 0

nodetool setcachecapacity

Sets global key and row cache capacities in megabytes.

   Tip: Overrides the configured value of the row_cache_size_in_mb (page 83)
   parameter in cassandra.yaml (page 83).

Synopsis

$ nodetool [connection_options] setcachecapacity
   [---] key-cache-capacity row-cache-capacity counter-cache-capacity
Table 264: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
DataStax Enterprise tools

The JMX port number.

-\texttt{pw, --password jmxpassword}
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-\texttt{pwf, --password-file jmx_password_filepath}
  The filepath to the file that stores JMX authentication credentials.

-\texttt{u, --username jmx_username}
  The user name for authenticating with secure JMX.

Command arguments

counter-cache-capacity
  Corresponds to the \texttt{counter_cache_size_in_mb} (page 84) in cassandra.yaml. By default, the database uses the smaller of minimum of 2.5\% of heap or 50 MB.
  
  • number - the number of keys saved by each cache
  • 0 - disable counter caching

key-cache-capacity
  Key cache capacity in MB units.

row-cache-capacity
  Row cache capacity in MB units, corresponds to the \texttt{row_cache_size_in_mb} (page 83) parameter in cassandra.yaml. By default, row caching is zero (disabled).

\texttt{nodedtool setcachekeystosave}

Sets the global number of keys saved by each cache for faster post-restart warmup.

\textbf{Tip:} Overrides the configured value of the \texttt{row_cache_keys_to_save} (page 83) parameter in cassandra.yaml.

Synopsis

$ \texttt{nodedtool [connection_options] setcachekeystosave key-cache-keys-to-save row-cache-keys-to-save counter-cache-keys-to-save}$

\textbf{Table 265: Legend}

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>\textit{Italics}</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  
  The JMX port number.

- **-pw, --password jmxpassword**
  
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  
  The user name for authenticating with secure JMX.

### Command arguments

- **counter-cache-keys-to-save**
Counter cache capacity in MB units.

**key-cache-keys-to-save**
Corresponds to the `key_cache_keys_to_save (deprecated)` parameter in cassandra.yaml. Key cache limiting is disabled by default, meaning all keys will be saved.
- number - the number of keys saved by each cache
- 0 - disable counter caching

**row-cache-keys-to-save**
Corresponds to the `row_cache_keys_to_save` parameter in cassandra.yaml.
- number - the number of keys saved by each cache
- 0 - disable counter caching

The row-cache-keys-to-save argument, which is disabled by default.

**nodetool setcompactionthreshold**

Sets minimum and maximum compaction thresholds for a table.

SSTables are compacted concurrently to avoid wasting memory or running out of memory when compacting highly overlapping SSTables.

**Tip:** The `max_threshold` table property sets an upper bound on the number of SSTables that may be compacted in a single minor compaction, as described in *How is data updated?*.

**Synopsis**

```
$ nodetool [connection_options] setcompactionthreshold
[--] keyspace_name table_name minthreshold maxthreshold
```

**Table 266: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

- **--**
  Separates an option from an argument that could be mistaken for a option.

- **keyspace_name**
  The keyspace name.
DataStax Enterprise tools

**maxthreshold**
Resets or overrides internal setting of 32. How many SSTables of a similar size must be present before a minor compaction is scheduled.

**minthreshold**
Minimum threshold.

**table_name**
The table name.

**Examples**

**Set minimum compaction throttling**

```
$ nodetool setcompactionthreshold cycling comments 6 28
```

**nodetool setcompactionthroughput**

Sets the throughput capacity for compaction in the system, or disables throttling. Overwrites the compaction_throughput_mb_per_sec (page 70) setting in cassandra.yaml.

**Tip:** To view the current setting, use nodetool getcompactionthroughput (page 962).

**Synopsis**

```
$ nodetool <options> setcompactionthroughput -- <value_in_mb>
```

**Tarball path:**

*installation_location/resources/cassandra/bin*

**Table 267: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address.</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number.</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path.</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password.</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>Remote JMX agent username.</td>
</tr>
<tr>
<td>value_in_mb</td>
<td></td>
<td>The throughput capacity in megabytes (MB) per second for compaction. To disable throttling, set to 0.</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>
Description

Set value_in_mb to 0 to disable throttling.

**nodetool setconcurrentcompactors**

Sets number of concurrent compactors.

Synopsis

```
$ nodetool [connection_options] setconcurrentcompactors
[--] num_compactors
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- `-h, --host hostname`
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- `-p, --port jmx_port`
  The JMX port number.
- `-pw, --password jmxpassword`
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- `-pwf, --password-file jmx_password_filepath`
  The filepath to the file that stores JMX authentication credentials.
- `-u, --username jmx_username`
  The user name for authenticating with secure JMX.

Command arguments

--
Separates an option from an argument that could be mistaken for a option.

`num_compactors`
Number of concurrent compactors.

Examples

**Set 2 concurrent compactors**

```
$ nodetool setconcurrentcompactors 2
```

**nodetool setconcurrentviewbuilders**

Sets the number of simultaneous materialized view builder tasks allowed to run concurrently. When a view is created, the node ranges are split into (`num_processors * 4`) builder tasks and submitted to this executor. Overrides the `concurrent_materialized_view_builders` ([page 76](#)) setting in `cassandra.yaml`.

Synopsis

```
$ nodetool [connection_options] setconcurrentviewbuilders number
```
### Table 269: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
The JMX port number.

- **pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

- **number**
  The number of concurrent materialized view builder tasks. Must be greater than 0.

### Examples

**Allow 6 concurrent materialized view builder tasks**

```
$ nodetool setconcurrentviewbuilders 6
```

### nodetool sethintedhandoffthrottlekb

Sets hinted handoff throttle in KB/sec per delivery thread.

When a node detects that a node for which it is holding hints has recovered, hints are sent to that node. This command sets the maximum sleep interval per delivery thread after delivering each hint. The interval shrinks proportionally to the number of nodes in the cluster. For example, if there are two nodes in the cluster, each delivery thread uses the maximum interval; if there are three nodes, each node throttles to half of the maximum interval, because the two nodes are expected to deliver hints simultaneously.

### Synopsis

```
$ nodetool [connection_options] sethintedhandoffthrottlekb
[---] value_in_KB/sec
```

### Table 270: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td><code>[ ]</code></td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td><code>( )</code></td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
  The JMX port number.
- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.
- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

--
Separates an option from an argument that could be mistaken for a option.

**value_in_kb_per_sec**

- value - the number of milliseconds that the database generates hints for an unresponsive node
- 0 - disables throttling

Examples

Set hinted handoff throttle at 64 KB/sec per delivery thread

$ nodetool sethintedhandoffthrottlekb 64

**nodetool setinterdcstreamthroughput**

Sets the inter-datacenter throughput capacity in megabits per second (Mbps) streaming.

*Note:* Since it is a subset of total throughput, **inter_dc_stream_throughput_outbound_megabits_per_sec (page 77)** should be set to a value less than or equal to **stream_throughput_outbound_megabits_per_sec (page 77)**.

Synopsis

$ nodetool [connection_options] setinterdcstreamthroughput [--] value_in_megabits

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{key:value}</td>
<td>Map collection. Braces ({}) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (‘) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

-h, --host hostname
   The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port
   The JMX port number.

-pw, --password jmxpassword
   The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-pwf, --password-file jmx_password_filepath
   The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username
   The user name for authenticating with secure JMX.

Command arguments

--
   Separates an option from an argument that could be mistaken for a option.

value_in_megabits
   - Mb - streaming throughput capacity in Mb per second
   - 0 - disables throttling
Examples

Set inter-datacenter throughput capacity at 64 megabits per second streaming

```bash
$ nodetool setinterdcstreamthroughput 64
```

**nodetool setlogginglevel**

Sets the log level threshold for a given component or class.

**Tip:** Use this command to set logging levels for services instead of modifying the logback-text.xml file.

**Synopsis**

```bash
$ nodetool [connection_options] setlogginglevel
[--] component | class level
```

**Table 272: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
**Syntax conventions**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (‘) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity=&quot;xml_entity_type&quot;'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

--

Separates an option from an argument that could be mistaken for a option.

**class**

The following values are valid for the log class qualifier:

- org.apache.cassandra
- org.apache.cassandra.db
- org.apache.cassandra.service.StorageProxy

**component**

The following values are valid for the log components qualifier:

- bootstrap
- compaction
- cql
- repair
- ring
streaming

level

If class qualifier and level arguments to the command are empty or null, logging levels are reset to the initial configuration.

The valid values for setting the log level include ALL for logging information at all levels, TRACE through ERROR, and OFF for no logging. TRACE creates the most verbose log, and ERROR, the least.

- ALL
- TRACE
- DEBUG
- INFO (Default)
- WARN
- ERROR
- OFF

Examples

Set StorageProxy service to debug level

```
$ nodetool setlogginglevel org.apache.cassandra.service.StorageProxy DEBUG
```

**Note:** Extended logging for compaction is supported and requires table configuration. The extended compaction logs are stored in a separate file.

nodetool setmaxhintwindow

Sets the maximum time that the database generates hints for an unresponsive node.

Synopsis

```
$ nodetool [connection_options] setmaxhintwindow [--] value_in_ms
```

Table 273: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th><strong>Syntax conventions</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**  
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**  
  The JMX port number.

- **-pw, --password jmxpassword**  
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**  
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**  
  The user name for authenticating with secure JMX.

### Command arguments

--
Separates an option from an argument that could be mistaken for a option.

**value_in_ms**
- value - milliseconds
- 0 - disables throttling

Examples

**Set time that database generates hints for an unresponsive node to 120 milliseconds**

```bash
$ nodetool setmaxhintwindow 120
```

**nodetool setstreamthroughput**

Sets the throughput capacity in megabits per second (Mb/s) for outbound streaming in the system. Overwrites the `stream_throughput_outbound_megabits_per_sec (page 77)` setting in `cassandra.yaml`.

**Note:**

If `inter_dc_stream_throughput_outbound_megabits_per_sec (page 77)` is set, since it is a subset of total throughput, its value should be less than or equal to `stream_throughput_outbound_megabits_per_sec (page 77)`.

**Tip:** To view the current setting, use `nodetool getstreamthroughput (page 980)`.

**Synopsis**

```bash
$ nodetool [connection_options] setstreamthroughput
[--] value_in_megabits
```

**Table 274: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

- **--**
  Separates an option from an argument that could be mistaken for a option.

- **value_in_megabits**
• Mb - streaming throughput capacity in Mb per second
• 0 - disables throttling

Examples

Set throughput capacity at 64 megabits per second for outbound streaming

$ nodetool setstreamthroughput 64

nodetool settimeout

Temporarily sets the timeout for the given timeout type by overriding the corresponding setting in cassandra.yaml:

• read - read_request_timeout_in_ms (page 85)
• range - range_request_timeout_in_ms (page 85)
• write - write_request_timeout_in_ms (page 85)
• counterwrite - counter_write_request_timeout_in_ms (page 85)
• cascontention - cas_contention_timeout_in_ms (page 85)
• truncate - truncate_request_timeout_in_ms (page 85)
• misc, such as general rpc_timeout_in_ms

To persist the setting, change the cassandra.yaml setting.

To discover the current timeouts, use nodetool gettimeout (page 982).

Synopsis

$ nodetool [connection_options] settimeout
[---] timeout_type timeout_in_ms

Table 275: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

-h, --host hostname  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port  
The JMX port number.

-pw, --password jmxpassword  
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

-pwf, --password-file jmx_password_filepath  
The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username  
The user name for authenticating with secure JMX.

**Command arguments**

--  
Separates an option from an argument that could be mistaken for a option.

timeout_in_ms  
Time to wait in milliseconds.
DataStax Enterprise tools

0 - Disables socket streaming timeout.

**timeout_type**

The timeout type: read, range, write, counterwrite, cascontention, truncate, streamingsocket, or misc (general rpc_timeout_in_ms).

Examples

**Set write timeout for 15 ms**

```
$ nodetool settimeout write 15
```

**Disable truncate timeout**

```
$ nodetool settimeout truncate 0
```

**nodetool settraceprobability**

Sets the probability for tracing any given request to value.

Probabilistic tracing identifies which queries are responsible for intermittent query performance problems. You can trace some or all statements sent to a cluster. Tracing a request usually requires at least 10 rows to be inserted.

A probability of 1.0 traces everything whereas lesser amounts (for example, 0.10) only sample a certain percentage of statements. Take care on large and active systems, as system-wide tracing will have a performance impact. Unless you are under a very light load, tracing all requests (probability 1.0) will probably overwhelm your system. Start with a small fraction, for example, 0.001 and increase only if necessary.

The trace information is stored in a system_traces keyspace that holds the sessions and events tables that can be easily queried to answer questions, such as what the most time-consuming query has been since a trace was started. Query the parameters map and thread column in the system_traces.sessions and system_traces.events tables for probabilistic tracing information.

**Tip:** To discover the current trace probability setting, use `nodetool gettraceprobability` (page 984).

**Synopsis**

```
$ nodetool [connection_options] settraceprobability
    [--] value
```

**Table 276: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th><strong>Lowercase</strong></th>
<th>Not literal.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

**-h, --host hostname**  
The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

**-p, --port jmx_port**  
The JMX port number.

**-pw, --password jmxpassword**  
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
-pwf, --password-file jmx_password_filepath
The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username
The user name for authenticating with secure JMX.

Command arguments

--
Separates an option from an argument that could be mistaken for a option.

value

- 0 - disables trace probability. Default.
- number between 0 and 1 - trace probability to represent a percentage.
- 1 - enables for all requests.

Examples

Set probability for tracing a request at 60%

$ nodetool settraceprobability 0.6

Enable tracing for all requests

$ nodetool settraceprobability 1

Disable request tracing

$ nodetool settraceprobability 0

nodetool sjk

Runs Swiss Java Knife (SJK) commands to execute, troubleshoot, and monitor the database.

Tip: See Using nodetool sjk.

Synopsis

$ nodetool [connection_options] sjk
[---] [args [args ...]]

Table 277: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
DataStax Enterprise tools

The filepath to the file that stores JMX authentication credentials.

-\texttt{u, --username jmx\_username}\hfill
The user name for authenticating with secure JMX.

Command arguments

--
\texttt{\textemdash}\hfill
Separates an option from an argument that could be mistaken for a option.

\texttt{args}\hfill
Arguments passed as is to ‘Swiss Army Knife’.

Examples

**Get status for the EndpointStateTracker MBean**

\begin{verbatim}
$ nodetool sjk mx -b com.datastax.bdp:type=core,name=EndpointStateTracker -f Blacklisted --get
\end{verbatim}

**Set status to true for the EndpointStateTracker MBean**

\begin{verbatim}
$ nodetool sjk mx -b com.datastax.bdp:type=core,name=EndpointStateTracker -f Blacklisted --set -v true
\end{verbatim}

**Get status of node1**

\begin{verbatim}
$ nodetool sjk mx -b com.datastax.bdp:type=core,name=EndpointStateTracker -mc -op getBlacklistedStatus -a node1
\end{verbatim}

\textbf{nodetool snapshot}

Creates a backup by taking a snapshot of table data. A snapshot is a hardlink to the SSTable files in the data directory for a schema table at the moment the snapshot is executed.

The snapshot directory path is: data/\texttt{keyspace\_name/table-UID/}
\texttt{snapshots/snapshot\_name}. Data is backed up into multiple .\texttt{db} files and table schema is saved to schema.cql. The schema.cql file captures the structure of the table at the time of snapshot because restoring the snapshot requires the table to have the same structure. See this DataStax Support knowledge base article Manual Backup and Restore, with Point-in-time and table-level restore.

\textbf{Warning:} Always run \texttt{nodetool cleanup (page 911)} before taking a snapshot for restore. Otherwise \textit{invalid} replicas, that is replicas that have been superseded by new, valid replicas on newly added nodes can get copied to the target when they should not. This results in old data showing up on the target.
**Note:** Before upgrading DataStax Enterprise, be sure to create a back up of all keyspaces. See taking a snapshot.

**Synopsis**

```
$ nodetool [connection_options] snapshot
[--table table_name | -kt keyspace_name.table_name,...]
[-sf] [-t snapshotname] [--]
[ keyspace_name [keyspace_name...] ]
```

**Table 278: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>‘&lt;schema&gt; ... &lt;/schema&gt;’</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

--
Separates an option from an argument that could be mistaken for a option.

- **--table, -cf, --column-family table_name**
  Table name in the specified keyspace.

- **-kt, --kt-list, -kc, --kc.list keyspace_name.table_name,...**
  Comma-separated list of keyspace_name.table_name with no spaces after the comma. For example, cycling.cyclist,basketball.players

- **-sf, --skip_flush**
  Do not flush tables before creating the snapshot.

  **Caution:** Snapshot will not contain unflushed data.

- **-t snapshotname, --tag snapshotname**
  The snapshot filepath. When not specified, the current time is used for the directory name. For example, 1489076973698.

Examples

Take snapshot of all keyspaces on the node

```
$ nodetool snapshot
```

A message displays with the name of the snapshot directory:

```
Requested creating snapshot(s) for [all keyspaces] with snapshot name [1489076973698] and options {skipFlush=false}
Snapshot directory: 1489076973698
```
Create snapshot of single keyspace in the cycling_2017-3-9 filepath

```
$ nodetool snapshot -t cycling_2017-3-9 cycling
```

The following output appears:

```
Requested creating snapshot(s) for [cycling] with snapshot name [2015.07.17]
Snapshot directory: cycling_2017-3-9
```

Take snapshot of single keyspace with two tables

The cycling keyspace contains two tables, cyclist_name and upcoming_calendar. The snapshot creates multiple snapshot directories named cycling_2017-3-9. A number of .db files containing the data are located in these directories, along with table schema. For example, from the DSE installation directory:

```
$ ls -l data/cycling/cyclist_name-9e516080f30811e689e40725f37c761d/
snapshots/cycling_2017-3-9
```

manifest.json
mc-1-big-CompressionInfo.db
mc-1-big-Data.db
mc-1-big-Digest.crc32
mc-1-big-Filter.db
mc-1-big-Index.db
mc-1-big-Statistics.db
mc-1-big-Summary.db
mc-1-big-TOC.txt
schema.cql

Take snapshot of multiple (mykeyspace and cycling) keyspaces

```
$ nodetool snapshot mykeyspace cycling
```

Requested creating snapshot(s) for [mykeyspace, cycling] with snapshot name [1391460334889]
Snapshot directory: 1391460334889

Take snapshot of single table

Take a snapshot of only the cyclist_name table in the cycling keyspace.

```
$ nodetool snapshot --table cyclist_name cycling
```

Requested creating snapshot(s) for [cycling] with snapshot name [1391461910600]
The resulting snapshot directory 1391461910600 contains data files and the schema of cyclist_name table in data/cycling/cyclist_name-a882dca02aaf11e58c7b8b496c707234/snapshots.

**Take snapshot of multiple tables in different keyspaces**

Take a snapshot the cyclist_name table in the cycling keyspace and the sample_times table in the test keyspace. For the -kt command argument, list tables in a comma-separated list with no spaces.

```
$ nodetool snapshot -kt cycling.cyclist_name,test.sample_times
```

Requested creating snapshot(s) for [cycling.cyclist_name,test.sample_times] with snapshot name [1431045288401]

Snapshot directory: 1431045288401

**nodetool status**

Provides information about the cluster, such as the state, load, and IDs.

A frequently used command, nodetool status provides the following information:

- **Status** - U (up) or D (down)
  
  Indicates whether the node is functioning or not.

- **State** - N (normal), L (leaving), J (joining), M (moving)
  
  The state of the node in relation to the cluster.

- **Address**
  
  The node's URL.

- **Load** - updates every 90 seconds
  
  The amount of file system data in the data directory, excluding all content in the snapshots subdirectories. Because all SSTable data files are included, any data that is not cleaned up (such as TTL-expired cell or tombstoned data) is counted.

- **Tokens**
  
  The number of tokens set for the node.

- **Owns**


The percentage of the data owned by the node per datacenter times the replication factor. For example, a node can own 33% of the ring, but shows 100% if the replication factor is 3.

- **Host ID**
  The network ID of the node.

- **Rack**
  The rack or, in the case of Amazon EC2, the availability zone of the node.

**Synopsis**

```
$ nodetool [connection_options] status
[-r] [--] [keyspace_name]
```

**Table 279: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key: value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt;&gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

**--**

Separates an option from an argument that could be mistaken for a option.

**keyspace_name**

The keyspace name.

- **-r, --resolve-ip**
  Node domain names instead of IPs.

Examples

**Get cluster status on all keyspaces**

```
$ nodetool status
```

`Datacenter: datacenter1
============================
Status=Up/Down
// State=Normal/Leaving/Joining/Moving`
Get cluster status on a single keyspace

$ nodetool status mykeyspace

Datacenter: datacenter1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving

<table>
<thead>
<tr>
<th>-- Address</th>
<th>Load</th>
<th>Tokens</th>
<th>Owns</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rack UN</td>
<td>127.0.0.1</td>
<td>47.66 KB</td>
<td>1</td>
<td>33.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>aaa1b7c1-6049-4a08-ad3e-3697a0e30e10 rack1</td>
</tr>
</tbody>
</table>

nodetool statusbackup

Provides status of incremental backup.

Synopsis

$ nodetool [connection_options] statusbackup

Table 280: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Literal string'</td>
<td>Single quotation (‘’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (‘’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

**Get status of incremental backup**

```
$ nodetool -u joe -pw P@ssw0rd! statusbackup
```
**nodetool statusbinary**

Provides the status of the native transport that defines the format of the binary message.

**Synopsis**

```
$ nodetool [connection_options] statusbinary
```

**Table 281: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

This command takes no arguments.

**Examples**

**Get status of native transport**

```
$ nodetool -u joe -pw P@ssw0rd! statusbinary

running
```

**nodetool statusgossip**

Provides status of gossip.

**Synopsis**

```
$ nodetool [connection_options] statusgossip
```

**Table 282: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
DataStax Enterprise tools

The user name for authenticating with secure JMX.

**Command arguments**

This command takes no arguments.

**Examples**

**Get status of gossip.**

```bash
$ nodetool -u joe -pw P@ssw0rd! statusgossip
```

**nodetool statushandoff**

Provides status of storing future hints.

**Synopsis**

```
$ nodetool [connection_options] statushandoff
```

**Table 283: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th><strong>Syntax conventions</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>&lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

This command takes no arguments.

### Examples

**Get status of storing future hints**

```
$ nodetool -u joe -pw P@ssw0rd! statushandoff
```
Hinted handoff is running

**nodetool stop**

Stops all compaction operations from continuing to run, typically run on a node where compaction has a negative impact on performance. After the compaction stops, the remaining operations in the queue are continued. Eventually, the compaction is restarted.

**Synopsis**

```
$ nodetool [connection_options] stop
   [-id compaction_id] [--] compaction_type
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (' ') marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
  The JMX port number.
- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.
- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

- **--**
  Separates an option from an argument that could be mistaken for a option.
- **-id, --compaction-id compaction_id**
  Stop a single compaction operation by the specified ID. Find IDs whose name starts with `compaction_` in transaction log files in the table transactions folder.

**compaction_type**

Supported compaction types:

- COMPACTION
- VALIDATION
- CLEANUP
- SCRUB
- VERIFY
- INDEX_BUILD

**nodetool stopdaemon**

Stops cassandra daemon.

**Synopsis**

```
$ nodetool [connection_options] stopdaemon
```
### Table 285: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

-h, --host hostname

The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

-p, --port jmx_port
The JMX port number.

`-pw, --password jmxpassword`
The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

`-pwf, --password-file jmx_password_filepath`
The filepath to the file that stores JMX authentication credentials.

`-u, --username jmx_username`
The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Stop cassandra daemon

```
$ nodetool -u joe -pw P@ssw0rd! stopdaemon
```

**nodetool tablehistograms**

Initial troubleshooting and performance metrics that provide current performance statics for read and write latency on a table during the past fifteen minutes.

Synopsis

```
nodetool options tablehistograms [ -- ] keyspace_name table_name
```

Tarball path:

```
installation_location/resources/cassandra/bin
```

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address.</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number.</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path.</td>
</tr>
<tr>
<td>-pw</td>
<td>--password-file</td>
<td>Password.</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>Remote JMX agent username.</td>
</tr>
</tbody>
</table>

`keyspace_name` Name of keyspace.

`table_name` Name of table.
### Description

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

### nodetool tablehistograms

shows table performance statistics over the past fifteen minutes, including read/write latency, partition size, cell count, and number of SSTables. Use this tool to analyze performance and tune individual tables and ensure that the percent latency level meets the SLA for the data stored in the table.

#### Example

For example, to get statistics for the DSE Search wiki demo **solr** table, use this command:

```bash
nodetool tablehistograms wiki solr
```

#### Output:

```
wiki/solr histograms
Percentile Size  SSTables Cell Count Write Latency (micros) Read Latency (micros) Partition Size (bytes)
50% 1.00 126.93 654.95
2759 3
75% 1.00 152.32 1358.10
5722 3
95% 1.00 785.94 5839.59
17084 3
98% 1.00 1629.72 12108.97
29521 3
99% 1.00 2346.80 12108.97
42510 3
Min 1.00 73.46 219.34
104 3
Max 1.00 2346.80 12108.97
219342 3
```

The output shows the percentile rank of read and write latency values, the partition size, and the cell count for the table.

### nodetool tablestats

Provides statistics about one or more tables. Statistics are updated after SSTables change through compaction or flushing.

**Tip:** DataStax Enterprise uses the `metrics-core` library to make the output more informative and easier to understand.
<table>
<thead>
<tr>
<th>Name of statistic</th>
<th>Example value</th>
<th>Brief description</th>
<th>Related information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyspace</td>
<td>libdata</td>
<td>Name of the <strong>keyspace</strong>.</td>
<td>Keyspace and table</td>
</tr>
<tr>
<td>Table</td>
<td>libout</td>
<td>Name of this <strong>table</strong>.</td>
<td></td>
</tr>
<tr>
<td>SSTable count</td>
<td>3</td>
<td>Number of <strong>SSTables</strong> containing data for this table.</td>
<td>Table statistics</td>
</tr>
<tr>
<td>Space used (live)</td>
<td>9592399</td>
<td>Total number of bytes of disk space used by all active SSTables belonging to this table.</td>
<td>Storing data on disk in SSTables</td>
</tr>
<tr>
<td>Space used (total)</td>
<td>9592399</td>
<td>Total number of bytes of disk space used by SSTables belonging to this table, including obsolete SSTables waiting for GC management.</td>
<td>Same as above.</td>
</tr>
<tr>
<td>Space used by snapshots (total)</td>
<td>0</td>
<td>Total number of bytes of disk space used by snapshot of this table's data.</td>
<td>About snapshots</td>
</tr>
<tr>
<td>Off heap memory used (total)</td>
<td></td>
<td>Total number of bytes of off heap memory used for memtables, Bloom filters, index summaries and compression metadata for this table.</td>
<td></td>
</tr>
<tr>
<td>SSTable Compression Ratio</td>
<td>0.367…</td>
<td>Ratio of size of compressed SSTable data to its uncompressed size.</td>
<td>Types of compression options.</td>
</tr>
<tr>
<td>Number of partitions (estimate)</td>
<td>3</td>
<td>The number of partition keys for this table.</td>
<td>Not the number of primary keys. This gives you the estimated number of partitions in the table.</td>
</tr>
<tr>
<td>Memtable cell count</td>
<td>1022550</td>
<td>Number of cells (storage engine rows x columns) of data in the <strong>memtable</strong> for this table.</td>
<td>How the database reads and writes data</td>
</tr>
<tr>
<td>Name of statistic</td>
<td>Example value</td>
<td>Brief description</td>
<td>Related information</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Memtable data size</td>
<td>32028148</td>
<td>Total number of bytes in the memtable for this table.</td>
<td>Total amount of live data stored in the memtable, excluding any data structure overhead.</td>
</tr>
<tr>
<td>Memtable off heap memory used</td>
<td>0</td>
<td>Total number of bytes of off-heap data for this memtable, including column related overhead and partitions overwritten.</td>
<td>The maximum amount is set in <code>cassandra.yaml</code> by the property <code>memtable_offheap_space_in_mb</code> (page 71).</td>
</tr>
<tr>
<td>Memtable switch count</td>
<td>3</td>
<td>Number of times a full memtable for this table was swapped for an empty one.</td>
<td>Increases each time the memtable for a table is flushed to disk. See How memtables are measured article.</td>
</tr>
<tr>
<td>Local read count</td>
<td>11207</td>
<td>Number of requests to read tables in the keyspace since startup.</td>
<td></td>
</tr>
<tr>
<td>Local read latency</td>
<td>0.048 ms</td>
<td>Round trip time in milliseconds to complete the most recent request to read the table.</td>
<td></td>
</tr>
<tr>
<td>Local write count</td>
<td>17598</td>
<td>Number of local requests to update the table since startup.</td>
<td></td>
</tr>
<tr>
<td>Local write latency</td>
<td>0.054 ms</td>
<td>Round trip time in milliseconds to complete an update to the table.</td>
<td>How are consistent read and write operations handled?</td>
</tr>
<tr>
<td>Pending flushes</td>
<td>0</td>
<td>Estimated number of reads, writes, and cluster operations pending for this table.</td>
<td>Important: Monitor this metric to watch for blocked or overloaded memtable flush writers. The <code>nodetool tpmstat</code> (page 1130) tool does not report on blocked flushwriters.</td>
</tr>
<tr>
<td>Name of statistic</td>
<td>Example value</td>
<td>Brief description</td>
<td>Related information</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>Percent repaired</td>
<td>100.0</td>
<td>Percentage of data (uncompressed) marked as repaired across all non-system tables on a node. Tables with a replication factor of 1 are excluded.</td>
<td></td>
</tr>
<tr>
<td>Bytes repaired</td>
<td>0.000KiB</td>
<td>Size of table data repaired on disk.</td>
<td></td>
</tr>
<tr>
<td>Bytes unrepairied</td>
<td>0.000KiB</td>
<td>Size of table data unrepairied on disk.</td>
<td></td>
</tr>
<tr>
<td>Bytes pending repair</td>
<td>0.000KiB</td>
<td>Size of table data isolated for an ongoing incremental repair.</td>
<td></td>
</tr>
<tr>
<td>Bloom filter false positives</td>
<td>0</td>
<td>Number of false positives reported by this table’s Bloom filter.</td>
<td>Tuning bloom filters</td>
</tr>
<tr>
<td>Bloom filter false ratio</td>
<td>0.00000</td>
<td>Fraction of all bloom filter checks resulting in a false positive from the most recent read.</td>
<td></td>
</tr>
<tr>
<td>Bloom filter space used, bytes</td>
<td>11688</td>
<td>Size in bytes of the bloom filter data for this table.</td>
<td></td>
</tr>
<tr>
<td>Bloom filter off heap memory</td>
<td>8</td>
<td>The number of bytes of offheap memory used for Bloom filters for this table.</td>
<td></td>
</tr>
<tr>
<td>Index summary off heap memory</td>
<td>41</td>
<td>The number of bytes of off heap memory used for index summaries for this table.</td>
<td></td>
</tr>
<tr>
<td>Compression metadata off heap</td>
<td>8</td>
<td>The number of bytes of off heap memory used for compression offset maps for this table.</td>
<td></td>
</tr>
</tbody>
</table>
## DataStax Enterprise tools

<table>
<thead>
<tr>
<th>Name of statistic</th>
<th>Example value</th>
<th>Brief description</th>
<th>Related information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compacted partition minimum</td>
<td>1110</td>
<td>Size in bytes of the smallest compacted partition for this table</td>
<td></td>
</tr>
<tr>
<td>Compacted partition maximum bytes</td>
<td>126934</td>
<td>Size in bytes of the largest compacted partition for this table.</td>
<td></td>
</tr>
<tr>
<td>Compacted partition mean bytes</td>
<td>2730</td>
<td>The average size of compacted partitions for this table.</td>
<td></td>
</tr>
<tr>
<td>Average live cells per slice (last five minutes)</td>
<td>0.0</td>
<td>Average number of cells scanned by single key queries during the last five minutes.</td>
<td></td>
</tr>
<tr>
<td>Maximum live cells per slice (last five minutes)</td>
<td>0.0</td>
<td>Maximum number of cells scanned by single key queries during the last five minutes.</td>
<td></td>
</tr>
<tr>
<td>Average tombstones per slice (last five minutes)</td>
<td>0.0</td>
<td>Average number of tombstones scanned by single key queries during the last five minutes.</td>
<td></td>
</tr>
<tr>
<td>Maximum tombstones per slice (last five minutes)</td>
<td>0.0</td>
<td>Maximum number of tombstones scanned by single key queries during the last five minutes.</td>
<td></td>
</tr>
<tr>
<td>Dropped mutations</td>
<td>0.0</td>
<td>The number of mutations (INSERT, UPDATE, or DELETE) started on this table but not completed.</td>
<td>A high number of dropped mutations can indicate an overloaded node.</td>
</tr>
</tbody>
</table>

### Synopsis

```
$ nodetool [connection_options] tablestats
[-F json | yaml] [-H] [-i] [--]
[keystore_name.table_name ...]
```
### Table 288: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>Repeatable. An ellipsis ( … ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

**-h, --host hostname**

The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

**-p, --port jmx_port**
The JMX port number.

**-pw, --password jmxpassword**

The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

**-pwf, --password-file jmx_password_filepath**

The filepath to the file that stores JMX authentication credentials.

**-u, --username jmx_username**

The user name for authenticating with secure JMX.

**Command arguments**

**--**

Separates an option from an argument that could be mistaken for a option.

**-F, --format json | yaml**

The format for the output. The default is plain text.

**-H, --human-readable**

Display bytes in human readable form: KiB (kibibyte), MiB (mebibyte), GiB (gibibyte), TiB (tebibyte).

**-i**

Ignore list of tables and display remaining tables.

**keyspace [tables]**

Run compaction on an entire keyspace or specified tables; use a space to separate table names.

- If you do not specify a keyspace or table, a major compaction is run on all keyspaces and tables.
- If you specify only a keyspace, a major compaction is run on all tables in that keyspace.
- If you specify one or more tables, a major compaction is run on those tables.

**Examples**

**Get table metrics on a single table in default format**

```bash
$ nodetool tablestats cycling.birthday_list
```

```
Total number of tables: 68
----------------
Keyspace : cycling
  Read Count: 0
  Read Latency: NaN ms
  Write Count: 20
  Write Latency: 0.05625 ms
  Pending Flushes: 0
  Table: birthday_list
  SSTable count: 0
  Space used (live): 0
  Space used (total): 0
  Space used by snapshots (total): 0
  Off heap memory used (total): 0
```
Get metrics on two tables in yaml format

```
$ nodetool tablestats -F yaml cycling.calendar cycling.birthday_list

total_number_of_tables: 68
cycling:
  write_latency_ms: 0.05625
tables:
  calendar:
    average_tombstones_per_slice_last_five_minutes: .NaN
    bloom_filter_off_heap_memory_used: '0'
    bytes_pending_repair: 0
    memtable_switch_count: 0
    maximum_tombstones_per_slice_last_five_minutes: 0
    memtable_cell_count: 12
    memtable_data_size: '854'
    average_live_cells_per_slice_last_five_minutes: .NaN
    local_read_latency_ms: NaN
    local_write_latency_ms: '0.046'
    pending_flushes: 0
    compacted_partition_minimum_bytes: 0
```
local_read_count: 0
sstable_compression_ratio: -1.0
dropped_mutations: '0'
bloom_filter_false_positives: 0
off_heap_memory_used_total: '0'
memtable_off_heap_memory_used: '0'
index_summary_off_heap_memory_used: '0'
bloom_filter_space_used: '0'
sstables_in_each_level: []
compacted_partition_maximum_bytes: 0
space_used_total: '0'
local_write_count: 12
compression_metadata_off_heap_memory_used: '0'
number_of_partitions_estimate: 3
bytes_repaired: 0
maximum_live_cells_per_slice_last_five_minutes: 0
space_used_live: '0'
compacted_partition_mean_bytes: 0
bloom_filter_false_ratio: '0.00000'
bytes_unrepaired: 0
percent_repaired: 100.0
space_used_by_snapshots_total: '0'
birthday_list:
  average_tombstones_per_slice_last_five_minutes: .NaN
bloom_filter_off_heap_memory_used: '0'
bytes_pending_repair: 0
memtable_switch_count: 0
maximum_tombstones_per_slice_last_five_minutes: 0
memtable_cell_count: 6
memtable_data_size: '799'
average_live_cells_per_slice_last_five_minutes: .NaN
local_read_latency_ms: NaN
local_write_latency_ms: '0.035'
pending_flushes: 0
compacted_partition_minimum_bytes: 0
local_read_count: 0
sstable_compression_ratio: -1.0
dropped_mutations: '0'
bloom_filter_false_positives: 0
off_heap_memory_used_total: '0'
memtable_off_heap_memory_used: '0'
index_summary_off_heap_memory_used: '0'
bloom_filter_space_used: '0'
sstables_in_each_level: []
compacted_partition_maximum_bytes: 0
space_used_total: '0'
local_write_count: 6
compression_metadata_off_heap_memory_used: '0'
number_of_partitions_estimate: 5
bytes_repaired: 0
maximum_live_cells_per_slice_last_five_minutes: 0
space_used_live: '0'
compacted_partition_mean_bytes: 0
bloom_filter_false_ratio: '0.00000'
nodetool toppartitions

Samples the activity in a table during the specified duration and reports the most active partitions.

Synopsis

$ nodetool [connection_options] toppartitions
[-a samplers] [-k num_partitions] [-s size] [--]
keyspace_name table_name duration

Table 289: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens (<code>--</code>). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>&lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks (<code>'</code>) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

**--**

Separates an option from an argument that could be mistaken for a option.

- **-a samplers, samplers2**
  Comma-separated list of samplers. Default is all.

**duration**

Duration in milliseconds.

- **-k num_partitions**
  Number of top partitions. Default is 10.

**keyspace_name**

The keyspace name.

- **-s size**
  Capacity of stream summary. A value closer to actual cardinality of partitions yields more accurate results. Default is 256.

**table_name**

The table name.
Examples

Sample the most active partitions for the table `test.users` for 1,000 milliseconds.

```
$ nodetool toppartitions test users 1000
```

The output of `nodetool toppartitions` is similar to the following:

<table>
<thead>
<tr>
<th>WRITES Sampler:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cardinality: ~2 (256 capacity)</td>
</tr>
<tr>
<td>Top 4 partitions:</td>
</tr>
<tr>
<td>Partition</td>
</tr>
<tr>
<td>4b504d39354f37353131</td>
</tr>
<tr>
<td>3738313134394d353530</td>
</tr>
<tr>
<td>4f363735324e324e4d30</td>
</tr>
<tr>
<td>303535324e4b4d504c30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>READS Sampler:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cardinality: ~3 (256 capacity)</td>
</tr>
<tr>
<td>Top 4 partitions:</td>
</tr>
<tr>
<td>Partition</td>
</tr>
<tr>
<td>4d4e30314f374e313730</td>
</tr>
<tr>
<td>4f363735324e324e4d30</td>
</tr>
<tr>
<td>303535324e4b4d504c30</td>
</tr>
<tr>
<td>4e355030324e344d3030</td>
</tr>
</tbody>
</table>

For each of the samplers used (WRITES and READS in the example), `toppartitions` reports:

- The cardinality of the sampled operations (that is, the number of unique operations in the sample set)

- The $n$ partitions in the specified table that had the most traffic in the specified time period (where $n$ is the value of the $-k$ argument, or ten if $-k$ is not explicitly set in the command).

For each Partition, `toppartitions` reports:

- **Partition**
  - The partition key

- **Count**
  - The number of operations of the specified type that occurred during the specified time period.

- **+/-**
  - The margin of error for the **Count** statistic

**Note:** To keep the `toppartitions` reporting from slowing performance, the database does not keep an exact count of operations, but uses sampling techniques to create an approximate number. (This example reports on a sample cluster; a production system might generate...
DataStax Enterprise tools

millions of reads or writes in a few seconds.) The +/- figure allows you to judge the accuracy of the toppartitions reporting.

nodetool tpstats

Prints usage statistics of thread pools. The DataStax Enterprise (DSE) database is based on a staged event-driven architecture (SEDA).

The database separates different tasks into stages connected by a messaging service. Each stage has a queue and a thread pool. Some stages skip the messaging service and queue tasks immediately on a different stage when it exists on the same node. The database can back up a queue if the next stage is too busy and lead to performance bottlenecks, as described in Monitoring a DataStax Enterprise cluster.

Reports are updated after SSTables change through compaction or flushing.

Report columns

The nodetool tpstats command report includes the following columns:

- **Active**
  - The number of Active threads.

- **Pending**
  - The number of Pending requests waiting to be executed by this thread pool.

- **Completed**
  - The number of tasks Completed by this thread pool.

- **Blocked**
  - The number of requests that are currently Blocked because the thread pool for the next step in the service is full.

- **All-Time Blocked**
  - The total number of All-Time Blocked requests, which are all requests blocked in this thread pool up to now.

Report rows

The follow list describes the task or property associated with the task reported in the nodetool tpstats output.

General metrics

The following report aggregated statistics for tasks on the local node:

- **BackgroundIoStage**
  - Completes background tasks like submitting hints and deserializing the row cache.

- **CompactionExecutor**
  - Running compaction.

- **GossipStage**
  - Distributing node information via Gossip. Out of sync schemas can cause issues. You may have to sync using nodetool resetlocalschema. (page 1060)

- **HintsDispatcher**
  - Dispatches a single hints file to a specified node in a batched manner.
**InternalResponseStage**
- Responding to non-client initiated messages, including bootstrapping and schema checking.

**MemtableFlushWriter**
- Writing memtable contents to disk. May back up if the queue is overruns the disk I/O, or because of sorting processes.
  
  **Warning:** nodetool tpstats no longer reports blocked threads in the MemtableFlushWriter pool. Check the Pending Flashes (page 1120) metric reported by nodetool tblestats.

**MemtablePostFlush**
- Cleaning up after flushing the memtable (discarding commit logs and secondary indexes as needed).

**MemtableReclaimMemory**
- Making unused memory available.

**PendingRangeCalculator**
- Calculating pending ranges per bootstraps and departed nodes. Reporting by this tool is not useful — see Developer notes.

**PerDiskMemtableFlushWriter_N**
- Activity for the memtable flush writer of each disk.

**ReadRepairStage**
- Performing read repairs. Usually fast, if there is good connectivity between replicas.

**Thread per core (TPC) task metrics**

All actions in the TPC loop are labeled and therefore observable. Tasks marked Pendable are throttled, limited to the value set for tpc_concurrent_requests_limit in the cassandra.yaml (by default, 128). Thread per core messages are prepended with TPC/type, where:

- TPC/N are metrics for the core number (when --cores is specified).
- TPC/other are metrics for tasks executed that are not on TPC threads.
- TPC/all are the aggregate task metrics for all cores.

**UNKNOWN**
- Unknown task.

**FRAME DECODE**
- Asynchronous frame decoding.

**READ LOCAL**
- Single-partition read request from a local node generated directly from clients.

**READ REMOTE**
- Single-partition read request from a remote replica.

**READ TIMEOUT**
- Signals read timeout errors.

**READ DEFERRED**
- Single-partition read request that will be first scheduled on an event loop (Pendable)

**READ RESPONSE**
- Single-partition read response.

**READ RANGE LOCAL**
Partition range read request from a local node generated directly from clients.

**READ_RANGE_REMOTE**
Partition range read request from a remote replica.

**READ_RANGE_NODESINC**
Partition range read originating from NodeSync.

**READ_RANGE_INTERNAL**
Range reads to internal tables.

**READ_RANGE_RESPONSE**
Partition range read response.

**READ_FROM_ITERATOR**
Switching thread to read from an iterator.

**READ_SECONDARY_INDEX**
Switching thread to read from secondary index.

**READ_DISK_ASYNC**
Waiting for data from disk.

**WRITE_LOCAL**
Write request from a local node generated directly from clients.

**WRITE_REMOTE**
Write request from a remote replica

**WRITE_INTERNAL**
Writes to internal tables.

**WRITE_RESPONSE**
Write response

**WRITE_DEFRAAGMENT**
Write issued to defragment data that required too many sstables to read (Pendable)

**WRITE_MEMTABLE**
Switching thread to write in memtable when not already on the correct thread

**WRITE_POST_COMMITLOG_SEGMENT**
Write request is waiting for the commit log segment to be allocated

**WRITE_POST_COMMITLOG_SYNC**
Write request is waiting for commit log to sync to disk

**WRITE_POST_MEMTABLE_FULL**
Write request is waiting for space in memtable

**BATCH_REPLAY**
Replaying a batch mutation

**BATCH_STORE**
Store a batchlog entry request (Pendable)

**BATCH_STORE_RESPONSE**
Store a batchlog entry response

**BATCH_REMOVE**
Remove a batchlog entry (Pendable)

**COUNTER_ACQUIRE_LOCK**
Acquiring counter lock.

**EXECUTE_STATEMENT**
Executing a statement.

**CAS**
Executing compare-and-set (LWT).

**LWT_PREPARE**
Preparation phase of light-weight transaction (Pendable).

**LWT_PROPOSE**
Proposal phase of light-weight transaction (Pendable).

**LWT_COMMIT**
Commit phase of light-weight transaction (Pendable).

**TRUNCATE**
Truncate request (Pendable).

**NODESYNC_VALIDATION**
NodeSync validation of a partition.

**AUTHENTICATION**
Authentication request.

**AUTHORIZATION**
Authorization request.

**TIMED_UNKNOWN**
Unknown timed task.

**TIMED_TIMEOUT**
Scheduled timeout task.

**EVENTLOOP_SPIN**
Number of busy spin cycles done by this TPC thread when it has no tasks to perform.

**EVENTLOOP_YIELD**
Number of Thread.yield() calls done by this TPC thread when it has no tasks to perform.

**EVENTLOOP_PARK**
Number of LockSupport.park() calls done by this TPC thread when it has no tasks to perform.

**HINT_DISPATCH**
Hint dispatch request (Pendable).

**HINT_RESPONSE**
Hint dispatch response.

**NETWORK_BACKPRESSURE**
Scheduled network backpressure.

Droppable messages

The database generates the messages listed below, but discards them after a timeout. The `nodetool tpstats` command reports the number of messages of each type that have been dropped. You can view the messages themselves using a JMX client.

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Stage</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>n/a</td>
<td>Deprecated</td>
</tr>
<tr>
<td>_TRACE</td>
<td>n/a (special)</td>
<td>Used for recording traces (nodetool settraceprobability) Has a special executor (1 thread, 1000 queue depth) that throws away messages on insertion instead of within the execute</td>
</tr>
</tbody>
</table>

DSE 6.7 Developer Guide (Latest version)
<table>
<thead>
<tr>
<th>Message Type</th>
<th>Stage</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUTATION</td>
<td>MutationStage</td>
<td>If a write message is processed after its timeout (write_request_timeout_in_ms) it either sent a failure to the client or it met its requested consistency level and will relay on hinted handoff and read repairs to do the mutation if it succeeded.</td>
</tr>
<tr>
<td>COUNTER_MUTATION</td>
<td>MutationStage</td>
<td>If a write message is processed after its timeout (write_request_timeout_in_ms) it either sent a failure to the client or it met its requested consistency level and will relay on hinted handoff and read repairs to do the mutation if it succeeded.</td>
</tr>
<tr>
<td>READ_REPAIR</td>
<td>MutationStage</td>
<td>Times out after write_request_timeout_in_ms.</td>
</tr>
<tr>
<td>READ</td>
<td>ReadStage</td>
<td>Times out after read_request_timeout_in_ms. No point in servicing reads after that point since it would of returned error to client</td>
</tr>
<tr>
<td>RANGE_SLICE</td>
<td>ReadStage</td>
<td>Times out after range_request_timeout_in_ms.</td>
</tr>
<tr>
<td>PAGED_RANGE</td>
<td>ReadStage</td>
<td>Times out after request_timeout_in_ms.</td>
</tr>
<tr>
<td>REQUEST_RESPONSE</td>
<td>RequestResponseStage</td>
<td>Times out after request_timeout_in_ms. Response was completed and sent back but not before the timeout</td>
</tr>
</tbody>
</table>

Synopsis

$ nodetool [connection_options] tpstats
[-C] [-F json | yaml]

Table 290: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italicics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (‘) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (-- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (‘) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
-pwf, --password-file jmx_password_filepath
The filepath to the file that stores JMX authentication credentials.

-u, --username jmx_username
The user name for authenticating with secure JMX.

Command arguments

-C, --cores
Include data for each core. The number of cores is determined by the tpc_cores
(page 78).

-F, --format json | yaml
The format for the output. The default is plain text.

Examples

Run nodetool tpstats on the host labcluster

```bash
$ nodetool tpstats -C
```

Command output is:

<table>
<thead>
<tr>
<th>Pool Name</th>
<th>Active</th>
<th>Pending (w/ Backpressure)</th>
<th>Delayed</th>
<th>Completed</th>
<th>Blocked</th>
<th>All time blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompactionExecutor</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>90552</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemtableFlushWriter</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>131</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemtablePostFlush</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>438</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemtableReclaimMemory</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>131</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PendingRangeCalculator</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PerDiskMemtableFlushWriter_0</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>130</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TPC/0</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>15488</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>TPC/0/EVENTLOOP_PROCESSED_TASKS</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>2243409</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TPC/0/EVENTLOOP_SCHEDULED_TASKS</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>3352216</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TPC/0/EVENTLOOP_SELECTOR_EVENTS</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>1121343</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TPC/0/EVENTLOOP_SELECT_CALLS</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>1121307</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TPC/0/EVENTLOOP_SELECT_NOW_CALLS</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>1121360</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TPC/0/EVENTLOOP_SPIN</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>10091799</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TPC/0/READ_DISK_ASYNC</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>54</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Message type</td>
<td>Dropped</td>
<td>Latency waiting in queue (micros)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------</td>
<td>----------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/0/READ_INTERNAL</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>3704</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/0/READ_RANGE_INTERNAL</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/0/READ_RANGE_SWITCH_FOR_RESPONSE</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/0/READ_SWITCH_FOR_RESPONSE</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>3704</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/0/TIMED_TIMEOUT</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>1109491</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/0/UNKNOWN</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/0/WRITE_INTERNAL</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/0/WRITE_SWITCH_FOR_MEMTABLE</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>8017</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/0/WRITE_SWITCH_FOR_RESPONSE</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/EVENTLOOP_PROCESSED_TASKS</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>2243419</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/EVENTLOOP_SCHEDULED_TASKS</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>3352225</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/EVENTLOOP_SELECTOR_EVENTS</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>1121341</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/EVENTLOOP_SELECT_CALLS</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>1121306</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/EVENTLOOP_SELECT_NOW_CALLS</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>1121361</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/EVENTLOOP_SPIN</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>10091817</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/READ_DISK_ASYNC</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>54</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/READ_INTERNAL</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>3704</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/READ_RANGE_INTERNAL</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/READ_RANGE_SWITCH_FOR_RESPONSE</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/READ_SWITCH_FOR_RESPONSE</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>3704</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/TIMED_TIMEOUT</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>1109489</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/UNKNOWN</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/WRITE_INTERNAL</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/WRITE_SWITCH_FOR_MEMTABLE</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>8017</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC/all/WRITE_SWITCH_FOR_RESPONSE</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A (N/A)</td>
<td>2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Message type: Dropped, Latency waiting in queue (micros)
nodetool truncatehints

Truncates all hints on the local node or for one or more endpoints.

Synopsis

$ nodetool [connection_options] truncatehints
[--] [endpoint ...]

Table 291: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (‘’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({}) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (‘’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

--
Separates an option from an argument that could be mistaken for a option.

**endpoint**
Endpoint address or addresses. IP address or hostname.

**nodetool upgradesstables**

Rewrites SSTables for tables that are not running the current version of DataStax Enterprise to upgrade to current version. Use this command when upgrading your server or changing compression options.

See [sstableupgrade](#) for SSTable compatibility with current DSE version.

**Synopsis**

```bash
$ nodetool [connection_options] upgradesstables
[-a] [-j num_jobs] [--] [keyspace_name table_name [table_name ...]]
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>'&lt; &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (') surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.
- **-p, --port jmx_port**
  The JMX port number.
- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.
- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.
- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

**Command arguments**

--
Separates an option from an argument that could be mistaken for a option.

- **-a, --include-all-sstables**
  Upgrade target SSTables, including SSTables already on the current DSE version.
- **-j, --jobs num_jobs**
  - num_jobs - Number of SSTables affected simultaneously.
  - 0 - Use all available compaction threads.

- **keyspace_name**
  The keyspace name.
- **table_name**
  One or more table names, separated by a space.

**Examples**

**Upgrade all SSTables in the cycling keyspace and the cyclist_name table**

```
$ nodetool -u user1 -pw password1 upgradesstables --include-all-sstables
cycling cyclist_name
```
Upgrade all SSTables in the cycling keyspace

```
$ nodetool -u user1 -pw password1 upgradesstables --include-all-sstables cycling
```

Upgrade all SSTables in all keyspaces and tables

```
$ nodetool -u user1 -pw password1 upgradesstables
```

If the SSTables are already on the current version, the command returns immediately and no action is taken.

**nodetool verify**

Checks the data checksum for one or more specified tables.

**Synopsis**

```
$ nodetool [connection_options] verify
[-e] [--] keyspace_name table_name [table_name ...]
```

**Table 293: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

### Command arguments

- **-e, --extended-verify**
  Each cell data, beyond simply checking SSTable checksums.

  `--`
  Separates an option from an argument that could be mistaken for a option.

- **keyspace_name**
  The keyspace name.

- **table_name**
  One table name, or many table names separated with a space.

### Examples

**Verify data checksum**
$ nodetool -u username -pw password verify cycling cyclist_name

nodetool version

Provides the DSE database version.

Synopsis

$ nodetool [connection_options] version

Table 294: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group.Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
Definition

The short form and long form parameters are comma-separated.

Connection options

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pwf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.

- **-u, --username jmx_username**
  The user name for authenticating with secure JMX.

Command arguments

This command takes no arguments.

Examples

Run nodetool version

```
$ nodetool version
```

```
ReleaseVersion: 4.0.0.602
```

**nodetool viewbuildstatus**

Shows the progress of a materialized view build.

Synopsis

```
$ nodetool [connection_options] viewbuildstatus
keyspace_name view_name | keyspace_name.view_name
```

**Table 295: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (() ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>’ &lt;schema&gt; ... &lt;/schema&gt; ’</td>
<td>Search CQL only: Single quotation marks (’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Connection options**

- **-h, --host hostname**
  The hostname or IP address of a remote node or nodes. When omitted, the default is the local machine.

- **-p, --port jmx_port**
  The JMX port number.

- **-pw, --password jmxpassword**
  The JMX password for authenticating with secure JMX. If a password is not provided, you are prompted to enter one.

- **-pf, --password-file jmx_password_filepath**
  The filepath to the file that stores JMX authentication credentials.
-u, --username jmx_username
   The user name for authenticating with secure JMX.

Command arguments

keyspace_name
   Keyspace name. By default, all keyspaces.
view
   The name of the view.

dse commands

The dse commands for starting the database and connecting an external client to a DataStax Enterprise node and performing common utility tasks.

About dse commands

The dse commands provide controls for starting and using DataStax Enterprise (DSE).

dse subcommands

Specify one dse subcommand and none or more optional command arguments.

   Note: When multiple flags are used, list them separately on the command line. For example, ensure there is a space between -k and -s in dse cassandra -k -s.

DSE Multi-Instance commands

To run standard DataStax Enterprise commands for nodes on a DSE Multi-Instance host machine, specify the node name using this syntax:

   sudo dse dse-nodeId subcommand [command_arguments]

For details, see DSE Multi-Instance commands.

dse command connection options

Options to authenticate connections to the database and to JMX for dse commands.

Synopsis

```
$ dse
[-f config_file | 
- u username -p password]
[-a jmx_username [-b jmx_password]]
command [options]
```
# Table 296: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code> &lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Specify how to connect and authenticate to the database for dse commands.

This list shows short form (-f filename) and long form (--config-file=filename):

**-f, --config-file config_filename**

File path to configuration file that stores credentials. The credentials in this configuration file override the ~/.dserc credentials. If not specified, then use ~/.dserc if it exists.

The configuration file can contain DataStax Enterprise and JMX login credentials. For example:
username=username
password=password
jmx_username=jmx_username
jmx_password=jmx_password

The credentials in the configuration file are stored in clear text. DataStax recommends restricting access to this file only to the specific user.

-u username
   Role to authenticate for database access.
-p, --password password
   Password to authenticate for database access.
-a, --jmxusername jmx_username
   User name for authenticating with secure local JMX.
-b, --jmxpassword jmx_password
   Password for authenticating with secure local JMX. If you do not provide a password, you are prompted to enter one.

Examples

To authenticate a connection to the database

$ dse -u user1 -p mypassword

To authenticate a connection using a configuration file

$ dse -f configfile

dse add-node

For DSE Multi-Instance, simplifies adding and configuring a node on a host machine. When optional parameters are absent, the default values remain unchanged.

   Important: The user running the command must have permissions for writing to the directories that DSE uses, or use sudo.

   Restriction: DSE Multi-Instance commands are supported only on package installations.

Synopsis

$ dse add-node -n nodeId
   [--advrep-directory advrepdirectory]
   [--analytics]
   [--cdc-directory=cdcdirectory]
   [--cluster=clustername]
   [--commit-directory=commitdirectory]
   [--cpus=number_of_cpus]
   [--dc=datacenter_placement]
```bash
[---data-directory=databasename]
[---dsefs] [---dsefs-directory=dsefrdirectory]
[---graph]
[---hadoop-logs=hadooplogsdirectory]
[help]
[---hints-directory=hintsdirectory]
[---jmxport=jmx_port]
[---listen-address=listen_IP_address]
[---logs-directory=alllogsdirectory]
[---max-heap-size=heapsize]
[---native-transport-address=native_transport_IP_address]
[---num-tokens=number_of_tokens]
[---pig-logs=piglogdirectory]
[---rack=rack_placement]
[---rpc-address=rpc_IP_address]
[---saved-caches-directory=savedcachesdirectory]
[---search]
[---seeds=IP_address1,IP_address2,...]
[---spark-local-directory=sparklocaldirectory]
[---spark-log-directory=sparklogdirectory]
[---spark-worker-cores=number_of_cores]
[---spark-worker-directory=sparkworkerdirectory]
[---spark-worker-memory=memory]
[---tomcat-logs=tomcatlogsdirectory]
[---unix-group=groupname]
[---unix-username=username]
```

### Table 297: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks (‘) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### New node configuration options:

- **n=nodeId, --node-id=nodeId**
  Required. For DSE Multi-Instance, the alphanumeric node name for the new node. The specified node name is automatically prepended with dse- so that the resulting node ID is dse-nodeId. For example, if you specify node1, the resulting node name is dse-node1.

- **--advrep-directory=advrepdirectory**
  Optional. The DSE Advanced Replication data directory.
  Default: /var/lib/dse-nodeId/advrep

- **--analytics**
  Enable DSE Analytics.

- **--cdc-directory=cdcdirectory**
  Optional. The CDC raw data directory.
  Default: /var/lib/dse-nodeId/cdc_raw

- **--cluster=clusternname**
  Optional. The name of the DataStax Enterprise cluster that the new node belongs to. Only non-whitespace values are supported.

- **--cpus=number_of_cpus**
  Optional. The number of cores.

- **--commit-directory=commitdirectory**
  Optional. The commit log directory.
  Default: /var/lib/dse-nodeId/commitlog

- **--dc=datacenter_placement**
  Optional. The data center placement.

- **--data-directory=datadirectory**
  Optional. The root directory for storing data.
  Default: /var/lib/dse-nodeId/data

- **--dsefs**
  Optional. Enable DSEFS.

- **--dsefs-directory=dsefsdatadirectory**
  Optional. The DSEFS data directory.
DataStax Enterprise tools

Default: /var/lib/dse-nodeId/dsefs

--graph
Optional. Enable DSE Graph.

--hadoop-logs=hadooplogsdirectory
Optional. The log directory for Hadoop logs.
Default: logs-directory/hadoop

--help
Optional. Send dse add-node option descriptions to standard output.

--hints-directory=hintsdirectory
Optional. The hints directory.
Default: /var/lib/dse-nodeId/hints

--jmxport=jmx_port
Optional. The DSE JMX metrics monitoring port.

--listen-address=listen_IP_address
Optional. The IP address or hostname that DSE binds to when connecting to other nodes.

--logs-directory=alllogsdirectory
Optional. The root directory for all of the logs.
Default: /var/log/dse-nodeId

--max-heap-size=heapsize
Optional. The Java heap size. If you omit MB the size is interpreted as megabytes.

--num-tokens=number_of_tokens
Optional. The number of tokens.

--pig-logs=piglogdirectory
The log directory for Pig logs.
Default: logs-directory/pig

--rack=rack_placement
Optional. The rack placement.

--rpc-address=rpc_IP_address
Optional. The IP address or hostname that DSE binds to for RPC requests.

--saved-caches-directory=savedcachesdirectory
Optional. The saved caches directory.
Default: /var/lib/dse-nodeId/saved_caches

--search
Optional. Enable DSE Search.

--seeds=IP_address1,IP_address2,...
Optional. A comma-separated list of IP addresses of the nodes to be used as seed nodes.

--spark-local-directory=sparklocaldirectory
Optional. The local directory for Spark Worker.
Default: /var/lib/dse-nodeId/spark/rdd

--spark-log-directory=sparklogdirectory
Optional. The log directory for Spark Worker.
Default: /var/log/dse-nodeId/spark/worker

--spark-worker-cores=number_of_cores
Optional. The maximum number of cores used by Spark executors.

--spark-worker-directory=sparkworkerdirectory
Optional. The data directory for Spark Worker.
Default: /var/lib/dse-nodeId/spark/worker

--spark-worker-memory=memory
  Optional. The maximum amount of memory used by Spark executors. Specify unit of measure with k (kilobytes), m (megabytes), g (gigabytes).

--tomcat-logs=tomcatlogsdirectory
  Optional. The log directory for tomcat logs.
  Default: logs-directory/tomcat

--unix-group=groupname
  Optional. The UNIX group that owns the node configuration.
  Default: cassandra

--unix-username=username
  Optional. The UNIX user that owns the node configuration.
  Default: cassandra

Examples

Add node1

$ dse add-node node1

The dse-node1 is created on the local machine.

Add a node that will join the cluster payroll on startup

$ dse add-node payrollnode --cluster payroll --listen-address 192.168.0.0
  --rpc-address 192.168.0.1 --seeds 192.168.0.2

The payrollnode is created with the specified configuration options.

**dse advrep**

**dse advrep commands**

A list of commands for DSE Advanced Replication.

**About the dse advrep command**

The command line tool provides commands and options for configuring and using DSE Advanced Replication.

**Synopsis**

$ dse advrep [connection_options] [command] [sub_command]
  [sub_command_options]

The default port for DSE Advanced Replication is 9042.
### Table 298: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>' Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

#### Using dse advrep command line help

To view a listing of dse advrep commands:

```
$ dse advrep help
```

To view help for a specific command:
$ dse advrep help command [ sub_command ]

Connection options

JMX authentication is supported by some dse commands. Other dse commands authenticate with the user name and password of the configured user. The connection option short form and long form are comma separated.

**Note:** You can provide authentication credentials in several ways, see Credentials for authentication.

General connection options:

--separator field_separator
The field separator for use with the --no-pretty-print command.

--verbose
Print verbose messages for command.

--verbose
Displays which arguments are recognized as Spark configuration options and which arguments are forwarded to the Spark shell.

--no-pretty-print
If not specified, data is printed using tabular output. If specified, data is printed as a comma separated list unless a separator is specified.

--cipher-suites ssl_cipher_suites
Specify comma-separated list of SSL cipher suites for connection to DSE when SSL is enabled. For example, --cipher-suites c1,c2,c3.

--host hostname
The DSE node hostname or IP address.

--jmx-port jmx_port
The remote JMX agent port number. Default: 7199.

--jmx-pwd jmx_password
The password for authenticating with secure local JMX. If you do not provide a password, you are prompted to enter one.

--jmx-user jmx_username
The username for authenticating with secure local JMX.

--kerberos-enabled true | false
Whether Kerberos authentication is enabled for connections to DSE. For example, --kerberos-enabled true.

--keystore-password keystore_password
Keystore password for connection to DSE when SSL client authentication is enabled.

--keystore-path ssl_keystore_path
Path to the keystore for connection to DSE when SSL client authentication is enabled.

--keystore-type ssl_keystore_type
Keystore type for connection to DSE when SSL client authentication is enabled. JKS is the type for keys generated by the Java keytool binary, but other types are possible, depending on user environment.
-p password
The password to authenticate for database access. Can use the DSE_PASSWORD environment variable.

--ssl
Whether SSL is enabled for connection to DSE. --ssl-enabled true is the same as --ssl.

--ssl-protocol ssl_protocol
SSL protocol for connection to DSE when SSL is enabled. For example, --ssl-protocol ss14.

-t token
Specify delegation token which can be used to login, or alternatively, DSE_TOKEN environment variable can be used.

--truststore_password ssl_truststore_password
Truststore password to use for connection to DSE when SSL is enabled.

--truststore_path ssl_truststore_path
Path to the truststore to use for connection to DSE when SSL is enabled. For example, --truststore-path /path/to/ts.

--truststore-type ssl_truststore_type
Truststore type for connection to DSE when SSL is enabled. JKS is the type for keys generated by the Java keytool binary, but other types are possible, depending on user environment. For example, --truststore-type jks2.

-u username
User name of a DSE authentication account. Can use the DSE_USERNAME environment variable.

Examples
This connection example specifies that Kerberos is enabled and lists the replication channels:

```
$ dse advrep --host ip-10-200-300-138.example.lan --kerberos-enabled=true conf list
```

To use the server YAML files:

```
$ dse advrep --use-server-config conf list
```

To list output without pretty-print with a specified separator:

```
dse advrep --no-pretty-print --separator "|" destination list-conf
```

This output will result:

```
destination|name|value
mydest|addresses|192.168.200.100
mydest|transmission-enabled|true
mydest|driver-ssl-cipher-suites|TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_ECDH_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_3DES_EDE_CBC_SHA,TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA,TLS_RSA_WITH_RC4_128_MD5,TLS_EMPTY_RENEGOTIATION_INFO_SCSV
mydest|driver-ssl-enabled|false
mydest|driver-ssl-protocol|TLS
mydest|name|mydest
```
**dse advrep channel create**

Creates a replication channel for change data to flow between source clusters and destination clusters.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```bash
$ dse advrep channel create
   --source-keyspace  keyspace_name
   --source-table    source_table_name
   --source-id       source_id_name
   --source-id-column source_id_column_name
   --destination     destination
   --destination-keyspace destination_keyspace_name
   --destination-table destination_table_name
   [ --fifo-order | --lifo-order ] [ --collection-enabled (true|false) ]
   [ --priority channel_priority ] [ --transmission-enabled (true|false) ]
```

**Table 299: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;' &lt;schema&gt; ... '&lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

---

**--source-keyspace keyspace_name (required)**

The source cluster keyspace to replicate.

**--source-table source_table_name (required)**

The source table to replicate.

**--source-id id**

A unique identifier for all data that comes from a particular source node.

**--source-id-column source_id**

The column that identifies the source id in the destination table.

**--destination destination (required)**

The destination where the replication will be sent; the user names the destination.

**--destination-keyspace keyspace_name**

The destination keyspace to which replication will be sent.

**--destination-table table_name**

The destination table to which replication will be sent.

**--fifo-order**

First in, first out channel (FIFO) replication order. Default.

**--lifo-order**

Last in, last out (LIFO) channel replication order.

**--collection-enabled ( true | false )**

Whether to enable the source table for replication collection on creation.

**--transmission-enabled ( true | false )**

Whether to replicate data collector for the table to the destination.
--priority channel_priority
    The order in which the source table log files are transmitted.

Examples

To create a replication source channel:

```bash
$ dse advrep channel create --source-keyspace foo --source-table bar --source-id sourcel --source-id-column source_id --destination mydest --destination-keyspace foo --destination-table bar --collection-enabled true --priority 1
```

with a result:

```
$ Created channel dc=Cassandra keyspace=foo table=bar to mydest
```

The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same. You can also set the source-id and source-id-column differently from the global setting.

**dse advrep channel update**

Updates a replication channel configuration.

A replication channel is a defined channel of change data between source clusters and destination clusters.

To update a channel, specify a new value for one or more options.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```bash
$ dse advrep channel update
    --source-keyspace keyspace_name
    --source-table source_table_name
    --source-id source_id_name
    --source-id-column source_id_column_name
    --destination destination
    --destination-keyspace destination_keyspace_name
    --destination-table destination_table_name
    [ --fifo-order | --lifo-order ]
    [ --collection-enabled (true|false) ] [ --transmission-enabled (true|false) ]
    [ --priority channel_priority ]
```
## Table 300: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--source-keyspace **keyspace_name** (required)
   
   The source cluster keyspace to replicate.

--source-table **source_table_name** (required)
   
   The source table to replicate.

--source-id **id**
   
   A unique identifier for all data that comes from a particular source node.

--source-id-column **source_id**
   
   The column that identifies the source id in the destination table.
--destination destination (required)
The destination where the replication will be sent; the user names the destination.

--destination-keyspace keyspace_name
The destination keyspace to which replication will be sent.

--destination-table table_name
The destination table to which replication will be sent.

--fifo-order
First in, first out channel (FIFO) replication order. Default.

--lifo-order
Last in, last out (LIFO) channel replication order.

--collection-enabled (true | false)
Whether to enable the source table for replication collection on creation.

--transmission-enabled (true | false)
Whether to replicate data collector for the table to the destination.

--priority channel_priority
The order in which the source table log files are transmitted.

Examples

To update a replication source channel configuration:

$ dse advrep --verbose channel update --source-keyspace demo --source-table sensor_readings --destination mydest --lifo-order

with a result as seen using dse advrep channel status:

|dc   |keyspace|table           |collecting|transmitting|
|replication order|priority|dest ks|dest table      |src id |src id col|dest   |dest enabled|
|Cassandra|demo    |sensor_readings |true      |true        |LIFO
|2       |demo   |sensor_readings |source1|source_id |mydest |
true

The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same. You can also set the source-id and source-id-column differently from the global setting.

dse advrep channel delete

Deletes a replication channel.

A replication channel is a defined channel of change data between source clusters and destination clusters.
To delete a channel, you must specify source information and the destination and data-center for the channel.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep channel delete
    --source-keyspace keyspace_name
    --source-table source_table_name
    --destination destination
    --data-center-id data_center_id
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>'&lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks (‘) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Examples

**To create a replication source channel:**

```bash
$ dse advrep channel delete --source-keyspace foo --source-table bar --destination mydest --data-center-id Cassandra
```

with a result:

```
Deleted channel dc=Cassandra keyspace=foo table=bar to mydest
```

The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same.

**dse advrep channel pause**

Pauses replication for a channel for change data to flow from a source cluster to a destination cluster.

A replication channel is a defined channel of change data between source clusters and destination clusters.

Pause collection of data or transmission of data between a source cluster and destination cluster.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```bash
$ dse advrep channel pause
  --source-keyspace keyspace_name
```
```bash
--source-table source_table_name
--destinations destination [ , destination ]
--data-center-ids data_center_id [ , data_center_id ]
--collection
--transmission
```

### Table 302: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

```bash
--source-keyspace keyspace_name
The source cluster keyspace to replicate.
--source-table source_table_name
```
The source table to replicate.

--destinations destination [, destination ]
   The destinations where the replication are sent.

--data-center-ids data_center_id [, data_center_id ]
   The datacenters for this channel, which must exist.

--collection
   No data for the source table is collected.

--transmission
   No data for the source table is sent to the configured destinations.

Examples

To pause a replication source channel:

$ dse advrep channel pause --source-keyspace foo --source-table bar --
  destinations mydest --data-center-ids Cassandra

with a result:

Channel dc=Cassandra keyspace=foo table=bar collection to mydest was
  paused

The source datacenter will be the datacenter in which the command is run. The keyspace
and table names on the destination can be different than on the source, but in this
example they are the same.

dse advrep channel resume

Resumes replication for a channel.

A replication channel is a defined channel of change data between source clusters and
destination clusters.

A channel can resume either the collection or transmission of replication between a source
cluster and destination cluster.

   Restriction: Command is supported only on nodes configured for DSE Advanced
   Replication.

Synopsis

$ dse advrep channel resume
   --source-keyspace keyspace_name
   --source-table source_table_name
   --destinations destination [, destination ]
   --data-center-ids data_center_id [, data_center_id ]
   --collection
   --transmission
### Table 303: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**--source-keyspace keyspace_name**  
The source cluster keyspace to replicate.

**--source-table source_table_name**  
The source table to replicate.

**--destinations destination [, destination ]**  
The destinations where the replication are sent.

**--data-center-ids data_center_id [, data_center_id ]**  
The datacenters for this channel, which must exist.
--collection
   No data for the source table is collected.
--transmission
   No data for the source table is sent to the configured destinations.

Examples

To resume a replication source channel:

```
$ dse advrep channel resume --source-keyspace foo --source-table bar --destinations mydest --data-center-ids Cassandra
```

with a result:

```
Channel dc=Cassandra keyspace=foo table=bar collection to mydest was resumed
```

The source datacenter will be the datacenter in which the command is run. The keyspace
and table names on the destination can be different than on the source, but in this
example they are the same.

dse advrep channel status

Prints status of a replication channel.

A replication channel is a defined channel of change data between source clusters and
destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced
Replication.

Synopsis

```
$ dse advrep channel status
   --data-center-id data_center_id
   --source-keyspace keyspace_name
   --source-table source_table_name
   --destination destination
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**--source-keyspace keyspace_name**  
The source cluster keyspace to replicate.

**--source-table source_table_name**  
The source table to replicate.

**--destination destination**  
The destination where the replication will be sent; the user names the destination.

**--data-center-id data_center_id**  
The datacenter for this channel.

Examples

**To print the status of a replication channel:**

```
$ dse advrep channel status --source-keyspace foo --source-table bar --destination mydest --data-center-id Cassandra
```

with a result:
The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same.

**dse advrep channel truncate**

Truncates a channel to prevent replicating all messages that are currently in the replication log.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```bash
$ dse advrep channel truncate
   --source-keyspace keyspace_name
   --source-table source_table_name
   --destinations destination [ , destination ]
   --data-center-ids data_center_id [ , data_center_id ]
```

**Table 305: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Examples

To truncate a replication channel to prevent replicating all messages that are currently in the replication log:

$$ dse advrep channel status --source-keyspace foo --source-table bar --destinations mydest --data-center-ids Cassandra $$

with a result:

Channel dc=Cassandra keyspace=foo table=bar to mydest was truncated
The source datacenter will be the datacenter in which the command is run. The keyspace and table names on the destination can be different than on the source, but in this example they are the same.

**dse advrep conf list**

Lists configuration settings for advanced replication.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep conf list
```

**Table 306: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ([--]). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Examples

**To list configuration settings:**

```bash
$ dse advrep conf list
```

The result:

```
<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_log_file</td>
<td>auditLog</td>
</tr>
<tr>
<td>permits</td>
<td>8</td>
</tr>
<tr>
<td>audit_log_enabled</td>
<td>true</td>
</tr>
</tbody>
</table>
```

The number of permits is 8, audit logging is enabled, and the audit log file name is auditLog.

**dse advrep conf remove**

Removes configuration settings for advanced replication.

A replication channel is a defined channel of change data between source clusters and destination clusters.

### Synopsis

```bash
$ dse advrep conf remove
  --separator field_separator
  --audit-log-enabled true|false
  --audit-log-compression none|gzip
  --audit-log-file log_file_name
  --audit-log-max-life-span-mins number_of_minutes
  --audit-log-rotate-mins number_of_minutes
  --permits number_of_permits
  --collection-max-open-files number_of_files
  --collection-time-slice-count number_of_files
  --collection-time-slice-width time_period_in_seconds
```
Table 307: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (()) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>

--audit-log-compression true|false
Enable or disable audit logging.

--audit-log-compression none|gzip
Enable audit log compression. Default: none

--audit-log-file *log_file_name*
Define the audit log filename.

--audit-log-rotate-max number_of_minutes
Define the maximum number of minutes for the audit log lifespan.

--audit-log-rotate-mins number_of_minutes
Define the number of minutes before the audit log will rotate.

--permits number_of_permits
Maximum number of messages that can be replicated in parallel over all destinations. Default: 1024

--collection-max-open-files number_of_files
Number of open files kept.

--collection-time-slice-count number_of_files
Specify the number of files which are open in the ingestor simultaneously.

--collection-time-slice-width time_period_in_seconds
Specify the time period in seconds for each data block ingested. Smaller time widths mean more files, whereas larger timer widths mean larger files, but more data to resend on CRC mismatches.

--collection-expire-after-write
Specify if the collection expires after the write occurs.

--invalid-message-log none|system_log|channel_log
Specify where error information is stored for messages that could not be replicated. Default: channel_log
Examples

To remove advanced replication configuration:

```
$ dse advrep conf remove --permits 8
```

with a result:

```
Removed config permits
```

```
dse advrep conf update
```

Updates configuration settings for advanced replication.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

Synopsis

```
$ dse advrep conf update
   --audit-log-enabled true|false
   --audit-log-compression none|gzip
   --audit-log-file log_file_name
   --audit-log-max-life-span-mins number_of_minutes
   --audit-log-rotate-mins number_of_minutes
   --permits number_of_permits
   --collection-max-open-files number_of_files
   --collection-time-slice-count number_of_files
   --collection-time-slice-width time_period_in_seconds
   --collection-expire-after-write
   --invalid-message-log
```

Table 308: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td><strong>Syntax conventions</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--audit-log-compression true|false
   Enable or disable audit logging.
--audit-log-compression none|gzip
   Enable audit log compression. Default: none
--audit-log-file log_file_name
   Define the audit log filename.
--audit-log-rotate-max number_of_minutes
   Define the maximum number of minutes for the audit log lifespan.
--audit-log-rotate-mins number_of_minutes
   Define the number of minutes before the audit log will rotate.
--permits number_of_permits
   Maximum number of messages that can be replicated in parallel over all destinations. Default: 1024
--collection-max-open-files number_of_files
   Number of open files kept.
--collection-time-slice-count number_of_files
   Specify the number of files which are open in the ingestor simultaneously.
--collection-time-slice-width time_period_in_seconds
DataStax Enterprise tools

Specify the time period in seconds for each data block ingested. Smaller time widths mean more files, whereas larger timer widths mean larger files, but more data to resend on CRC mismatches.

--collection-expire-after-write
   Specify if the collection expires after the write occurs.

--invalid-message-log none|system_log|channel_log
   Specify where error information is stored for messages that could not be replicated. Default: channel_log

Examples

To update configuration settings:

```
$ dse advrep conf update --permits 8 --audit-log-enabled true --audit-log-file auditLog
```

with a result:

```
Updated audit_log_file from null to auditLog
Updated permits from null to 8
Updated audit_log_enabled from null to true
```

dse advrep destination create

Creates a replication destination.

A replication channel is a defined channel of change data between source clusters and destination clusters.

Restriction: Command is supported only on nodes configured for DSE Advanced Replication.

Synopsis

```
$ dse advrep destination create
   --name destination_name
   --addresses address_name [ , address_name ]
   [ --transmission-enabled (true|false) ]
   --driver-user user_name
   --driver-pwd password
   --driver-used-hosts-per-remote-dc number_of_hosts
   --driver-connections number_of_connections
   --driver-connections-max number_of_connections
   --driver-local-dc data_center_name
   --driver-allow-remote-dcs-for-local-cl true|false
   --driver-consistency-level [ ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL QUORUM|EACH QUORUM|SERIAL|LOCAL SERIAL|LOCAL ONE ]
   --driver-compression [ snappy|lz4 ]
   --driver-connect-timeout timeout_in_milliseconds
   --driver-read-timeout timeout_in_milliseconds
   --driver-max-requests-per-connection number_of_requests
```
Table 309: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve uppercase.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

---

**--name destination_name (required)**

Designate the name of the destination.

**--addresses address_name [, address_name ] (required)**

Identify the IP addresses of the destinations.

**--transmission-enabled true | false**

Specify if data collector for the table should be replicated to the destination.

**--driver-user user_name**

Specify the username for the destination.

**--driver-pwd password**

Specify the password for the destination.

**--driver-used-hosts-per-remote-dc number_of_hosts**

Define the number of hosts per remote datacenter that the datacenter-aware round robin policy considers available for use.

**--driver-connections number_of_connections**

Specify the number of connections that the driver creates.

**--driver-connections-max number_of_connections**

Specify the maximum number of connections that the driver creates.

**--driver-local-dc data_center_name**

Specify the name of the datacenter that is considered local.

**--driver-consistency-level ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL_QUORUM|EACH_QUORUM|SERIAL|LOCAL_SERIAL|LOCAL_ONE**

Specify the consistency level for the destination.

**--driver-compression snappy|lz4**

Specify a compression algorithm for data files.

**--driver-connect-timeout timeout_in_milliseconds**

Specify the timeout for the driver connection.

**--driver-read-timeout timeout_in_milliseconds**

Specify the timeout for the driver reads.

**--driver-max-requests-per-connection number_of_requests**

Specify the maximum number of requests per connection.

**--driver-ssl-enabled true|false**

Enable SSL connection for the destination.

**--driver-ssl-cipher-suites**

Specify the SSL cipher suites to use for driver connections.

**--driver-ssl-protocol**

Specify the SSL protocol to use for driver connections.

**--driver-keystore-path**

Specify the SSL keystore path to use for driver connections.

**--driver-keystore-password**

Specify the SSL keystore password to use for driver connections.

**--driver-keystore-type**

Specify the SSL keystore type to use for driver connections.

**--driver-truststore-path**

---
Specify the SSL truststore path to use for driver connections.

--driver-truststore-password
Specify the SSL truststore password to use for driver connections.

--driver-truststore-type
Specify the SSL truststore type to use for driver connections.

Examples

To update a replication destination:

```
$ dse advrep --verbose destination update --name mydest --addresses 10.200.182.148 --transmission-enabled true
```

with a result:

```
Destination mydest created
```

dse advrep destination create

Creates a replication destination.

A replication channel is a defined channel of change data between source clusters and
destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

Synopsis

```
$ dse advrep destination update
   --name destination_name
   --addresses address_name [ , address_name ]
   [ --transmission-enabled (true|false) ]
   --driver-user user_name
   --driver-pwd password
   --driver-used-hosts-per-remote-dc
   --driver-connections
   --driver-connections-max
   --driver-local-dc
   --driver-allow-remote-dcs-for-local-cl true|false
   --driver-consistency-level [ ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL_QUORUM|EACH QUORUM|SERIAL|LOCAL_SERIAL|LOCAL_ONE ]
   --driver-compression [ snappy|lz4 ]
   --driver-connect-timeout timeout_in_milliseconds
   --driver-read-timeout timeout_in_milliseconds
   --driver-max-requests-per-connection number_of_requests
   --driver-ssl-enabled true|false
   --driver-ssl-cipher-suites
   --driver-ssl-protocol
   --driver-ssl-keystore-path
   --driver-ssl-keystore-password
   --driver-ssl-keystore-type
   --driver-ssl-truststore-path
```
## Table 310: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**--name destination_name (required)**
- Designate the name of the destination.

**--addresses address_name [, address_name ] (required)**
- Identify the IP addresses of the destinations.

**--transmission-enabled true | false**
Specify if data collector for the table should be replicated to the destination.

--driver-user user_name
  Specify the username for the destination.

--driver-pwd password
  Specify the password for the destination.

--driver-used-hosts-per-remote-dc number_of_hosts
  Define the number of hosts per remote datacenter that the datacenter-aware round robin policy considers available for use.

--driver-connections number_of_connections
  Specify the number of connections that the driver creates.

--driver-connections-max number_of_connections
  Specify the maximum number of connections that the driver creates.

--driver-local-dc data_center_name
  Specify the name of the datacenter that is considered local.

--driver-consistency-level ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL_QUORUM|EACH_QUORUM|SERIAL|LOCAL_SERIAL|LOCAL_ONE
  Specify the consistency level for the destination.

--driver-compression snappy|lz4
  Specify a compression algorithm for data files.

--driver-connect-timeout timeout_in_milliseconds
  Specify the timeout for the driver connection.

--driver-read-timeout timeout_in_milliseconds
  Specify the timeout for the driver reads.

--driver-max-requests-per-connection number_of_requests
  Specify the maximum number of requests per connection.

--driver-ssl-enabled true|false
  Enable SSL connection for the destination.

--driver-ssl-cipher-suites
  Specify the SSL cipher suites to use for driver connections.

--driver-ssl-protocol
  Specify the SSL protocol to use for driver connections.

--driver-keystore-path
  Specify the SSL keystore path to use for driver connections.

--driver-keystore-password
  Specify the SSL keystore password to use for driver connections.

--driver-keystore-type
  Specify the SSL keystore type to use for driver connections.

--driver-truststore-path
  Specify the SSL truststore path to use for driver connections.

--driver-truststore-password
  Specify the SSL truststore password to use for driver connections.

--driver-truststore-type
  Specify the SSL truststore type to use for driver connections.

Examples

To create a replication destination:
DataStax Enterprise tools

$ dse advrep --verbose destination update --name mydest --addresses 10.200.182.148 --driver-consistency-level ANY

with a result:

Destination mydest updated
Updated addresses from 10.200.182.148 to 10.200.182.1648
Updated driver_consistency_level from ONE to ANY
Updated name from mydest to mydest

Notice that any option included causes a change to occur.

dse advrep destination delete

Deletes a given replication destination.

A replication channel is a defined channel of change data between source clusters and destination clusters.

Restriction: Command is supported only on nodes configured for DSE Advanced Replication.

Synopsis

$ dse advrep destination delete
   --name destination_name

Table 311: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve uppercase.</td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ( {} ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt;&gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### `--name destination_name (required)`

Designate the name of the destination.

### Examples

#### To delete a replication destination:

```bash
$ dse advrep destination delete --name mydest
```

with a result:

```
Destination mydest removed
```

### dse advrep destination list

Lists all replication destinations.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

### Synopsis

```bash
$ dse advrep destination list
```
### Table 312: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Examples

**To list all replication destinations:**

```
$ dse advrep destination list
```

with a result:
DataStax Enterprise tools

<table>
<thead>
<tr>
<th>name</th>
<th>enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>mydest</td>
<td>true</td>
</tr>
</tbody>
</table>

---

dse advrep destination list-conf

Lists all configuration for a given replication destination.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```bash
$dse advrep destination list-conf
   --separator field_separator
   --name destination_name
```

**Table 313: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>&lt;datatype1,datatype2&gt;</strong></td>
</tr>
<tr>
<td><strong>cql_statement;</strong></td>
</tr>
<tr>
<td><strong>[ -- ]</strong></td>
</tr>
<tr>
<td><strong>' &lt;schema&gt; ... &lt;/schema&gt; '</strong></td>
</tr>
<tr>
<td><strong>@xml_entity=’xml_entity_type’</strong></td>
</tr>
</tbody>
</table>

**--name destination_name (required)**

Designate the name of the destination.

### Examples

#### To list the configuration for a replication destination:

```bash
$ dse advrep destination list-conf --name mydest
```

with a result:

```plaintext
KEYS: ---- [addresses, transmission-enabled, driver-ssl-cipher-suites, driver-ssl-enabled, driver-ssl-protocol, name, driver-connect-timeout, driver-max-requests-per-connection, driver-connections-max, driver-connections, driver-compression, driver-consistency-level, driver-allow-remote-dcs-for-local-cl, driver-used-hosts-per-remote-dc, driver-read-timeout]

<table>
<thead>
<tr>
<th>destination</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mydest</td>
<td>addresses</td>
<td>10.200.180.162</td>
</tr>
<tr>
<td>mydest</td>
<td>transmission-enabled</td>
<td>true</td>
</tr>
<tr>
<td>mydest</td>
<td>driver-ssl-cipher-suites</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256,</td>
</tr>
</tbody>
</table>
```
<p>| TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384, |
| TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384, |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA256, |
| TLS_DHE_DSS_WITH_AES_256_CBC_SHA256, |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA, |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, |
| TLS_RSA_WITH_AES_256_CBC_SHA, |
| TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA, |
| TLS_ECDH_RSA_WITH_AES_256_CBC_SHA, |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA, |
| TLS_DHE_DSS_WITH_AES_256_CBC_SHA, |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256, |
| TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, |
| TLS_DHE_DSS_WITH_AES_128_CBC_SHA256, |
| TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, |
| TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, |
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, |
| TLS_RSA_WITH_AES_256_GCM_SHA384, |
| TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384, |
| TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384, |
| TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, |
| TLS_DHE_DSS_WITH_AES_256_GCM_SHA384, |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, |
| TLS_RSA_WITH_AES_128_GCM_SHA256, |
| TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256, |
| TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256, |
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, |
| TLS_DHE_DSS_WITH_AES_128_GCM_SHA256, |
| TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA, |
| TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA, |
| SSL_RSA_WITH_3DES_EDE_CBC_SHA, |
| TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA, |
| TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA, |
| SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA, |
| SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA, |
| TLS_ECDHE_ECDSA_WITH_RC4_128_SHA, |
| TLS_ECDHE_RSA_WITH_RC4_128_SHA, |
| SSL_RSA_WITH_RC4_128_SHA, |
| TLS_ECDH_ECDSA_WITH_RC4_128_SHA, |
| TLS_ECDH_RSA_WITH_RC4_128_SHA, |
| SSL_RSA_WITH_RC4_128_MD5, |
| TLS_EMPTY_RENEGOTIATION_INFO_SCSV |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver-ssl-enabled</td>
<td>false</td>
</tr>
<tr>
<td>driver-ssl-protocol</td>
<td>TLS</td>
</tr>
<tr>
<td>name</td>
<td>mydest</td>
</tr>
<tr>
<td>driver-connect-timeout</td>
<td>15000</td>
</tr>
<tr>
<td>driver-max-requests-per-connection</td>
<td>1024</td>
</tr>
<tr>
<td>driver-connections-max</td>
<td>8</td>
</tr>
<tr>
<td>driver-connections</td>
<td>1</td>
</tr>
<tr>
<td>driver-compression</td>
<td>lz4</td>
</tr>
<tr>
<td>driver-consistency-level</td>
<td>ONE</td>
</tr>
<tr>
<td>driver-allow-remote-dcs-for-local-cl</td>
<td>false</td>
</tr>
<tr>
<td>driver-used-hosts-per-remote-dc</td>
<td>0</td>
</tr>
<tr>
<td>driver-read-timeout</td>
<td>15000</td>
</tr>
</tbody>
</table>

**dse advrep destination remove-conf**

Removes configuration for a destination.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep destination remove-conf
   --name  destination_name
   --addresses address_name [ , address_name ]
   [ --transmission-enabled (true|false) ]
```
--driver-user user_name
--driver-pwd password
--driver-used-hosts-per-remote-dc
--driver-connections
--driver-connections-max
--driver-local-dc
--driver-allow-remote-dcs-for-local-cl true|false
--driver-consistency-level [ ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL_QUORUM|EACH_QUORUM|SERIAL|LOCAL_SERIAL|LOCAL_ONE ]
--driver-compression [ snappy|lz4 ]
--driver-connect-timeout timeout_in_milliseconds
--driver-read-timeout timeout_in_milliseconds
--driver-max-requests-per-connection number_of_requests
--driver-ssl-enabled true|false
--driver-ssl-cipher-suites
--driver-ssl-protocol
--driver-ssl-keystore-path
--driver-ssl-keystore-password
--driver-ssl-keystore-type
--driver-ssl-truststore-path
--driver-ssl-truststore-password
--driver-ssl-truststore-type

Table 314: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeateable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key: value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( <code>&lt; &gt;</code> ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( <code>;</code> ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( <code>--</code> ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &amp;lt;schema&amp;gt; ... &amp;lt;/schema&amp;gt;</code></td>
<td>Search CQL only: Single quotation marks ( <code>&amp;apos;</code> ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--name destination_name (required)
Designate the name of the destination.

--addresses address_name [, address_name ] (required)
Identify the IP addresses of the destinations.

--transmission-enabled true | false
Specify if data collector for the table should be replicated to the destination.

--driver-user user_name
Specify the username for the destination.

--driver-pwd password
Specify the password for the destination.

--driver-used-hosts-per-remote-dc number_of_hosts
Define the number of hosts per remote datacenter that the datacenter-aware round robin policy considers available for use.

--driver-connections number_of_connections
Specify the number of connections that the driver creates.

--driver-connections-max number_of_connections
Specify the maximum number of connections that the driver creates.

--driver-local-dc data_center_name
Specify the name of the datacenter that is considered local.

--driver-consistency-level ANY|ONE|TWO|THREE|QUORUM|ALL|LOCAL_QUORUM|EACH_QUORUM|SERIAL|LOCAL_SERIAL|LOCAL_ONE
Specify the consistency level for the destination.

--driver-compression snappy|lz4
Specify a compression algorithm for data files.

--driver-connect-timeout timeout_in_milliseconds
Specify the timeout for the driver connection.

--driver-read-timeout timeout_in_milliseconds
Specify the timeout for the driver reads.

--driver-max-requests-per-connection number_of_requests
Specify the maximum number of requests per connection.

--driver-ssl-enabled true|false
Enable SSL connection for the destination.

--driver-ssl-cipher-suites
Specify the SSL cipher suites to use for driver connections.

--driver-ssl-protocol
Specify the SSL protocol to use for driver connections.

--driver-keystore-path
Specify the SSL keystore path to use for driver connections.

--driver-keystore-password
Specify the SSL keystore password to use for driver connections.

--driver-keystore-type
Specify the SSL keystore type to use for driver connections.

--driver-truststore-path
Specify the SSL truststore path to use for driver connections.

--driver-truststore-password
Specify the SSL truststore password to use for driver connections.

--driver-truststore-type
Specify the SSL truststore type to use for driver connections.

Examples

To remove configuration for a replication destination:

$ dse advrep --verbose destination remove-conf --transmission-enabled true

with a result:

Removed config transmission-enabled

dse advrep metrics list

Lists advanced replication JMX metrics.

A replication channel is a defined channel of change data between source clusters and destination clusters.

Restriction: Command is supported only on nodes configured for DSE Advanced Replication.

Synopsis

$ dse advrep metrics list
   --metric group metric_group
   --metric-type metric_type

Table 315: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (( )) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (' ') marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**--metric group metric_group**

The source cluster keyspace for which to show count.

**--metric-type metric_type**

The source table for which to show count.

**Examples**

To display the JMX metrics:

```
$ dse advrep --host localhost --port 7199 metrics list
```
To display JMX metrics for a particular metric group:

```
$ dse advrep --host localhost --port 7199 metrics list --metric-group Tables
```

with a result:

---
<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>MessagesDelivered</td>
<td>3000</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>CommitLogsToConsume</td>
<td>1</td>
</tr>
<tr>
<td>Tables</td>
<td>MessagesReceived</td>
<td>3000</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessageAddErrors</td>
<td>0</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>CommitLogsDeleted</td>
<td>0</td>
</tr>
</tbody>
</table>
---
<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Count</th>
<th>RateUnit</th>
<th>MeanRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReplicationLog</td>
<td>MessagesAdded</td>
<td>3000</td>
<td>events/second</td>
<td>0.020790529428089743</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesDeleted</td>
<td>0</td>
<td>events/second</td>
<td>0.0</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesAcknowledged</td>
<td>3000</td>
<td>events/second</td>
<td>0.020790529428089743</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>CommitLogMessagesRead</td>
<td>30740</td>
<td>events/second</td>
<td>0.21303361656215317</td>
</tr>
</tbody>
</table>
---
<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission</td>
<td>AvailablePermits</td>
<td>30000</td>
</tr>
</tbody>
</table>
---
To display JMX metrics for a particular metric type:

```
$ dse advrep --host localhost --port 7199 metrics list --metric-type MessagesAdded
```

with a result:

```
<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Count</th>
<th>RateUnit</th>
<th>MeanRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>FifteenMinuteRate</td>
<td>OneMinuteRate</td>
<td></td>
<td>MeanRate</td>
<td></td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesAdded</td>
<td>3000</td>
<td>events/second</td>
<td>0.020827685267120057</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.100068258619765E-28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.964393875E-314</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.515866021410421E-82</td>
</tr>
</tbody>
</table>
```

dse advrep replog count

Returns the messages that have not been replicated.

A replication channel is a defined channel of change data between source clusters and destination clusters.

**Restriction:** Command is supported only on nodes configured for DSE Advanced Replication.

**Synopsis**

```
$ dse advrep replog count
    --source-keyspace keyspace_name
    --source-table source_table_name
    --destination destination
    --data-center-id data_center_id
```

**Table 316: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Examples

#### To verify the record count held in a replication log:

```bash
$ dse advrep replog count --destination mydest --source-keyspace foo --source-table bar
```

with a result:
dse advrep replog analyze-audit-log

Reads the audit log and prints a summary.

A replication channel is a defined channel of change data between source clusters and destination clusters.

Restriction: Command is supported only on nodes configured for DSE Advanced Replication.

Synopsis

```
$ dse advrep replog analyze-audit-log
   --file audit_log_filename
```

Table 317: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve uppercase.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt; &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity=' xml_entity_type '</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

---

---

**--file audit_log_filename**

The audit log file to create.

**Examples**

**To analyze the data in a replication log:**

```
$ dse advrep replog analyze-audit-log --file auditLog
```

with a result:

```makefile
foo, bar : inserts = 1000, insertErrors = 0
foo, bar : reads = 1000, sent = 0, deletes = 1000, readingErrors = 0, deletingErrors = 0
```

---

**dse beeline**

Starts the Beeline *(page 265)* shell.

**Restriction:** Command is supported only on nodes with analytics workloads.

**Synopsis**

```
$ dse beeline
```

This command takes no arguments.

**dse cassandra**

Starts the database in transactional mode. Command options start the database in other modes and enable advanced features on a node. See *Starting DataStax Enterprise (page 1440)*.

To change the DSE system properties on start up, see *Setting system properties during startup (page 158)*.

**Synopsis**

```
dse cassandra [-k] [-s] [-g]
```
Note: When multiple flags are used, list them separately on the command line. For example, ensure there is a space between `-k` and `-s` in `dse cassandra -k -s`.

Table 318: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>' Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Options

-k
DataStax Enterprise tools

Start the node in analytics mode. The first time the node starts up the analytics workload type is configured.

-g
Start the node in graph mode. The first time the node starts up the graph workload type is configured.

-s
Start the node in search mode. The first time the node starts up the search workload type is configured.

-E
Change JVM error file.

-f
Start a real-time transactional node in the foreground.

-h
Display the usage and listing of the dse commands.

-H
Change JVM HeapDumpPath.

-p pidfilepath
Create the pid file. The pid file is typically used by monitoring processes and init scripts. Not compatible with -f option.

Examples

Start a node in transactional mode

$ dse cassandra

In the foreground, start a node in transactional mode

$ dse cassandra -f

Start a node in DSE Analytics mode

$ dse cassandra -k

Start a node in SearchAnalytics mode

$ dse cassandra -k -s

Ensure there is a space between -k and -s in dse cassandra -k -s.

Start a node in DSE Analytics, DSE Graph, and DSE Search modes

$ dse cassandra -k -g -s

Ensure there is a space between -k, -g, and -s in dse cassandra -k -g -s.
**Start a node in DSE Search mode and change the location of the search index data on the server**

$ dse cassandra -s -Ddse.solr.data.dir=filepath

See [Managing the location of DSE Search data](#).

**Start a node in transactional mode without joining the ring**

$ dse cassandra -Dcassandra.join_ring=false

**Start a node in transactional mode to test compaction and compression strategies**

$ dse cassandra -Dcassandra.write_survey=true

Experiment with different strategies and benchmark write performance differences without affecting the production workload. See [Testing compaction and compression](#).

**Start a node in transactional mode and pass the dead node IP address**

$ dse cassandra -Dcassandra.replace_address=10.91.176.160

**Start a node in transactional mode and create pid.txt**

$ dse cassandra -p pid.txt

dse cassandra-stop

Stops the DataStax Enterprise process.

See [Stopping a node](#).

**Synopsis**

$ cassandra-stop -p pid

**Table 319: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>&quot;Literal string&quot;</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**pid**  
DataStax Enterprise (cassandra) process id.

### Examples

**Stop by process id**

```bash
cassandra-stop -p 41234
```

**dse exec**

Sets the environment variables required to run third-party tools that integrate with Spark:

- SPARK_HOME to point to the DSE Spark directory
- HADOOP_CONF_DIR to point to the Hadoop configuration directory within DSE
- Sets other environment variables required by DSE Spark to enable custom DSE
- Executes the given shell command
This command is typically used for third-party tools that integrate with Spark (page 214).

Synopsis

$ exec [-cl] [-a name] [command [arguments ...]] [redirection ...]

Table 320: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
Examples
See Using DSE Spark with third party tools and integrations (page 214).

dse fs

Starts the DSE File System (DSEFS). The DSEFS prompt shows the current working directory, which is the default DSEFS search directory.

See DSEFS (DataStax Enterprise file system) (page 285).

Synopsis

```
$ dse fs
[--prefer-contact-points -h IP_address1,IP_address2,...]
[dsefs_shell_command [command_options]]
```

**dsefs_shell_command**

The dse fs command supports all dsefs commands (page 1269).

**--prefer-contact-points -h IP_address1,IP_address2,...**

Give precedence to the specified hosts, regardless of proximity, when issuing DSEFS commands. As long as the specified hosts are available, DSEFS will not switch to other DSEFS nodes in the cluster.

Without these options, DSEFS switches to the closest available DSEFS node.

Examples

Start DSEFS

```
$ dse fs
```

Connected to DataStax Enterprise File System 6.7.0 at DSE cluster Test Cluster
Type help to get the list of available commands.
dsefs dsefs://127.0.0.1:5598/ >

DSEFS starts on the closest available DSEFS node.

Start DSEFS

```
$ dse fs 10.0.0.2,10.0.0.5
```

Connected to DataStax Enterprise File System 6.7.0 at DSE cluster Test Cluster
Type help to get the list of available commands.
dsefs dsefs://127.0.0.1:5598/ >

DSEFS starts with precedence to the specified hosts, regardless of proximity.
See DSEFS (DataStax Enterprise file system) *(page 285).*

**dse gremlin-console**

DSE Gremlin Console automatically connects at startup to DataStax Enterprise (DSE) server, as configured in the `remote.yaml` *(page 149)* file. Override the configured host and port from the command line.

**Synopsis**

\[
\]$ \text{dse } [-u \text{ username } [-p \text{ password}]] \text{ gremlin-console } [\text{hostname}[:\text{port}]] \text{[options]}
\]

**Table 321: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

## DSE connection parameters

When starting the Gremlin Console using this command it automatically connects to the host specified in the `remote.yaml (page 149).`

- **-u username**
  - When DataStax Enterprise Authentication is enabled use this option to login to the database.
  
  **Tip:** Set the user name in a file or as an environment variable.

- **-p password**
  - Optional password for DSE authentication. If omitted when a user name is specified, the password prompt appears.
  
  **Tip:** Set the password in a file or as an environment variable.

- **hostname**
  - The hostname of the DataStax Enterprise to which the console connects. Overrides the setting in the `remote.yaml (page 149).`

- **port**
  - Port number of the DataStax Enterprise database port, default is 9042. Overrides the setting in the `remote.yaml (page 149).`

## Options

Gremlin console options.

- **-C, --color**
  - Disable use of ANSI colors.

- **-D, --debug**
  - Enabled debug console output.

- **-Q, --quiet**
  - Suppress superfluous console output.

- **-V, --verbose**
  - Enable verbose Console output

- **-e, --execute=SCRIPT_NAME [ARG1 ARG2 ...]**
  - Execute the specified script and close the console on completion.

- **-h, --help**
  - Display this help message.

- **-i, --interactive=SCRIPT_NAME [ARG1 ARG2 ...]**
  - Execute the specified script and leave the console open on completion.

- **-l**
  - Set the logging level of components that use standard logging output independent of the Console.

- **-v, --version**
DataStax Enterprise tools

Display the version.

**dse list-nodes**

Lists the nodes that are configured for the DSE Multi-Instance host machine.

Since the default DataStax Enterprise node is called dse, the `dse list-nodes` command always returns at least the dse node, even if nodes were not added with the `dse add-node` command.

**Restriction:** DSE Multi-Instance commands are supported only on package installations.

Synopsis

```
$ dse list-nodes
```

**Table 322: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

This command takes no arguments and lists the nodes that are configured for the DSE Multi-Instance host machine.

Examples

**List the nodes**

$ dse list-nodes

dse pyspark

Starts the Spark Python shell.

See the DataFrames documentation (page 253) for an example of using PySpark, and the PySpark API documentation.

**Synopsis**

$ dse spark pyspark

**Table 323: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Literal string'</td>
<td>Single quotation (‘’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (‘’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

This command takes no arguments.

**dse remove-node**

Removes a node that is configured for the DSE Multi-Instance host machine.

**Important:** The user running the command must have permissions for writing to the directories that DSE uses, or use sudo.

**Restriction:** DSE Multi-Instance commands are supported only on package installations.

**Synopsis**

```sh
$ dse remove-node nodeId [--yes]
```

**Table 324: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th><strong>Syntax</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity=’xml_entity_type’</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Examples

**Remove the node payrollnode**

```
$ dse remove-node payrollnode
```

or the equivalent command with the prepended `dse-`:

```
$ dse remove-node dse-payrollnode
```

The prompt for node deletion is displayed:

```
##############################
```
Remove the node dse-payrollnode with explicit confirmation

$ dse remove-node dse-payrollnode --yes

dse spark

Enters interactive Spark shell and offers basic auto-completion.

**Restriction:** Command is supported only on nodes with analytics workloads.

For details on using Spark with DSE, see:

- Accessing database data from Spark *(page 190)*
- BYOS (Bring Your Own Spark) *(page 266)*
- Importing graphs using DseGraphFrame *(page 632)*
- Starting Spark *(page 185)*

**Synopsis**

$ dse connection_options spark
[-framework dse|spark-2.0] [--help] [--verbose]
[--conf name=spark.value|sparkproperties.conf]
[--executor-memory mem]
[--jars additional-jars]
[--master dse://?appReconnectionTimeoutSeconds=secs]
[--properties-file path_to_properties_file]
[--total-executor-cores cores]
[-i app_script_file]

**Table 325: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

In general, Spark submission arguments (--submission_args) are translated into system properties -Dname=value and other VM parameters like classpath. The application arguments (-app_args) are passed directly to the application.

Configure the Spark shell with these arguments:

```
--conf name=spark.value|sparkproperties.conf
    An arbitrary Spark option to the Spark configuration prefixed by spark.
    • name-spark.value
    • sparkproperties.conf - a configuration

--executor-memory mem
    The amount of memory that each executor can consume for the application. Spark uses a 512 MB default. Specify the memory argument in JVM format using the k, m, or g suffix.

-framework dse|spark-2.0
    The classpath for the Spark shell. When not set, the default is dse.
```
• dse - Sets the Spark classpath to the same classpath that is used by the DSE server.

• spark-2.0 - Sets a classpath that is used by the open source Spark (OSS) 2.0 release to accommodate applications originally written for open source Apache Spark. Uses a BYOS (Bring Your Own Spark) JAR with shaded references to internal dependencies to eliminate complexity when porting an app from OSS Spark.

  **Note:** If the code works on DSE, applications do not require the spark-2.0 framework. Full support in the spark-2.0 framework might require specifying additional dependencies. For example: hadoop-aws is included on the dse server path but is not present on the OSS Spark-2.0 classpath. In this example, applications that use S3 or other AWS APIs must include their own aws-sdk on the runtime classpath. This additional runtime classpath is required only for applications that cannot run on the DSE classpath.

  --help
  
  Shows a help message that displays all options except DataStax Enterprise Spark shell options.

  -i app_script_file
  
  Spark shell application argument that runs a script from the specified file.

  --jars path_to_additional_jars
  
  A comma-separated list of paths to additional JAR files.

  --master dse://?appReconnectionTimeoutSeconds=secs
  
  A custom timeout value when submitting the application, useful for troubleshooting Spark application failures. The default timeout value is 5 seconds.

  --properties-file path_to_properties_file
  
  The location of the properties file that has the configuration settings. By default, Spark loads the settings from spark-defaults.conf.

  --total-executor-cores cores
  
  The total number of cores the application uses.

  --verbose
  
  Displays which arguments are recognized as Spark configuration options and which arguments are forwarded to the Spark shell.

Examples

Start the Spark shell

$ dse spark

Start the Spark shell with case-sensitivity

DseGraphFrame and Spark SQL are case insensitive by default. Column names that differ only in case will result in conflicts. The Spark property `spark.sql.caseSensitive=true` avoids case conflicts.
$ dse spark --conf spark.sql.caseSensitive=true

**Set the timeout value to 10 seconds**

$ dse spark --master dse://?appReconnectionTimeoutSeconds=10

Useful for troubleshooting, see [Detecting Spark application failures (page 189)](#).

### dse spark-jobserver

Starts and stops the Spark Jobserver that is bundled with DSE.

**Restriction:** Command is supported only on nodes with analytics workloads.

See [Spark Jobserver (page 269)](#).

**Synopsis**

```
$ dse spark-jobserver start
[--properties-file path_to_properties_file]
[--executor-memory memory] [--total-executor-cores cores]
[--conf name=spark.value] [--jars path_to_additional_jars]
[--help] [--verbose] | stop
```

<table>
<thead>
<tr>
<th><strong>Syntax conventions</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( <code> </code> ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>&lt;datatype1, datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( <code>&lt;</code> ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( <code>--</code> ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks ( <code>‘</code> ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**start**

Starts the Spark Jobserver.

**help**

Displays options and usage instructions. Use `nodesync help subcommand` for more information on a specific command.

**--verbose**

Displays which arguments are recognized as Spark configuration options and which arguments are forwarded to the Spark shell.

**stop**

Stops the Spark Jobserver.

For the `dse spark-jobserver start` command, apply one or more valid `spark-submit` options.

**--properties-file path_to_properties_file**

The location of the properties file that has the configuration settings. By default, Spark loads the settings from `spark-defaults.conf`.

**--executor-memory mem**

The amount of memory that each executor can consume for the application. Spark uses a 512 MB default. Specify the memory argument in JVM format using the k, m, or g suffix.

**--total-executor-cores cores**

The total number of cores the application uses.

**--conf name=spark.value|sparkproperties.conf**

An arbitrary Spark option to the Spark configuration prefixed by `spark`.

- name-spark.value
- sparkproperties.conf - a configuration

**--jars path_to_additional_jars**

A comma-separated list of paths to additional JAR files.
Examples

**Start the Spark Jobserver without submit options**

dse spark-jobserver start

**Start the Spark Jobserver with submit option**

dse spark-jobserver start --properties-file spark.conf

See spark-submit options *(page 235)*.

**Stop the Spark Jobserver**

dse spark-jobserver stop

**dse spark-history-server**

Starts and stops the Spark history server, the front-end application that displays logging data from all nodes in the Spark cluster.

**Restriction:** Configuration is required for the Spark history server. See Spark history server *(page 234)*.

**Synopsis**

```
$ dse spark-history-server
   start [--properties-file properties_file] stop
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { ) ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### start

Starts the Spark history server to load the event logs from Spark jobs that were run with event logging enabled. The Spark history server can be started from any node in the cluster.

---

**--properties-file properties_file**

The properties file to overwrite the default Spark configuration in `conf/spark-defaults.conf`. The properties file can include settings like the authentication method and credentials and event log location.

---

### stop

Stops the Spark history server.

### Examples

#### Start the Spark history server on the local node

```bash
dse spark-history-server start
```

The Spark history server is started with the default configuration in `conf/spark-defaults.conf`.

#### Start the Spark history server with a properties file

```bash
dse spark-history-server start --properties-file sparkproperties.conf
```
The Spark history server is started with the configuration specified in `sparkproperties.conf`.

**dse spark-sql**

Starts the Spark SQL shell in DSE to interactively perform Spark SQL queries.

The Spark SQL shell in DSE automatically creates a Spark session and connects to the Spark SQL Thrift server (page 254) to handle the underlying JDBC connections. See Using Spark SQL to query data (page 244).

**Synopsis**

```
$ dse spark-sql
```

**Table 328: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks (<code>'</code>) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

This command accepts no parameters.

Examples

Start the Spark SQL shell

```bash
$ dse spark-sql
```

The log file is at `/home/ubuntu/.spark-sql-shell.log`

```bash
spark-sql>
```

At the spark-sql prompt, you can interactively perform Spark SQL queries.

**dse spark-sql-thriftserver**

Starts and stops the Spark SQL Thriftserver. The Spark SQL Server uses a JDBC and an ODBC interface for client connections to DSE.

Configuration is required for the Spark SQL Thriftserver. See [Using the Spark SQL Thriftserver (page 254)](#).

Synopsis

```bash
$ dse spark-sql-thriftserver start [--conf spark_prop] [--hiveconf hive_prop] | stop
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax conventions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
<td>...</td>
</tr>
<tr>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
<td>'Literal string'</td>
</tr>
<tr>
<td>Map collection. Braces ( { ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
<td>{ key:value }</td>
</tr>
<tr>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
<td>&lt;datatype1,datatype2&gt;</td>
</tr>
<tr>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
<td>cql_statement;</td>
</tr>
<tr>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
<td>[ -- ]</td>
</tr>
<tr>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
</tr>
<tr>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
<td>@xml_entity='xml_entity_type'</td>
</tr>
</tbody>
</table>

### Examples

**Start**

Starts the Spark SQL Thriftserver. The user who runs the command to start the Spark SQL Thriftserver requires permissions to write to the Spark directories.

```
$ dse spark-sql-thriftserver start
```

**--conf spark_prop**

Pass in general Spark configuration settings, like spark.cores.max=4.

**-hiveconf config_file**

Pass in a hive configuration property, like hive.server2.thrift.port=10001.

**Stop**

Stops the Spark SQL Thriftserver.

**Examples**

**Start the Spark SQL Thriftserver with default Spark and Hive options**

```
$ dse spark-sql-thriftserver start
```

**Start the Spark SQL Thriftserver with a Spark configuration option**

```
$ dse spark-sql-thrift-server start --conf spark.cores.max=4
```

**Start the Spark SQL Thriftserver with a Hive configuration option**
Stop the Spark SQL Thriftserver

```
$ dse spark-sql-thriftserver stop
```

**dse spark-submit**

Launches applications on a cluster to enable use of Spark cluster managers through a uniform interface. This command supports the same options as Apache Spark `spark-submit`.

**Restriction:** Command is supported only on nodes with analytics workloads.

**Synopsis**

```
$ dse spark-submit
--class class_name
jar_file other_options|
--status|--kill driver_id [--master master_ip_address]
```

**Table 330: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>'&lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

This command supports the same options as Apache Spark spark-submit. Unlike the standard behavior for the Spark status and kill options, in DSE deployments these options do not require the Spark Master IP address.

**kill** **driver_id**
Kill a Spark application running in the DSE cluster.

**master** **master_ip_address**
The IP address of the Spark Master running in the DSE cluster.

**status** **driver_id**
Get the status of a Spark application running in the DSE cluster.

**Examples**

Run the HTTP response example program (located in the dse-demos directory) on two nodes:

```bash
$ dse spark-submit --class com.datastax.HttpSparkStream target/HttpSparkStream.jar -d 2
```

To submit an application using cluster mode using the supervise option to restart in case of failure:

```bash
$ dse spark-submit --deploy-mode cluster --supervise --class com.datastax.HttpSparkStream target/HttpSparkStream.jar -d $NUM_SPARK_NODES
```

To submit an application using cluster mode when TLS is enabled

Pass the SSL configuration with standard Spark commands to use secure HTTPS on port 4440.

```bash
$ dse spark-submit --conf spark.ssl.ui.enabled=true --conf spark.ssl.ui.keyPassword=ctool_keystore --conf spark.ssl.ui.keyStore=/home/automaton/ctool_security/ctool_keystore
```
To set the driver host to a publicly accessible IP address

```
$ dse spark-submit --conf spark.driver.host=203.0.113.0 myApplication.jar
```

To get the status of a driver

Unlike the Apache Spark option, you do not have to specify the Spark Master IP address.

```
$ dse spark-submit --status driver-20180726160353-0019
```

Result when the driver exists:

```
Driver driver-20180726160353-0019 found: state=<state>, worker=<workerId> (<workerHostPort>)
```

To kill a driver

Unlike the Apache Spark option, you do not have to specify the Spark Master IP address.

```
$ dse spark-submit --kill driver-20180726160353-0019
```

dse SparkR

Starts the R shell configured with DSE Spark to automatically set the Spark session within R. See Using SparkR with DataStax Enterprise (page 256).

Restriction: Command is supported only on nodes with analytics workloads.

Synopsis

```
$ dse SparkR
```

Table 331: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (( )) identify a group to choose from. Do not type the parentheses.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>&lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

This command accepts no parameters.

Examples

**Starts the R shell configured with DSE Spark**

```
$ dse sparkR
```

**dse -v**

Sends the DataStax Enterprise version number to standard output.

This command does not require authentication.
Synopsis

$ dse -v

dse client-tool

About dse client-tool

The dse client-tool command line interface connects an external client to a DataStax Enterprise node and performs common utility tasks.

Using dse client-tool command line help

To show a listing of the dse client-tool subcommands:

$ dse client-tool help

To show the command line help for a specific dse client-tool subcommand:

$ dse client-tool help subcommand

For example:

$ dse client-tool help spark

dse client-tool connection options

You must authenticate connections to an external client for dse client-tool commands.

JMX authentication is supported by some dsetool commands. Other dsetool commands authenticate with the user name and password of the configured user. The connection option short form and long form are comma separated.

Note:

You can provide authentication credentials in several ways, see Credentials for authentication.

To enable dsetool to use Kerberos authentication, see Using dsetool with Kerberos enabled cluster.

Different sources of configuration properties are used to connect external clients to a DSE node: DSE configuration in dse.yaml and cassandra.yaml.

Note: You can provide authentication credentials in several ways, see Credentials for authentication. The dse client-tool subcommands use DSE Unified Authentication, like the Java and other language drivers, not JMX authentication like dsetool.
RPC permissions over the native protocol leverage DSE authentication and role-based access abilities. To configure external client access to DataStax Enterprise commands, see Authorizing remote procedure calls (RPC).

DSE proxy authentication can be used with dse client-tool, and delegation tokens can be generated for the proxy authenticated role. If the role alice is authenticated, and alice uses proxy authorization to the role bob, alice’s delegation token can be used authenticate as alice and authorize as bob. If bob loses login permissions, the token can still be used to login as alice, because the token reflects alice’s authentication. If alice loses authorization permissions for bob, the token cannot be used to login.

Synopsis

```bash
$ dse client-tool
[--a proxy_auth_username] [-u username] [-p password]
[--port port] [--host hostname]
[--sasl-protocol-name dse_service_principal]
[--keystore-path ssl_keystore_path]
[--keystore-password keystore_password]
[--keystore-type ssl_keystore_type]
[--truststore-path ssl_truststore_path]
[--truststore-password ssl_truststore_password]
[--truststore-type ssl_truststore_type]
[--cipher-suites ssl_cipher_suites]
[--kerberos-enabled (true | false)]
[--ssl-enabled (true | false)]
[--use-server-config]
[--t delegation token]
[--ssl-protocol ssl_protocol]
command [options]
```

Table 332: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (’’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--cipher-suites ssl_cipher_suites
Specify comma-separated list of SSL cipher suites for connection to DSE when SSL is enabled. For example, --cipher-suites c1,c2,c3.

--host hostname
The DSE node hostname or IP address.

--kerberos-enabled true | false
Whether Kerberos authentication is enabled for connections to DSE. For example, --kerberos-enabled true.

--keystore-password keystore_password
Keystore password for connection to DSE when SSL client authentication is enabled.

--keystore-path ssl_keystore_path
Path to the keystore for connection to DSE when SSL client authentication is enabled.

--keystore-type ssl_keystore_type
Keystore type for connection to DSE when SSL client authentication is enabled. JKS is the type for keys generated by the Java keytool binary, but other types are possible, depending on user environment.

-p password
The password to authenticate for database access. Can use the DSE_PASSWORD environment variable.

--port port
The native protocol RPC connection port (Thrift).

--sasl-protocol-name dse_service_principal
SASL protocol name, that is, the DSE service principal name.

--ssl
Whether SSL is enabled for connection to DSE. `--ssl-enabled true` is the same as `--ssl`.

`--ssl-protocol ssl_protocol`
SSL protocol for connection to DSE when SSL is enabled. For example, `--ssl-protocol ssl4`.

`-t token`
Specify delegation token which can be used to login, or alternatively, DSE_TOKEN environment variable can be used.

`--truststore_password ssl_truststore_password`
Truststore password to use for connection to DSE when SSL is enabled.

`--truststore_path ssl_truststore_path`
Path to the truststore to use for connection to DSE when SSL is enabled. For example, `--truststore-path /path/to/ts`.

`--truststore-type ssl_truststore_type`
Truststore type for connection to DSE when SSL is enabled. JKS is the type for keys generated by the Java keytool binary, but other types are possible, depending on user environment. For example, `--truststore-type jks2`.

`-u username`
User name of a DSE authentication account. Can use the DSE_USERNAME environment variable.

`-a proxy_auth_username`
DSE authorization username if proxy authentication is used.

`--use-server-config`
Read parameters from server yaml configuration files. It assumes this node is properly configured.

**dse client-tool cassandra**

Performs token management and partitioner discovery.

**Restriction:** Token management commands require Kerberos authentication mode.

**Synopsis**

```
dse connection_options client-tool cassandra
  (cancel-token token |
  generate-token [username] |
  renew-token token |
  partitioner)
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Commands

- **cancel-token token**
  Cancel the specified token.

- **generate-token [username]**
  Generate delegation token (page 218) to access Kerberos DSE from non-Kerberos clusters.
  - When the username is not specified, the current user is the token renewer. Only DSE processes can renew a token.
  - When the username is specified as the token renewer, that user can renew and cancel the token.

- **partitioner**
  Returns the partitioner that is being used by the node.

- **renew-token token**
  Renew the specified token.
Examples

**Generate token for the current user**

```$ dse client-tool cassandra --generate-token```

**Generate token with user AdminAlicia as the token renewer**

```dse client-tool cassandra generate-token --token-renewer AdminAlicia```

**Return the current partitioner**

```dse client-tool cassandra partitioner```

**Cancel specified token**

```dse client-tool cassandra cancel-token token```

**Renew specified token**

```dse client-tool cassandra renew-token token```

### dse client-tool configuration export

Exports the DataStax Enterprise client configuration from a remote node.

To run Spark commands against a remote cluster, you must copy the exported file from the remote node to the local client machine.

**Synopsis**

```dse client-tool connection_options configuration export filename```

#### Table 334: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Examples

**To export the DataStax Enterprise client configuration from the remote node:**

```bash
dse client-tool configuration export dse-config.jar
```

### dse client-tool configuration byos-export

Exports the DSE node configuration to a Spark-compatible file that can be copied to a node in the external Spark cluster and used with the Spark shell.

See [Generating the BYOS configuration file](page 266).

### Synopsis

```
dse client-tool connection_options configuration byos-export
[--default-properties path_to_existing_properties_file]
```
DataStax Enterprise tools

[--export-credentials]
[--generate-token [--token-renewer username]]

Table 335: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--default-properties spark_propfile_path dse_spark_propfile_path
The path to the default Spark properties file and the DataStax Enterprise Spark properties file to merge properties from both.

--export-credentials
Store current DSE user and password in the generated configuration file.

file
The file name for the generated Spark-compatible file. For example, byos.properties.

`--generate-token`  
Generates digest authentication token to support access to DSE clusters secured with Kerberos from non-Kerberos clusters.

`--set-keystore-password password`  
The keystore password for connection to the database when SSL client authentication is enabled.

`--set-keystore-path path`  
The path to the SSL keystore when SSL client authentication is enabled. All nodes must store the keystore in the same location.

`--set-keystore-type type`  
The keystore type when SSL client authentication is enabled. If not specified, the default is JKS.

`--set-truststore-password password`  
Include the specified truststore password in the configuration file.

`--set-truststore-path path`  
The path to SSL truststore on Spark nodes. All nodes must store the truststore in the same location.

`--set-truststore-type type`  
The truststore type when SSL client authentication is enabled. If not specified, the default is JKS.

`--token-renewer userid`  
User with permission to renew or cancel the token. When not specified, only the DSE process can renew the generated token.

Examples

You can export the DSE node configuration to a Spark-compatible file with various options.

**Generate the byos.properties file in your home directory**

```
dse client-tool configuration byos-export ~/byos.properties
```

**Merge the default Spark properties with the DSE Spark properties**

```
dse client-tool configuration byos-export --default-properties /usr/lib/spark/conf/spark-defaults.conf /home/user1/.dse/byos.conf
```

**dse client-tool configuration import**

Imports configuration file and generates local configuration files and a cqlshrc file with settings from the imported file so the DSE client applications can remotely access the running DSE cluster.

Run this command on a client node to set up the local DSE installation for integrated client applications.
Synopsis

dse client-tool connection_options configuration import file
[--cqlshrc [file]]
[--force]

Table 336: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

--cqlshrc

Generate a cqlshrc file for the DSE client node.

file
Path to cqlshrc file to be generated. When a file is not specified, the default file is the ~/.cassandra/cqlshrc file.

--force
Force an overwrite of existing configuration files. By default, the import command fails if the configuration files already exist.

--set-keystore-password password
The keystore password for connection to the database when SSL client authentication is enabled.

--set-keystore-path path
The path to the SSL keystore when SSL client authentication is enabled. All nodes must store the keystore in the same location.

--set-keystore-type type
The keystore type when SSL client authentication is enabled. If not specified, the default is JKS.

--set-truststore-password password
Include the specified truststore password in the configuration file.

--set-truststore-path path
Path to SSL truststore on Spark nodes. All nodes must store the truststore in the same location.

--set-truststore-type type
The truststore type when SSL client authentication is enabled. If not specified, the default is JKS.

Examples

Run the import command on the client node.

**Import the configuration file with default values:**

```bash
dse client-tool configuration import dse-config.jar
```

**Create a local cqlshrc file with the default name:**

```bash
dse client-tool configuration import dse-config.jar --cqlshrc
```

**Force an overwrite of the existing configuration file:**

```bash
dse client-tool configuration import dse-config.jar --force
```

---

**dse client-tool spark**

Perform operations related to integrated Spark.

**Synopsis**

```bash
dse client-tool connection_options spark
(master-address | leader-address | version | sql-schema (--exclude | --keyspace | --table | --decimal | --all)
```
### Table 337: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**leader-address**

Returns the IP address of the currently selected Spark Master for the datacenter.

**master-address**

Returns the localhost IP address used to configure Spark applications. The address is returned as URI:

```plaintext
dse://ip:port?
connection.local_dc=dc_name;connection.host=cs_list_contactpoints;
```
The `connection.host=cs_list_contactpoints` option is a comma separated list of IP addresses of additional contact points. The additional contact points are up to five randomly selected nodes from the datacenter.

**Note:** DSE automatically connects Spark applications to the Spark Master. You do not need to use the IP address of the current Spark Master in the connection URI.

### metastore-migrate --from_version --to_version
Migrate Spark SQL metastore from one DSE version to another DSE version.
- --from_version - the version to migrate metastore from
- --to_version - the version to migrate metastore to

### version
Returns the version of Spark that is bundled with DataStax Enterprise.

### sql-schema (--exclude | --keyspace | --table | --decimal | --all)
Exports the SQL table creation query with these options:
- --table tablename - comma-separated list of tables to include
- --exclude csvlist - comma-separated list of tables to exclude
- --all - includes all keyspaces
- --keyspace csvlist - comma-separated list of keyspaces to include

### Examples

**View the Spark connection URL for this datacenter:**

```
$ dse client-tool spark master-address
dse://10.200.181.62:9042?
connection.local_dc=Analytics;connection.host=10.200.181.63;
```

**View the IP address of the current Spark Master in this datacenter:**

```
$ dse client-tool spark leader-address 10.200.181.62
```

**Generate Spark SQL schema files**

You can use the generated schema files with Spark SQL on external Spark clusters.

```
$ dse client-tool --use-server-config spark sql-schema --all > output.sql
```

**Migrate Spark metastore**

To map custom external tables from DSE 5.0.11 to the DSE 6.7.0 release format of the Hive metastore used by Spark SQL after upgrading:
dse client-tool spark metastore-migrate --from 5.0.11 --to 6.7.0

**dse client-tool alwayson-sql**

Perform operations related to AlwaysOn SQL.

**Synopsis**

```
dse client-tool connection_options alwayson-sql (status | stop | start | restart | reconfig)
```

**Table 338: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>|</td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[-- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@xml_entity=&quot;xml_entity_type&quot;</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### status

Get the AlwaysOn SQL service status of the datacenter. With the `--dc datacenter name` option, get the status of the specified datacenter.

The returned status is one of:

- **RUNNING**: the server is running and ready to accept client requests.
- **STOPPED_AUTO_RESTART**: the server is being started but is not yet ready to accept client requests.
- **STOPPED_MANUAL_RESTART**: the server was stopped with either a `stop` or `restart` command. If the server was issued a `restart` command, the status will be changed to `STOPPED_AUTO_RESTART` as the server starts again.
- **STARTING**: the server is actively starting up but is not yet ready to accept client requests.

### stop

Manually stop the AlwaysOn SQL service. With the `--dc datacenter name` option, manually stop the service on the specified datacenter.

### start

Manually start the AlwaysOn SQL service. With the `--dc datacenter name` option, manually start the service on the specified datacenter. The service will start automatically if it's been enabled (page 257).

### restart

Manually restart a running AlwaysOn SQL service. With the `--dc datacenter name` option, manually restart the service on the specified datacenter.

### reconfig

Manually reconfigure the AlwaysOn SQL service. With the `--dc datacenter name` option, manually reconfigure the service on specified datacenter. Running this command will tell the service to re-read the configuration options.

The `alwayson_sql_options` section in `dse.yaml`, described in detail at AlwaysOn SQL options (page 134), has options for setting the ports, timeout values, log location, and other Spark or Hive configuration settings. Additional configuration options are located in `spark-alwayson-sql.conf`.

### Examples

**Stop a running service:**

```
$ dse client-tool alwayson-sql stop
```

**Start the service on a particular datacenter:**

```
$ dse client-tool alwayson-sql --dc dc-west start
```
Force the service to stop:

```
$ dse client-tool alwayson-sql stop
```

Reread the configuration options for a running service:

```
$ dse client-tool alwayson-sql reconfig
```

dse nodesync

The NodeSync service (page 167) continuous background repair is enabled on a per table basis.

Modifies CQL nodesync property on one or more tables, enables nodesync tracing and monitoring.

**Important:** Tables with NodeSync enabled will be skipped for repair operations run against all or specific keyspaces. For individual tables, running the repair command will be rejected when NodeSync is enabled.

**Synopsis**

```
[dse] nodesync
[(-ca cql_Authprovider | --cql-auth-provider cql_Authprovider)]
[(-cp cql_password | --cql-password cql_password)]
[(-cs | --cql-ssl)]
[(-cu cql_username | --cql-username cql_username)]
[(-h cql_host | --host cql_host)]
[help]
[(-jp jmx_password | --jmx-password jmx_password)]
[(-jpf jmx_password_file | --jmx-password-file jmx_password_file)]
[(-js | --jmx-ssl)]
[(-ju jmx_username | --jmx-username jmx_username)]
[(-p cql_port | --port cql_port )]
```

**Table 339: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key: value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1, datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Main options

The following options apply to all nodesync commands.

- ca, --cql-auth-provider cql_Authprovider
  CQL auth provider class name.
- cp, --cql-password cql_password
  CQL password.
- cs | --cql-ssl
  Use SSL for CQL connection.
- cu, --cql-username cql_username
  CQL username.
- h, --host cql_host
  Connect to the specified remote CQL host.
- help
  Displays options and usage instructions. Use nodesync help subcommand for more information on a specific command.
- jp, --jmx-password jmx_password
  JMX password.
- jpf, --jmx-password-file jmx_password_file
  Path to JMX password file.
- js | --jmx-ssl

DSE 6.7 Developer Guide (Latest version)  Page 1241
Use SSL for JMX.

-ju, --jmx-username jmx_username
  JMX username.

-p, --port cql_port
  Connection port for CQL.

-k, --keyspace keyspace_name
  Specify a default keyspace for unqualified table names or wildcards in the table_list.

--quiet
  Suppress warning and error messages.

-v | --verbose
  Display all messages.

--
  Separates table list from the rest of the command.

table_list
  Target tables using any of the following methods:
  
  • Qualified table names: keyspace_name.table_name. For example, cycling.comments.
  
  • Default keyspace -k (page 1246) option with:
    
    # Unqualified table names. For example -k cycling cyclist_alt_stats
    comments cyclist_races.
    
    # An asterisk in double quotes to select all tables. For example, -k cycling
    "*".

-n, --nodes node_list
  Only disable tracing on the listed nodes. Specify the host name or IP address in a comma separated list.
  Default: all nodes.

--quiet
  Suppresses messages from displaying on stdout.

distributed=true | false
  Whether to distribute and apply the operation to all nodes in the local datacenter.
  
  • True applies the operation to all nodes in the local datacenter.
  
  • False applies the operation only to the node it was sent to. False works only when recovery=true.
  
  Default: true
  
  Warning: Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

generateResources=true | false
  Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with schema= and solrconfig=.
  
  Valid values:
- true - Automatically generate search index schema and configuration resources if resources do not already exist.
- false - Default. Do not automatically generate search index resources.

Examples

**Display top level help**

```
$ nodesync help
```

```
usage: nodesync [(js | --jmx-ssl)] [(p <cqlPort> | --port <cqlPort>)]
                [(c <cqlUsername> | --cql-username <cqlUsername>)]
                [(jmxPassword <jmxPassword> | --jmx-password <jmxPassword>)]
                [(jmxPasswordFile <jmxPasswordFile> | --jmx-password-file <jmxPasswordFile>)]
                [(ca <cqlAuthProvider> | --cql-auth-provider <cqlAuthProvider>)]
                [(jmxUsername <jmxUsername> | --jmx-username <jmxUsername>)]
                [(c <cqlPassword> | --cql-password <cqlPassword>)] [(c | --cql-ssl)]
                [(h <cqlHost> | --host <cqlHost>)] <command> [args]
```

The most commonly used nodesync commands are:
- disable Disable NodeSync on the specified tables
- enable Enable NodeSync on the specified tables
- help Display help information
- tracing Enable/disable tracing for NodeSync
- validation Monitor/manage user-triggered validations

See 'nodesync help <command>' for more information on a specific command.

**nodesync disable**

Disables NodeSync on one or more target tables by setting the nodesync enabled property to false.

**Synopsis**

```
[dse] nodesync main_options disable
        [(-k keyspace_name | --keyspace keyspace_name)]
        [--quiet]
        [(-v | --verbose)]
        [--] [(table_list | “*”)]
```

**Table 340: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Main options

The following options apply to all `nodesync` commands.

- **-ca, --cql-auth-provider cql_Authprovider**
  CQL auth provider class name.
- **-cp, --cql-password cql_password**
  CQL password.
- **-cs | --cql-ssl**
  Use SSL for CQL connection.
- **-cu, --cql-username cql_username**
  CQL username.
- **-h, --host cql_host**
  Connect to the specified remote CQL host.
- **help**
  Displays options and usage instructions. Use `nodesync help subcommand` for more information on a specific command.
-jp, --jmx-password jmx_password
  JMX password.
-jpf, --jmx-password-file jmx_password_file
  Path to JMX password file.
-js | --jmx-ssl
  Use SSL for JMX.
-ju, --jmx-username jmx_username
  JMX username.
-p, --port cql_port
  Connection port for CQL.
-k, --keyspace keyspace_name
  Specify a default keyspace for unqualified table names or wildcards in the table_list.
--quiet
  Suppress warning and error messages.
-v | --verbose
  Display all messages.
--
  Separates table list from the rest of the command.
table_list
  Target tables using any of the following methods:
  • Qualified table names: keyspace_name.table_name. For example, cycling.comments.
  • Default keyspace -k (page 1246) option with:
    # Unqualified table names. For example -k cycling cyclist_alt_stats comments cyclist_races.
    # An asterisk in double quotes to select all tables. For example, -k cycling "*".
-n, --nodes node_list
  Only disable tracing on the listed nodes. Specify the host name or IP address in a comma separated list.
  Default: all nodes.
--quiet
  Suppresses messages from displaying on stdout.
distributed=true | false
  Whether to distribute and apply the operation to all nodes in the local datacenter.
  • True applies the operation to all nodes in the local datacenter.
  • False applies the operation only to the node it was sent to. False works only when recovery=true.
  Default: true

  Warning: Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

generateResources=true | false
Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with schema= and solrconfig=.

Valid values:
- true - Automatically generate search index schema and configuration resources if resources do not already exist.
- false - Default. Do not automatically generate search index resources.

Disable options

The following options apply to the disable subcommand:

- **-k, --keyspace keyspace_name**
  Specify a default keyspace for unqualified table names or wildcards in the table_list.

- **--quiet**
  Suppress warning and error messages.

- **-v | --verbose**
  Display all messages.

- **--**
  Separates table list from the rest of the command.

**table_list**

Target tables using any of the following methods:
- Qualified table names: `keyspace_name.table_name`. For example, cycling.comments.
- Default keyspace `-k (page 1246)` option with:
  - Unqualified table names. For example `-k cycling cyclist_alt_stats comments cyclist_races`.
  - An asterisk in double quotes to select all tables. For example, `-k cycling "*"`.

Examples

**Disable on single table**

Set nodesync enabled to false on one table:

```
$ nodesync disable demo.health_data
```

No messages returned on success.

**Disable on list of tables in different keyspaces**

Set nodesync enabled to false on some tables in a keyspace:

```
$ nodesync disable -v -k demo -- test1 test2 test3
```

Displays a message for each table that was disabled:
Nodesync disabled for demo.test1
Nodesync disabled for demo.test2
Nodesync disabled for demo.test3

**Disable on all tables in a keyspace**

Set nodesync enabled to false on all tables in a keyspace:

```
$ nodesync disable -v -k demo "*"
```

Displays a message for each table that was disabled:

Nodesync disabled for demo.test2
Nodesync disabled for demo.health_data
Nodesync disabled for demo.test1
Nodesync disabled for demo.test
Nodesync disabled for demo.test3

**Disable on list of tables in different keyspaces**

Set nodesync enabled to false on all tables in a keyspace:

```
$ nodesync disable -v demo.test demo.test3 cycling.comments
cycling.cyclist_races
```

Displays a message for each table that was disabled:

Nodesync disabled for demo.test
Nodesync disabled for cycling.comments
Nodesync disabled for demo.test3
Nodesync disabled for cycling.cyclist_races

**nodesync enable**

Sets nodesync enabled to true on target tables.

**Tip:** Default setting is true.

**Synopsis**

```
[dse] nodesync main_options enable
[(-k keyspace_name | --keyspace keyspace_name)]
[--quiet]
[(-v | --verbose)]
[--] [table_list | "*"
```

**Table 341: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td><strong>Syntax conventions</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Main options**

The following options apply to all nodesync commands.

- **-ca, --cql-auth-provider cql_Authprovider**
  CQL auth provider class name.

- **-cp, --cql-password cql_password**
  CQL password.

- **-cs | --cql-ssl**
  Use SSL for CQL connection.

- **-cu, --cql-username cql_username**
  CQL username.

- **-h, --host cql_host**
  Connect to the specified remote CQL host.
help
Displays options and usage instructions. Use nodesync help subcommand for
more information on a specific command.

- jp, --jmx-password jmx_password
  JMX password.
- jpf, --jmx-password-file jmx_password_file
  Path to JMX password file.
- js | --jmx-ssl
  Use SSL for JMX.
- ju, --jmx-username jmx_username
  JMX username.
- p, --port cql_port
  Connection port for CQL.
- k, --keyspace keyspace_name
  Specify a default keyspace for unqualified table names or wildcards in the
table_list.
--quiet
  Suppress warning and error messages.
- v | --verbose
  Display all messages.
--
  Separates table list from the rest of the command.
table_list
  Target tables using any of the following methods:
  • Qualified table names: keyspace_name.table_name. For example,
cycling.comments.
  • Default keyspace -k (page 1246) option with:

    # Unqualified table names. For example -k cycling cyclist_alt_stats
    comments cyclist_races.
    # An asterisk in double quotes to select all tables. For example, -k cycling
    "*".

- n, --nodes node_list
  Only disable tracing on the listed nodes. Specify the host name or IP address in a
comma separated list.
  Default: all nodes.
--quiet
  Suppresses messages from displaying on stdout.
distributed=true | false
  Whether to distribute and apply the operation to all nodes in the local datacenter.
  • True applies the operation to all nodes in the local datacenter.
  • False applies the operation only to the node it was sent to. False works only
  when recovery=true.
  Default: true
Warning: Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

generateResources=true | false
Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with schema= and solrconfig=.

Valid values:
- true - Automatically generate search index schema and configuration resources if resources do not already exist.
- false - Default. Do not automatically generate search index resources.

Enable options
The following options apply to the enable subcommand:

-k, --keyspace keyspace_name
Specify a default keyspace for unqualified table names or wildcards in the table_list.

--quiet
Suppress warning and error messages.

-v | --verbose
Display all messages.

Separates table list from the rest of the command.

table_list
Target tables using any of the following methods:
- Qualified table names: keyspace_name[table_name]. For example, cycling.comments.
- Default keyspace -k (page 1246) option with:
  # Unqualified table names. For example -k cycling cyclist_alt_stats comments cyclist_races.
  # An asterisk in double quotes to select all tables. For example, -k cycling "*".

Examples

Enable single table
Set nodesync enabled to true on one table:

$ nodesync enable demo.health_data

No messages returned on success.

Enable multiple tables
Set nodesync enabled to true on two tables:
$ nodesync enable demo.health_data cycling.comment

No messages returned on success.

**Enable all tables in a keyspace**

Set nodesync enabled to true on two tables:

```
$ nodesync enable -v -k cycling "*"
```

A list of tables that are enabled is returned.

Nodesync enabled for cycling.comments
Nodesync enabled for cycling.cyclist_alt_stats
Nodesync enabled for cycling.cyclist_races

**nodesync tracing**

Provides detailed transaction information related to internal NodeSync operations by capturing events in the `system_traces` keyspace. When tracing is enabled a session id displays in standard output and an entry with the high-level details is written to the `system_traces.session` table. More detailed data for each operation is written to the `system_traces.events` table.

**Tip:** By default, Tracing information is saved for 7 days.

**Synopsis**

```
nodesync main_options tracing command
```

**Table 342: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Literal keyword.</td>
</tr>
<tr>
<td><strong>Lowercase</strong></td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td><strong>[ ]</strong></td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td><strong>( )</strong></td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td>**</td>
<td>**</td>
</tr>
<tr>
<td><strong>...</strong></td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Literal string'</td>
<td>Single quotation (‘’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (‘’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Main options

The following options apply to all nodesync commands.

- **-ca, --cql-auth-provider cql_Authprovider**
  CQL auth provider class name.
- **-cp, --cql-password cql_password**
  CQL password.
- **-cs | --cql-ssl**
  Use SSL for CQL connection.
- **-cu, --cql-username cql_username**
  CQL username.
- **-h, --host cql_host**
  Connect to the specified remote CQL host.
- **help**
  Displays options and usage instructions. Use nodesync help subcommand for more information on a specific command.
- **-jp, --jmx-password jmx_password**
  JMX password.
- **-jpf, --jmx-password-file jmx_password_file**
  Path to JMX password file.
- **-js | --jmx-ssl**
  Use SSL for JMX.
- **-ju, --jmx-username jmx_username**
  JMX username.
- **-p, --port cql_port**
  Connection port for CQL.
-k, --keyspace keyspace_name
    Specify a default keyspace for unqualified table names or wildcards in the
table_list.

--quiet
    Suppress warning and error messages.

-v | --verbose
    Display all messages.

--
    Separates table list from the rest of the command.

table_list
    Target tables using any of the following methods:
    
    • Qualified table names: keyspace_name.table_name. For example,
cycling.comments.
    • Default keyspace -k (page 1246) option with:
      # Unqualified table names. For example -k cycling cyclist_alt_stats
      comments cyclist_races.
      # An asterisk in double quotes to select all tables. For example, -k cycling
      "*".

-n, --nodes node_list
    Only disable tracing on the listed nodes. Specify the host name or IP address in a
comma separated list.
    Default: all nodes.

--quiet
    Suppresses messages from displaying on stdout.

distributed=true | false
    Whether to distribute and apply the operation to all nodes in the local datacenter.
    • True applies the operation to all nodes in the local datacenter.
    • False applies the operation only to the node it was sent to. False works only
when recovery=true.
    Default: true

    Warning: Distributing a re-index to an entire datacenter degrades
    performance severely in that datacenter.

generateResources=true | false
    Whether to automatically generate search index resources based on the existing
CQL table metadata. Cannot be used with schema= and solrconfig=.
    Valid values:
    • true - Automatically generate search index schema and configuration
    resources if resources do not already exist.
false - Default. Do not automatically generate search index resources.

**nodesync tracing disable**

Turns off NodeSync tracing.

**Synopsis**

```bash
nodesync main_options tracing disable
[(-n node_list | --nodes node_list)]
[--quiet]
[(-v | --verbose)]
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt; &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Main options

The following options apply to all `nodesync` commands.

- `-ca`, `--cql-auth-provider cql_Authprovider`
  CQL auth provider class name.
- `-cp`, `--cql-password cql_password`
  CQL password.
- `-cs | --cql-ssl`
  Use SSL for CQL connection.
- `-cu, --cql-username cql_username`
  CQL username.
- `-h, --host cql_host`
  Connect to the specified remote CQL host.
- `help`
  Displays options and usage instructions. Use `nodesync help` subcommand for more information on a specific command.
- `-jp, --jmx-password jmx_password`
  JMX password.
- `-jpf, --jmx-password-file jmx_password_file`
  Path to JMX password file.
- `-js | --jmx-ssl`
  Use SSL for JMX.
- `-ju, --jmx-username jmx_username`
  JMX username.
- `-p, --port cql_port`
  Connection port for CQL.
- `-k, --keyspace keyspace_name`
  Specify a default keyspace for unqualified table names or wildcards in the `table_list`.
- `--quiet`
  Supress warning and error messages.
- `-v | --verbose`
  Display all messages.
- `--`
  Separates table list from the rest of the command.

**table_list**

Target tables using any of the following methods:
- Qualified table names: `keyspace_name.table_name`. For example, `cycling.comments`.
- Default keyspace `-k (page 1246)` option with:
Unqualified table names. For example `-k cycling cyclist_alt_stats comments cyclist_races`.

An asterisk in double quotes to select all tables. For example, `-k cycling "*"`.

-n, --nodes node_list
Only disable tracing on the listed nodes. Specify the host name or IP address in a comma separated list.
Default: all nodes.

--quiet
Suppresses messages from displaying on stdout.

distributed=true | false
Whether to distribute and apply the operation to all nodes in the local datacenter.
- True applies the operation to all nodes in the local datacenter.
- False applies the operation only to the node it was sent to. False works only when recovery=true.
Default: true

Warning: Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

generateResources=true | false
Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with schema= and solrconfig=.
Valid values:
- true - Automatically generate search index schema and configuration resources if resources do not already exist.
- false - Default. Do not automatically generate search index resources.

disable options

-n, --nodes node_list
Only disable tracing on the listed nodes. Specify the host name or IP address in a comma separated list.
Default: all nodes.

--quiet
Suppresses messages from displaying on stdout.

Examples

Disable tracing on all nodes

```
$ nodesync tracing disable
```

Disable tracing on the local node
nodesync tracing enable

Enables tracing.

Synopsis

```

nodesync main_options tracing enable
[(-c | --color)]
[(-f | --follow)]
[(-l level_name | --level level_name)]
[(-n node_list | --nodes node_list)]
[--quiet]
[(-t seconds | --timeout seconds)]
[--tables table_list]
[(-v | --verbose)]
```

Table 344: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ] Separates the command line options from the command arguments with</td>
</tr>
<tr>
<td>two hyphens ( -- ). This syntax is useful when arguments might be mistaken</td>
</tr>
<tr>
<td>for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
</tr>
<tr>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema</td>
</tr>
<tr>
<td>declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
</tr>
<tr>
<td>Search CQL only: Identify the entity and literal value to overwrite the</td>
</tr>
<tr>
<td>XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Main options

The following options apply to all `nodesync` commands.

- **-ca, --cql-auth-provider cql_Authprovider**
  - CQL auth provider class name.

- **-cp, --cql-password cql_password**
  - CQL password.

- **-cs | --cql-ssl**
  - Use SSL for CQL connection.

- **-cu, --cql-username cql_username**
  - CQL username.

- **-h, --host cql_host**
  - Connect to the specified remote CQL host.

- **help**
  - Displays options and usage instructions. Use `nodesync help subcommand` for more information on a specific command.

- **-jp, --jmx-password jmx_password**
  - JMX password.

- **-jpf, --jmx-password-file jmx_password_file**
  - Path to JMX password file.

- **-js | --jmx-ssl**
  - Use SSL for JMX.

- **-ju, --jmx-username jmx_username**
  - JMX username.

- **-p, --port cql_port**
  - Connection port for CQL.

- **-k, --keyspace keyspace_name**
  - Specify a default keyspace for unqualified table names or wildcards in the `table_list`.

- **--quiet**
  - Suppress warning and error messages.

- **-v | --verbose**
  - Display all messages.

- **--**
  - Separates `table_list` from the rest of the command.

- **table_list**
  - Target tables using any of the following methods:
• Qualified table names: `keyspace_name.table_name`. For example, `cycling.comments`.

• Default keyspace `-k (page 1246)` option with:
  
  # Unqualified table names. For example `-k cycling cyclist_alt_stats comments cyclist_races`.
  
  # An asterisk in double quotes to select all tables. For example, `-k cycling "*"`.

-`n, --nodes node_list`
  Only disable tracing on the listed nodes. Specify the host name or IP address in a comma separated list.
  Default: all nodes.

--quiet
  Suppresses messages from displaying on stdout.

**distributed=true | false**
  Whether to distribute and apply the operation to all nodes in the local datacenter.
  
  • True applies the operation to all nodes in the local datacenter.
  
  • False applies the operation only to the node it was sent to. False works only when recovery=true.

  Default: true
  
  **Warning:** Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

**generateResources=true | false**
  Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with schema= and solrconfig=.

  Valid values:
  
  • true - Automatically generate search index schema and configuration resources if resources do not already exist.
  
  • false - Default. Do not automatically generate search index resources.

Enable options

-`c, --color`
  If --follow is used, color each trace event according from which host it originates from

-`f, --follow`
  After having enabled tracing, continuously show the trace events, showing new events as they come. Note that this won't exit unless you either manually exit (with Ctrl-c) or use a timeout (--timeout option).

-`l <levelStr>, --level <levelStr>`
  The tracing level: either 'low' or 'high'. If omitted, the 'low' level is used. Note that the 'high' level is somewhat verbose and should be used with care.
-n, --nodes node_list
    Only disable tracing on the listed nodes. Specify the host name or IP address in a
    comma separated list.
    Default: all nodes.

--quiet
    Suppresses messages from displaying on stdout.

-t <timeoutStr>, --timeout <timeoutStr>
    Timeout on the tracing; after that amount of time, tracing will be automatically
    disabled (and if --follow is used, the command will return). This default in seconds,
    but a 's', 'm' or 'h' suffix can be used for seconds, minutes or hours respectively.

--tables <tableStr>
    A comma separated list of fully-qualified table names to trace. If omitted, all tables
    are trace.

-v, --verbose
    Verbose output.

Examples

Enable tracing on all nodes

```
$ nodesync tracing enable
```

When the CQL host and JMX port is not specified, the local IP and default port are used.

Tracing

```
Warning: Do not forget to stop tracing with 'nodesync tracing disable'.
Enabled tracing. Session id is e60dfd70-eb5a-11e7-8bde-b5dcb560a8ef
```

**nodesync tracing show**

Display the events of a NodeSync tracing session.

Synopsis

```
nodesync main_options tracing show
    [[(--c | --color)]
    [[(--f | --follow)]
    [[|--i <traceIdStr> | --id <traceIdStr>]]
    [[|--n <nodeList> | --nodes <nodeList>]]
    [--quiet]
    [[(--t <timeoutStr> | --timeout <timeoutStr>)]
    [(-v | --verbose)]
```

**Table 345: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

## Main options

The following options apply to all nodesync commands.

- **-ca, --cql-auth-provider cql_Authprovider**
  
  CQL auth provider class name.

- **-cp, --cql-password cql_password**
  
  CQL password.

- **-cs | --cql-ssl**
  
  Use SSL for CQL connection.

- **-cu, --cql-username cql_username**
  
  CQL username.

- **-h, --host cql_host**
  
  Connect to the specified remote CQL host.

- **help**
  
  Connect to the specified remote CQL host.
Displays options and usage instructions. Use `nodesync help` subcommand for more information on a specific command.

- **-jp, --jmx-password** `jmx_password`
  JMX password.

- **-jpf, --jmx-password-file** `jmx_password_file`
  Path to JMX password file.

- **-js | --jmx-ssl**
  Use SSL for JMX.

- **-ju, --jmx-username** `jmx_username`
  JMX username.

- **-p, --port** `cql_port`
  Connection port for CQL.

- **-k, --keyspace** `keyspace_name`
  Specify a default keyspace for unqualified table names or wildcards in the `table_list`.

- **--quiet**
  Suppress warning and error messages.

- **-v | --verbose**
  Display all messages.

- **--**
  Separates table list from the rest of the command.

**table_list**
Target tables using any of the following methods:

- Qualified table names: `keyspace_name.table_name`. For example, `cycling.comments`.
- Default keyspace `-k` *(page 1246)* option with:
  
  ```
  # Unqualified table names. For example -k cycling cyclist_alt_stats comments cyclist_races.
  # An asterisk in double quotes to select all tables. For example, -k cycling "*".
  ```

- **-n, --nodes** `node_list`
  Only disable tracing on the listed nodes. Specify the host name or IP address in a comma separated list.
  Default: all nodes.

- **--quiet**
  Suppresses messages from displaying on `stdout`.

**distributed=true | false**
Whether to distribute and apply the operation to all nodes in the local datacenter.

- True applies the operation to all nodes in the local datacenter.
- False applies the operation only to the node it was sent to. False works only when recovery=true.

Default: true

**Warning:** Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.
**generateResources=true | false**

Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with schema= and solrconfig=.

Valid values:
- **true** - Automatically generate search index schema and configuration resources if resources do not already exist.
- **false** - Default. Do not automatically generate search index resources.

**show options**

-c, --color Colorize each trace event according from which host it originates from. Has no effect if the trace only come from a single host.
- ca <cqlAuthProvider>, --cql-auth-provider <cqlAuthProvider> CQL auth provider class name
- cp <cqlPassword>, --cql-password <cqlPassword> CQL password
- cs, --cql-ssl Enable SSL for CQL
- cu <cqlUsername>, --cql-username <cqlUsername> CQL username
- -f, --follow Continuously show the trace events, showing new events as they come. Note that this won't exit unless you either manually exit (with Ctrl-c) or use a timeout (--timeout option)
- h <cqlHost>, --host <cqlHost> CQL contact point address
- i <traceIdStr>, --id <traceIdStr> The trace ID to show. If omitted, this will check if some nodes have tracing enable, and if all node that have it use the same trace ID, it will default to showing that. Otherwise, the command error out.
- jp <jmxPassword>, --jmx-password <jmxPassword> JMX password
- jpf <jmxPasswordFile>, --jmx-password-file <jmxPasswordFile> Path to the JMX password file
- js, --jmx-ssl Enable SSL for JMX
- ju <jmxUsername>, --jmx-username <jmxUsername> JMX username
- n <nodeList>, --nodes <nodeList> Comma separated of nodes address on which to act; if omitted, all (live) nodes will be included
- p <cqlPort>, --port <cqlPort> CQL port number
- -q, --quiet Quiet output; don't print warnings
- t <timeoutStr>, --timeout <timeoutStr> When --follow is used, automatically exit after the provided amount of time elapses. This default to seconds, but a 's', 'm' or 'h' suffix can be used for seconds, minutes or hours respectively.
- v, --verbose Verbose output

**Examples**

```shell
$ nodesync tracing show -i e60dfd70-eb5a-11e7-8bde-b5dcb560a8ef
```

Starting NodeSync tracing on /10.200.176.186 (elapsed: 2.7ms)
Adding continuous proposer for demo.health_data (elapsed: 6.9m)
[#] Skipping (10,-9223372036854775808) of demo.health_data, state updated: was recently validated by another node (2h ago, previously know: 2h ago) (elapsed: 6.9m)
[#0] Starting validation on (-9223372036854775798,10] of demo.health_data (validated 2h ago) (elapsed: 6.9m)
[#1] Starting validation on (-9223372036854775808,-9223372036854775798) of demo.health_data (validated 2.1h ago) (elapsed: 6.9m)
[#0] Completed validation (full_in_sync) in 4ms: validated 0B and repaired 0B (elapsed: 6.9m)
[#2] Starting validation on (10,-9223372036854775808] of demo.health_data (validated 2h ago) (elapsed: 6.9m)
nodesync tracing status

Enable/disable tracing for NodeSync.

Synopsis

```
nodesync main_options tracing  status

[(-n <nodeList> | --nodes <nodeList>)]
[--quiet]
[(-v | --verbose)]
```

Table 346: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (' ') marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (-- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (‘’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Main options

The following options apply to all `nodesync` commands.

- **-ca, --cql-auth-provider cql_Authprovider**  
  CQL auth provider class name.

- **-cp, --cql-password cql_password**  
  CQL password.

- **-cs | --cql-ssl**  
  Use SSL for CQL connection.

- **-cu, --cql-username cql_username**  
  CQL username.

- **-h, --host cql_host**  
  Connect to the specified remote CQL host.

- **help**  
  Displays options and usage instructions. Use `nodesync help` subcommand for more information on a specific command.

- **-jp, --jmx-password jmx_password**  
  JMX password.

- **-jpf, --jmx-password-file jmx_password_file**  
  Path to JMX password file.

- **-js | --jmx-ssl**  
  Use SSL for JMX.

- **-ju, --jmx-username jmx_username**  
  JMX username.

- **-p, --port cql_port**  
  Connection port for CQL.

- **-k, --keyspace keyspace_name**  
  Specify a default keyspace for unqualified table names or wildcards in the `table_list`.

- **--quiet**  
  Suppress warning and error messages.

- **-v | --verbose**  
  Display all messages.

- **--**  
  Separates table list from the rest of the command.

### table_list

Target tables using any of the following methods:

- **Qualified table names**: `keyspace_name.table_name`. For example, `cycling.comments`.
Default keyspace -k (page 1246) option with:

- Unqualified table names. For example -k cycling cyclist_alt_stats comments cyclist_races.
- An asterisk in double quotes to select all tables. For example, -k cycling "*".

-n, --nodes node_list
Only disable tracing on the listed nodes. Specify the host name or IP address in a comma separated list.
Default: all nodes.

--quiet
Suppresses messages from displaying on stdout.

distributed=true | false
Whether to distribute and apply the operation to all nodes in the local datacenter.
- True applies the operation to all nodes in the local datacenter.
- False applies the operation only to the node it was sent to. False works only when recovery=true.
Default: true

Warning: Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

generateResources=true | false
Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with schema= and solrconfig=.

Valid values:
- true - Automatically generate search index schema and configuration resources if resources do not already exist.
- false - Default. Do not automatically generate search index resources.

Examples

```
$ nodesync validation
```

**nodesync validation**

Monitor and manage user-triggered validations.

**Synopsis**

```
[dse] nodesync main_options validation
  (cancel id |
   list |
   submit [(-r KB | --rate KB)] [--] table_name [range ...] )
```
Main options

The following options apply to all nodesync commands.

- **ca**, **--cql-auth-provider** `cql_Authprovider`
  CQL auth provider class name.

- **cp**, **--cql-password** `cql_password`
  CQL password.

- **cs | --cql-ssl**
  Use SSL for CQL connection.

- **cu**, **--cql-username** `cql_username`
  CQL username.

- **h, --host** `cql_host`
  Connect to the specified remote CQL host.

**help**
Displays options and usage instructions. Use `nodesync help subcommand` for more information on a specific command.

- **jp**, **--jmx-password** `jmx_password`
  JMX password.

- **jpf, --jmx-password-file** `jmx_password_file`
  Path to JMX password file.

- **js | --jmx-ssl**
  Use SSL for JMX.

- **ju**, **--jmx-username** `jmx_username`
  JMX username.

- **p, --port** `cql_port`
  Connection port for CQL.

- **k, --keyspace** `keyspace_name`
  Specify a default keyspace for unqualified table names or wildcards in the `table_list`.

  **--quiet**
  Suppress warning and error messages.

- **v | --verbose**
  Display all messages.

  **--**
  Separates table list from the rest of the command.

**table_list**
Target tables using any of the following methods:

- Qualified table names: `keyspace_name.table_name`. For example, `cycling.comments`.
- Default keyspace `-k (page 1246)` option with:
  
  # Unqualified table names. For example `-k cycling cyclist_alt_stats comments cyclist_races`.
  
  # An asterisk in double quotes to select all tables. For example, `-k cycling "*"`.  

DSE 6.7 Developer Guide (Latest version)
-n, --nodes node_list
  Only disable tracing on the listed nodes. Specify the host name or IP address in a comma separated list.
  Default: all nodes.
--quiet
  Suppresses messages from displaying on stdout.
distributed=true | false
  Whether to distribute and apply the operation to all nodes in the local datacenter.
  • True applies the operation to all nodes in the local datacenter.
  • False applies the operation only to the node it was sent to. False works only when recovery=true.
  Default: true
  Warning: Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.
generateResources=true | false
  Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with schema= and solrconfig=.
  Valid values:
  • true - Automatically generate search index schema and configuration resources if resources do not already exist.
  • false - Default. Do not automatically generate search index resources.

validation options

cancel id
  Cancel the specified user-triggered validation.
list [(-a | --all)]
  List user validations. Use -a to list all, running or validations that completed in the past day.
  Default: Only running validations are displayed.
submit [options] table_name [range]
  Submit a forced user validation.
  -r KB, --rate KB Rate to be used just for this validation, in KB per second.
--quiet
  Suppress warning and error messages.
--
  Separates command-line options from the list of argument. Use when arguments might be mistaken for command-line options.
table_name [token_range]
  Keyspace qualified table name, optionally followed by token ranges in the form (x, y). If no token ranges are specified, then all the tokens are validated.
-v | --verbose
Display all messages.

**dsefs commands**

The DSEFS functionality supports operations including uploading, downloading, moving, and deleting files, creating directories, and verifying the DSEFS status.

**append**

Appends a local file to a remote file.

**Tip:** Refer to files in the local file system by prefixing paths with the `file:` prefix.

**Synopsis**

```
$ append source_filepath destination_filepath
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td><em>'Literal string'</em></td>
<td>Single quotation (' ') marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt; &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>'@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

#### Definition

The short form and long form parameters are comma-separated.

#### Command arguments

**destination_filepath**

Explicit or relative filepath.
- If destination path ends with name, destination entry is given that name.
- If the destination path ends with a backslash (\), original source file name is used.
- **Wildcard (page 294)** characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.

**source_filepath**

Explicit or relative filepath.
- **Wildcard (page 294)** characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.

#### Examples

**Append local file to remote file**

```
$ dsefs dsefs://127.0.0.1:5598/ > append file:/home/cal09 dsefs:/data2/cal10
```

**cat**

Concatenates files and prints on the standard output.

**Synopsis**

```
$ cat filepath [filepath ...]
```
Table 348: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Command arguments

filepath

Explicit or relative filepath.

- Wildcard (page 294) characters are supported.
DataStax Enterprise tools

- Explicit file system prefixes `dsefs:` and `file:` are supported.
- `..` is the parent directory.

Examples

Print one file in the DSE filesystem to standard output

dsefs file:/home/ > cat calSept

<table>
<thead>
<tr>
<th>September 2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Su</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>30</td>
</tr>
</tbody>
</table>

Print two files in the DSE filesystem to standard output

dsefs file:/home/ > cat calSept calOct

<table>
<thead>
<tr>
<th>September 2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Su</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>October 2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>Su</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>27</td>
</tr>
</tbody>
</table>

cd

Changes the working directory in DSEFS. The DSEFS shell remembers the last working directory of each file system separately.

Tip: The DSEFS prompt identifies the current working directory in DSEFS:

- `dsefs dsefs://127.0.0.1:5598/` > is the default directory
- `dsefs dsefs://127.0.0.1:5598/dir2/` is the current working directory `dir2`
- `dsefs file:/` > is the current directory on the local file system
Synopsis

$ cd filepath

Table 349: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Command arguments
### filepath

Explicit or relative filepath.

- **Wildcard** ([page 294]) characters are supported.
- Explicit file system prefixes `dsefs:` and `file:` are supported.
- `..` is the parent directory.

### Examples

#### Change directory in DSEFS

```bash
$ dsefs dsefs://127.0.0.1:5598/ > cd tmp
dsefs dsefs://127.0.0.1:5598/tmp/ >
```

#### Change directory to the last working directory on the local file system

```bash
$ dsefs dsefs://127.0.0.1:5598/ > cd file:/
dsefs file:/home/user1/path/to/local/files
```

#### Change directory to the parent directory on the local file system

```bash
$ dsefs file:/home/user1/path/to/local/files > cd ..
dsefs file:/home/user1/path/to/local >
```

#### Go back to the last working directory in DSEFS

```bash
$ dsefs file:/home/user1/path/to/local/files > cd dsefs:
dsefs dsefs://127.0.0.1:5598/ >
```

### chgrp

Changes group ownership for files or directories.

#### Synopsis

```bash
$ chgrp [-R] [-v] group_name filepath [filepath ...]
```

#### Table 350: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity=’xml_entity_type’</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Command arguments**

**filepath**

Explicit or relative filepath.

- *Wildcard (page 294)* characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.

**group_name**
DataStax Enterprise tools

    Group name.
-R, --recursive
    Remove directories and their contents recursively.
-v, --verbose
    Turn on verbose output.

Examples

Change group ownership of myFile to admin

```bash
$ dsefs dsefs://127.0.0.1:5598/ > chgrp admin file:/home/myFile
```

**chmod**

Changes permission mode for owner, group, and others.

**Synopsis**

```bash
$ chmod [-R] [-v] permission_mode filepath [filepath ...]
```

**Table 351: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Command arguments

**filepath**

Explicit or relative filepath.

- Wildcard (page 294) characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.

**permission_mode**

Octal representation of permission mode for owner, group, and others:

- 0 – no permission
- 1 – execute
- 2 – write
- 3 – write and execute
- 4 – read
- 5 – read and execute
- 6 – read and write
- 7 – read, write, and execute

-R, --recursive

Remove directories and their contents recursively.

-v, --verbose

Turn on verbose output.

### Examples

**Change permission to make file readable, writable and executable by all users**

```
$ dsefs dsefs://127.0.0.1:5598/ > chmod 777 file:/home/myFile
```

**Change permission to make file readable, writable and executable by owner and only executable by group and others**
$ dsefs dsefs://127.0.0.1:5598/ > chmod 711 file:/home/myFile

chown

Changes ownership and/or group ownership for files or directories.

Synopsis

$ chgrp [-R] [-v] [-u username] [-g group_name] filepath [filepath ...]

Table 352: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
Definition
The short form and long form parameters are comma-separated.

Command arguments

**filepath**
Explicit or relative filepath.
- Wildcard ([page 294](#)) characters are supported.
- Explicit file system prefixes `dsefs:` and `file:` are supported.
- `..` is the parent directory.

**-g, --group group_name**
New owner group name.

**-R, --recursive**
Remove directories and their contents recursively.

**-u, --user username**
Set new owner username.

**-v, --verbose**
Turn on verbose output.

Examples

**Recursively change ownership to admin group for two files**

```
$ dsefs dsefs://127.0.0.1:5598/ > chown -R -g admin file:/home/myFile
file:/data2/myFile2
```

**Change ownership to John Doe**

```
$ dsefs dsefs://127.0.0.1:5598/ > chown -u jdoe dsefs:/home/myFile
```

**cp**
Copies files and directories within a file system or between two file systems. Downloads contents to the client from the source file system and uploads contents to the target filesystem. Copying multiple files is done sequentially.

**Note:** Copies are created with default attributes. Source file and directory attributes like permissions, ownership, block size, and redundancy are not preserved.

**Synopsis**

```
$ cp [-b size_in_bytes]
[--force-sync] [--no-force-sync]
[-o] [-r]
source_path [source_path ...] destination_path
```
Table 353: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Command arguments

-b, --block-size size_in_bytes
   Preferred block size in bytes for files. Ignored when the destination path is a file system other than DSEFS.

destination_path
Explicit path.

- If more than one source entry is copied, this path must point to an existing directory and end with a backslash (/).
- Supports explicit file system prefixes:
  
  # dsefs: - path to a remote file on the DSEFS file system dsefs://
  # ip_address/path/to/a/remote/file
  # file: - path to a local file file:path/to/a/local/file

- Wildcard (page 294) characters are supported.

--force-sync
Synchronize files in this directory with the storage device when closed. Files created in the directory inherit the option.

--no-force-sync
Do not synchronize files in this directory with the storage device when closed. Files created in the directory inherit the option.

-n, --redundancy-factor num_nodes
Create number of replicas of file data, similar to the replication factor in the database keystores, but more granular.

- One number greater than the number of nodes that are allowed to fail before data loss occurs. For example, set this value to 3 to allow 2 nodes to fail.
- For simple replication, use a value that is equivalent to the replication factor.
- Default value is inherited from the parent directory.

-o, --overwrite
If destination file exists, overwrite.

-r, -R, --recursive
Copy directories and their contents recursively.

source_filepath
Explicit or relative filepath.

- Wildcard (page 294) characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.

Examples

Copy file from source to overwrite file in destination

```
$ dsefs file:/home/user1/test > cp -o dsefs:archive.tgz another-archive-copy.tgz
```

**df**

Reports file system status and disk space usage.
Synopsis

$ df [-h]

Table 354: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Command arguments
-h, --human-readable
Display human-readable sizes. For example, 1.25k, 234M, or 2G.

Examples

Get file system status and disk space usage

```
$ dsefs dsefs://127.0.0.1:5598/ > df
```

<table>
<thead>
<tr>
<th>Location</th>
<th>Status</th>
<th>DC</th>
<th>Rack</th>
<th>Host</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144e587c-11b1-4d74-80f7-dc5e0c744aca up GraphAnalytics rack1</td>
<td>node1.example.com 10.200.179.38 5598 /var/lib/dsefs/data 0</td>
<td>29289783296 5368709120</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98ca0435-fb36-4344-b5b1-8d776d35c7d6 up GraphAnalytics rack1</td>
<td>node2.example.com 10.200.179.39 5598 /var/lib/dsefs/data 0</td>
<td>29302099968 5368709120</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
echo
dispaly a line of text.
```

Synopsis

```
$ echo text_to_display
```

Table 355: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
Syntax conventions | Description
--- | ---
{ key:value } | Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
<datatype1,datatype2> | Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
cql_statement; | End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ] | Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' <schema> ... </schema> ' | Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@xml_entity='xml_entity_type' | Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

Definition

The short form and long form parameters are comma-separated.

Command arguments

text_to_display

Text to display.

Examples

Display File copied

$ dsefs dsefs://127.0.0.1:5598/ > echo File copied

exit

Exits DSEFS command shell.

Synopsis

$ exit

Table 356: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Command arguments**

This command takes no arguments.

**Examples**

**Exit DSEFS command shell**
DataStax Enterprise tools

```bash
$ dsefs dsefs://127.0.0.1:5598/ > exit
```

**fsck**

Performs file system consistency check and repairs file system errors. Only a superuser may run `fsck`.

**Synopsis**

```bash
$ fsck
```

**Table 357: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
Definition
The short form and long form parameters are comma-separated.

Command arguments
This command takes no arguments.

Examples
Check file system and repair errors

```
$ dsefs dsefs://127.0.0.1:5598/ > fsck
```

get

A special case of `cp (page 1279)` that copies a DSEFS remote file to the local file system. If a relative source path is given, it is resolved in the last DSEFS working directory, regardless of the current working directory. Similarly, if a relative destination path is given, it is always resolved in the last local working directory. Filepaths can be absolute and can point to any file system.

Synopsis

```
$ get source_filepath destination_filepath
```

Table 358: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Literal string'</td>
<td>Single quotation (‘’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt;&gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--) . This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (‘’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Command arguments

**destination_filepath**

Explicit or relative filepath.

- If destination path ends with name, destination entry is given that name.
- If the destination path ends with a backslash (/), original source file name is used.
- **Wildcard (page 294)** characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.

**source_filepath**

Explicit or relative filepath.

- **Wildcard (page 294)** characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.

### Examples

**Copy DSEFS remote file to local filesystem**
$ dsefs dsefs://127.0.0.1:5598/ > dsefs / > get archive.tgz
local_archive.tgz

Is

Lists directory contents.

Synopsis

$ ls [-R] [-l] [-h] [-1] [directory_name [directory_name ...]]

Table 359: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Command arguments

**directory_name**
Directory on DSEFS file system.

- **Wildcard (page 294)** characters are supported.
- **..** is the parent directory.

**-h, --human-readable**
Display human-readable sizes. For example, 1.25k, 234M, or 2G.

**-l, --long**
Use long listing format.

**-R, --recursive**
Remove directories and their contents recursively.

**-1, --single-column**
List one file per line.

Examples

**List directory contents**

```
$ dsefs dsefs://127.0.0.1:5598/ > ls file:
```

```
bin  cdrom  dev  home  lib32  lost+found  mnt  proc  run  srv  tmp  var
    initrd.img.old  vmlinuz.old
boot  data  etc  lib  lib64  media  opt  root  sbin  sys  usr
    initrd.img  vmlinuz
```

**List directory contents with one file per line**

```
$ dsefs dsefs://127.0.0.1:5598/ > ls -1 file:
```

```
bin
cdrom
dev
home
lib32
lost+found
mnt
proc
run/nsrv
```
mkdir

Creates new directory or directories.

Synopsis

$ mkdir [-p] [-b size_in_bytes] [-n num_nodes]
[-c encoder_name] [-m permission_mode]
[--no-force-sync] [--force-sync]
new_directory_name [new_directory_name ...]

Table 360: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ( { ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

## Definition

The short form and long form parameters are comma-separated.

### Command arguments

- **-b, --block-size size_in_bytes**
  Preferred block size in bytes for files in new directory.
  Default is 64 MB.

- **-c, --compression-encoder encoder_name**
  The compression encoder name. DSE ships with the LZ4 compression encoder.

- **new_directory_name**
  New directory on DSEFS file system.
  - Explicit file system prefixes dsefs: and file: are supported.

- **--force-sync**
  Synchronize files in this directory with the storage device when closed. Files created in the directory inherit the option.

- **-m, --permission-mode permission_mode**
  Octal representation of permission mode for owner, group, and others:
  - 0 – no permission
  - 1 – execute
  - 2 – write
  - 3 – write and execute
  - 4 – read
  - 5 – read and execute
  - 6 – read and write
  - 7 – read, write, and execute
**--no-force-sync**
Do not synchronize files in this directory with the storage device when closed. Files created in the directory inherit the option.

**-n, --redundancy-factor num_nodes**
Create number of replicas of file data, similar to the replication factor in the database keyspaces, but more granular.

- One number greater than the number of nodes that are allowed to fail before data loss occurs. For example, set this value to 3 to allow 2 nodes to fail.
- For simple replication, use a value that is equivalent to the replication factor.
- Default value is 3.

Ignored when the destination path is a file system other than DSEFS.

**-p, --parents**
If needed, makes parent directories. If parent directories exist, no error.

**Examples**

**Make new directory with 32-MB block sizes, redundancy factor or 2, files synchronize on close**

```bash
$ dsefs dsefs://127.0.0.1:5598/ > mkdir -b 32000000 -n 2 --force-sync file:new_directory
```

**mv**

Moves a file or directory.

**Synopsis**

```
$ mv source_filepath [source_filepath ...] destination_filepath
```

**Table 361: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Command arguments

**destination_filepath**

Explicit or relative filepath.

- If destination path ends with name, destination entry is given that name.
- If the destination path ends with a backslash (/), original source file name is used.
- **Wildcard (page 294)** characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.

**source_filepath**

Explicit or relative filepath.

- **Wildcard (page 294)** characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.
Examples

Move file from source file to destination directory

$ dsefs dsefs://127.0.0.1:5598/ > mv file:/home/myFile dsefs:/data2/myDirectory

put

A special case of `cp (page 1279)` that copies a local file to the DSE filesystem. If a relative source path is given, it is resolved in the last local working directory, regardless of the current working directory. Similarly, if a relative destination path is given, it is always resolved in the last DSEFS working directory. As in `cp`, both paths may be absolute and are allowed to point to any file system. If the destination path points to a different file system than DSEFS, the block size and redundancy options are ignored.

Synopsis


Table 362: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( <code>&lt;</code> ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( <code>;</code> ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( <code>--</code> ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks ( <code>'</code> ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Command arguments**

**-b, --block-size size_in_bytes**
Preferred block size in bytes for files. Ignored when the destination path is a file system other than DSEFS.

**-c, --compression-encoder encoder_name**
The compression encoder name. DSE ships with the LZ4 compression encoder.

**destination_filepath**
Explicit or relative filepath.
- If destination path ends with name, destination entry is given that name.
- If the destination path ends with a backslash ( `/` ), original source file name is used.
- Wildcard (page 294) characters are supported.
- Explicit file system prefixes `dsefs:` and `file:` are supported.
- .. is the parent directory.

**-f, --compression-frame-size frame_size_in_bytes**
Compress frame to preferred frame size in bytes. Frame is a subject of compression. The bigger the frame, the bigger chance for high compression ratio.
In most cases, the default value is sufficient. Default frame size in bytes is 131072 bytes.

**--force-sync**
Synchronize files in this directory with the storage device when closed. Files created in the directory inherit the option.

**-m, --permission-mode permission_mode**
Octal representation of permission mode for owner, group, and others:
-n, --redundancy-factor num_nodes
Create number of replicas of file data, similar to the replication factor in the
database keyspaces, but more granular.

- One number greater than the number of nodes that are allowed to fail before
data loss occurs. For example, set this value to 3 to allow 2 nodes to fail.
- For simple replication, use a value that is equivalent to the replication factor.
- Default value is inherited from the parent directory.

--no-force-sync
Do not synchronize files in this directory with the storage device when closed. Files
created in the directory inherit the option.

-o, --overwrite
If destination file exists, overwrite.

source_filepath
Explicit or relative filepath.

- Wildcard (page 294) characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.

Examples

Copy local bluefile to remote greenfile

$ dsefs dsefs://127.0.0.1:5598/ > put file:/bluefile greenfile

pwd
Prints full filepath of current working directory.

Synopsis

$ pwd [directory_path]

Table 363: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>Literal string</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Command arguments

**directory_path**

Current working directory.

### Examples

**Print working directory**
```
$ dsefs dsefs://127.0.0.1:5598/ > pwd dsefs://myDirectory

dsefs:/home/user1/new_directory
```

**realpath**

Prints the resolved absolute path; all but the last component must exist

**Synopsis**

```
$ realpath [-e] [-m] filepath [filepath ...]
```

**Table 364: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (' ') marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
Table 365: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (' ') marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks (' ') surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Command arguments**

**new_name**

New file name.

**filepath**

Explicit or relative filepath.

- **Wildcard (page 294)** characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.
## Examples

### Rename myFile to cyclist

```sh
$ dsefs dsefs://127.0.0.1:5598/ > rename file:/home/myFile cyclist
```

### rm

Removes files or directories.

**Synopsis**

```sh
$ rm [-R] [-v] filepath [filepath ...]
```

### Table 366: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>'&lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks (') surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Command arguments

**filepath**

Explicit or relative filepath.

- **Wildcard** *(page 294)* characters are supported.
- Explicit file system prefixes `dsefs:` and `file:` are supported.
- `..` is the parent directory.

**-R, --recursive**

Remove directories and their contents recursively.

**-v, --verbose**

Turn on verbose output.

### Examples

**Remove files**

```bash
$ dsefs dsefs://127.0.0.1:5598/ > rm file:/home/myFile dsefs:/home/remoteFile
```

**Remove directory**

```bash
$ dsefs dsefs://127.0.0.1:5598/ > rm file:/home/
```

**rmdir**

Removes empty directory or directories.

### Synopsis

```
$ rmdir filepath [filepath ...]
```

### Table 367: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
</tbody>
</table>
Syntax conventions | Description
--- | ---
**Lowercase** | Not literal.
**Italics** | Variable value. Replace with a valid option or user-defined value.
[ ] | Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.
( ) | Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.
| | Or. A vertical bar ( | ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
... | Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.
'Literal string' | Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ key:value } | Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
<datatype1,datatype2> | Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
cql_statement; | End CQL statement. A semicolon (;) terminates all CQL statements.
[ -- ] | Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' <schema> ... </schema> ' | Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.
@xml_entity='xml_entity_type' | Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**Definition**
The short form and long form parameters are comma-separated.

**Command arguments**

**filepath**
Explicit or relative filepath.
- **Wildcard (page 294)** characters are supported.
- Explicit file system prefixes dsefs: and file: are supported.
- .. is the parent directory.
Examples

Remove empty directory

```bash
$ dsefs dsefs://127.0.0.1:5598/ > rmdir file:/home/
```

**stat**

Displays file or directory status.

**Synopsis**

```bash
$ stat [-v] filepath [filepath ...]
```

<table>
<thead>
<tr>
<th>Table 368: Legend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax conventions</td>
</tr>
<tr>
<td>UPPERCASE</td>
</tr>
<tr>
<td>Lowercase</td>
</tr>
<tr>
<td><em>Italics</em></td>
</tr>
<tr>
<td>[ ]</td>
</tr>
<tr>
<td>( )</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>'Literal string'</td>
</tr>
<tr>
<td>{ key:value }</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
</tr>
<tr>
<td>cql_statement;</td>
</tr>
<tr>
<td>[ -- ]</td>
</tr>
<tr>
<td>Syntax conventions</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td><code>&lt;schema&gt; ... &lt;/schema&gt;</code></td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Command arguments**

**filepath**

Explicit or relative filepath.

- **Wildcard** (page 294) characters are supported.
- Explicit file system prefixes `dsefs:` and `file:` are supported.
- `..` is the parent directory.

- **-v, --verbose**
  
  Turn on verbose output.

**Examples**

**Get status of directory**

```
$ dsefs dsefs://127.0.0.1:5598/ > stat file:new_directory
```

DIRECTORY file:/home/user1/new_directory:

- **Owner** user1
- **Group** user1
- **Permission** `rwxr-xr-x`
- **Created** 2017-01-15 13:10:06+0200
- **Modified** 2017-01-15 13:10:06+0200
- **Accessed** 2017-01-15 13:10:06+0200
- **Size** 4096

**truncate**

Truncates file or files to a specified length.

To retain only metadata, set file size to 0 bytes. Also useful to keep an empty file for processes without deleting and recreating a file.

**Synopsis**

```
$ truncate [-s size_in_bytes] filepath [filepath ...]
```
### Table 369: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>’ &lt;schema&gt; ... &lt;/schema&gt; ’</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity=’xml_entity_type’</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Command arguments**

**filepath**

Explicit or relative filepath.

- Wildcard (page 294) characters are supported.
• Explicit file system prefixes \texttt{dsefs:} and \texttt{file:} are supported.
• .. is the parent directory.

\texttt{-s, --size size\_in\_bytes}
Set new file size in bytes.

**Examples**

**Truncate file to 0 bytes**

```
$ \texttt{dsefs dsefs://127.0.0.1:5598/} \textgg{} \texttt{truncate \,-s 0 \, file:/home/myFile}
```

**umount**

Unmounts file system storage locations from file hierarchy. Only a superuser may run \texttt{umount}.

**Synopsis**

```
$ \texttt{umount \[-f\] \, location\_UUID \[\, location\_UUID \, ...\]}
```

**Table 370: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>{ }</td>
<td>Optional. Square brackets { } surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar ( ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cql_statement;</code></td>
</tr>
<tr>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
</tr>
<tr>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
| `' <schema> ... </schema> 
`                                               |
| Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration. |
| `@xml_entity='xml_entity_type'`                                              |
| Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files. |

### Definition
The short form and long form parameters are comma-separated.

### Command arguments

- **-f, --force**
  Force unmounting, even if location is unavailable.

- **location_UUID**
  UUID of location.

### Examples

**Unmount location from DSEFS**

```
$ dsefs dsefs://127.0.0.1:5598/ > umount dcd9dd1f-46c8-4b47-b3e3-
aa431156021a
```

### dsetool

**About dsetool**

dsetool is a command line interface for DSE operations.

**Synopsis**

```
$ dsetool [connection_options] command command_args
```

**Table 371: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (' ') marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (‘ ‘) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Using dsetool command line help

To view a listing of dsetool commands:

```bash
$ dsetool help
```

To view help for a specific command:

```bash
$ dsetool command help
```
dsetool commands for DSE Search

Search CQL commands are distributed to the entire data center. The dsetool commands for DSE Search distribute search index changes to the data center by default, and are node-specific only when the distributed flag is set to false.

**Connection options**

Options for connecting to your cluster with the dsetool utility.

**Synopsis**

```
$ dsetool [connection_options] command command_args
```

**Table 372: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (' ') marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks (' ') surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
JMX authentication is used by some dsetool commands. Other dsetool commands authenticate with the user name and password of the configured user. The connection option short form and long form are comma separated.

Note:

You can provide authentication credentials in several ways, see Credentials for authentication.

To enable dsetool to use Kerberos authentication, see Using dsetool with Kerberos enabled cluster.

Specify how to connect and authenticate the dsetool command.

This list shows short form (-f filename) and long form (--config-file=filename):

- **-a, --jmxusername jmx_username**  
  User name for authenticating with secure local JMX.

- **-b, --jmxpassword jmx_password**  
  Password for authenticating with secure local JMX. If you do not provide a password, you are prompted to enter one.

- **-c, --cassandra_port dse_port**  
  DSE port number.

- **--cipher-suites ssl_cipher_suites**  
  Specify comma-separated list of SSL cipher suites for connection to DSE when SSL is enabled. For example, --cipher-suites c1,c2,c3.

- **-f, --config-file config_filename**  
  File path to configuration file that stores credentials. The credentials in this configuration file override the ~/.dserc credentials. If not specified, then use ~/.dserc if it exists.

  The configuration file can contain DataStax Enterprise and JMX login credentials. For example:

  ```
  username=username
  password=password
  jmx_username=jmx_username
  jmx_password=jmx_password
  ```

  The credentials in the configuration file are stored in clear text. DataStax recommends restricting access to this file only to the specific user.

- **-h, --host IP_address**  
  Connect to the specified hostname or IP address. Do not connect to the local node.

- **-j, --jmxport jmx_port**  
  Remote JMX agent port number.
--keystore-path ssl_keystore_path
    Path to the keystore for connection to DSE when SSL client authentication is enabled.
--keystore-password keystore_password
    Keystore password for connection to DSE when SSL client authentication is enabled.
--keystore-type ssl_keystore_type
    Keystore type for connection to DSE when SSL client authentication is enabled. JKS is the type for keys generated by the Java keytool binary, but other types are possible, depending on user environment.
-l, --username username
    Role to authenticate for database access.
-p, --password password
    Password to authenticate for database access.
-s, --port solr_port
    Solr port.
--ssl true | false
    Whether to use SSL for native connections.
--ssl-protocol ssl_protocol
    SSL protocol for connection to DSE when SSL is enabled. For example, --ssl-protocol ssl4.
--sslauth true | false
    Whether to use SSL client authentication.
--truststore_password ssl_truststore_password
    Truststore password to use for connection to DSE when SSL is enabled.
--truststore_path ssl_truststore_path
    Path to the truststore to use for connection to DSE when SSL is enabled. For example, --truststore-path /path/to/ts.
--truststore-type ssl_truststore_type
    Truststore type for connection to DSE when SSL is enabled. JKS is the type for keys generated by the Java keytool binary, but other types are possible, depending on user environment. For example, --truststore-type jks2.

dsetool core_indexing_status
Retrieves the dynamic indexing status of a search index on a DSE Search node and displays the percent complete and an estimated completion time in milliseconds.

   Restriction: Command is supported only on nodes with DSE Search workloads.

Synopsis

```bash
$ dsetool core_indexing_status [keyspace_name.]table_name [--all] [--progress]
```
## Table 373: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Retrieves the dynamic indexing status (INDEXING, FINISHED, or FAILED) of the specified index or indexes, where:

**[keyspace_name.]table_name**

The search index table name is required. The keyspace name is optional. The case of keyspace and table names is preserved. You must use the correct case for the keyspace and table names.

**--all**

Retrieve the dynamic indexing status of the specified search index on all nodes.

**--progress**
DataStax Enterprise tools

Display the percent complete and an estimated completion time in milliseconds. This option is ignored and is assumed true. The command always displays the percent complete and an estimated completion time in milliseconds.

See Verifying indexing status (page 344).

Examples

These examples use the demo keyspace and health_data table.

To view the indexing status for the local node:

```
$ dsetool core_indexing_status demo.health_data
```

The results are displayed:

```
[demo.health_data]: INDEXING, 38% complete, ETA 452303 milliseconds (7 minutes 32 seconds)
```

To view the indexing status for a search index on a specified node:

```
$ dsetool -h 200.192.10.11 core_indexing_status demo.health_data
```

To view indexing status of all search indexes in the data center:

```
$ dsetool core_indexing_status demo.health_data --all
```

The results are displayed for 3 nodes in the data center:

<table>
<thead>
<tr>
<th>Address</th>
<th>Core Indexing Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>200.192.10.11</td>
<td>FINISHED</td>
</tr>
<tr>
<td>200.192.10.12</td>
<td>FINISHED</td>
</tr>
<tr>
<td>200.192.10.23</td>
<td>FINISHED</td>
</tr>
</tbody>
</table>

**dsetool create_core**

Creates the search index table on the local node. Supports DSE authentication with `-l username -p password`.

The CQL command to create a search index is CREATE SEARCH INDEX.

**Restriction:** Command is supported only on nodes with DSE Search workloads. Auto-generated schemas have default DocValues enabled. See Creating a search index with default values for details on docValues.

**Note:** If one or more nodes fail to create the core in distributed operations, an error message indicates the failing node or nodes. If it failed to create the core immediately, issue the create again. If it failed to create on some nodes, issue a reload for those nodes to load the newly created core.
Synopsis

```
$ dsetool create_core keyspace_name.table_name
[coreOptions=yamlFile | coreOptionsInline=key1:value1#key2:value2#...]
[distributed=true|false]
[generateResources=true|false]
[schema=path]
[solrconfig=path]
```

Table 374: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

keyspace_name.table_name
Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**coreOptions=yamlFile**
When generateResources=true, specify a customized YAML-formatted file of options. The contents of the file are the same options that can be specified with coreOptionsInline. See Changing auto-generated search index settings (page 340).

**coreOptionsInline=key1:value1#key2:value2#...**
Use this key-value pair syntax `key1:value1#key2:value2#` to specify values for these settings: See Changing auto-generated search index settings (page 340).

**auto_soft_commit_max_time:ms**
The maximum auto soft commit time in milliseconds.

**default_query_field:field**
The query field to use when no field is specified in queries.

**enable_string_copy_fields:( true | false )**
true | false - Generate non-stored string copy fields for non-key text fields. Text data can be tokenized or non tokenized. True creates a non-stored, non-tokenized copy field, so that you can have text both ways. Default: false.

**exclude_columns: col1, col2, col3, ...**
A comma-separated (CSV) list of columns to exclude.

**generate_DocValues_for_fields: ( * | field1, field2, ...**
Specify the fields to automatically configure DocValues in the generated search index schema. Specify '*' to add all possible fields:

```
generate_DocValues_for_fields: '*'
```

or specify a comma-separated list of fields, for example:

```
generate_DocValues_for_fields: uuidfield, bigintfield
```

**Note:** Solr does not support DocValue on boolean fields.

**distributed=true | false**
Whether to distribute and apply the operation to all nodes in the local datacenter.

- True applies the operation to all nodes in the local datacenter.
- False applies the operation only to the node it was sent to. False works only when recovery=true.

Default: true

**Warning:** Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

**generateResources=true | false**
Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with schema= and solrconfig=.

Valid values:

- true - Automatically generate search index schema and configuration resources if resources do not already exist.
false - Default. Do not automatically generate search index resources.

**include_columns**
A comma-separated (CSV) list of columns to include. Empty = includes all columns.

**index_merge_factor**
How many segments of equal size to build before merging them into a single segment.

**index_ram_buffer_size**
The index ram buffer size in megabytes (MB).

**lenient**
Ignore non-supported type columns and continue to generate resources, instead of erroring out when non-supported type columns are encountered. Default: false

**resource_generation_profiles**
To minimize index size, specify a CSV list of profiles to apply while generating resources.

### Table 375: Resource generation profiles

<table>
<thead>
<tr>
<th>Profile name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceSavingAll</td>
<td>Applies spaceSavingNoJoin and spaceSavingSlowTriePrecision profiles.</td>
</tr>
<tr>
<td>spaceSavingNoJoin</td>
<td>Do not index a hidden primary key field. Prevents joins across cores.</td>
</tr>
<tr>
<td>spaceSavingSlowTriePrecision</td>
<td>trie fields precisionStep to '0', allowing for greater space saving but slower querying.</td>
</tr>
</tbody>
</table>

**Note:** Using spaceSavings profiles disables auto generation of DocValues.

For example:

```
resource_generation_profiles: spaceSavingNoJoin,
spaceSavingSlowTriePrecision
```

**rt**
Enable live indexing to increase indexing throughput. Enable live indexing on only one search index per cluster.

```
rt=true
```

**recovery**
Whether to delete and recreate the search index if it is not able to load due to corruption. Valid values:

- true - If search index is unable to load, recover the index by deleting and recreating it.
- false - Default. No recovery.

**reindex**
Whether to reindex the data when search indexes are auto-generated with generateResources=true. Reindex works on a datacenter (DC) level. Reindex only
once per search-enabled DC. Repeat the reindex command on other data centers as required.

Valid values:
- true - Default. Reindexes the data. Accepts reads and keeps the current search index while the new index is building.
- false - Does not reindex the data. You can check and customize search index resources before indexing.

**schema=path**
Path of the UTF-8 encoded search index schema file. Cannot be specified when generateResources=true.

**solrconfig=path**
Path of the UTF-8 encoded search index configuration file. Cannot be specified when generateResources=true.

**Examples**

**Automatically generate search index for the health_data table in the demo keyspace**

```bash
$ dsetool create_core demo.health_data generateResources=true
```

**Override the default and reindex existing data, specify the reindex=true option**

```bash
$ dsetool create_core demo.health_data generateResources=true reindex=true
```

The `generateResources=true` option generates resources only if resources do not exist in the `solr_resources` table.

**Use options in a YAML-formatted file**

To turn on live indexing, also known as real-time (RT) indexing, the contents of the `rt.yaml` file are:

```
rt: true
```

```bash
$ dsetool create_core udt_ks.users generateResources=true reindex=true coreOptions=rt.yaml
```

**Enable encryption with inline options**

Specify the class for directoryFactory to `solr.EncryptedFSDirectoryFactory`:

```bash
$ dsetool create_core keyspace_name.table_name generateResources=true coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```
$ dsetool create_core demo.health_data generateResources=true
coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"

dsetool createsystemkey

Creates an encryption/decryption key for transparent data encryption (TDE).

See Transparent data encryption.

Synopsis

$ dsetool createsystemkey
[cipher_algorithm[/mode/padding]
[length] [key_name]
[-d filepath] [-k=kmip_groupname
[-t kmip_template] [-n namespace]]

Table 376: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens (<code>--</code>). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks (‘’) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### cipher_algorithm[/mode/padding]

DSE supports the following JCE cipher algorithms:

- AES/CBC/PKCS5Padding (valid with length 128, 192, or 256).
- AES/ECB/PKCS5Padding (valid with length 128, 192, or 256)
- DES/CBC/PKCS5Padding (valid with length 56)
- DESede/CBC/PKCS5Padding (valid with length 112 or 168)
- Blowfish/CBC/PKCS5Padding (valid with length 32-448)
- RC2/CBC/PKCS5Padding (valid with length 40-128)

Default value: AES/CBC/PKCS5Padding (with length 128)

### `-d filepath, --directory filepath`

Key file output directory. Enables creating key files before DSE is installed. This option is typically used by IT automation tools like Ansible. When no directory is specified, keys are saved to the value of `system_key_directory` (page 114) in `dse.yaml`.

### `length`

Required if cipher_algorithm is specified. Key length is not required for HMAC algorithms. Default value: 128 (with the default cipher algorithm AES/CBC/PKCS5Padding)

### `key_name`

Unique file name for the generated system key file. Encryption key files can have any valid Unix name. When no name is specified, the default file name is `system_key`. The default key file name is not configurable.

### `-k=kmp_groupname`

The name of the KMIP group that is defined in the `kmip_hosts` (page 115) section of `dse.yaml`.

### `-t kmip_template`

The key template on the specified KMIP provider.

### `-n namespace`

Namespace on the specified KMIP provider.

**Examples**

**To create a local key file:**

```
$ dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 system_key2
```
where system_key2 is the unique file name for the generated key file.

**To create an off-server key file:**

```
$ dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 system_key2 -kmip=group2
```

where group2 is the key server group defined in the `kmip_hosts (page 115)` section of `dse.yaml`.

**To create a local key file in a specific directory:**

```
$ dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 -d /mydir
```

See [Setting up local encryption keys](#).

**dsetool encryptconfigvalue**

Encrypts sensitive configuration information. This command takes no arguments and prompts for the value to encrypt.

**Example**

```
$ dsetool encryptconfigvalue
```

**dsetool get_core_config**

Displays the XML for the specified search index config. Supports DSE authentication with `[-l username -p password]`.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

**Synopsis**

```
$ dsetool get_core_config keyspace_name.table_name [current=true|false]
```

**Table 377: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**keyspace_name.table_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**current=true|false**

Optionally specify to view the current (active) configuration.

- true - Returns the active live search index config.
- false - Default. Returns the pending (latest uploaded) search index configuration.

### Examples

The following examples view the search index config for the demo keyspace and health_data table.

To view the pending (latest uploaded) configuration:

```bash
$ dsetool get_core_config demo.health_data
```
DataStax Enterprise tools

The XML for the auto-generated configuration is displayed:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<config>
  <abortOnConfigurationError>${solr.abortOnConfigurationError:true}</abortOnConfigurationError>
  <luceneMatchVersion>LUCENE_6_0_0</luceneMatchVersion>
  <dseTypeMappingVersion>2</dseTypeMappingVersion>
  <directoryFactory class="solr.StandardDirectoryFactory" name="DirectoryFactory"/>
  <indexConfig>
    <rt>false</rt>
    <rtOffheapPostings>true</rtOffheapPostings>
    <useCompoundFile>false</useCompoundFile>
    <ramBufferSizeMB>512</ramBufferSizeMB>
    ...
  </indexConfig>
  <requestHandler>
    <admin>
      <defaultQuery>*:*</defaultQuery>
    </admin>
  </requestHandler>
</config>
```

To view the pending (latest uploaded) search index configuration:

```bash
$ dsetool get_core_config demo.health_data current=true
```

To save the XML output to a file:

```bash
$ dsetool get_core_config demo.health_data > /Users/maryjoe/Documents/search/health_data_config.xml
```

The health_data_config.xml file is created.

**dsetool get_core_schema**

Displays the XML for the pending or active search index schema. Supports DSE authentication with `-l username -p password`.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

**Synopsis**

```bash
$ dsetool get_core_schema keyspace_name.table_name [current=true|false]
```

**Table 378: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**keyspace_name.table_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**current=true|false**

Optionally specify to view the current (active) schema.

- true - Returns the current live search index schema.
- false - Default. Returns the latest uploaded search index schema.

**Examples**

The following examples view the search index schema for the demo keyspace and health_data table.
To save the XML output to a file:

$ dsetool get_core_schema demo.health_data > /Users/maryjoe/Documents/search/health_data_schema.xml

The health_data_schema.xml file is created.

To view the pending (latest uploaded) search index schema:

$ dsetool get_core_schema demo.health_data

To view the active (currently loaded) search index schema:

$ dsetool get_core_schema demo.health_data current=true

The XML for the schema is displayed:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
  <types>
    <fieldType class="org.apache.solr.schema.TextField" name="TextField">
      <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <filter class="solr.LowerCaseFilterFactory"/>
      </analyzer>
    </fieldType>
    <fieldType class="org.apache.solr.schema.TrieIntField" name="TrieIntField"/>
  </types>
  <fields>
    <field indexed="true" multiValued="false" name="grade_completed" stored="true" type="TextField"/>
    ...
    <field indexed="true" multiValued="false" name="fips" stored="true" type="TextField"/>
  </fields>
  <uniqueKey>(id,age)</uniqueKey>
</schema>
```

dsetool help

Provides a listing of dsetool commands and parameters.

Synopsis

$ dsetool help
## Table 379: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Typing `dsetool` or `dsetool help` provides a listing of dsetool commands and parameters.

**Note:** Help is not available on a single command.

### dsetool index_checks (experimental)

Optional and experimental. Reads the full index and optionally performs sanity checks. No repairs or fixes occur. Run only when index is inactive. No writes are allowed while index check is running.
**Note:** Running this index check is time consuming and implies a hard commit.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

**Synopsis**

```bash
$ dsetool index_checks keyspace_name.table_name
[coreOptions=yamlFilepath] | [coreOptionsInline=options]
--index_checks=true|false
--index_checks_stop=true|false
```

**Table 380: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis (... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>’&lt;schema&gt; ... &lt;/schema&gt; ‘</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### keyspace_name.table_name

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

### coreOptions=yamlFilepath

When auto-generation is on with generateResources=true, the file path to a customized YAML-formatted file of options. See Changing auto-generated search index settings (page 340).

### coreOptionsInline=key1:value1#key2:value2#...

Use this key-value pair syntax `key1:value1#key2:value2#` to specify values for these settings:

- `auto_soft_commit_max_time:ms`
- `default_query_field:field`
- `distributed:( true | false )`
- `enable_string_copy_fields:( true | false )`
- `exclude_columns: col1, col2, col3, ...`
- `generate_DocValues_for_fields:( * | field1, field2, ... )`
- `generateResources:( true | false )`

See Changing auto-generated search index settings (page 340).

### --index_checks=true|false

Specify to run the index check.

- `true` - Runs the index check to verify index integrity. Reads the full index and has performance impact.
- `false` - Default. Does not run the index check.

### --index_checks_stop=true|false

Specify to stop the index check.

- `true` - Requests the index check to stop.
- `false` - Does not stop the index check.

### Examples

**Important**: Ensure that indexing is inactive before doing an index check.

#### To do an index check:

```
$ dsetool index_checks demo.health_data
```

The LUKE handler information is displayed:

```
LUKE handler info:
```
**dsetool infer_solr_schema**

Automatically infers and proposes a schema that is based on the specified keyspace and table. Search indexes are not modified. Supports DSE authentication with `[-l username -p password]`.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

**Synopsis**

```bash
$ dsetool infer_solr_schema keyspace_name.table_name
[coreOptions=yamlFilepath]| [coreOptionsInline=options]
```

**Table 381: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

#### keyspace_name.table_name

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

#### coreOptions=yamlFilepath

When auto-generation is on with generateResources=true, the file path to a customized YAML-formatted file of options. See Changing auto-generated search index settings (page 340).

#### coreOptionsInline=key1:value1#key2:value2#...

Use this key-value pair syntax key1:value1#key2:value2# to specify values for these settings:

- auto_soft_commit_max_time:ms
- default_query_field:field
- distributed:( true | false )
- enable_string_copy_fields:( true | false )
- exclude_columns: col1, col2, col3, ...
- generate_DocValues_for_fields:( * | field1, field2, ... )
- generateResources:( true | false )

See Changing auto-generated search index settings (page 340).

### Examples

To automatically infer and propose a schema that is based on the specified keyspace and table with the tuples and UDTs, specify the keyspace and table that contains tuples and UDTs:
dsetool inmemorystatus

Provides the memory size, capacity, and percentage for this node and the amount of memory each table is using. The unit of measurement is MB. Bytes are truncated.

Synopsis

$ dsetool inmemorystatus [keyspace_name.table_name]

Table 382: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>&quot;Literal string&quot;</td>
<td>Single quotation (‘) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>&quot;&lt;schema&gt; ... &lt;/schema&gt;&quot;</td>
<td>Search CQL only: Single quotation marks (‘) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
Syntax conventions | Description
--- | ---
`@xml_entity='xml_entity_type'` | Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

[keystore_name.table_name]
The keyspace name and table name.

Examples

To view the status for all tables:

```bash
$dsetool inmemorystatus
```

The results for all tables are displayed:

<table>
<thead>
<tr>
<th>Max Memory to Lock:</th>
<th>3276MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Total Memory Locked:</td>
<td>0MB</td>
</tr>
<tr>
<td>Current Total Memory Not Able To Lock:</td>
<td>0MB</td>
</tr>
<tr>
<td>No MemoryOnlyStrategy tables found.</td>
<td></td>
</tr>
</tbody>
</table>

To view the status for a specific table:

```bash
$dsetool inmemorystatus demo.health_data
```

**dsetool insights_config**

Enables and disables DSE Metrics Collector and configures reporting frequency and storage options.

Synopsis

```bash
$dsetool insights_config
--show_config | --mode [DISABLED|ENABLED_NO_STORAGE| ENABLED_WITH_LOCAL_STORAGE]
--metric_sampling_interval_in_seconds seconds
--config_refresh_interval_in_seconds seconds
--data_dir_max_size_in_mb dir_size
--node_system_info_report_period ISO-8601_duration_string
```

Table 383: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**--config_refresh_interval_in_seconds seconds**

How often the DSE Metrics Collector configuration changes are pushed to all nodes in the cluster. If nodes are down when a change is made, the change will propagate when the node is back up.

Default: 30

**--data_dir_max_size_in_mb mb**

When local storage is enabled, the limit on how much DSE Metrics Collector data will be stored on disk. The maximum size of the data directory must not exceed 2 GB.

Default: 1024 (1 GB)

**--metric_sampling_interval_in_seconds seconds**

The frequency that metrics are reported to DSE Metrics Collector.

Default: 30

**--mode**

Enables and disables DSE Metrics Collector (page 893) and configures storage options:
• **DISABLED** - disables metrics collection.
• **ENABLED_NO_STORAGE** - enables metrics collection and starts reporting metrics. Typically used when collectd is configured to report to a real-time monitoring system.
• **ENABLED_WITH_LOCAL_STORAGE** - enables metrics collection and reporting with local storage on disk. Default.

**--node_system_info_report_period duration**
The repeating time interval, in ISO-8601 format, for gathering diagnostic information about the node. For example, PT1H is 1 hour, PT5M is 5 minutes, and PTM200S is 200 seconds.

Default: PT1H (1 hour)

**--show_config**
Prints the current configuration for DSE Metrics Collector.

Examples

To view the current DSE Metrics Collector configuration

```
$ dsetool insights_config --show_config
```

The results of the default configuration:

```
{
    "mode" : "DISABLED",
    "config_refresh_interval_in_seconds" : 30,
    "metric_sampling_interval_in_seconds" : 30,
    "data_dir_max_size_in_mb" : 1024,
    "node_system_info_report_period" : "PT1H"
}
```

To enable metrics collection when collectd is configured to report to a real-time monitoring system

```
$ dsetool insights_config --mode ENABLED_NO_STORAGE
```

To enable metrics collection with local storage

```
$ dsetool insights_config --mode ENABLED_WITH_LOCAL_STORAGE
```

To configure 1500 MB for the DSE Metrics Collector local data directory

```
$ dsetool insights_config --data_dir_max_size_in_mb 1500
```

**Note:** The maximum size of the local data directory must not exceed 2 GB.

To change the node system reporting duration to 1 week

```
```
Use a ISO-8601 time duration string.

```
$ dsetool insights_config --node_system_info_report_period P1W
```

**To disable metrics collection**

```
$ dsetool insights_config --mode DISABLED
```

**To configure the metric sampling interval for 60 seconds**

```
$ dsetool insights_config --metric_sampling_interval_in_seconds 60
```

**To configure 120 seconds for the configuration refresh interval**

Push configuration changes to all nodes in the cluster every 2 minutes:

```
$ dsetool insights_config --config_refresh_interval_in_seconds 120
```

### dsetool insights_filters

Configures filters to include and exclude specific metrics for DSE Metrics Collector.

By default, the following metrics are always excluded:

- Thread Per Core (TPC) metrics at each core level
- Keystore level metrics
- DSE internal table metrics (except system_auth, paxos, and batchlog metrics)

Use a regular expression (regex) to specify which metrics to include or exclude from the filter. See [Filtering metrics](page 897).

**Synopsis**

```
$ dsetool insights_filters
--show_filters | --remove_all_filters |
--add --global|--insights_only --allow regex | --deny regex|
--remove --global | --insights_only --allow regex | --deny regex
```

#### Table 384: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

---

**--show_filters**
- Prints the current filters for DSE Metrics Collector.

**--remove_all_filters**
- Remove all metrics filters for DSE Metrics Collector.

**--add --global|--insights_only regex**
- Include metrics that match this regular expression and apply the filter with scope of --global or --insights_only.

**--deny --global|--insights_only regex**
- Exclude metrics that match this regular expression and apply the filter with scope of --global or --insights_only.

**--global**
- Metrics filter scope includes metrics reported locally and insights data files.

**--insights_only**
- Limit metrics filter scope to insights data files only. Appropriate for diagnostic use.
Example: Example filters

**Show all active filters**

```
$ dsetool insights_filters --show_filters
```

**Remove all active filters**

```
$ dsetool insights_filters --remove_all_filters
```

**Add a global filter to deny all metrics matching KeyspaceMetrics**

```
```

**Remove a global filter to deny metrics for a specific keyspace**

```
$ dsetool insights_filters --remove --global --deny "org\.apache\.cassandra\..metrics\.(keyspace|table)\..\+(name_of_keyspace)\.."
```

**Add a filter to insights data files that deny grace period metrics**

```
$ dsetool insights_filters --add --insights_only --deny .+gc.+
```

---

**dsetool list_index_files**

Lists all index files for a search index on the local node. The results show file name, encryption, disk usage, decrypted size, and encryption overhead. The index file is encrypted only when the backing CQL table is encrypted and the search index uses EncryptedFSDirectoryFactory; otherwise, the index file is decrypted.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

**Synopsis**

```
$ dsetool list_index_files keyspace_name table_name [--index directory]
```

**Table 385: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td><strong>Syntax conventions</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**keyspace_name.table_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**--index**

The data directory that contains the index files.

- If not specified, the default directory is inferred from the search index name.
- directory - A specified file path to the solr.data directory that contains the search index files.
Examples

To list the index files:

$ dsetool list_index_files

The results show file name, encryption, disk usage, decrypted size, and encryption overhead:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Encryption</th>
<th>Disk usage</th>
<th>Encryption overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>segments_1</td>
<td>N/A</td>
<td>7124 bytes</td>
<td>N/A</td>
</tr>
<tr>
<td>write.lock</td>
<td>N/A</td>
<td>3240 bytes</td>
<td>N/A</td>
</tr>
</tbody>
</table>

To list the index files in a specified directory:

$ dsetool list_index_files /My_data_dir

dsetool list_core_properties

Lists the properties and values in the dse-search.properties resource for the search index.

Tip: See Load balancing for distributed search queries.

Synopsis

$ dsetool list_core_properties

Table 386: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>...</code></td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td><code>'Literal string'</code></td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; ' </code></td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type' </code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### keyspace_name.table_name

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

### Examples

To view properties set in the dse-search.properties resource:

```
$ dsetool list_core_properties demo.health_data
```

Result for default values:

```
shard.shuffling.strategy=RANDOM
```

### dsetool list_subranges

Lists the subranges of data in a keyspace by dividing a token range into a number of smaller subranges. Useful when the specified range is contained in the target node’s primary range.

#### Synopsis

```
$ dsetool list_subranges keyspace_name table_name keys_per_range start_token end_token
```
### Table 387: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Literal keyword.</td>
</tr>
<tr>
<td><strong>Lowercase</strong></td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**keyspace_name table_name**  
Keypspace table pair. 

**keys_per_range**  
The approximate number of rows per subrange. 

**start_token**  
The start token of a specified range of tokens. 

**end_token**  
The end token of a specified range of tokens.
Example
To run the command:

```
$ dsetool list_subranges demo health_data 10000
113427455640312821154458202477256070485 0
```

The subranges are output and can be used as input to the `nodetool repair` \(\text{(page 1054)}\) command.

<table>
<thead>
<tr>
<th>Start Token</th>
<th>End Token</th>
<th>Estimated Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1134274556403128211544582024</td>
<td>132425442795624521227151664</td>
<td>11264</td>
</tr>
<tr>
<td>132425442795624521227151664</td>
<td>15140957604838922734725799</td>
<td>11136</td>
</tr>
<tr>
<td>15140957604838922734725799</td>
<td>0</td>
<td>11264</td>
</tr>
</tbody>
</table>

**dsetool listjt**

Lists all Job Tracker nodes grouped by the datacenter that is local to them.

**Restriction:** Command is supported only on nodes with analytics workloads.

**Synopsis**

```
$ dsetool listjt
```

**Table 388: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens (--). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; ' </code></td>
<td>Search CQL only: Single quotation marks (‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

This command takes no arguments.

**Examples**

```bash
$ dsetool listjt
```

### dsetool managekmip list

Verifies communication with the specified Key Management Interoperability Protocol (KMIP) server and lists the encryption/decryption keys on that server.

**Synopsis**

```bash
$ dsetool managekmip list kmip_group_name [namespace=key_namespace]
```

**Table 389: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td><code>[ ]</code></td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td><code>( )</code></td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
</tbody>
</table>
**Syntax conventions**

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or. A vertical bar (</td>
<td>) separates alternative elements. Type any one of the elements. Do not type the vertical bar.</td>
<td></td>
</tr>
<tr>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
<td></td>
<td><code>...</code></td>
</tr>
<tr>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
<td>‘Literal string’</td>
<td><code>'Literal string'</code></td>
</tr>
<tr>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
<td><code>{ key:value }</code></td>
<td><code>{ key:value }</code></td>
</tr>
<tr>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
</tr>
<tr>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
<td><code>cql_statement;</code></td>
<td><code>cql_statement;</code></td>
</tr>
<tr>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
<td>[ -- ]</td>
<td>[ -- ]</td>
</tr>
<tr>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
<td><code>'&lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td><code>'&lt;schema&gt; ... &lt;/schema&gt;'</code></td>
</tr>
<tr>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td><code>@xml_entity='xml_entity_type'</code></td>
</tr>
</tbody>
</table>

**kmip_groupname**

The user-defined name of the KMIP group that is configured in the kmip_hosts (page 115) section of dse.yaml.

**namespace=key_namespace**

Namespace on the specified KMIP provider.

**Examples**

**Get a list of the available keys and states from the KMIP server:**

```
$ dsetool managekmip list kmipgrouptwo
```

The results show that the KMIP server named vormetricgroup has two keys:

<table>
<thead>
<tr>
<th>Keys on vormetricgroup:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>Protect Stop Date</td>
</tr>
<tr>
<td>02-449</td>
</tr>
<tr>
<td>Deactivated</td>
</tr>
</tbody>
</table>
**dsetool managekmip expirekey**

Expires encryption/decryption keys on a Key Management Interoperability Protocol (KMIP) server. Database stops using the key for encryption at the specified time and continues to use the expired key to decrypt existing data. Data re-keying is not required. Use this command to satisfy security policies that require periodically switching the encryption key.

DataStax recommends following best practices for key management permission policies. See [Expiring an encryption key](#).

**Synopsis**

```bash
$ dsetool managekmip expirekey kmip_group_name kmip_key_id [date_time]
```

**Table 390: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;schema&gt; ... &lt;/schema&gt;</code></td>
<td>Search CQL only: Single quotation marks (<code>'</code>) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
| `@xml_entity='xml_entity_type'` | Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

### kmip_groupname

The user-defined name of the KMIP group that is configured in the `kmip_hosts` section of `dse.yaml`.

### kmip_key_id

The key id on the KMIP provider.

### date_time

After the specified date_time, new data will not be encrypted with the key. Data can be decrypted with the key after this expire date/time. Format of datetime is YYYY-MM-DD HH:MM:SS:T. For example, use 2016-04-13 20:05:00:0 to expire the encryption key at 8:05 p.m. on 13 April 2016.

### Examples

#### To immediately expire an encryption key:

```bash
$ dsetool managekmip expirekey kmipgrouptwo 02-540
```

Encryption for new data is prevented, but decryption with the key is still allowed. Because the expire date/time is not specified, the key is expired immediately.

#### To expire an encryption key at a specific date and time:

```bash
$ dsetool managekmip expirekey kmipgrouptwo 02-540 2017-04-13 20:05:00:0
```

### dsetool managekmip revoke

Permanently disables the key on the KMIP server. Database can no longer use the key for encryption, but continues to use the key for decryption of existing data. Re-encrypt existing data before completely removing the key from the KMIP server. Use this command as the first step when replacing a compromised key.

### Synopsis

```bash
$ dsetool managekmip revoke kmip_group_name kmip_key_id
```

### Table 391: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key: value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**kmip_groupname**

The user-defined name of the KMIP group that is configured in the kmip_hosts (page 115) section of dse.yaml.

**kmip_key_id**

The key id on the KMIP provider.

**Examples**

**To revoke a key to prevent decryption:**
dsetool managekmip destroy

Completely removes the key from the KMIP server. Database can no longer use the key for encryption or decryption. Existing data that has not been re-encrypted becomes inaccessible.

**Important:** Use this command only after revoking a key and re-encrypting existing data.

**Synopsis**

```
$ dsetool managekmip destroy kmip_group_name kmip_key_id
```

**Table 392: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks (`) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### kmip_groupname

The user-defined name of the KMIP group that is configured in the `kmip_hosts` (page 115) section of `dse.yaml`.

### kmip_key_id

The key id on the KMIP provider.

### Examples

**To revoke a key to prevent decryption:**

```bash
$ dsetool managekmip revoke kmipgrouptwo 02-540
```

**After you revoke a key, you can destroy it:**

```bash
$ dsetool managekmip destroy kmipgrouptwo 02-540
```

### dsetool node_health

Retrieves a dynamic score between 0 and 1 that describes the health of a DataStax Enterprise node. Node health is a score-based representation of how fit a node is to handle search queries. The node health composite score is based on dropped mutations and uptime. A higher score indicates better node health. Nodes that have a large number of dropped mutations and nodes that are just started have a lower health score.

See Collecting node health and indexing status scores.

### Synopsis

```bash
$ dsetool node_health [--all]
```

### Table 393: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key: value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1, datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

---

**Examples**

**To retrieve the health score of the local node:**

```
$ dsetool node_health
```

The result displays a number between 0 and 1:

```
Node Health [0,1]: 0.7
```

**To retrieve the health score of a specified node:**

```
$ dsetool -h 200.192.10.11 node_health
```

**To retrieve the health score of all nodes:**

```
$ dsetool node_health --all

dsetool partitioner

Returns the fully qualified classname of the IPartitioner that is used by the cluster.

Synopsis

$ dsetool partitioner

Table 394: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
This command takes no arguments.

Examples

```
$ dsetool partitioner
```

The partitioner in use is displayed:

```
org.apache.cassandra.dht.Murmur3Partitioner
```

**dsetool perf**

Temporarily changes the running parameters for the CQL Performance Service. Histogram tables provide DSE statistics that can be queried with CQL.

Changes made with performance object subcommands do not persist between restarts and are useful only for short-term diagnostics.

**Note:** To make these changes permanent, change the CQL Performance Service options (page 122) in dse.yaml.

See DSE Performance Service diagnostic table reference and Collecting histogram diagnostics.

**Synopsis**

```
$ dsetool perf subcommand values
```

**Table 395: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces (<code>{ }</code>) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (<code>&lt; &gt;</code>) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (<code>;</code>) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens (<code>--</code>). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; </code>'</td>
<td>Search CQL only: Single quotation marks (<code>'</code>) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### clustersummary enable|disable
Whether to enable the collection of database-level statistics for the cluster.

### cqlslowlog enable|disable
Whether to enable the collection of CQL queries that exceed the specified time threshold.

### cqlslowlog threshold
The CQL slow log threshold as a percentile of the actual request times:
- `[0,1]` is a percentile threshold
- `>1` is an absolute threshold in milliseconds
- `1.0` logs no queries
- `99.9` logs 0.1% of the slowest queries
- `95.0` logs 5% of the slowest queries
- `50.0` logs 50% of the slowest queries
- `0.0` logs all queries

### cqlslowlog skip_writing_to_db
Keeps slow queries in-memory only.

### cqlslowlog write_to_db
Writes data to the database. When data writes to the database, the threshold must be `>= 2000 ms` to prevent a high load on database.

Temporary equivalent of `cql_slow_log_options.skip_writing_to_db: false` setting in `dse.yaml` (page 123).

### cqlslowlog set_num_slowest_queries
The number of slow queries to keep in-memory.

### cqlslowlog recent_slowest_queries
The specified number of the most recent slow queries to retrieve.

### cqlsysteminfo enable|disable
Whether to collect CQL system performance information statistics.

**dbsummary enable|disable**
Whether to collect database summary statistics.

**histograms enable|disable**
Whether to collect table histograms that measure the distribution of values in a stream of data. Histogram tables provide DSE statistics that can be queried with CQL. The data in the diagnostic histogram tables is cumulative since the DSE server was started.

**resourcelatencytracking enable|disable**
Whether to collect resource latency tracking statistics.

**solrcachestats enable|disable**
Whether to collect Solr cache statistics.

**solrindexingerrorlog enable|disable**
Whether to log Solr indexing errors.

**solrindexstats enable|disable**
Whether to collect Solr indexing statistics.

**solrlatencysnapshots enable|disable**
Whether to collect Solr latency snapshots.

**solrrequesthandlerstats enable|disable**
Whether to collect Solr request handler statistics.

**solrslowlog threshold enable|disable**
Whether to log the Solr slow sub-query log and set the Solr slow log threshold in milliseconds.

**solrupdatehandlerstats enable|disable**
Whether to collect Solr update handler statistics.

**userlatencytracking enable|disable**
Whether to enable user latency tracking.

**Examples**

These example commands make temporarily changes only. Changes made with performance object subcommands do not persist between restarts and are useful only for short-term diagnostics.

See [Collecting database summary diagnostics](#).

**To enable the collection of database-level statistics data:**

```
$ dsetool perf clustersummary enable
```

**To disable the collection of database-level statistics data:**

```
$ dsetool perf clustersummary disable
```

See [Collecting slow queries](#).

**To keep slow queries in-memory only:**

```
$ dsetool perf cqlslowlog skip_writing_to_db

To set the number of slow queries to keep in-memory:

$ dsetool perf cqlslowlog set_num_slowest_queries 5

To write slow queries to the database:

$ dsetool perf cqlslowlog write_to_db

To disable collecting information on slow queries:

$ dsetool perf cqlslowlog disable

To change the threshold to collect information on 5% of the slowest queries:

$ dsetool perf cqlslowlog 95.0

To enable collecting information to identify slow search queries:

$ dsetool perf solrslowlog enable

To change the threshold value (in milliseconds) at which a sub-query is slow enough to be reported:

$ dsetool perf solrslowlog 200

dsetool read_resource

Reads the specified search index config or schema. Supports DSE authentication with [-l username -p password].

Restriction: Command is supported only on nodes with DSE Search workloads.

Synopsis

$ dsetool read_resource keyspace_name.table_name name=res_filename

Table 396: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**keyspace_name.table_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**res_filename**

The name of the search index resource file to read.

**Examples**

**To read the resource:**

```
$ dsetool read_resource demo.health_data stopwords.xml
```
DataStax Enterprise tools

After reading the resource, then upload the search index.

**dsetool rebuild_indexes**

Rebuilds secondary indexes.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

**Synopsis**

```
$ dsetool rebuild_indexes keyspace_name.table_name [index1,index2,...]
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td><em>[]</em></td>
<td>Optional. Square brackets ([]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td><em>( )</em></td>
<td>Group. Parentheses (( )) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>&quot;Literal string&quot;</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>&quot;&lt;schema&gt; ... &lt;/schema&gt;&quot;</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### keyspace_name.table_name

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

### index1,index2,...

Include one or a comma-separated list of secondary indexes to rebuild. If indexes are not specified, rebuilds all indexes.

#### Examples

**To rebuild all secondary indexes:**

```
$ dsetool rebuild_indexes demo.health_data
```

**To rebuild only the specified secondary indexes:**

```
$ dsetool rebuild_indexes demo.health_data index1, index2
```

### dsetool reload_core

Reloads the search index to recognize changes to schema or configuration. Supports DSE authentication with `-l username -p password`.

**Note:** To reload the core and prevent reindexing, accept the default values `reindex=false` and `deleteAll=false`.

See [Reloading the search index](#) for details.

#### Synopsis

```
$ dsetool reload_core keyspace_name.table_name
[coreOptions=yamlFile | coreOptionsInline=key1:value1#key2:value2#...] [deleteAll=true|false] [distributed=true|false] [reindex=(true|false)] [schema=path] [solrconfig=path]
```

#### Table 398: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
</tbody>
</table>
## Syntax conventions

<table>
<thead>
<tr>
<th><strong>Syntax conventions</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt; &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### keyspace_name.table_name

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

### schema=path

Path of the UTF-8 encoded search index schema file. Cannot be specified when generateResources=true.

### solrconfig=path

Path of the UTF-8 encoded search index configuration file. Cannot be specified when generateResources=true.

### distributed=(true | false)

Whether to distribute and apply the operation to all nodes in the local datacenter.

- True applies the operation to all nodes in the local datacenter.
• False applies the operation only to the node it was sent to. False works only when recovery=true.

**Warning:** Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

**reindex=(true | false)**

Whether to reindex the data when search indexes are auto-generated with generateResources=true. Reindex works on a datacenter (DC) level. Reindex only once per search-enabled DC. Repeat the reindex command on other data centers as required.

Valid values:

• true - Default. Reindexes the data. Accepts reads and keeps the current search index while the new index is building.
• false - Does not reindex the data. You can check and customize search index resources before indexing.

**deleteAll=(true|false)**

• true - deletes the already existing index before reindexing; search results will return either no or partial data while the index is rebuilding.
• false - does not delete the existing index, causing the reindex to happen in-place; search results will return partially incorrect results while the index is updating. Default.

During reindexing, a series of criteria routes sub-queries to the nodes most capable of handling them. See Shard routing for distributed queries.

**Examples**

**To make the pending search index active:**

```
$ dsetool reload_core demo.health_data
coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```

**To upload the changed resource file:**

```
$ dsetool reload_core demo.health_data
coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```

**dsetool ring**

Lists the nodes in the ring. For more readable output, use dsetool status.

**Synopsis**

```
$ dsetool ring
```
Table 399: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

This command requires no input.

Examples

```
$ dsetool ring
```

dsetool set_core_property

Sets the properties and values in the dse-search.properties resource for the search index.
Tip: See Load balancing for distributed search queries.

Synopsis

```
$ dsetool set_core_property keyspace_name.table_name
shard.set.cover.finder=DYNAMIC|STATIC
shard.shuffling.strategy=HOST|QUERY|HOST_QUERY|RANDOM|SEED
```

Table 400: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

`keyspace_name.table_name`
Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

For shard.set.cover.finder:

**DYNAMIC**
Use randomization in token range and endpoint selection for load balancing.
Default.

**STATIC**
Requires load balanced client. Suitable for 8+ vnodes. The same query on a node uses the same token ranges and endpoints. Creates fewer token filters, more performant than DYNAMIC.

When shard.set.cover.finder=DYNAMIC, values for shard.shuffling.strategy:

**HOST**
Shards are selected based on the host that received the query.

**QUERY**
Shards are selected based on the query string.

**HOST_QUERY**
Shards are selected by host x query.

**RANDOM**
Suitable only for 8 or fewer vnodes. Different random set of shards are selected with each request (default).

**SEED**
Selects the same shard from one query to another.

Examples

To not use randomization to select token ranges and endpoints:

```
$ dsetool set_core_property demo.health_data
   shard.set.cover.finder=STATIC
```

To use default randomization to select token ranges and endpoints:

```
$ dsetool set_core_property demo.health_data
   shard.set.cover.finder=RANDOM
```

**dsetool sparkmaster cleanup**

Drops and recreates the Spark Master recovery table.

**Synopsis**

```
$ dsetool sparkmaster cleanup [datacenter]
```
## Table 401: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

This command has an optional argument `datacenter`. If a datacenter is specified, it will remove the recovery data for that datacenter.

### Examples

```
$ dsetool sparkmaster cleanup
```
dsetool sparkworker restart

Manually restarts the Spark Worker on the selected node, without restarting the node.

Synopsis

$ dsetool sparkworker restart

Table 402: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
This command accepts no parameters.

Examples

```bash
$ dsetool sparkworker restart
```

**dsetool status**

Lists the nodes in their ring, including the node type and node health. When the datacenter workloads are the same type, the workload type is listed. When the datacenter workloads are heterogeneous, the workload type is shown as mixed.

**Synopsis**

```bash
$ dsetool status
```

**Table 403: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;schema&gt; ... &lt;/schema&gt;</td>
<td>Search CQL only: Single quotation marks (‘) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

This command accepts no parameters.

### Examples

```bash
$ dsetool status
```

## dsetool stop_core_reindex

Stops reindexing for the specified search index on the node where the command is run. Optionally, specify a timeout in minutes so that the core waits to stop reindexing until the specified timeout is reached, then gracefully stops the indexing. The default timeout is 1 minute.

### Synopsis

```bash
$ dsetool stop_core_reindex keyspace_name.table_name [timeout_min]
```

### Table 404: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation (‘) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( <code>&lt;</code> ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( <code>;</code> ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( <code>--</code> ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks ( <code>‘</code> ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### keyspace_name.table_name

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

### timeout_min

The number of minutes to wait to gracefully stop the indexing.

### Examples

**To stop reindexing after the default 1 minute timeout:**

```
$ dsetool stop_core_reindex demo.health_data
```

**To reindexing after 6 minutes:**

```
$ dsetool stop_core_reindex demo.health_data 6
```

### dsetool tieredtablestats

Outputs tiered storage information, including SSTables, tiers, timestamps, and sizes. Provides information on every table that uses tiered storage.

### Synopsis

```
$ dsetool tieredtablestats keyspace_name.table_name [-v]
```

### Table 405: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**keyspace_name.table_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

-v

Output statistics for each SSTable, in addition to the tier summaries.

**Examples**

**To monitor all tables using tiered storage:**

```
$ dsetool tieredtablestats
```
Output of command:

```
ks.tbl
  Tier 0:
  Summary:
    max_data_age: 1449178580284
    max_timestamp: 1449168678515945
    min_timestamp: 1449168678515945
    reads_120_min: 5.2188117172945374E-5
    reads_15_min: 4.415612774014863E-7
    size: 4839
  SSTables:
    /mnt2/ks/tbl-257cecf1988311e58be1ff4e6f1f6740/ma-3-big-Data.db:
      estimated_keys: 256
      level: 0
      max_data_age: 1449178580284
      max_timestamp: 1449168678515945
      min_timestamp: 1449168678515945
      reads_120_min: 5.2188117172945374E-5
      reads_15_min: 4.415612774014863E-7
      rows: 1
      size: 4839
  Tier 1:
  Summary:
    max_data_age: 1449178580284
    max_timestamp: 1449168749912092
    min_timestamp: 1449168749912092
    reads_120_min: 0.0
    reads_15_min: 0.0
    size: 4839
  SSTables:
    /mnt3/ks/tbl-257cecf1988311e58be1ff4e6f1f6740/ma-4-big-Data.db:
      estimated_keys: 256
      level: 0
      max_data_age: 1449178580284
      max_timestamp: 1449168749912092
      min_timestamp: 1449168749912092
      reads_120_min: 0.0
      reads_15_min: 0.0
      rows: 1
      size: 4839
```

To monitor the `health_data` table using tiered storage:

```
$ dsetool tieredtablestats demo.health_data
```

To monitor the `health_data` table with output for each SSTable:
$ dsetool tieredtablestats demo.health_data -v

dsetool tsreload

Reloads the truststores without a restart. Specify client or server.

Synopsis

$ dsetool tsreload client|server

Table 406: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>
client
Reloads the truststore that is used for encrypted client-to-node communications.

server
Reloads the server truststore that is used for encrypted node-to-node (internode) SSL communications.

dsetool unload_core

Removes a search index. Supports DSE authentication with \([-l \text{username} -p \text{password}]\).

To drop a search index from a table and delete all related data for the entire cluster, see DROP SEARCH INDEX.

The removal of the secondary index from the table schema is always distributed.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

**Synopsis**

```
$ dsetool unload_core keyspace_name.table_name
[deleteDataDir=(true|false)]
[deleteResources=(true|false)]
[distributed=(true|false)]
```

**Table 407: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( <code>&lt; </code> ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt; '</code></td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

#### deleteDataDir=( true | false )

Whether to delete index data and any other artifacts in the `solr.data` directory.

Valid values:
- true - Deletes index data and any other artifacts in the `solr.data` directory. It does not delete DataStax Enterprise data.
- false - Default. Does not delete index data or other artifacts.

#### deleteResources=( true | false )

Whether to delete the config and schema resources associated with the search index.

Valid values:
- true - Deletes index resources.
- false - Default. Does not delete index resources.

#### distributed=true | false

Whether to distribute and apply the operation to all nodes in the local datacenter.

- True applies the operation to all nodes in the local datacenter.
- False applies the operation only to the node it was sent to. False works only when recovery=true.

Default: true

**Warning:** Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

#### dsetool upgrade_index_files

Upgrades all DSE Search index files.

**Requirements:**
• The remote node that contains the encryption configuration must be running.
• The local node is offline.
• The user that runs this command must have read and write permissions to the directory that contains the index files.

Synopsis

```
$ dsetool upgrade_index_files keyspace_name.table_name
   -h IP_address [-c port]
   [--backup] [--workspace directory] [--index directory]
```

Table 408: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### keyspace_name.table_name

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

### -h IP_address

Required. Node hostname or IP address of the remote node that contains the encryption configuration that is used for index encryption. The remote node must be running.

### -c port

The DSE port on the remote node that contains the encryption configuration.

### --backup

Preserves the index files from the current index as a backup after successful upgrade. The preserved index file backup is moved to the --workspace directory. When not specified, index files from the current index are deleted.

### --workspace directory

The workspace directory for the upgrade process. The upgraded index is created in this directory. When not specified, the default directory is the same directory that contains the search index files.

### --index directory

The data directory that contains the search index files. When not specified, the default directory is inferred from the search index name.

### Examples

**To perform offline index encryption:**

```
$ dsetool upgrade_index_files demo.health_data
```

See [Migrating encrypted tables from earlier versions](#) and [Encrypting new Search indexes](#).

### dsetool write_resource

Uploads the specified search index config or schema.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

Resource files are stored internally in the database. You can configure the maximum resource file size or disable resource upload with the `resource_upload_limit` ([page 120](#)) option in `dse.yaml`.

Supports DSE authentication with `[-l username -p password]`. 
Synopsis

$ dsetool write_resource keyspace_name.table_name name=res_filename
        file=path_to_file_to_upload

Table 409: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ‘ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

keyspace_name.table_name

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

res_filename
DataStax Enterprise tools

The name of the search index resource file to upload.

file

The file path of the file to upload.

Examples

To write the resource:

```bash
$ dsetool write_resource demo.health_data stopwords.xml
```

To specify the uploaded resource file and the path to the resource file:

```bash
$ dsetool write_resource demo.health_data name=ResourceFile.xml file=/myPath1/myPath2/schemaFile.xml
```

DataStax Enterprise stress tools

**cassandra-stress tool**

The `cassandra-stress` tool is a Java-based stress testing utility for basic benchmarking and load testing a DataStax Enterprise cluster.

Data modeling choices can greatly affect application performance. Significant load testing over several trials is the best method for discovering issues with a particular data model. The `cassandra-stress` tool is an effective tool for populating a cluster and stress testing CQL tables and queries. Use `cassandra-stress` to:

- Quickly determine how a schema performs.
- Understand how your database scales.
- Optimize your data model and settings.
- Determine production capacity.

The `cassandra-stress` tool also supports a YAML-based profile for defining specific schemas with various compaction strategies, cache settings, and types. Sample files are located in the `tools` directory:

- `cqlstress-counter-example.yaml`
- `cqlstress-example.yaml`
- `cqlstress-insanity-example.yaml`

The YAML file supports user-defined keyspace, tables, and schema. The YAML file can be used to design tests of reads, writes, and mixed workloads.

When started without a YAML file, `cassandra-stress` creates a keyspace, `keyspace1`, and tables, `standard1` or `counter1`, depending on what type of table is being tested. These elements are automatically created the first time you run a stress test and reused on subsequent runs. You can drop `keyspace1` using `DROP KEYSPACE`. You cannot change the default keyspace and tables names without using a YAML file (page 1387).
Usage:

- Package installations:

  
  ```
  $ cassandra-stress command [options]
  ```

- Tarball installations:

  
  ```
  $ cd install_location/tools 
  bin/cassandra-stress command [options]
  ```

cassandra-stress options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter_read</td>
<td>Multiple concurrent reads of counters. The cluster must first be populated by a counter_write test.</td>
</tr>
<tr>
<td>counter_write</td>
<td>Multiple concurrent updates of counters.</td>
</tr>
<tr>
<td>help</td>
<td>Display help: cassandra-stress help</td>
</tr>
<tr>
<td>legacy</td>
<td>Legacy support mode.</td>
</tr>
<tr>
<td>mixed</td>
<td>Interleave basic commands with configurable ratio and distribution. The cluster must first be populated by a write test.</td>
</tr>
<tr>
<td>print</td>
<td>Inspect the output of a distribution definition.</td>
</tr>
<tr>
<td>read</td>
<td>Multiple concurrent reads. The cluster must first be populated by a write test.</td>
</tr>
<tr>
<td>user</td>
<td>Interleave user provided queries with configurable ratio and distribution.</td>
</tr>
<tr>
<td>version</td>
<td>Print the cassandra-stress version.</td>
</tr>
<tr>
<td>write</td>
<td>Multiple concurrent writes against the cluster.</td>
</tr>
</tbody>
</table>

**Important:** Additional sub-options are available for each option in the following table. To get more detailed information on any of these, enter:

```
$ cassandra-stress help option
```

When entering the help command, be sure to precede the option name with a hyphen, as shown.
<table>
<thead>
<tr>
<th>Sub-option</th>
<th>Description</th>
</tr>
</thead>
</table>
| -col       | Column details, such as size and count distribution, data generator, names, and comparator. **Usage:**
|            | -col names=? [slice] [super=?] [comparator=?] [timestamp=?] [size=DIST(?)]
|            | or
|            | -col [n=DIST(?)] [slice] [super=?] [comparator=?] [timestamp=?] [size=DIST(?)] |
| -errors    | How to handle errors when encountered during stress testing. **Usage:**
|            | -errors [retries=N] [ignore] [skip-read-validation]
|            | • retries=N Number of times to try each operation before failing.
|            | • ignore Do not fail on errors.
|            | • skip-read-validation Skip read validation and message output. |
| -graph     | Graph results of cassandra-stress tests. Multiple tests can be graphed together. **Usage:**
|            | -graph file=? [revision=?] [title=?] [op=?] |
| -insert    | Insert specific options relating to various methods for batching and splitting partition updates. **Usage:**
|            | -insert [revisit=DIST(?)] [visits=DIST(?)] partitions=DIST(?)
|            | [batchtype=?] select-ratio=DIST(?) row-population-ratio=DIST(?) |
| -log       | Where to log progress and the interval to use. **Usage:**
|            | -log [level=?] [no-summary] [file=?] [hdrfile=?] [interval=?]
<p>|            | [no-settings] [no-progress] [show-queries] [query-log-file=?] |</p>
<table>
<thead>
<tr>
<th>Sub-option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-mode</code></td>
<td>Thrift or CQL with options.</td>
</tr>
<tr>
<td></td>
<td><code>-mode thrift [smart] [user=?] [password=?]</code></td>
</tr>
<tr>
<td></td>
<td><code>-mode native [unprepared] cql3 [compression=?] [port=?]</code></td>
</tr>
<tr>
<td></td>
<td><code>-mode simplenative [prepared] cql3 [port=?]</code></td>
</tr>
<tr>
<td><code>-node</code></td>
<td>Nodes to connect to.</td>
</tr>
<tr>
<td></td>
<td><code>-node [datacenter=?] [whitelist] [file=?] []</code></td>
</tr>
<tr>
<td><code>-pop</code></td>
<td>Population distribution and intra-partition visit order.</td>
</tr>
<tr>
<td></td>
<td><code>-pop seq=? [no-wrap] [read-lookback=DIST(?)] [contents=?]</code></td>
</tr>
<tr>
<td></td>
<td><code>-pop [dist=DIST(?)] [contents=?]</code></td>
</tr>
<tr>
<td><code>-port</code></td>
<td>Specify port for connecting Cassandra nodes. Port can be specified for Cassandra native protocol, Thrift protocol or a JMX port for retrieving statistics.</td>
</tr>
<tr>
<td></td>
<td><code>-port [native=?] [thrift=?] [jmx=?]</code></td>
</tr>
<tr>
<td>Sub-option</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **-rate** | Set the rate using the following options:  
  
  `-rate threads=N [throttle=N] [fixed=N]`  
  
  where  
  
  - `threads=N` number of clients to run concurrently.  
  - `throttle=N` throttle operations per second across all clients to a maximum rate (or less) with no implied schedule. Default is 0.  
  - `fixed=N` expect fixed rate of operations per second across all clients with implied schedule. Default is 0.  
  
  OR  
  
  `-rate [threads>=N] [threads<=N] [auto]`  
  
  Where  
  
  - `threads>=N` run at least this many clients concurrently. Default is 4.  
  - `threads<=N` run at most this many clients concurrently. Default is 1000.  
  - `auto` stop increasing threads once throughput saturates. |
| **-schema** | Replication settings, compression, compaction, and so on.  
  Usage:  
  
  `-schema [replication(?)] [keyspace=?] [compaction(?)] [compression=?]` |
| **-sendto** | Specify a server to send the stress command to.  
  Usage:  
  
  `-sendto <host>` |
| **-tokenrange** | Token range settings.  
  Usage:  
  
  `-tokenrange [no-wrap] [split-factor=?] [savedata=?]` |
| **-transport** | Custom transport factories.  
  Usage:  
  
  `-transport [factory=?] [truststore=?] [truststore-password=?] [keystore=?] [keystore-password=?] [ssl-protocol=?] [ssl-alg=?] [store-type=?] [ssl-ciphers=?]` |

Additional command-line parameters can modify how cassandra-stress runs:

Additional cassandra-stress parameters
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cl=?</td>
<td>Set the consistency level to use during cassandra-stress. Options are ONE, QUORUM, LOCAL_QUORUM, EACH_QUORUM, ALL, and ANY. Default is LOCAL.ONE.</td>
</tr>
<tr>
<td>clustering=DIST(?)</td>
<td>Distribution clustering runs of operations of the same kind.</td>
</tr>
<tr>
<td>duration=?</td>
<td>Specify the time to run, in seconds, minutes or hours.</td>
</tr>
<tr>
<td>err&lt;?</td>
<td>Specify a standard error of the mean; when this value is reached, cassandra-stress will end. Default is 0.02.</td>
</tr>
<tr>
<td>n=?</td>
<td>Specify the number of operations to run.</td>
</tr>
<tr>
<td>n&gt;?</td>
<td>Specify a minimum number of iterations to run before accepting uncertainly convergence.</td>
</tr>
<tr>
<td>n&lt;?</td>
<td>Specify a maximum number of iterations to run before accepting uncertainly convergence.</td>
</tr>
<tr>
<td>no-warmup</td>
<td>Do not warmup the process, do a cold start.</td>
</tr>
<tr>
<td>ops(?)</td>
<td>Specify what operations to run and the number of each. (only with the user option)</td>
</tr>
<tr>
<td>profile=?</td>
<td>Designate the YAML file to use with cassandra-stress. (only with the user option)</td>
</tr>
<tr>
<td>truncate=?</td>
<td>Truncate the table created during cassandra-stress. Options are never, once, or always. Default is never.</td>
</tr>
</tbody>
</table>

Example: Simple read and write examples

```
# Insert (write) one million rows
$ cassandra-stress write n=1000000 -rate threads=50

# Read two hundred thousand rows.
$ cassandra-stress read n=200000 -rate threads=50

# Read rows for a duration of 3 minutes.
$ cassandra-stress read duration=3m -rate threads=50

# Read 200,000 rows without a warmup of 50,000 rows first.
$ cassandra-stress read n=200000 no-warmup -rate threads=50
```

Example: View schema help

```
$ cassandra-stress help -schema
```
DataStax Enterprise tools

replication([strategy=?][factor=?][<option 1..N>=?]):
  Define the replication strategy and any parameters
  strategy=?
  (default=org.apache.cassandra.locator.SimpleStrategy)  The replication strategy to use
  factor=? (default=1)
  The number of replicas
  keyspace=? (default=keyspace1)
  The keyspace name to use

compaction([strategy=?][<option 1..N>=?]):
  Define the compaction strategy and any parameters
  strategy=?
  The compaction strategy to use
  compression=?
  Specify the compression to use for SSTable, default:no compression

Example: Populate the database

Generally it is easier to let cassandra-stress create the basic schema and then modify it in CQL:

#Load one row with default schema
$ cassandra-stress write n=1 cl=one -mode native cql3 -log
  file=create_schema.log

#Modify schema in CQL
$ cqlsh

#Run a real write workload
$ cassandra-stress write n=1000000 cl=one -mode native cql3 -schema
  keyspace="keyspace1" -log file=load_1M_rows.log

Example: Change the replication strategy

Changes the replication strategy to NetworkTopologyStrategy and targets one node named existing.

$ cassandra-stress write n=500000 no-warmup -node existing -schema
  "replication(strategy=NetworkTopologyStrategy, existing=2)"
Example: Run a mixed workload

When running a mixed workload, you must escape parentheses, greater-than and less-than signs, and other such things. This example invokes a workload that is one-quarter writes and three-quarters reads.

```
$ cassandra-stress mixed ratio\(write=1, read=3\) n=100000 cl=ONE -pop dist=UNIFORM\(1..1000000\) -schema keyspace="keyspace1" -mode native cql3 -rate threads\(>=16) threads\(<=256\) -log file=~\mixed_autorate_50r50w_1M.log
```

Notice the following in this example:

1. The `ratio` parameter requires backslash-escaped parenthesis.

2. The value of `n` used in the read phase is different from the value used in write phase. During the write phase, `n` records are written. However in the read phase, if `n` is too large, it is inconvenient to read all the records for simple testing. Generally, `n` does not need be large when validating the persistent storage systems of a cluster.

   The `-pop dist=UNIFORM\(1..1000000\)` portion says that of the `n=100,000` operations, select the keys uniformly distributed between 1 and 1,000,000. Use this when you want to specify more data per node than what fits in DRAM.

3. In the `rate` section, the greater-than and less-than signs are escaped. If not escaped, the shell attempts to use them for IO redirection: the shell tries to read from a non-existent file called `=256` and create a file called `=16`. The `rate` section tells cassandra-stress to automatically attempt different numbers of client threads and not test less that 16 or more than 256 client threads.

Example: Standard mixed read/write workload keyspace for a single node

```
CREATE KEYSPACE "keyspace1" WITH replication = {
  'class': 'SimpleStrategy',
  'replication_factor': '1'
};
USE "keyspace1";
CREATE TABLE "standard1" (  
  key blob,
  "C0" blob,
  "C1" blob,
  "C2" blob,
  "C3" blob,
  "C4" blob,
  PRIMARY KEY (key)
) WITH
```
DataStax Enterprise tools

```
bloom_filter_fp_chance=0.010000 AND
caching='KEYS_ONLY' AND
comment='' AND
gc_grace_seconds=864000 AND
index_interval=128 AND
replicate_on_write='true' AND
default_time_to_live=0 AND
speculative_retry='99.0PERCENTILE' AND
memtable_flush_period_in_ms=0 AND
compaction={"class": 'SizeTieredCompactionStrategy'} AND
compression={"class": 'LZ4Compressor'};
```

Example: Split up a load over multiple cassandra-stress instances on different nodes

This example demonstrates loading into large clusters, where a single cassandra-stress load generator node cannot saturate the cluster. In this example, $NODES is a variable whose value is a comma delimited list of IP addresses such as 10.0.0.1, 10.0.0.2, and so on.

```
#On Node1
$ cassandra-stress write n=1000000 cl=one -mode native cql3 -
schema keyspace="keyspace1" -pop seq=1..1000000 -log file=~/
node1_load.log -node $NODES

#On Node2
$ cassandra-stress write n=1000000 cl=one -mode native cql3 -
schema keyspace="keyspace1" -pop seq=1000001..2000000 -log file=~/
node2_load.log -node $NODES
```

Example: Run cassandra-stress with authentication

The following example shows using the -mode option to supply a username and password:

```
$ cassandra-stress -mode native cql3 user=cassandra
password=cassandra no-warmup cl=QUORUM
```

**Note:** Check the documentation of the transport (page ) option for SSL authentication.

Example: Run cassandra-stress with authentication and SSL encryption

The following example shows using the -mode option to supply a username and password, and the -transportation option for SSL parameters:

**Note:** Cassandra authentication and SSL encryption must already be configured before executing `cassandra-stress` with these options. The example shown above uses self-signed CA certificates.

Example: Run cassandra-stress using the truncate option

This option must be inserted before the `mode` option, otherwise the cassandra-stress tool won't apply truncation as specified.

The following example shows the `truncate` command:

```bash
$ cassandra-stress write n=100000000 cl=QUORUM truncate=always -schema keyspace=keyspace-rate threads=200 -log file=write_$NOW.log
```

Example: Use a YAML file to run cassandra-stress

This example uses a YAML file named `cqlstress-example.yaml`, which contains the keyspace and table definitions, and a query definition. The keyspace name and definition are the first entries in the YAML file:

```yaml
keyspace: perftesting
keyspace_definition:

    CREATE KEYSPACE perftesting WITH replication = {
        'class': 'SimpleStrategy',
        'replication_factor': 3
    };
```

The table name and definition are created in the next section using CQL:

```yaml
table: users
table_definition:

    CREATE TABLE users (  
        username text,  
        first_name text,  
        last_name text,  
        password text,  
        email text,  
        last_access timeuuid,  
```
In the `extra_definitions` section you can add secondary indexes or materialized views to the table:

```sql
extra_definitions:
- CREATE MATERIALIZED VIEW perftesting.users_by_first_name AS SELECT * FROM perftesting.users WHERE first_name IS NOT NULL and username IS NOT NULL PRIMARY KEY (first_name, username);
- CREATE MATERIALIZED VIEW perftesting.users_by_first_name2 AS SELECT * FROM perftesting.users WHERE first_name IS NOT NULL and username IS NOT NULL PRIMARY KEY (first_name, username);
- CREATE MATERIALIZED VIEW perftesting.users_by_first_name3 AS SELECT * FROM perftesting.users WHERE first_name IS NOT NULL and username IS NOT NULL PRIMARY KEY (first_name, username);
```

The population distribution can be defined for any column in the table. This section specifies a uniform distribution between 10 and 30 characters for `username` values in generated rows, that the values in the generated rows will create, a uniform distribution between 20 and 40 characters for generated `startdate` over the entire Cassandra cluster, and a Gaussian distribution between 100 and 500 characters for `description` values.

```text
columnspec:
- name: username
  size: uniform(10..30)
- name: first_name
  size: fixed(16)
- name: last_name
  size: uniform(1..32)
- name: password
  size: fixed(80) # sha-512
- name: email
  size: uniform(16..50)
- name: startdate
  cluster: uniform(20..40)
- name: description
  size: gaussian(100...500)
```

After the column specifications, you can add specifications for how each batch runs. In the following code, the `partitions` value directs the test to use the column definitions above to insert a fixed number of rows in the partition in each batch:

```text
insert:
  partitions: fixed(10)
  batchtype: UNLOGGED
```

The last section contains a query, `read1`, that can be run against the defined table.

```text
queries:
read1:
```
The following example shows using the `user` option and its parameters to run `cassandra-stress` tests from `cqlstress-example.yaml`:

```bash
$ cassandra-stress user profile=tools/cqlstress-example.yaml n=1000000 ops\(insert=3,read1=1\) no-warmup cl=QUORUM
```

Notice that:

- The `user` option is required for the `profile` and `opt` parameters.
- The value for the `profile` parameter is the path and filename of the `.yaml` file.
- In this example, `-n` specifies the number of `batches` that run.
- The values supplied for `ops` specifies which operations run and how many of each. These values direct the command to `insert` rows into the database and run the `read1` query.

How many times? Each insert or query counts as one batch, and the values in `ops` determine how many of each type are run. Since the total number of batches is 1,000,000, and `ops` says to run three inserts for each query, the result will be 750,000 inserts and 250,000 of the `read1` query.

Use escaping backslashes when specifying the `ops` value.

For more information, see Improved Cassandra 2.1 Stress Tool: Benchmark Any Schema – Part 1.

Example: Use the `-graph` option

In Cassandra 3.2 and later, the `-graph` option provides visual feedback for `cassandra-stress` tests. A file must be named to build the resulting HTML file. A `title` and `revision` are optional, but `revision` must be used if multiple stress tests are graphed on the same output.

```bash
$ cassandra-stress user profile=tools/cqlstress-example.yaml ops \(insert=1\) -graph file=test.html title=test revision=test1
```

An interactive graph can be displayed with a web browser:
Interpreting the output of cassandra-stress

Each line reports data for the interval between the last elapsed time and current elapsed time.

Created keyspaces. Sleeping 1s for propagation.
Sleeping 2s...
Warming up WRITE with 50000 iterations...
Running WRITE with 200 threads for 1000000 iteration

<p>| type     | total ops, | op/s, | pk/s, | row/s, | mean, | med, | .95, | .99, | .999, | max, | .99, | .999, | max, | .99, | .999, | max, | .99, | .999, | max, | .99, | .999, | max, | .99, | .999, | max, |
|----------|------------|-------|-------|--------|-------|------|------|------|-------|------|------|-------|------|------|-------|------|------|-------|------|------|-------|------|------|-------|------|------|-------|------|
|          | total,     | ms,   | sum ms, | sdv ms, | mb    | total, | ms,   | sum ms, | sdv ms, | total, | ms,   | sum ms, | sdv ms, | total, | ms,   | sum ms, | sdv ms, | total, | ms,   | sum ms, | sdv ms, | total, | ms,   | sum ms, | sdv ms, |
|          | 43148      | 239.3 | 255.4, | 1.0, | 0.00000, | 0, | 1, | 4.6, | 1.5, | 42991, | 42991, | 42991, | 4.6, | 1.5, | 42991, | 42991, | 42991, | 4.6, | 1.5, |
|          | 98715      | 204.6 | 264.5, | 2.3, | 0.00705, | 0, | 1, | 4.6, | 1.7, | 43857, | 43857, | 43857, | 4.6, | 1.7, | 43857, | 43857, | 43857, | 4.6, | 1.7, |
|          | 157777     | 251.7 | 286.3, | 3.5, | 0.02393, | 0, | 1, | 4.1, | 1.4, | 47283, | 47283, | 47283, | 4.1, | 1.4, | 47283, | 47283, | 47283, | 4.1, | 1.4, |
|          | 312940     | 251.7 | 286.3, | 3.5, | 0.02393, | 0, | 1, | 4.1, | 1.4, | 47283, | 47283, | 47283, | 4.1, | 1.4, | 47283, | 47283, | 47283, | 4.1, | 1.4, |
|          | 43148      | 239.3 | 255.4, | 1.0, | 0.00000, | 0, | 1, | 4.6, | 1.5, | 42991, | 42991, | 42991, | 4.6, | 1.5, | 42991, | 42991, | 42991, | 4.6, | 1.5, |
|          | 98715      | 204.6 | 264.5, | 2.3, | 0.00705, | 0, | 1, | 4.6, | 1.7, | 43857, | 43857, | 43857, | 4.6, | 1.7, | 43857, | 43857, | 43857, | 4.6, | 1.7, |
|          | 157777     | 251.7 | 286.3, | 3.5, | 0.02393, | 0, | 1, | 4.1, | 1.4, | 47283, | 47283, | 47283, | 4.1, | 1.4, | 47283, | 47283, | 47283, | 4.1, | 1.4, |
|          | 43148      | 239.3 | 255.4, | 1.0, | 0.00000, | 0, | 1, | 4.6, | 1.5, | 42991, | 42991, | 42991, | 4.6, | 1.5, | 42991, | 42991, | 42991, | 4.6, | 1.5, |
|          | 98715      | 204.6 | 264.5, | 2.3, | 0.00705, | 0, | 1, | 4.6, | 1.7, | 43857, | 43857, | 43857, | 4.6, | 1.7, | 43857, | 43857, | 43857, | 4.6, | 1.7, |
|          | 157777     | 251.7 | 286.3, | 3.5, | 0.02393, | 0, | 1, | 4.1, | 1.4, | 47283, | 47283, | 47283, | 4.1, | 1.4, | 47283, | 47283, | 47283, | 4.1, | 1.4, |
|          | 43148      | 239.3 | 255.4, | 1.0, | 0.00000, | 0, | 1, | 4.6, | 1.5, | 42991, | 42991, | 42991, | 4.6, | 1.5, | 42991, | 42991, | 42991, | 4.6, | 1.5, |
|          | 98715      | 204.6 | 264.5, | 2.3, | 0.00705, | 0, | 1, | 4.6, | 1.7, | 43857, | 43857, | 43857, | 4.6, | 1.7, | 43857, | 43857, | 43857, | 4.6, | 1.7, |
|          | 157777     | 251.7 | 286.3, | 3.5, | 0.02393, | 0, | 1, | 4.1, | 1.4, | 47283, | 47283, | 47283, | 4.1, | 1.4, | 47283, | 47283, | 47283, | 4.1, | 1.4, |
| Results: | op rate    | 46751 | 46751 | WRITE:46751 | WRITE:46751 |
|          | partition rate | 46751 | 46751 | WRITE:46751 | WRITE:46751 |
|          | row rate    | 46751 | 46751 | WRITE:46751 | WRITE:46751 |
|          | latency mean | 4.3 | 4.3 | WRITE:4.3 | WRITE:4.3 |</p>
<table>
<thead>
<tr>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>total ops</td>
<td>Running total number of operations during the run.</td>
</tr>
<tr>
<td>op/s</td>
<td>Number of operations per second performed during the run.</td>
</tr>
<tr>
<td>pk/s</td>
<td>Number of partition operations per second performed during the run.</td>
</tr>
<tr>
<td>row/s</td>
<td>Number of row operations per second performed during the run.</td>
</tr>
<tr>
<td>mean</td>
<td>Average latency in milliseconds for each operation during that run.</td>
</tr>
<tr>
<td>med</td>
<td>Median latency in milliseconds for each operation during that run.</td>
</tr>
<tr>
<td>.95</td>
<td>95% of the time the latency was less than the number displayed in the column.</td>
</tr>
<tr>
<td>.99</td>
<td>99% of the time the latency was less than the number displayed in the column.</td>
</tr>
<tr>
<td>.999</td>
<td>99.9% of the time the latency was less than the number displayed in the column.</td>
</tr>
<tr>
<td>max</td>
<td>Maximum latency in milliseconds.</td>
</tr>
<tr>
<td>time</td>
<td>Total operation time.</td>
</tr>
<tr>
<td>stderr</td>
<td>Standard error of the mean. It is a measure of confidence in the average throughput number; the smaller the number, the more accurate the measure of the cluster’s performance.</td>
</tr>
<tr>
<td>gc: #</td>
<td>Number of garbage collections.</td>
</tr>
<tr>
<td>max ms</td>
<td>Longest garbage collection in milliseconds.</td>
</tr>
<tr>
<td>sum ms</td>
<td>Total of garbage collection in milliseconds.</td>
</tr>
<tr>
<td>sdv ms</td>
<td>Standard deviation in milliseconds.</td>
</tr>
</tbody>
</table>
### DataStax Enterprise tools

<table>
<thead>
<tr>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mb</td>
<td>Size of the garbage collection in megabytes.</td>
</tr>
</tbody>
</table>

## fs-stress tool

### Synopsis

```
fs-stress [options] dsefs_directory listen_address
```

The default IP address is the `listen_address (page 65)` property in the cassandra.yaml file. If not using localhost, specify the correct IP address.

`fs-stress` is located in the `tools` directory of your installation.

The default location of the `tools` directory depends on the type of installation:

- Package installations: `/usr/share/dse/tools`
- Tarball installations: `installation_location/dse/tools`

### Table 411: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Description

The `fs-stress` tool performs stress testing of the DSE File System (DSEFS) layer.

<table>
<thead>
<tr>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>progress</td>
<td>Total progress of the stress operation.</td>
</tr>
<tr>
<td>bytes</td>
<td>Total bytes written/read.</td>
</tr>
<tr>
<td>curr rate</td>
<td>Current rate of bytes being written/read per second.</td>
</tr>
<tr>
<td>avg rate</td>
<td>Average rate of bytes being written/read per second.</td>
</tr>
<tr>
<td>max latency</td>
<td>Maximum latency in milliseconds during the current reporting window.</td>
</tr>
</tbody>
</table>

### SSTable utilities

SSTable utility tools are diagnostic tools for analyzing, using, upgrading, and changing DataStax Enterprise SSTables.

#### About SSTable tools

For the following SSTable utility tools, stop DSE (page 1442) before running the command:

- `sstabledump (page 1396)`
- `sstableexpiredblockers (page 1404)`
- `sstablelevelreset (page 1406)`
- `sstablémetadata (page 1411)`
- `sstableofflinerelevel (page 1415)`
- `sstablerepaireddset (page 1421)`
- `sstablesplit (page 1426)`

**Tip:** SSTable tools work offline from the DataStax Enterprise database. If you need to pass a JVM parameter, specify it in the command line. For example, to change the max heap size:
SSTable tools are located in several locations.

The default location of the SSTable tools depends on the type of installation and the tool:
- **Package installations:** /usr/bin/
- **Tarball installations:** installation_location/resources/cassandra/tools/bin or installation_location/resources/cassandra/bin

**sstabledowngrade**

Downgrades the SSTables in the given table or snapshot to the version of OSS Apache Cassandra™ that is compatible with the current version of DSE.

**Note:** The sstabledowngrade command cannot be used to downgrade system tables or downgrade DSE versions.

**Synopsis**

```bash
$ sstabledowngrade
[--debug]
[-h]
[-k]
[-b]
[--keep-generation]
[-o output-div]
[--sstable-files sstable]
[ -t rate-limit]
keyspace_name
table_name
[snapshot_name]
```

**Table 412: SSTable compatibility and upgrade version**

<table>
<thead>
<tr>
<th>DSE version</th>
<th>SSTable</th>
<th>sstableloader (page 1408) supported format-version</th>
<th>sstableupgrade (page 1428) and nodetool upgradesstables (page 1140) supported format-version</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7.x</td>
<td>bti</td>
<td>big-ma, big-mb, big-mc, and bti-aa</td>
<td>big-ma, big-mb, and big-mc</td>
</tr>
<tr>
<td>6.0.x</td>
<td>aa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.1.x</td>
<td>big</td>
<td>big-ka, big-ma, big-mb, and big-mc</td>
<td>big-ka, big-ma, and big-mb</td>
</tr>
<tr>
<td>5.0.x</td>
<td>mc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSE version</td>
<td>SSTable format</td>
<td>sstableloader (page 1408) supported version</td>
<td>sstableupgrade (page 1428) and nodetool upgradesstables (page 1140) supported format-version</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
<td>-------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>5.0.x</td>
<td>mb</td>
<td>big-ka, big-ma, and big-mb</td>
<td>big-ka and big-ma</td>
</tr>
<tr>
<td>5.0.x</td>
<td>ma</td>
<td>big-ka and big-ma</td>
<td>big-k* only</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Command arguments**

--debug
Display stack traces.

-h, --help
Display the usage and listing of the commands.

-k, --keep-source
Do not delete the source SSTables.

-b, --backups
Rewrite incremental backups for the given table. May not be combined with the snapshot_name option.

--keep-generation
Keep the SSTable generation. Do use with the --keep-source option.

-o, --output-dir
Rewritten files are placed in output-dir/keyspace-name/table-name-and-id.

--sstable-files
Instead of processing all SSTables in the default data directories, process only the tables specified via this option. If a single SSTable file, only that SSTable is processed. If a directory is specified, all SSTables within that directory are processed. Snapshots and backups are not supported with this option.

-t, --throughput
Set to limit the maximum disk read rate in MB/s.

**keyspace_name**
Keyspace name. Required. Overrides the client_encryption_options (page 97) in cassandra.yaml.

**table_name**
Table name. Required.

**snapshot_name**
Snapshot name.

- Only rewrites the specified snapshot.
DataStax Enterprise tools

- Replaces files in the given snapshot and breaks any hard links to live SSTables.
- Required when before attempting to restore a snapshot taken in a different DSE version than the one that is currently running.

Examples

Upgrade events table on cycling keyspace

```
$ sstabledowngrade cycling events
```

Found 1 sstables to rewrite.
Rewriting TrieIndexSSTableReader(path='/var/lib/cassandra/data/cycling/events-2118bc7054af11e987feb76774f7ab56/aa-1-bti-Data.db') to BIG/mc.
Rewrite of TrieIndexSSTableReader(path='/var/lib/cassandra/data/cycling/events-2118bc7054af11e987feb76774f7ab56/aa-1-bti-Data.db') to BIG/mc complete.

sstabledump

Dumps contents of given SSTable to standard output in JSON format.

**Restriction:** DataStax Enterprise must be stopped before you run this command.

Synopsis

```
$ sstabledump sstable_filepath
[-d] [-e] [-k partition_key]
[-l] [-t] [-x partition_key]
```

Table 413: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Single quotation (’’) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
<td>'Literal string'</td>
<td></td>
</tr>
<tr>
<td>Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
<td>{ key:value }</td>
<td>Map collection.</td>
</tr>
<tr>
<td>Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple.</td>
</tr>
<tr>
<td>A semicolon ( ; ) terminates all CQL statements.</td>
<td>cql_statement;</td>
<td>End CQL statement.</td>
</tr>
<tr>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
<td>[ -- ]</td>
<td></td>
</tr>
<tr>
<td>Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only:</td>
</tr>
<tr>
<td>Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only:</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Command arguments

- **-d**
  - Display a CQL row per line.
- **-e**
  - Display a list of partition keys.
- **-k, --key partition_key**
  - Partition keys to include.
- **-l**
  - Output JSON lines, by partition.
- **-t**
  - Print raw timestamps instead of ISO 8601 date strings.
- **-x, --exclude-key partition_key**
  - Partition key to exclude. Ignored if -y option is given.

### Examples

**Verify DataStax Enterprise is not running**

```
$ nodetool status
```

**Datacenter: Graph
================================**
Status=Up/Down
| State=Normal/Leaving/Joining/Moving
-- Address        Load       Tokens       Owns    Host ID
Rack
UN 10.200.177.92  265.04 KiB  1            ?       980cab6a-2e5d-44c6-b897-0733dde580ac  rack1
DN 10.200.177.94  426.21 KiB  1            ?       7ecbbc0c-627d-403e-b8cc-a2daa93d9ad3  rack1

Dump contents of SSTable

Restriction: DataStax Enterprise must be stopped before you run this command.

$ sstabledump /var/lib/cassandra/data/cycling/birthday_list-f4f24621ce3f1le89d32bdccab3aa9c6f/aa-l-bti-Statistics.db

[{
    "partition": {
        "key": [ "Claudio HEINEN" ],
        "position": 0
    },
    "rows": [
        {
            "type": "row",
            "position": 90,
            "liveness_info": { "tstamp": "2018-10-12T16:58:00.368228Z" },
            "cells": [
                { "name": "blist_", "deletion_info": { "marked_deleted": "2018-10-12T16:58:00.368227Z", "local_delete_time": "2018-10-12T16:58:00Z" } },
                { "name": "blist_", "path": [ "bday" ], "value": "27/07/1992" },
                { "name": "blist_", "path": [ "blist_age" ], "value": "23" },
                { "name": "blist_", "path": [ "blist_nation" ], "value": "GERMANY" }
            ]
        }
    ],
    "partition": {
        "key": [ "Claudio VANDELLI" ],
        "position": 91
    },
    "rows": [
        {
            "type": "row",
            "position": 179,
            "liveness_info": { "tstamp": "2018-10-12T16:58:00.354443Z" },
            "cells": [
                { "name": "blist_", "path": [ "bday" ], "value": "27/07/1992" }
            ]
        }
    ]
}]}
Show a row per line in standard output of the cycling.birthday_list table

**Restriction:** DataStax Enterprise must be stopped before you run this command.

```
$ sstabledump /var/lib/cassandra/data/cycling/birthday_list-e439b9222bc511e8891b23da85222d3d/aa-2-bti-Data.db -d
```

```
```
Display a list of partition keys from the cycling.birthday_list table

Restriction: DataStax Enterprise must be stopped before you run this command.

```
$ sstabledump /var/lib/cassandra/data/cycling/birthday_list-e439b9222bc511e8891b23da85222d3d/aa-2-bti-Data.db -e

[ [ "Claudio HEINEN" ], [ "Claudio VANDELLI" ], [ "Luc HAGENAARS" ], [ "Toine POELS" ], [ "Allan DAVIS" ], [ "Laurence BOURQUE" ] ]
```

Display all rows in the partition

Restriction: DataStax Enterprise must be stopped before you run this command.

```
$ sstabledump /var/lib/cassandra/data/cycling/birthday_list-e439b9222bc511e8891b23da85222d3d/aa-2-bti-Data.db -k "Claudio HEINEN"

[
{
  "partition" : {
    "key" : [ "Claudio HEINEN" ],
    "position" : 0
  },
  "rows" : [
    {
      "type" : "row",
      "position" : 75,
      "cells" : [
        { "name" : "blist", "path" : [ "age" ], "value" : "23" },
        { "name" : "blist", "path" : [ "bday" ], "value" : "27/07/1992" },
        { "name" : "blist", "path" : [ "nation" ], "value" : "GERMANY" }
      ]
    }
  ]
}
]
Display all rows except those in the specified partition

**Restriction:** DataStax Enterprise must be stopped before you run this command.

```
$ sstabledump /var/lib/cassandra/data/cycling/birthday_list-e439b9222bc511e8891b23da85222d3d/aa-2-bti-Data.db -x "Claudio HEINEN"
```

```
[
  {
    "partition": {
      "key": [ "Claudio VANDELLI" ],
      "position": 0
    },
    "rows": [
      {
        "type": "row",
        "position": 151,
        "liveness_info": { "tstamp": "2018-03-19T22:35:57.437559Z" },
        "cells": [
          { "name": "blist", "path": [ "age" ], "value": "54" },
          { "name": "blist", "path": [ "bday" ], "value": "27/07/1961" },
          { "name": "blist", "path": [ "nation" ], "value": "ITALY" }
        ]
      }
    ]
  },
  {
    "partition": {
      "key": [ "Luc HAGENAARS" ],
      "position": 152
    },
    "rows": [
      {
        "type": "row",
        "position": 231,
        "liveness_info": { "tstamp": "2018-03-19T22:35:57.448698Z" },
        "cells": [
          { "name": "blist", "path": [ "age" ], "value": "28" },
          { "name": "blist", "path": [ "bday" ], "value": "27/07/1987" },
          { "name": "blist", "path": [ "nation" ], "value": "NETHERLANDS" }
        ]
      }
    ]
  }]
}]
```
"key" : [ "Toine POELS" ],
"position" : 232
},
"rows" : [ {
  "type" : "row",
  "position" : 309,
  "cells" : [
    { "name" : "blist", "path" : [ "age" ], "value" : "52" },
    { "name" : "blist", "path" : [ "bday" ], "value" : "27/07/1963" },
    { "name" : "blist", "path" : [ "nation" ], "value" : "NETHERLANDS" }]
],
},
"partition" : {
  "key" : [ "Allan DAVIS" ],
  "position" : 310
},
"rows" : [ {
  "type" : "row",
  "position" : 383,
  "cells" : [
    { "name" : "blist", "path" : [ "age" ], "value" : "35" },
    { "name" : "blist", "path" : [ "bday" ], "value" : "27/07/1980" },
    { "name" : "blist", "path" : [ "nation" ], "value" : "AUSTRALIA" }]
],
},
"partition" : {
  "key" : [ "Laurence BOURQUE" ],
  "position" : 384
},
"rows" : [ {
  "type" : "row",
  "position" : 460,
  "cells" : [
    { "name" : "blist", "path" : [ "age" ], "value" : "23" },
    { "name" : "blist", "path" : [ "bday" ], "value" : "27/07/1992" },
    { "name" : "blist", "path" : [ "nation" ], "value" : "CANADA" }]
]
Display each row in its own JSON map

Restriction: DataStax Enterprise must be stopped before you run this command.

```
$sstabledump /var/lib/cassandra/data/cycling/birthday_list-e439b9222bc511e8891b23da85222d3d/aa-2-bti-Data.db -l
```

```json
{
    "partition":{
        "key": ["Claudio HEINEN"],
        "position": 0,
        "rows": [
            {
                "type": "row",
                "position": 75,
                "liveness_info": {
                    "tstamp": "2018-03-19T22:35:57.445075Z"
                },
                "cells": [
                    {
                        "name": "blist",
                        "path": ["age"],
                        "value": "23"
                    },
                    {
                        "name": "blist",
                        "path": ["bday"],
                        "value": "27/07/1992"
                    },
                    {
                        "name": "blist",
                        "path": ["nation"],
                        "value": "GERMANY"
                    }
                ]
            }
        ]
    },
    "partition":{
        "key": ["Claudio VANDELLI"],
        "position": 76,
        "rows": [
            {
                "type": "row",
                "position": 151,
                "liveness_info": {
                    "tstamp": "2018-03-19T22:35:57.437559Z"
                },
                "cells": [
                    {
                        "name": "blist",
                        "path": ["age"],
                        "value": "54"
                    },
                    {
                        "name": "blist",
                        "path": ["bday"],
                        "value": "27/07/1961"
                    },
                    {
                        "name": "blist",
                        "path": ["nation"],
                        "value": "ITALY"
                    }
                ]
            }
        ]
    },
    "partition":{
        "key": ["Luc HAGENAARS"],
        "position": 152,
        "rows": [
            {
                "type": "row",
                "position": 231,
                "liveness_info": {
                    "tstamp": "2018-03-19T22:35:57.448698Z"
                },
                "cells": [
                    {
                        "name": "blist",
                        "path": ["age"],
                        "value": "28"
                    },
                    {
                        "name": "blist",
                        "path": ["bday"],
                        "value": "27/07/1987"
                    },
                    {
                        "name": "blist",
                        "path": ["nation"],
                        "value": "NETHERLANDS"
                    }
                ]
            }
        ]
    },
    "partition":{
        "key": ["Toine POELS"],
        "position": 232,
        "rows": [
            {
                "type": "row",
                "position": 309,
                "liveness_info": {
                    "tstamp": "2018-03-19T22:35:57.451068Z"
                },
                "cells": [
                    {
                        "name": "blist",
                        "path": ["age"],
                        "value": "52"
                    },
                    {
                        "name": "blist",
                        "path": ["bday"],
                        "value": "27/07/1963"
                    },
                    {
                        "name": "blist",
                        "path": ["nation"],
                        "value": "NETHERLANDS"
                    }
                ]
            }
        ]
    },
    "partition":{
        "key": ["Allan DAVIS"],
        "position": 310,
        "rows": [
            {
                "type": "row",
                "position": 383,
                "liveness_info": {
                    "tstamp": "2018-03-19T22:35:57.430478Z"
                },
                "cells": [
                    {
                        "name": "blist",
                        "path": ["age"],
                        "value": "35"
                    },
                    {
                        "name": "blist",
                        "path": ["bday"],
                        "value": "27/07/1980"
                    },
                    {
                        "name": "blist",
                        "path": ["nation"],
                        "value": "AUSTRALIA"
                    }
                ]
            }
        ]
    },
    "partition":{
        "key": ["Laurence BOURQUE"],
        "position": 384,
        "rows": [
            {
                "type": "row",
                "position": 460,
                "liveness_info": {
                    "tstamp": "2018-03-19T22:35:57.441360Z"
                },
                "cells": [
                    {
                        "name": "blist",
                        "path": ["age"],
                        "value": "23"
                    },
                    {
                        "name": "blist",
                        "path": ["bday"],
                        "value": "27/07/1992"
                    },
                    {
                        "name": "blist",
                        "path": ["nation"],
                        "value": "CANADA"
                    }
                ]
            }
        ]
    }
}
**sstableexpiredblockers**

Outputs the SSTables that prevent an SSTable from dropping.

By identifying the blocking SSTables, you can take correction active so the database can drop entire SSTables during compaction. SSTables are dropped during compaction when they contain only expired tombstones and is guaranteed not to cover any data in other SSTables.

**Restriction:** DataStax Enterprise must be stopped before you run this command.

**Synopsis**

```
$ sstableexpiredblockers [--dry-run] keyspace_name table_name
```

**Table 414: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

The short form and long form parameters are comma-separated.

### Command arguments

--dry-run
   Test command syntax and environment. Do not execute the command.

**keyspace_name**
   Keyspace name. Required. Overrides the client_encryption_options (page 97) in cassandra.yaml.

**table_name**
   Table name. Required.

### Examples

#### Verify DataStax Enterprise is not running

```
$ nodetool status
```

```
Datacenter: Graph
============================
Status=Up/Down
   | State=Normal/Leaving/Joining/Moving
-- Address Load Tokens Owns Host ID
   | Rack
UN 10.200.177.92 265.04 KiB 1 ? 980cab6a-2e5d-44c6-b897-0733dde580ac rack1
DN 10.200.177.94 426.21 KiB 1 ? 7ecbbc0c-627d-403e-b8cc-a2daa93d9ad3 rack1
```

#### Output the blocking SSTables that prevent an SSTable from dropping

**Restriction:** DataStax Enterprise must be stopped before you run this command.

```
$ sstableexpiredblockers cycling cyclist_races
```

#### Test the output without executing sstableexpiredblockers

```
```
DataStax Enterprise tools

Restriction: DataStax Enterprise must be stopped before you run this command.

```
$ sstableexpiredblockers --dry-run cycling cyclist_races
```

**sstablelevelreset**

Uses **LeveledCompactionStrategy** to reset the level to zero on a set of SSTables. If the SSTable is already at level 0, no change occurs. If the SSTable is releveled, the metadata is rewritten to designate the level at 0.

Restriction: DataStax Enterprise must be stopped before you run this command.

**Synopsis**

```
$ sstablelevelreset [--really-reset] keyspace_name table_name
```

### Table 415: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>()</td>
<td>Group. Parentheses (() ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

The short form and long form parameters are comma-separated.

### Command arguments

**keyspace_name**
Keyspace name. Required. Overrides the `client_encryption_options` (page 97) in `cassandra.yaml`.

**--really-reset**
Specifies that DSE is stopped.

**table_name**
Table name. Required.

### Examples

**Verify DataStax Enterprise is not running**

```
$ nodetool status
```

**Reset cyclist_name to level 0**

**Restriction:** DataStax Enterprise must be stopped before you run this command.

```
$ sstablelevelreset cycling cyclist_name
```

Skipped `/var/lib/cassandra/data/cycling/cyclist_name-4157ef22ce4411e8949e33016bf887c0/aa-2-bti-Data.db` since it is already on level 0
Skipped /var/lib/cassandra/data/cycling/
cyclist_name-4157ef22ce4411e8949e33016bf887c0/aa-3-bti-Data.db since it is already on level 0

**sstableloader**

Streams a set of SSTable data files from the `sstable_directory` to a live cluster. The target keyspace and table are the parent directories of the `sstable_directory`.

For example, to load an SSTable named `Standard1-g-1-Data.db` into `Keyspace1/Standard1`, have the files `Standard1-g-1-Data.db` and `Standard1-g-1-Index.db` in directory `/path/to/Keyspace1/Standard1/`.

**Synopsis**

```
$ sstableloader
[-alg algorithm] [-ap authentication_provider]
[-ciphers cipher_suite] [-cph num_connections_per_host]
[-d initial_host] [-df dse.yaml_path]
[-f cassandra.yaml_path] [-h]
[-i node] [-idct throttle_speed]
[-ks keystore_path] [-kspw keystore_password]
[--no-progress] [-p native_transport_port]
[-prtl SSL_protocol] [pw password]
[-sp storage_port] [-ssp ssi_storage_port] [-st store_type]
[-t throttle_speed] [-ts truststore_path] [-tspw truststore_password]
[-u username] [-v]
sstable_directory
```

**Table 416: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italics</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ key:value }</code></td>
<td>Map collection. Braces ({ }) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>' &lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Command arguments

- **-alg,--ssl-alg algorithm**
  Client SSL algorithm. Default: SunX509.

- **-ap,--auth-provider authentication_provider**
  Custom AuthProvider class name. Can be combined with -u `username` and -pw `password` if the AuthProvider supports plain text credentials.

- **-ciphers, --ssl-ciphers cipher-suite**
  Comma-separated list of encryption suites for Client SSL.

- **-cph,--connections-per-host num_connections_per_host**
  Number of concurrent connections per host.

- **-d, --nodes initial_host**
  Required. Comma-separated list of hosts to connect to initially for ring information.

- **-df, --dse-conf-path dse_yaml_path**
  The dse.yaml filepath.

- **-f, --conf-path cassandra_yaml_path**
  The filepath to a cassandra.yaml config file to override only these encryption options that were set in the cassandra.yaml file that was read at startup:
  - `stream_throughput_outbound_megabits_per_sec (page 77)`
  - `server_encryption_options (page 95)`
  - `client_encryption_options (page 97)`

- **-h, --help**
  Display the usage and listing of the commands.

- **-i, --ignore node**
Comma-separated list of nodes to ignore.

- idct, --inter-dc-throttle throttle_speed
  Inter-datacenter throttle speed in Mbits. Default: unlimited.

- ks,--keystore keystore_path
  Filepath to keystore for SSL client-to-node encryption. Overrides the client_encryption_options (page 97) in cassandra.yaml.

- kspw,--keystore-password keystore_password
  Client SSL keystore password. Overrides the client_encryption_options (page 97) in cassandra.yaml.

--no-progress
  Do not display progress.

- p, --port native_transport_port
  Port for native connection. Default: 9042.

- prtcl,--ssl-protocol SSL_protocol
  Client SSL connections protocol. Overrides the server_encryption_options (page 95) in cassandra.yaml. Default: TLS.

- pw,--password password
  Cassandra authentication password.

- sp, --storage-port storage_port
  Port for internode communication. Default: 7000.

- ssp, --ssl-storage-port ssl_storage_port
  Port for TLS internode communication. Default: 7001.

sstable_directory
  The absolute path to the SSTable data directory. The data_file_directories (page 66) property in cassandra.yaml defines the default directory.

- st, --store-type store_type
  Client SSL store type.

- t, --throttle throttle_speed
  Throttle speed in Mbits. Default: unlimited.

- ts, --truststore truststore_path
  Client SSL filepath to truststore.

- tspw, --truststore-password truststore_password
  Client SSL truststore password.

- u, --username username
  Cassandra authentication username.

- v,--verbose
  Verbose output.

Examples

Package installation

$ sstableloader -d 110.82.155.1 /var/lib/cassandra/data/cycling/cyclist_name-9e516080f30811e689e40725f37c761d/snapshots/1527686840030

Tarball installation
sstablemetadata

Prints metadata about given SSTable or SSTables to standard output, including SSTable name, partitioner, tombstone details, compressor, TTL, token, min and max clustering values, SSTable level, partition size and statistics, and column information.

**Restriction:** DataStax Enterprise must be stopped before you run this command.

**Synopsis**

```
$ sstablemetadata
sstable_filepath [sstable_filepath ...]
[-c] [-g seconds] [-s] [-t time_unit] [-u]
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (;) terminates all CQL statements.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

The short form and long form parameters are comma-separated.

### Command arguments

- **-c, --colors**
  ANSI color sequence.

- **-g, --gc_grace_seconds seconds**
  Time to use when calculating droppable tombstones.

- **sstable_filepath**
  The explicit or relative filepath to the SSTable data file ending in Data.db.

- **-s, --scan**
  Full SSTable scan for additional details. Default: false.

- **-t, --timestamp_unit time_unit**
  Time unit that cell timestamps are written with.

- **-u, --unicode**
  Use Unicode to draw histograms and progress bars.

### Examples

These examples are generated using the cycling keyspace. See Setting up the Cycling keyspace.

### Verify DataStax Enterprise is not running

```
$ nodetool status
```

**Datacenter: Graph**

<table>
<thead>
<tr>
<th></th>
<th>Load</th>
<th>Tokens</th>
<th>Owns</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UN</td>
<td>265.04 KiB</td>
<td>1</td>
<td>?</td>
<td>980cab6a-2e5d-44c6-b897-0733dde580ac rack1</td>
</tr>
<tr>
<td>DN</td>
<td>426.21 KiB</td>
<td>1</td>
<td>?</td>
<td>7ecbbc0c-627d-403e-b8cc-a2daa93d9ad3 rack1</td>
</tr>
</tbody>
</table>

### Get information about SSTable
**Restriction:** DataStax Enterprise must be stopped before you run this command.

```
$ sstablemetadata /var/lib/cassandra/data/cycling/birthday_list- f4f24621ce3f11e89d32bdcab3a99c6f/aa-1-bti-Statistics.db
```

SSTable: /var/lib/cassandra/data/cycling/birthday_list- f4f24621ce3f11e89d32bdcab3a99c6f/aa-1-bti
Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
Bloom Filter FP chance: 0.01
Minimum timestamp: 1539363480354442 (10/12/2018 16:58:00)
Maximum timestamp: 1539363480374846 (10/12/2018 16:58:00)
SSTable min local deletion time: 1539363480 (10/12/2018 16:58:00)
SSTable max local deletion time: 2147483647 (no tombstones)
Compressor: org.apache.cassandra.io.compress.LZ4Compressor
Compression ratio: 0.6884057971014492
TTL min: 0
TTL max: 0
First token: -5189327806405140569 (Claudio HEINEN)
Last token: -428849430723689847 (Luc HAGENAARS)
minClusteringValues: []
maxClusteringValues: []
Estimated droppable tombstones: 0.3333333333333333
SSTable Level: 0
Repaired at: 0
Pending repair: --
Replay positions covered: {CommitLogPosition(segmentId=1539277782404, position=18441844)=CommitLogPosition(segmentId=1539277782404, position=18480562)}
totalColumnsSet: 3
totalsRows: 3
Estimated tombstone drop times:

<table>
<thead>
<tr>
<th>Drop Time</th>
<th>Count (%)</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>1539363480 (10/12/2018 16:58:00)</td>
<td>3 (100)</td>
<td></td>
</tr>
</tbody>
</table>

OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

Percentiles
50th 1663415872 (09/17/2022 11:57:52)
75th 1663415872 (09/17/2022 11:57:52)
95th 1663415872 (09/17/2022 11:57:52)
98th 1663415872 (09/17/2022 11:57:52)
99th 1663415872 (09/17/2022 11:57:52)
Min 1386179894 (12/04/2013 17:58:14)
Max 1663415872 (09/17/2022 11:57:52)

Partition Size:

<table>
<thead>
<tr>
<th>Size (bytes)</th>
<th>Count (%)</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>103 (103 B)</td>
<td>3 (100)</td>
<td></td>
</tr>
</tbody>
</table>

OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

Percentiles
50th 103 (103 B)
75th 103 (103 B)
95th 103 (103 B)
98th 103 (103 B)
99th 103 (103 B)
Min 87 (87 B)
Max 103 (103 B)
DataStax Enterprise tools

### Column Count:

<table>
<thead>
<tr>
<th>Columns</th>
<th>Count (%)</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3 (100)</td>
<td>OOOOOOOOOOOOOOOOOOOOOOOOOOOO</td>
</tr>
</tbody>
</table>

**Percentiles**

- 50th: 3
- 75th: 3
- 95th: 3
- 98th: 3
- 99th: 3
- Min: 3
- Max: 3

Estimated cardinality: 3

**EncodingStats**

- minTTL: 0
- minLocalDeletionTime: 1539363480 (10/12/2018 16:58:00)
- minTimestamp: 1539363480354442 (10/12/2018 16:58:00)

**KeyType**: org.apache.cassandra.db.marshal.UTF8Type

**ClusteringTypes**: []

**StaticColumns**: 

**RegularColumns**: 

- blist_:org.apache.cassandra.db.marshal.MapType(org.apache.cassandra.db.marshal.UTF8Type,org.apache.cassandra.db.marshal.UTF8Type)

---

### Get information about SSTable with username

**Restriction**: DataStax Enterprise must be stopped before you run this command.

```
$ sstablemetadata /var/lib/cassandra/data/cycling/cyclist_category-elf76e21ce4311e8949e33016bf887c0/aa-1-bti-Rows.db -u
```

**SSTable**: /var/lib/cassandra/data/cycling/cyclist_category-elf76e21ce4311e8949e33016bf887c0/aa-1-bti

**Partitioner**: org.apache.cassandra.dht.Murmur3Partitioner

- Bloom Filter FP chance: 0.01
- Minimum timestamp: 1539365167498813 (10/12/2018 17:26:07)
- Maximum timestamp: 1539365167524231 (10/12/2018 17:26:07)
- SSTable min local deletion time: 2147483647 (no tombstones)
- SSTable max local deletion time: 2147483647 (no tombstones)

**Compressor**: org.apache.cassandra.io.compress.LZ4Compressor

- Compression ratio: 1.0761904761904761
- TTL min: 0
- TTL max: 0

- First token: -798238132730727330 (One-day-races)
- Last token: -798238132730727330 (One-day-races)

- minClusteringValues: [367]
- maxClusteringValues: [198]

- Estimated droppable tombstones: 0.0
- SSTable Level: 0
- Repaired at: 0
- Pending repair: --

Replay positions covered: 

- commitlogposition(segmentId=1539277782404, position=19530606)=commitlogposition(segmentId=1539277782404, position=19541152)

**totalColumnsSet**: 4
totalRows: 2
Estimated tombstone drop times:

<table>
<thead>
<tr>
<th>Percentiles</th>
<th>Count (%)</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>50th</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>75th</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>95th</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>98th</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>99th</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Partition Size:

<table>
<thead>
<tr>
<th>Size (bytes)</th>
<th>Count (%)</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>124 (124 B)</td>
<td>1 (100)</td>
<td>#</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Percentiles</th>
<th>Count (%)</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>50th</td>
<td>124 (124 B)</td>
<td></td>
</tr>
<tr>
<td>75th</td>
<td>124 (124 B)</td>
<td></td>
</tr>
<tr>
<td>95th</td>
<td>124 (124 B)</td>
<td></td>
</tr>
<tr>
<td>98th</td>
<td>124 (124 B)</td>
<td></td>
</tr>
<tr>
<td>99th</td>
<td>124 (124 B)</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>104 (104 B)</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>124 (124 B)</td>
<td></td>
</tr>
</tbody>
</table>

Column Count:

<table>
<thead>
<tr>
<th>Columns</th>
<th>Count (%)</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1 (100)</td>
<td>#</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Percentiles</th>
<th>Count (%)</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>50th</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>75th</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>95th</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>98th</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>99th</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Estimated cardinality: 1
EncodingStats minTTL: 0
EncodingStats minLocalDeletionTime: 1442880000 (09/22/2015 00:00:00)
EncodingStats minTimestamp: 1539365167498813 (10/12/2018 17:26:07)
KeyType: org.apache.cassandra.db.marshal.UTF8Type
ClusteringTypes:
[org.apache.cassandra.db.marshal.ReversedType(org.apache.cassandra.db.marshal.Int32Type)]
StaticColumns:
StaticColumns:
RegularColumns: id:org.apache.cassandra.db.marshal.UUIDType,
lastname:org.apache.cassandra.db.marshal.UTF8Type

sstableofflinerelevel

Creates a decent leveling for the given keyspace and table.

**Restriction:** DataStax Enterprise must be stopped before you run this command.
Synopsis

$ sstableofflinerelevel [--dry-run] keyspace_name table_name

Table 418: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Command arguments
--dry-run
Test command syntax and environment. Do not execute the command.

**keyspace_name**
Keyspace name. Required. Overrides the client_encryption_options (page 97) in cassandra.yaml.

**table_name**
Table name. Required.

Examples

**Verify DataStax Enterprise is not running**

$ nodetool status

<table>
<thead>
<tr>
<th>Address</th>
<th>Load</th>
<th>Tokens</th>
<th>Owns</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UN 10.200.177.92</td>
<td>265.04 KiB</td>
<td>1</td>
<td>?</td>
<td>980cab6a-2e5d-44c6-b897-0733dde580ac</td>
</tr>
<tr>
<td>DN 10.200.177.94</td>
<td>426.21 KiB</td>
<td>1</td>
<td>?</td>
<td>7ecbcc0c-627d-403e-b8cc-a2daa93d9ad3</td>
</tr>
</tbody>
</table>

**Relevel calendar table on cycling keyspace**

**Restriction:** DataStax Enterprise must be stopped before you run this command.

$ sstableofflinerelevel cycling calendar

No sstables to relevel for cycling.calendar

**sstablepartitions**

Identifies large partitions of SSTables and outputs the partition size, row count, cell count, and tombstone count.

**Synopsis**

$ sstablepartitions

[ -b ] [-c cell_threshold]
[ -k partition_key]
[ -m ] [-o tombstone_count_threshold]
[ -r ] [-t partition_count_threshold]
[ -u ] [-x partition_keys | -y]

sstable_filepath | sstable_directory
### Table 419: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Italic</td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

The short form and long form parameters are comma-separated.

### Command arguments

- **-b, --backups**
  - Include backups in the data directories (recursive scans).
- **-c, --min-cells cell_threshold**
  - Partition cell count threshold.
- **-k, --key partition_key**
  - Partition keys to include.
-m, --csv
Produce CSV machine-readable output instead of JSON formatted output.
-o, --min-tombstones tombstone_threshold
Partition tombstone count threshold.
-r, --recursive
Recursively.

**sstable_directory**
The absolute path to the SSTable data directory. The `data_file_directories` (page 66) property in cassandra.yaml defines the default directory.

**sstable_filepath**
The explicit or relative filepath to the SSTable data file ending in Data.db.

-t, --min-size partition_threshold
Partition size threshold.
-u, --current-timestamp
Include timestamp in output. Timestamp is the number of seconds since epoch, unit time for TTL expired calculation.
-x, --exclude-key partition_key
Partition key to exclude. Ignored if -y option is given.
-y, --partitions-only
Only brief partition information. Exclude per-partition detailed row/cell/tombstone information from process and output.

**Examples**

**Analyze partition statistics for all SSTables a single table**

```bash
$ sstablepartitions -r /var/lib/cassandra/data/stresscql/blogposts-7dd6dfc289b511e8a4a329556a9391cc/
```

<table>
<thead>
<tr>
<th>Partition size</th>
<th>Row count</th>
<th>Cell count</th>
<th>Tombstone count</th>
</tr>
</thead>
<tbody>
<tr>
<td>p50</td>
<td>124</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p75</td>
<td>149</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p90</td>
<td>149</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>p95</td>
<td>179</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>p99</td>
<td>215</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>p999</td>
<td>258</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>min</td>
<td>51</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>max</td>
<td>8239</td>
<td>179</td>
<td>179</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Count</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>56696</td>
<td>137676</td>
</tr>
</tbody>
</table>

DSE 6.7 Developer Guide (Latest version)
Processing stresscql.blogposts-7dd6dfc289b511e8a4a329556a9391cc #4 (bti-aa) (230134 bytes uncompressed, 192999 bytes on disk)

<table>
<thead>
<tr>
<th>Partition size</th>
<th>Row count</th>
<th>Cell count</th>
<th>Tombstone</th>
</tr>
</thead>
<tbody>
<tr>
<td>p50</td>
<td>124</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p75</td>
<td>124</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p90</td>
<td>149</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p95</td>
<td>149</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p99</td>
<td>149</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p999</td>
<td>179</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>min</td>
<td>51</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>max</td>
<td>446</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>count</td>
<td>2169</td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>3626</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output only partitions with cell count threshold equal to or greater than 10

$ sstablepartitions -c 10 /var/lib/cassandra/data/stresscql/blogposts-7dd6dfc289b511e8a4a329556a9391cc/aa-4-bti-Data.db

Processing stresscql.blogposts-7dd6dfc289b511e8a4a329556a9391cc #4 (bti-aa) (230134 bytes uncompressed, 192999 bytes on disk)

Partition size            Row count           Cell count      Tombstone
p50                    124                    1                    1
p75                    124                    1                    1
p90                    149                    1                    1
p95                    149                    1                    1
p99                    149                    1                    1
p999                   179                    2                    2
min                     51                    0                    0
max                     446                   10                   10
count                 2169

Summary of stresscql.blogposts-7dd6dfc289b511e8a4a329556a9391cc #4 (bti-aa):
File: /home/dimitardimitrov/.ccm/c13529-master/node1/data0/stresscql/blogposts-7dd6dfc289b511e8a4a329556a9391cc/aa-4-bti-Data.db
1 partitions match
Keys: Fwl

Cc xD06iw_\]Q|\[[t[KzCI& $
$ (46776c0b4363097815114430361169775f7f5d511b3b08177c5b745b4b1306007a434926091a24)
live, position: 208502, size: 434, rows: 10, cells: 10, tombstones: 0
(row:0, range:0, complex:0, cell:0, row-TTLd:0, cell-TTLd:0)
Summary of stresscql.blogposts-7dd6dfc289b511e8a4a329556a9391cc #4 (bti-aa):
File: /home/dimitardimitrov/.ccm/c13529-master/node1/data0/stresscql/blogposts-7dd6dfc289b511e8a4a329556a9391cc/aa-4-bti-Data.db
1 partitions match
Keys: Fwl

Cc xD06iw_\]Q|\[[t[KzCI& $
### Output CSV machine-readable output

```
$ sstablepartitions -c 10 -m /var/lib/cassandra/data/stresscql/blogposts-7dd6dfc289b511e8a4a329556a9391cc/aa-4-bti-Data.db

key,keyBinary,live,offset,size,rowCount,cellCount,tombstoneCount,rowTombstoneCount,rangeTombstoneCount,complexTombstoneCount,cellTombstoneCount,rowTtlExpired,cellTtlExpired,directory,keyspace,table,index,snapshot,backup,generation,format,version
"Fwl
Cc xD06iw_]Q|t[KzCI$ 
",46776c0b436309781511430361169775f7f5d511b3b08177c5b745b4b1306007a434926091a24,true,208502,434,10,10,0,0,0,0,0,0,0,
/home/dimitarndimitrov/.ccm/c13529-master/node1/data0/stresscql/blogposts-7dd6dfc289b511e8a4a329556a9391cc/aa-4-bti-
Data.db,stresscql,tableposts,,,,4,bti,aa
```

### sstablerepairedset

Sets status as repaired or unrepaired on a given set of SSTables and updates the `repairedAt` field to denote the time of the repair. This metadata facilitates incremental repairs. Use this tool in the process of migrating an installation to incremental repair.

**Restriction:** DataStax Enterprise must be stopped before you run this command.

**Tip:** Use the following command to list all the *Data.db files in a keyspace:

```
$ find '/home/user/dse-6.7.2/data/keyspace1/' -iname "*Data.db*"
```

### Synopsis

```
$ sstablerepairedset --really-set [--is-repaired | --is-unrepaired] [-f sstable_list_file | sstable_filepath]
```

### Table 420: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( () ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ’ ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Command arguments**

- **-f sstable_list_file**  
The filepath to a file that contains a list of SSTables. For example, a *.txt file.
- **--is-repaired**  
Sets repaired status.
- **--is-unrepaired**  
Sets unrepaired status.
- **--really-set**  
Acknowledgement of potential command impact with DSE stopped.
- **sstable_filepath**
The explicit or relative filepath to the SSTable data file ending in Data.db.

Examples

Verify DataStax Enterprise is not running

```
$ nodetool status
```

```
Datacenter: Graph
========================
Status=Up/Down
| State=Normal/Leaving/Joining/Moving |
-- Address        Load       Tokens       Owns    Host ID  
|-------------------------------------|
```

```
| Rack | UN 10.200.177.92 265.04 KiB 1 | ? | 980cab6a-2e5d-44c6-b897-0733dde980ac rack1 |
```

```
| DN 10.200.177.94 426.21 KiB 1 | ? | 7ecbbc0c-627d-403e-b8cc-a2daa93d9ad3 rack1 |
```

Mark SSTable as repaired

**Restriction:** DataStax Enterprise must be stopped before you run this command.

```
$ sstablerepairedset --really-set --is-repaired /var/lib/cassandra/data/cycling/cyclist_category-e1f76e21ce4311e8949e33016bf887c0/aa-1-bti-Data.db
```

There is no command output.

Use file to list SSTables to mark as unrepaired

**Restriction:** DataStax Enterprise must be stopped before you run this command.

```
$ sstablerepairedset --is-unrepaired -f repairSetSSTables.txt
```

where the `repairSetSSTables.txt` file contains a list of SSTables (*Data.db) files, like:

```
/data/cycling/cyclist_by_country-82246fc065ff11e5a4c58b496c707234/ma-1-big-Data.db
/data/cycling/cyclist_by_birthday-8248246065ff11e5a4c58b496c707234/ma-1-big-Data.db
/data/cycling/cyclist_by_birthday-8248246065ff11e5a4c58b496c707234/ma-2-big-Data.db
/data/cycling/cyclist_by_age-8201305065ff11e5a4c58b496c707234/ma-1-big-Data.db
```
sstablescrub

Scrubs the SSTable for the provided table.

The sstablescrub utility is an offline version of nodetool scrub (page 1067). It attempts to remove the corrupted parts while preserving non-corrupted data. Because sstablescrub runs offline, it can correct errors that nodetool scrub cannot. If an SSTable cannot be read due to corruption, it will be left on disk.

If scrubbing results in dropping rows, new SSTables become unrepaired. However, if no bad rows are detected, the SSTable keeps its original repairedAt field, which denotes the time of the repair.

**Restriction:** DataStax Enterprise must be stopped before you run this command.

Synopsis

```
$ sstablescrub
keyspace_name table_name
```

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ‘ ’ ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>&lt;datatype1,datatype2&gt;</code></td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td><code>cql_statement;</code></td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td><code>[ -- ]</code></td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td><code>'&lt;schema&gt; ... &lt;/schema&gt;'</code></td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td><code>@xml_entity='xml_entity_type'</code></td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Command arguments**

--debug
Display stack traces.

-h, --help
Display the usage and listing of the commands.

`keyspace_name`
Keyspace name. Required. Overrides the `client_encryption_options (page 97)` in `cassandra.yaml`.

-m, --manifest-check
Check and repair only the leveled manifest. Do not scrub the SSTables.

-n, --no-validate
Do not validate columns using column validator.

-r, --reinsert-overflowed-ttl
Rewrite rows with overflowed expiration date affected by CASSANDRA-14092 with the maximum supported expiration date of 2038-01-19T03:14:06+00:00. Rows are rewritten with the original timestamp incremented by one millisecond to override/supersede any potential tombstone that might have been generated during compaction of the affected rows. See Recovering expired data caused by TTL year 2038 problem.

-s, --skip-corrupted
Skips corrupt rows in counter tables.

`table_name`
Table name. Required.

-v, --verbose
Verboso output.
Examples

**Verify DataStax Enterprise is not running**

```
$ nodetool status
```

<table>
<thead>
<tr>
<th>Datacenter: Graph</th>
<th>Status=Up/Down</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>State=Normal/Leaving/Joining/Moving</td>
</tr>
<tr>
<td>--</td>
<td>Address</td>
</tr>
<tr>
<td>RK</td>
<td>10.200.177.92</td>
</tr>
<tr>
<td>DN</td>
<td>10.200.177.94</td>
</tr>
</tbody>
</table>

**Restriction:** DataStax Enterprise must be stopped before you run this command.

**Scrub SSTable for the calendar table**

```
$ sstablescrub cycling calendar
```

**sstablesplit**

Splits SSTable files into multiple SSTables of a maximum designated size.

For example, if SizeTieredCompactionStrategy was used for a major compaction and results in an excessively large SSTable, split the table to ensure that compaction occurs before the next huge compaction.

**Restriction:** DataStax Enterprise must be stopped before you run this command.

**Synopsis**

```
$ sstablesplit [--debug] [-h] [--no_snapshot] [-s max_size_in_MB] sstable_filepath [sstable_filepath ...]
```

**Table 422: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
</tbody>
</table>
### Syntax conventions

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets (&lt; &gt;) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon (; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Command arguments

**--debug**

Display stack traces.

**-h, --help**

Display the usage and listing of the commands.

**--no-snapshot**

Do not snapshot SSTables before splitting.

**-s, --size max_size_in_MB**


**sstable_filepath**

Filepath to an SSTable.
Examples

Verify DataStax Enterprise is not running

```bash
$ nodetool status

Datacenter: Graph
--------------------------------
| State=Normal/Leaving/Joining/Moving |
-- Address        Load       Tokens       Owns    Host ID
| Rack             |
UN 10.200.177.92  265.04 KiB  1            ?       980cab6a-2e5d-44c6-b897-0733dde580ac  rack1
DN 10.200.177.94  426.21 KiB  1            ?       7ecbb0c-627d-403e-b8cc-a2daa93d9ad3  rack1
```

Restriction: DataStax Enterprise must be stopped before you run this command.

Split SSTables to 10 MB

```bash
$ sstablesplit /var/lib/cassandra/data/cycling/cyclist_category-elf76e21ce4311e8949e33016bf887c0/aa-1-bti-Statistics.db 10

Skipping inexisting file 10
Skipping /var/lib/cassandra/data/cycling/cyclist_category-elf76e21ce4311e8949e33016bf887c0/aa-1-bti-Data.db: it's size (0.000 MB) is less than the split size (50 MB)
No sstables needed splitting.
```

sstableupgrade

Upgrades the SSTables in the given table or snapshot to the current version of Cassandra.

Synopsis

```bash
$ sstableupgrade
[--debug]
[-h]
[-k]
[-b]
[--keep-generation]
[-o output-div]
[--sstable-files sstable]
[-t rate-limit]
keyspace_name
table_name
[snapshot_name]
```
Table 423: SSTable compatibility and upgrade version

<table>
<thead>
<tr>
<th>DSE version</th>
<th>SSTable</th>
<th>sstableloader (page 1408) supported format-version</th>
<th>sstableupgrade (page 1428) and nodetool upgradesstables (page 1140) supported format-version</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7.x</td>
<td>bti</td>
<td>aa</td>
<td>big-ma, big-mb, big-mc, and bti-aa</td>
</tr>
<tr>
<td>6.0.x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.1.x</td>
<td>big</td>
<td>mc</td>
<td>big-ka, big-ma, big-mb, and big-mc</td>
</tr>
<tr>
<td>5.0.x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.0.x</td>
<td>mb</td>
<td>big-ka, big-ma, and big-mb</td>
<td>big-ka and big-ma</td>
</tr>
<tr>
<td>5.0.x</td>
<td>ma</td>
<td>big-ka and big-ma</td>
<td>big-k* only</td>
</tr>
</tbody>
</table>

Definition

The short form and long form parameters are comma-separated.

Command arguments

--debug
Display stack traces.

-h, --help
Display the usage and listing of the commands.

-b, --backups
Rewrite incremental backups for the given table. May not be combined with the snapshot_name option.

-k, --keep-source
Do not delete the source SSTables.

--keep-generation
Keep the SSTable generation. Do use with the --keep-source option.

--sstable-files
Instead of processing all SSTables in the default data directories, process only the tables specified via this option. If a single SSTable file, only that SSTable is processed. If a directory is specified, all SSTables within that directory are processed. Snapshots and backups are not supported with this option.

-t, --throughput
Set to limit the maximum disk read rate in MB/s.

datastrax enterprise tools
Keyspace name. Required. Overrides the client_encryption_options (page 97) in cassandra.yaml.

**table_name**
Table name. Required.

**snapshot_name**
Snapshot name.
- Upgrades the specified snapshot before restoring a snapshot from a major version older than the currently running major Cassandra version.
- Replaces files in the given snapshot and breaks any hard links to live SSTables.

**Examples**

**Upgrade events table on cycling keyspace**

```
$ sstableupgrade cycling events
Found 0 sstables that need upgrading.
```

The SSTables are already on the current version, so the command returns immediately and no action is taken.

**sstableutil**

Lists SSTable files for the provided table.

**Synopsis**

```
$ sstableutil [-c] [-d] [-h] [-o]
[-t type] [-v]
keyspace_name table_name
```

**Table 424: Legend**

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Literal keyword.</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Not literal.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ] ) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>'&lt;schema&gt; ... &lt;/schema&gt;'</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

**Definition**

The short form and long form parameters are comma-separated.

**Command arguments**

- **-c, --cleanup**
  Clean up any outstanding transactions.

- **-d, --debug**
  Display stack traces.

- **-h, --help**
  Display the usage and listing of the commands.

- **keyspace_name**
  Keyspace name. Required. Overrides the `client_encryption_options (page 97)` in `cassandra.yaml`.

- **-o, --oplog**
  Include operation logs.

- **table_name**
  Table name. Required.

- **-t, --type type**
  Type of files:
  - all - all final and temporary files
  - tmp - only temporary files
DataStax Enterprise tools

- final - only final files

-v,--verbose
Verbose output.

Examples

List SSTables files for comments table on cycling keyspace

```
$ sstableutil cycling comments

Listing files...
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-1-bti-CompressionInfo.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-1-bti-Data.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-1-bti-Digest.crc32
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-1-bti-Filter.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-1-bti-Partitions.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-1-bti-Rows.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-1-bti-Statistics.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-1-bti-TOC.txt
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-2-bti-CompressionInfo.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-2-bti-Data.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-2-bti-Digest.crc32
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-2-bti-Filter.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-2-bti-Partitions.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-2-bti-Rows.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-2-bti-Statistics.db
/var/lib/cassandra/data/cycling/comments-
eae06ce2ce4211e8949e33016bf887c0/aa-2-bti-TOC.txt
```

**sstableverify**

Verifies the SSTable for the given table.

**Synopsis**

```
$ sstableverify
```
### Table 425: Legend

<table>
<thead>
<tr>
<th>Syntax conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Literal keyword.</td>
</tr>
<tr>
<td><strong>Lowercase</strong></td>
<td>Not literal.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Variable value. Replace with a valid option or user-defined value.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Or. A vertical bar (</td>
</tr>
<tr>
<td>...</td>
<td>Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.</td>
</tr>
<tr>
<td>'Literal string'</td>
<td>Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.</td>
</tr>
<tr>
<td>{ key:value }</td>
<td>Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.</td>
</tr>
<tr>
<td>&lt;datatype1,datatype2&gt;</td>
<td>Set, list, map, or tuple. Angle brackets ( &lt; &gt; ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.</td>
</tr>
<tr>
<td>cql_statement;</td>
<td>End CQL statement. A semicolon ( ; ) terminates all CQL statements.</td>
</tr>
<tr>
<td>[ -- ]</td>
<td>Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.</td>
</tr>
<tr>
<td>' &lt;schema&gt; ... &lt;/schema&gt; '</td>
<td>Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.</td>
</tr>
<tr>
<td>@xml_entity='xml_entity_type'</td>
<td>Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.</td>
</tr>
</tbody>
</table>

### Definition

The short form and long form parameters are comma-separated.

### Command arguments

**--debug**

Display stack traces.
DataStax Enterprise tools

-e, --extended
   Extended verification.
-h, --help
   Display the usage and listing of the commands.

keyspace_name
   Keyspace name. Required. Overrides the client_encryption_options (page 97) in cassandra.yaml.

table_name
   Table name. Required.

-v, --verbose
   Verbose output.

Examples

Verifies cyclist_name table on cycling keyspace

$ sstableverify cycling cyclist_name

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ sstableverify cycling cyclist_name</td>
<td></td>
</tr>
</tbody>
</table>

Verifying
TrieIndexSSTableReader(path='/var/lib/cassandra/data/cycling/cyclist_name-4157ef22ce4411e8949e33016bf887c0/aa-2-bti-Data.db')
   (0.151KiB)
Deserializing sstable metadata for TrieIndexSSTableReader(path='/var/lib/cassandra/data/cycling/cyclist_name-4157ef22ce4411e8949e33016bf887c0/aa-2-bti-Data.db')
Checking computed hash of TrieIndexSSTableReader(path='/var/lib/cassandra/data/cycling/cyclist_name-4157ef22ce4411e8949e33016bf887c0/aa-2-bti-Data.db')
Verifying TrieIndexSSTableReader(path='/var/lib/cassandra/data/cycling/cyclist_name-4157ef22ce4411e8949e33016bf887c0/aa-3-bti-Data.db')
   (0.131KiB)
Deserializing sstable metadata for TrieIndexSSTableReader(path='/var/lib/cassandra/data/cycling/cyclist_name-4157ef22ce4411e8949e33016bf887c0/aa-3-bti-Data.db')
Checking computed hash of TrieIndexSSTableReader(path='/var/lib/cassandra/data/cycling/cyclist_name-4157ef22ce4411e8949e33016bf887c0/aa-3-bti-Data.db')

DataStax tools

Tools that are installed separately and used across products. See DataStax tools.

Preflight check tool

The preflight check tool is a collection of tests that can be run on a DataStax Enterprise node to detect and fix node configurations. The tool can detect and optionally fix many invalid or suboptimal configuration settings, such as user resource limits, swap, and disk settings.
In package installations, the preflight check tool is located in `/usr/share/dse/tools/pfc`. It is not available in tarball installations.

**Usage**

```bash
cd /usr/share/dse/tools/pfc &&
sudo ./preflight_check options
```

<table>
<thead>
<tr>
<th>Table 426: Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Option</strong></td>
</tr>
<tr>
<td>--help</td>
</tr>
<tr>
<td>-h</td>
</tr>
<tr>
<td>--fix</td>
</tr>
<tr>
<td>-f</td>
</tr>
<tr>
<td>--yaml=YAML_LOCATION</td>
</tr>
<tr>
<td>--devices=DEVICES</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>--disk-duration=DISK_DURATION</td>
</tr>
<tr>
<td>--disk-threads=DISK_THREADS</td>
</tr>
<tr>
<td>--ssd=SSD</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>--nossd</td>
</tr>
</tbody>
</table>

**cluster_check and yaml_diff tools**

The cluster_check and yaml_diff tools check the differences between cassandra.yaml or dse.yaml files. This check is particularly useful during upgrades.

**Prerequisites:**

PyYAML must be installed. To install:
These examples check the differences between `cassandra.yaml` files.

- To check differences between YAML files:
  ```
  $ cd /usr/share/dse/tools/yamls &&
  ./yaml_diff path/to/cassandra.yaml path/to/cassandra.yaml.new
  ```

  The `Missing Settings` section of the report lists both missing and deprecated settings.

- To check the differences between each node's YAML in a datacenter:
  For ease of use, use password-less SSH access from the current node to all other nodes.

  ```
  $ cd /usr/share/dse/tools/yamls &&
  ./cluster_check /path/to/cassandra.yaml [/path/to/nodelist]
  ```

  The `nodelist` parameter is optional since the script checks for the list of IP addresses contained in `nodetool status`. The format for the `nodelist` file is one address per line.
Starting and stopping DataStax Enterprise

After you install and configure DataStax Enterprise on one or more nodes, start your cluster beginning with the seed nodes. In a mixed-workload DataStax Enterprise cluster, you must start the analytics seed node first.

Packaged installations include start-up and stop scripts for running DataStax Enterprise as a service. Binary tarballs do not.

OpsCenter provides a options in the Nodes UI for starting, stopping, and restarting DSE on a node. See Starting DSE on a node, Stopping DSE on a node, and Restarting DSE on a node.

Starting DataStax Enterprise as a service

Steps for starting the DataStax Enterprise (DSE) service when DataStax Enterprise was installed from RHEL or Debian packages.

All nodes types are DataStax Enterprise nodes and run the database.

Considerations for starting a cluster

Be aware of the following when starting a DataStax Enterprise cluster:

**Nodes must be segregated by datacenters**

Transactional, DSE Search, DSE Analytics, and SearchAnalytics (page 178) nodes must be in separate datacenters. For example, in a cluster with both DSE Search and transactional nodes, all DSE Search nodes must be in a one or more search datacenters and all transactional nodes must be in one or more datacenters.

DSE Graph can be enabled on any node in any datacenter.

**Deploying a mixed-workload cluster**

When deploying one or more datacenters for each type of node, first determine which nodes to start as transactional, analytic, DSE Graph only, DSE Graph plus other types, DSE Search, and SearchAnalytics nodes. Deploy in this order:

1. Analytic seed nodes.
2. Transactional or DSE Graph-only seed nodes.
3. DSE Search seed nodes.
4. SearchAnalytics nodes.
5. Remaining nodes one at a time. See Initializing multiple datacenters per workload type.

**DSE Analytics nodes**

Before starting DSE Analytics nodes, ensure that the replication factor (page 177) is configured correctly for the analytics keyspaces. Every time you add a new datacenter, you must manually increase the replication factor of the dse_leases keyspace for the new DSE Analytics datacenter.

**Start up commands**

Set the type of node in the /etc/default/dse file. (Start-up scripts are also available in /etc/init.d.)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPH_ENABLED=1</td>
<td>Starts the node as a DSE Graph node.</td>
</tr>
<tr>
<td>SPARK_ENABLED=1</td>
<td>Starts the node as a Spark node and starts the Spark Master service.</td>
</tr>
<tr>
<td>SOLR_ENABLED=1</td>
<td>Starts the node as a DSE Search node.</td>
</tr>
</tbody>
</table>

**Transactional-only or BYOS nodes**

Table 427: Examples

<table>
<thead>
<tr>
<th>Node type</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark Analytics node</td>
<td>SPARK_ENABLED=1</td>
</tr>
<tr>
<td></td>
<td>SOLR_ENABLED=0</td>
</tr>
<tr>
<td></td>
<td>GRAPH_ENABLED=0</td>
</tr>
<tr>
<td>or</td>
<td>SPARK_ENABLED=1</td>
</tr>
<tr>
<td>Note: No entry is the same as disabling it.</td>
<td></td>
</tr>
<tr>
<td>Spark Analytics, DSE Graph, and DSE Search node</td>
<td>SPARK_ENABLED=1</td>
</tr>
<tr>
<td></td>
<td>GRAPH_ENABLED=1</td>
</tr>
<tr>
<td></td>
<td>SOLR_ENABLED=1</td>
</tr>
<tr>
<td>BYOS (Bring Your Own Spark)</td>
<td>Set BYOS nodes as transactional nodes:</td>
</tr>
<tr>
<td>Spark nodes run in separate Spark cluster from a vendor other than DataStax.</td>
<td>All_NODE_TYPES=0 or not present.</td>
</tr>
<tr>
<td>Node type</td>
<td>Settings</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>DSE Graph and BYOS</td>
<td>GRAPH_ENABLED=1</td>
</tr>
<tr>
<td><strong>SearchAnalytics (page 178)</strong> nodes</td>
<td>SPARK_ENABLED=1</td>
</tr>
<tr>
<td></td>
<td>SOLR_ENABLED=1</td>
</tr>
</tbody>
</table>

An integrated DSE SearchAnalytics cluster allows analytics jobs to be performed using CQL queries.

**Prerequisites:** Be sure to read the Considerations for starting a cluster (*page 1437*).

You can also use OpsCenter to start and stop nodes.

1. If DataStax Enterprise is running, stop the node (*page 1443*).

2. Set the node type in the `/etc/default/dse` file. For example, to a Spark node:

   ```
   SPARK_ENABLED=1
   SOLR_ENABLED=0
   GRAPH_ENABLED=0
   ```

   **Note:** Alternately, you can omit the other start up entries and just use
   ```
   SPARK_ENABLED=1
   ```

3. Start DataStax Enterprise:

   ```
   $ sudo service dse start
   ```

   If the following error appears, see DataStax Enterprise times out when starting.

   ```
   WARNING: Timed out while waiting for DSE to start.
   ```

4. To verify that the cluster is running:

   ```
   $ nodetool status
   ```

   **Note:** If DSE has problems starting, see Troubleshooting starting and installing DSE.

The nodetool command shows the node type and the status. For a transactional node running in a normal state (UN) with virtual nodes (vnodes) enabled shows:

```
Datacenter: Cassandra
============
Status=Up/Down
| State=Normal/Leaving/Joining/Moving
```
Operations

<table>
<thead>
<tr>
<th>-- Address</th>
<th>Load</th>
<th>Tokens</th>
<th>Owns</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UN 127.0.0.1</td>
<td>82.43 KB</td>
<td>128</td>
<td>?</td>
<td>rack1</td>
</tr>
</tbody>
</table>

For example, a running node in a normal state (UN) with DSE Analytics without vnodes enabled shows:

Datacenter: Analytics
=============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
<table>
<thead>
<tr>
<th>-- Address</th>
<th>Load</th>
<th>Owns</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UN 172.16.222.136</td>
<td>103.24 KB</td>
<td>?</td>
<td>3c1d0657-0990-4f78-a3c0-3e0c37fc3a06 1647352612226902707 rack1</td>
</tr>
</tbody>
</table>

Starting DataStax Enterprise as a stand-alone process

Steps for starting the DataStax Enterprise (DSE) process when DataStax Enterprise was installed from a tarball.

All nodes types are DataStax Enterprise nodes and run the database.

Considerations for starting a cluster

Be aware of the following when starting a DataStax Enterprise cluster:

**Nodes must be segregated by datacenters**

Transactional, DSE Search, DSE Analytics, and SearchAnalytics (page 178) nodes must be in separate datacenters. For example, in a cluster with both DSE Search and transactional nodes, all DSE Search nodes must be in a one or more search datacenters and all transactional nodes must be in one or more datacenters.

DSE Graph can be enabled on any node in any datacenter.

**Deploying a mixed-workload cluster**

When deploying one or more datacenters for each type of node, first determine which nodes to start as transactional, analytic, DSE Graph only, DSE Graph plus other types, DSE Search, and SearchAnalytics nodes. Deploy in this order:

1. Analytic seed nodes.
2. Transactional or DSE Graph-only seed nodes.
3. DSE Search seed nodes.
4. SearchAnalytics nodes.
5. Remaining nodes one at a time. See Initializing multiple datacenters per workload type.

**DSE Analytics nodes**
Before starting DSE Analytics nodes, ensure that the replication factor (page 177) is configured correctly for the analytics keyspaces. Every time you add a new datacenter, you must manually increase the replication factor of the dse_leases keyspace for the new DSE Analytics datacenter.

**Start up commands**

1. Start the node from the *installation_location*.

2. Set the type.

<table>
<thead>
<tr>
<th>Node/datacenter</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactional only</td>
<td>bin/dse cassandra</td>
</tr>
<tr>
<td>DSE Graph</td>
<td>bin/dse cassandra -g</td>
</tr>
<tr>
<td>DSE Analytics with Spark</td>
<td>bin/dse cassandra -k</td>
</tr>
<tr>
<td>DSE Search</td>
<td>bin/dse cassandra -s</td>
</tr>
</tbody>
</table>

**Note:** When multiple flags are used, list them separately on the command line. For example, ensure there is a space between `-k` and `-s` in `dse cassandra -k -s`.

**Table 428: Starting examples**

<table>
<thead>
<tr>
<th>From the <em>installation_location</em>:</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark Analytics, DSE Graph, and DSE Search node</td>
<td>bin/dse cassandra -k -g -s</td>
</tr>
<tr>
<td>BYOS (Bring Your Own Spark)</td>
<td>bin/dse cassandra</td>
</tr>
<tr>
<td>Spark nodes run in separate Spark cluster from a vendor other than DataStax.</td>
<td>bin/dse cassandra -g</td>
</tr>
<tr>
<td>DSE Graph and BYOS</td>
<td>bin/dse cassandra -g</td>
</tr>
<tr>
<td>SearchAnalytics (page 178) nodes</td>
<td>bin/dse cassandra -k -s</td>
</tr>
</tbody>
</table>

**Prerequisites:** Be sure to read the Considerations for starting a cluster (page 1440).
You can also use **OpsCenter** to start and stop nodes.

1. If DataStax Enterprise is running, stop the node *(page 1443)*.

2. From the install directory, start the node. For example, to set a Spark node:
   ```
   bin/dse cassandra -k
   ```

3. To check that your ring is up and running:
   ```
   $ cd installation_location &&
   bin/nodetool status
   ```

   where the installation directory is either:
   - `/usr/share/dse`
   - DataStax Enterprise installation directory

   **Note:** If DSE has problems starting, see *Troubleshooting starting and installing DSE*.

The `nodetool` command shows the node type and the status. For a transactional node running in a normal state (UN) with virtual nodes (vnodes) enabled shows:

```
Datacenter: Cassandra
============
Status=Up/Down
  | State=Normal/Leaving/Joining/Moving
  -- Address    Load       Tokens  Owns    Host ID
     Rack
UN  127.0.0.1  82.43 KB   128     ?
   4072dc8-7843-43ae-9c98-7c532b1f517e  rack1
```

For example, a running node in a normal state (UN) with DSE Analytics without vnodes enabled shows:

```
Datacenter: Analytics
============
Status=Up/Down
  | State=Normal/Leaving/Joining/Moving
  -- Address    Load       Owns    Host ID
     Token                 Rack
UN  172.16.222.136  103.24 KB  ?       3c1d0657-0990-4f78-
   a3c0-3e0c37fc3a06  1647352612226902707   rack1
```

**Stopping a DataStax Enterprise node**

To speed up the restart process, before stopping the `dse` service, run `nodetool drain` *(page 940)*. This step writes the current memtables to disk. When you restart the node, the commit
log is not read which speeds the restart process. If you have durable writes set to false, which is unlikely, there is no commit log and you must drain the node to prevent losing data.

To stop DataStax Enterprise running as a service:

```bash
$ nodetool drain
$ sudo service dse stop
```

To stop DataStax Enterprise running as a stand-alone process:

Running nodetool drain before using the `cassandra-stop` command to stop a stand-alone process is not necessary because the `cassandra-stop` command drains the node before stopping it.

From the installation location:

```bash
$ bin/dse cassandra-stop
```

**Note:** Use `sudo` if required.

In the unlikely event that the `cassandra-stop` command fails because it cannot find the process DataStax Enterprise Java process ID (PID), the output instructs you to find the DataStax Enterprise Java process ID (PID) manually, and stop the process using its PID number.

```bash
ps auwx | grep dse
```

Use the PID, in the second column of the output, to stop the database.

```bash
bin/dse cassandra-stop -p PID
```

**Note:** If you have trouble, see Troubleshooting starting DataStax Enterprise.

### Clearing the data from DataStax Enterprise

Remove all data from any type of installation.

**Package installation**

To clear the data from the `default` directories:

1. After Stop (page 1442) the service.
2. Run one of the following commands:

   ```bash
   $ sudo rm -rf /var/lib/cassandra/*  # Remove all data
   ```
Tarball installation

To clear all data from the default directories:

1. **Stop** *(page 1437)* the DataStax Enterprise process.

2. Remove the data from the installation location:

   $ cd installation_location

   Run one of the following commands:

   $ sudo rm -rf data/* commitlog/* saved_caches/* hints/* ## Remove all data

   $ sudo rm -rf data/* ## Remove only the data directories
CQL

CQL (Cassandra Query Language) is a query language for the DataStax Enterprise database. See CQL for DataStax Enterprise 6.7.