Table of Contents

Getting started.................................................................................................................. 11
  About this guide............................................................................................................. 11
  New features.................................................................................................................. 12
  Key features.................................................................................................................. 15
Release notes.................................................................................................................... 18
  DSE 5.1.11..................................................................................................................... 19
    Cassandra enhancements for 5.1.11........................................................................... 25
    Upgrade advice for 5.1.11......................................................................................... 26
    Spark Cassandra Connector changes for 5.1.11....................................................... 29
    TinkerPop changes for DSE 5.1.11......................................................................... 31
  DSE 5.1.10..................................................................................................................... 31
    Cassandra enhancements for 5.1.10........................................................................... 35
    Upgrade advice for 5.1.10......................................................................................... 36
    TinkerPop changes for 5.1.10................................................................................... 39
  DSE 5.1.9....................................................................................................................... 39
  DSE 5.1.8....................................................................................................................... 40
    Cassandra enhancements for 5.1.8........................................................................... 46
    Upgrade advice for 5.1.8......................................................................................... 54
    TinkerPop changes for 5.1.8................................................................................... 57
  DSE 5.1.7....................................................................................................................... 57
    Cassandra enhancements for 5.1.7........................................................................... 62
    Upgrade advice for 5.1.7......................................................................................... 62
    Spark Cassandra Connector changes for 5.1.7....................................................... 67
    TinkerPop changes for 5.1.7................................................................................... 68
  DSE 5.1.6....................................................................................................................... 68
    Cassandra enhancements for 5.1.6........................................................................... 77
    Upgrade advice for 5.1.6......................................................................................... 77
Installing DSE........................................................................................................... 222

Which install method should I use?........................................................................ 222

DataStax Installers................................................................................................... 223

DataStax Installer (root permissions)......................................................................... 223

DataStax Installer (no root permissions).................................................................. 231

DataStax Installer for Mac OS X............................................................................... 239
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataStax Installer (unattended)</td>
<td>245</td>
</tr>
<tr>
<td>Binary tarball installer</td>
<td>252</td>
</tr>
<tr>
<td>Installing on Docker</td>
<td>256</td>
</tr>
<tr>
<td>Installing supporting software</td>
<td>256</td>
</tr>
<tr>
<td>Installing Oracle JDK on Debian or Ubuntu Systems</td>
<td>256</td>
</tr>
<tr>
<td>Installing Oracle JDK on RHEL-based Systems</td>
<td>258</td>
</tr>
<tr>
<td>Installing OpenJDK on Debian-based Systems</td>
<td>259</td>
</tr>
<tr>
<td>Installing OpenJDK on RHEL-based Systems</td>
<td>260</td>
</tr>
<tr>
<td>Uninstalling DSE</td>
<td>260</td>
</tr>
<tr>
<td>Default file locations</td>
<td>262</td>
</tr>
<tr>
<td>Package and Installer-Services installations</td>
<td>262</td>
</tr>
<tr>
<td>Tarball and Installer-No Services installations</td>
<td>267</td>
</tr>
<tr>
<td>Configuration</td>
<td>272</td>
</tr>
<tr>
<td>Recommended production settings</td>
<td>272</td>
</tr>
<tr>
<td>YAML and configuration properties</td>
<td>278</td>
</tr>
<tr>
<td>cassandra.yaml</td>
<td>278</td>
</tr>
<tr>
<td>dse.yaml</td>
<td>312</td>
</tr>
<tr>
<td>cassandra-rackdc.properties</td>
<td>355</td>
</tr>
<tr>
<td>cassandra-topology.properties</td>
<td>356</td>
</tr>
<tr>
<td>Configuring snitches for cloud providers</td>
<td>357</td>
</tr>
<tr>
<td>Ec2Snitch</td>
<td>357</td>
</tr>
<tr>
<td>Ec2MultiRegionSnitch</td>
<td>358</td>
</tr>
<tr>
<td>GoogleCloudSnitch</td>
<td>361</td>
</tr>
<tr>
<td>CloudstackSnitch</td>
<td>362</td>
</tr>
<tr>
<td>Start-up parameters</td>
<td>362</td>
</tr>
<tr>
<td>Choosing a compaction strategy</td>
<td>366</td>
</tr>
<tr>
<td>Configuring Virtual Nodes</td>
<td>367</td>
</tr>
<tr>
<td>Virtual node (vnode) configuration</td>
<td>367</td>
</tr>
<tr>
<td>DSE advanced functionality</td>
<td>370</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>DSE Graph Reference</td>
<td>931</td>
</tr>
<tr>
<td>DSE Advanced Replication</td>
<td>957</td>
</tr>
<tr>
<td>About DSE Advanced Replication</td>
<td>958</td>
</tr>
<tr>
<td>Architecture</td>
<td>958</td>
</tr>
<tr>
<td>Traffic between the clusters</td>
<td>961</td>
</tr>
<tr>
<td>Terminology</td>
<td>963</td>
</tr>
<tr>
<td>Getting started</td>
<td>964</td>
</tr>
<tr>
<td>Keyspaces</td>
<td>975</td>
</tr>
<tr>
<td>Data Types</td>
<td>976</td>
</tr>
<tr>
<td>Operations</td>
<td>977</td>
</tr>
<tr>
<td>CQL queries</td>
<td>996</td>
</tr>
<tr>
<td>Metrics</td>
<td>998</td>
</tr>
<tr>
<td>Managing invalid messages</td>
<td>1006</td>
</tr>
<tr>
<td>Managing audit logs</td>
<td>1007</td>
</tr>
<tr>
<td>DSE Advanced Replication command line tool</td>
<td>1009</td>
</tr>
<tr>
<td>Tools</td>
<td>1022</td>
</tr>
<tr>
<td>dsetool</td>
<td>1022</td>
</tr>
<tr>
<td>About dsetool</td>
<td>1022</td>
</tr>
<tr>
<td>Connection options</td>
<td>1023</td>
</tr>
<tr>
<td>autoj</td>
<td>1025</td>
</tr>
<tr>
<td>checkcfs</td>
<td>1025</td>
</tr>
<tr>
<td>core_indexing_status</td>
<td>1027</td>
</tr>
<tr>
<td>create_core</td>
<td>1029</td>
</tr>
<tr>
<td>createsystemkey</td>
<td>1032</td>
</tr>
<tr>
<td>encryptconfigvalue</td>
<td>1034</td>
</tr>
<tr>
<td>get_core_config</td>
<td>1034</td>
</tr>
<tr>
<td>get_core_schema</td>
<td>1036</td>
</tr>
<tr>
<td>help</td>
<td>1038</td>
</tr>
</tbody>
</table>
Stress tools

dse client-tool
write_resource

1068
ring
perf
1057
1047
list_index_files
list_subranges
managekmip list
managekmip expirekey
managekmip revoke
managekmip destroy
node_health
partitioner
perf
read_resource
rebuild_indexes
repaircfs
reload_core
ring
sparkmaster cleanup
sparkworker restart
status
stop_core_reindex
tieredtablestats
treload
unload_core
upgrade_index_files
write_resource
dse client-tool
Stress tools
Interpreting the output of cassandra-stress................................................. 1087

**Operations** .................................................................................................. 1090

Starting and stopping DSE ............................................................................. 1090
Starting as a service ......................................................................................... 1090
Starting as a stand-alone process .................................................................. 1093
Stopping a node ............................................................................................... 1095
Clearing data from DSE ................................................................................. 1096

**Studio** ........................................................................................................... 1098

About DataStax Studio .................................................................................... 1098
Release notes .................................................................................................... 1098
Studio 2.0.0 ...................................................................................................... 1098
Studio Getting started ..................................................................................... 1099
Installing and running DataStax Studio 2.0 .................................................. 1099
Starting and stopping Studio ......................................................................... 1101
Creating a simple notebook ........................................................................... 1102
Creating a connection ..................................................................................... 1104
Using SSL connections .................................................................................. 1106
Creating a graph in a notebook ..................................................................... 1107
Interact with data using CQL ......................................................................... 1112
Using notebooks ............................................................................................. 1115
Listing notebooks using Notebook Manager ................................................ 1116
Defining run behavior .................................................................................... 1116
Notebook cells ............................................................................................... 1118
Notebook code editor ..................................................................................... 1120
User data for notebooks .................................................................................. 1124
Configuring Studio ......................................................................................... 1125
Basic configuration options ......................................................................... 1125
Advanced configuration options ................................................................. 1128
Studio reference ............................................................................................. 1129
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connections page</td>
<td>1129</td>
</tr>
<tr>
<td>Default imports</td>
<td>1129</td>
</tr>
<tr>
<td>Keyboard shortcuts</td>
<td>1131</td>
</tr>
<tr>
<td>FAQ</td>
<td>1132</td>
</tr>
<tr>
<td>Troubleshooting DataStax Studio</td>
<td>1133</td>
</tr>
<tr>
<td>CQL</td>
<td>1134</td>
</tr>
</tbody>
</table>
Getting started

Information about using this guide, plus new and key features in DataStax Enterprise 5.1.

About the DataStax Enterprise 5.1 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 5.1 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. Other tasks can be accomplished using **OpsCenter**.

This guide includes documentation for:

**Install methods (page 222)**
- Types of installs generally used by developers.

**DSE Analytics (page 370)**
- DSE Analytics uses Apache Spark™ to perform analytic queries over large sets of data. Topics include starting, configuring, running commands against a remote cluster, accessing data, and a number of examples.

**DSE Graph (page 651)**
- A graph database for storing information about the relationships between entries. Topics include getting started, terminology, data modeling, anti-patterns, importing data, tools, and graph analytics.

**DSE Search (page 507)**
- DSE Search simplifies using search applications for data stored in a database. DSE Search integrates Apache Solr™ 6.0.1 to manage search indexes with a persistent store.

**DSEFS (page 475)**
- DataStax Enterprise File System is distributed file system for storing very large sets of data and encapsulated that data across a DSE cluster. It can be stored for...
Getting started

processing by DSE analytics and other tools. It's a replacement for CFS (Cassandra File System) and is roughly equivalent to HDFS.

**DataStax Studio (page 1098)**
DataStax Studio is an interactive tool for CQL (Cassandra Query Language) and DSE Graph.

**DataStax DevCenter**
DevCenter is not supported for DSE versions 5.1 and later.

**Other information sources**

<table>
<thead>
<tr>
<th>Architecture Guide</th>
<th>How the DataStax Enterprise database works.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrators Guide</td>
<td>Information about capacity planning, installation, configuration, migration, performance monitoring, security, backup, data recovery and more.</td>
</tr>
<tr>
<td>CQL for DSE 5.1</td>
<td>Cassandra Query Language (CQL) is a query language for the DataStax Enterprise database. You can interact with the database using the CQL shell, <code>cqlsh</code>, or DataStax Studio (page 1098).</td>
</tr>
<tr>
<td>Landing pages</td>
<td>Getting started with DSE, supported platforms, product compatibility, third-party software, resources for additional information, and earlier documentation.</td>
</tr>
<tr>
<td>OpsCenter</td>
<td>Installing and using DSE OpsCenter and Lifecycle Manager.</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>DSE Troubleshooting Guide</td>
<td>Various troubleshooting topics including Linux settings, search, analytics, security, starting DSE, and installing.</td>
</tr>
<tr>
<td>Upgrade Guide</td>
<td>Information on upgrading various versions of DataStax Enterprise and upgrading from Apache Cassandra to DataStax Enterprise.</td>
</tr>
<tr>
<td>Sources of support</td>
<td>DataStax Support, DataStax Academy forums, Stack Overflow for DataStax Enterprise, Stack Overflow for the DataStax Java client driver and the DataStax PHP driver.</td>
</tr>
</tbody>
</table>

**DataStax Enterprise 5.1 new features**

DataStax Enterprise 5.1 introduces the following new features and enhancements:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSE Analytics (page 370)</td>
<td>Support for Apache Spark™ 2.0 including SparkR. Improvements include operational improvements, performance improvements, structured streaming, DSE GraphFrames, Spark SQL, and geospatial types.</td>
</tr>
<tr>
<td>DSEFS (page 475)</td>
<td>DataStax Enterprise File System provides a distributed file system for storing very large data sets, such as Spark Streaming data and analytic processing. DSEFS replaces the CFS (Cassandra File System).</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DSE Graph <em>(page 651)</em></td>
<td>Improvements includes new fuzzy search; advanced configuration of search indexing; better data loading performance; improved geospatial querying; Kerberos support for Graph Loader; ability to customize graph visualizations; and stability improvements.</td>
</tr>
<tr>
<td>DSE Security</td>
<td>New security features include granular access control that allows permissions on table rows and search indexes; DSE Proxy management for web services; and JMX authentication integrated with DSE Unified Authentication (LDAP or internal). For details, see <a href="#">New security features in DSE 5.1</a>.</td>
</tr>
<tr>
<td>DSE Advanced Replication <em>(page 958)</em></td>
<td>Re-designed to use CDC (Change Data Capture) logs, CDC is ideal for configurable replication between sources and destinations. Suitable for environments where sporadic connectivity can occur, such as a network of microservices clusters that report data to a central analytics cluster.</td>
</tr>
<tr>
<td>Drivers</td>
<td>Support for new DSE 5.1 functionality, such as Unified Authentication, proxy login and execution, and the new data types. Additionally, the Java and Python drivers contain a DSE Graph fluent API that use the Gremlin Traversal API for programmatically building Gremlin queries.</td>
</tr>
<tr>
<td>DataStax Studio <em>(page 1098)</em></td>
<td>Added support for CQL (Cassandra Query Language). This new features provides the ability to visually navigate database objects, create and tune CQL queries. Studio features an intelligent CQL editor providing syntax highlighting, validation, intelligent code completion, configuration options, and query profiling. Improvements in DSE Studio for DSE Graph include better usability; more complete profiling for graph queries; and new customization capabilities for graph visualization, including coloring and sizing vertices by label or property value, and custom shapes and icons. Schema visualization is also improved.</td>
</tr>
<tr>
<td>DSE In-Memory</td>
<td>MemoryOnlyStrategy now works with compression.</td>
</tr>
<tr>
<td>Operations</td>
<td>Performance improvements including faster server startup. Improved help for CQL and cqlsh commands. Tab completion to cqlsh for DSE custom compaction strategies. Improvements to dsetool <em>(page 1022)</em> and dse client-tool <em>(page 1082)</em>.</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>New database features</strong></td>
<td></td>
</tr>
<tr>
<td><strong>cassandra-stress -graph option</strong></td>
<td>Results can be automatically graphed for data visualization.</td>
</tr>
<tr>
<td><strong>Clustering columns improvement</strong></td>
<td>Clustering columns without a secondary index can be used in a WHERE clause, provided the ALLOW FILTERING clause is also used.</td>
</tr>
<tr>
<td><strong>CREATE TABLE WITH ID</strong></td>
<td>If a table is accidentally dropped, recreate it with its ID and replay the commitlog to regain data.</td>
</tr>
<tr>
<td><strong>JBOD compaction and backup</strong></td>
<td>Updated SSTable partitioning by token range has improved JBOD compaction and backup. The nodetool relocatesstables command supports the improvement. For details, see the Improving JBOD blog.</td>
</tr>
<tr>
<td><strong>jvm.options file</strong></td>
<td>Garbage collection (GC) and other JVM options moved to the jvm.options (page 362) file from the cassandra-env.sh file.</td>
</tr>
<tr>
<td><strong>nodetool updates</strong></td>
<td></td>
</tr>
<tr>
<td>• nodetool compact --user-defined</td>
<td>Allows submission of a file list. Useful for low disk space or tombstone purging.</td>
</tr>
<tr>
<td>• nodetool proxyhistograms</td>
<td>CAS read and write latency is displayed for compare-and-set operations.</td>
</tr>
<tr>
<td>• nodetool getsstables --hex-format</td>
<td>Option for using a hex-formatted key to get SSTables.</td>
</tr>
<tr>
<td>• nodetool gettimeout</td>
<td>Prints value of a timeout.</td>
</tr>
<tr>
<td>• nodetool settimeout</td>
<td>Sets the value of a timeout in milliseconds.</td>
</tr>
<tr>
<td><strong>PER PARTITION LIMIT</strong></td>
<td>A query can be limited to return results from each partition, such as a Top 3 listing.</td>
</tr>
<tr>
<td><strong>sstableloader -ap option</strong></td>
<td>sstableloader (Bulkloader) can use third-party authentication.</td>
</tr>
<tr>
<td><strong>Static columns improvements</strong></td>
<td>Static columns can be indexed (experimental). Static columns can be used with SASI indexes (experimental).</td>
</tr>
<tr>
<td><strong>Timestamp in sub-second precision</strong></td>
<td>Timestamp defaults include sub-second precision.</td>
</tr>
<tr>
<td><strong>TTL for COPY FROM</strong></td>
<td>Specify a TTL value when copying from CSV files.</td>
</tr>
<tr>
<td><strong>user-defined type (UDT) improvements</strong></td>
<td>If a UDT has only non-collection fields, an individual field value can be updated or deleted.</td>
</tr>
</tbody>
</table>
For more details, see the DataStax Enterprise 5.1 release notes (page 18). To upgrade to DataStax Enterprise 5.1, see the DataStax Upgrade Guide.

DataStax Enterprise 5.1 key features

DataStax Enterprise, powered by the best distribution of Apache Cassandra™, seamlessly integrates your code, allowing applications to utilize a breadth of techniques to produce a mobile app or online applications. DSE makes it easy to distribute your data across datacenters or cloud regions, making your applications always-on, ready to scale, and able to create instant insight and experiences. DataStax Enterprise provides flexibility to deploy on any on-premise, cloud infrastructure, or hybrid cloud, plus the ability to use multiple operational workloads, such as analytics and search, without any operational performance degradation.

Database

DataStax Enterprise has added powerful new capabilities to the Apache Cassandra database with advanced security, tiered storage, row-level access control, and more. Thus, in the DataStax Enterprise 5.1 documentation, the Cassandra database and related commercial-only features are referred to cumulatively as the DataStax Enterprise database, the database, or DataStax Enterprise depending on the context.

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 370)

Provides real-time, streaming, and batch operational analytics with an enhanced version of Apache Spark™ 2.0 — a distributed, parallel data processing engine.

DSE Graph (page 651)

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 507)

Integrated with Apache Solr™ 6.0 to provide continuously available search. Index management CQL and cqlsh commands (page 536) 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).
Getting started

**DSE Advanced Replication (page 958)**
Supports configurable distributed data replication from source clusters to destination clusters bi-directionally. It is 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.

**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, antipatterns, 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 754)

**Developer tools**
Javadoc, demos, DataStax Studio (page 1098)

**Production tools**
OpsCenter and Lifecycle Manager, nodetool, dsetool (page 1022), DSE Graph Loader (page 831)

DataStax Drivers

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

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

**Drivers for DSE 5.0 and later**
These drivers can only be used with DataStax Enterprise and support the advanced functionality of DataStax Enterprise 5.1:
Getting started

- 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 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 5.1 release notes

DataStax Enterprise release notes cover cluster requirements, upgrade guidance, components, changes and enhancements, issues, and resolved issues for DataStax Enterprise (DSE) 5.1.

Note: Each point release includes a highlights and executive summary section to provide guidance and add visibility to important improvements.

Warning:

DSE 5.1.9 and 5.1.8 Avoid upgrading to DSE 5.1.8 or DSE 5.1.9 if you use TTL (time-to-live) with DSE Search live indexing (RT indexing). Upgrade to DSE 5.1.10 or later to use TTL with DSE Search live indexing. (DSP-16038 (page 40))

Requirement for Uniform Clusters

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 Basic, or all 5 Nodes must be DataStax Enterprise. Mixing different Subscriptions within a Cluster is not permitted. “Cluster” means a collection of Nodes running the Software which communicate with one another via Gossip, and “Gossip” means the mechanism within the Software enabling related Nodes to communicate with one another. For more information, see Enterprise Terms.

Before you upgrade

The latest version of DataStax Enterprise is 5.1.11.

• Be sure to read the DataStax Upgrade Guide. Upgrades to DSE 5.1 are supported only from DSE 5.0.
• Ensure that you upgrade to the latest patch release on your current version before you upgrade to a higher version. Fixes included in the latest patch release might help or smooth the upgrade process.
• Product compatibility
  # DataStax Enterprise 5.1 is compatible with:
    # DSE OpsCenter 6.5, 6.1
    # DataStax Studio 2.0.0
  
  # DataStax Drivers

  Depending on the driver version, you might need to recompile your client application code. See Upgrading DataStax drivers.

  # DataStax Bulk Loader
    # Can load data into DSE 5.0 or later
# Can unload data from any Apache Cassandra™ 2.1 or later data source

- Be sure to check compatibility for:
  
  # Products

  Review compatibility across DataStax products and other software.

  # Drivers

  Your driver might not be compatible with this version or require re-compiling.

## DSE 5.1.11

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

14 September 2018

- 5.1.11 Components *(page 19)*
- 5.1.11 Highlights *(page 19)*
- 5.1.11 Changes and enhancements *(page 20)*
- 5.1.11 Resolved issues *(page 22)*
- 5.1.11 Cassandra enhancements *(page 25)*
- 5.1.11 General upgrade advice *(page 26)*
- 5.1.11 TinkerPop changes *(page 31)*

### 5.1.11 Components

- Apache Solr™ 6.0.1.0.2304 (updated)
- Apache Spark™ 2.0.2.21 (updated)
- Apache Tomcat® 8.0.47
- DataStax Spark Cassandra Connector 2.0.10 (updated)
- DSE Java Driver 1.2.6
- Netty 4.0.54.Final (updated)
- Spark Jobserver 0.6.2.238 requires compatible API
- TinkerPop 3.2.9-20180507-f6ead8b2
- Select Hadoop libraries

DSE 5.1.11 is compatible with Apache Cassandra™ 3.11.0 and adds production-certified changes *(page 25)* and enhancements.

### 5.1.11 Highlights

Executive summary highlights for DSE 5.1.11:

- DSE Analytics and DSEFS *(page 20)*
- DSE Graph *(page 20)*
- DSE Search *(page 20)*
The executive summary highlights are just a top-level view. Be sure to review all 5.1.11 Changes and enhancements (page 20).

5.1.11 DSE Analytics and DSEFS highlights

- Improved security with Spark user isolation. (DSP-16093)
- Client and internode connection improvements. Configurable connections and pools. (DSP-14284, DSP-16065)
- Improved security: DSEFS uses an isolated native memory pool for file data and metadata sent between nodes. This isolation makes it harder to exploit potential memory management bugs. (DSP-16492)
- Fix for duration type in a keyspace that prevented DSEFS from starting. (DSP-16825)
- Fix for failures in Spark when wrong type of exceptions occur on file not found. (DSP-16933)

5.1.11 DSE Graph highlights

- Fix unresponsive nodes following Gremlin timeouts. (DSP-16544)

5.1.11 DSE Search highlights

- Fixes NoSuchMethodError or NoClassDefFoundError exceptions when attempting to use a Snowball-generated stemmer. (DSP-16116)
- DSE will not start without appropriate Tomcat JAR scanning exclusions. (DSP-16841)

5.1.11 Changes and enhancements

In addition to the 5.1.11 Highlights (page 19), review all changes and enhancements:

- DataStax Enterprise (page 20)
- DSE Analytics (page 20)
- DSEFS (page 21)
- DSE Graph (page 21)
- DSE Search (page 22)

5.1.11 DataStax Enterprise core changes and enhancements

- Tool sstablepartitions for identifying large partitions. (DB-803)
- Connections on non-serialization errors are not dropped. (DB-2233)
- Create a log message when DDL statements are executed. (DB-2383)
- Improved error handling and logging for TDE encryption key management. (DSP-15314)
- sstableloader supports custom config file locations. (DSP-16092)
- DataStax does more extensive testing on OpenJDK 8 due to the end of public updates for Oracle JRE/JDK 8. (DSP-16179)

5.1.11 DSE Analytics changes and enhancements
• DSE client applications, like Spark, hard stop if user home is not defined, does not exist, or the current user does not have write permissions. (DSP-15476)
• DSE pyspark libraries are added to PYTHONPATH for dse exec command. Add support for Jupyter integration (page 409). (DSP-16797)

5.1.11 DSEFS changes and enhancements

• DSEFS operations (page 483): chown, chgrp, and chmod support recursive (-R) and verbose (-v) flag. (DSP-14238)
• Client and internode connection improvements. (DSP-14284, DSP-16065)
  
  # DSEFS clients close idle connections after 60 seconds, configurable in dse.yaml (page 341).
  # Idle DSEFS internode connections are closed after 120 seconds. Configurable with new dse.yaml option internode_idle_connection_timeout_ms (page 341).
  # Configurable connection pool with core_max_concurrent_connections_per_host (page 342).
• Improved error message when performing an operation on a corrupted path. (DSP-16340)
• Security improvements:
  
  # Only super users are able to remove corrupted non-empty directories when authentication is enabled for DSEFS. (DSP-16340)
  # DSEFS uses an isolated native memory pool for file data and metadata sent between nodes. This isolation makes it harder to exploit potential memory management bugs. (DSP-16492)

5.1.11 DSE Graph changes and enhancements

• Improved Gremlin console authentication configuration. (DSP-9905)
• Maximum evaluation timeout is 1094 days. (DSP-16709)
  
  # Gremlin evaluation_timeout parameter:

  ```java
  schema.config().option('graph.traversal_sources.g.evaluation_timeout').set(Duration.ofDays(1094))
  ```

  # dse.yaml options: analytic_evaluation_timeout (page 348), realtime_evaluation_timeout (page 349)
• Default write consistency level (CL) for Graph is LOCAL_QUORUM. (DSP-17140)
  
  **Attention:** In earlier DSE versions, the default QUORUM write consistency level (CL) was not appropriate for multi-datacenter production environments.
• Added convenience methods for reading graph configuration: getEffectiveAllowScan and getEffectiveSchemaMode. (DSP-16650)
• The hardcoded default schema_mode (page 349) is changed from Development to Production. (DSP-16650)
5.1.11 DSE Search changes and enhancements

- Log fewer messages at INFO level in TTLIndexRebuildTask. (DSP-15600)
- Search index permissions can be applied at keyspace level. (DSP-15835)
- CQL solr_query supports Solr facet heatmaps. (DSP-16404)
- Drop operations (ALTER SEARCH INDEX SCHEMA DROP) on the schema now require including at least one attribute on the element being dropped and support dropping only one element at a time. (DSP-15947)

The required attributes by element are:

- field - name
- fieldType - name
- dynamicField - name
- copyField - source, dest

5.1.11 Resolved issues

Resolved issues for:

- DataStax Enterprise core (page 22)
- DSE Analytics (page 23)
- DSEFS (page 23)
- DSE Graph (page 23)
- DSE Search (page 24)

5.1.11 DataStax Enterprise resolved issues

- Set MX4J_ADDRESS to 127.0.0.1 if not explicitly set. (DB-1950)
- Digest mismatch for same data between nodes with flushed memtables and nodes with non-flushed memtables. (DB-1980)
- Fix handling of start bound in legacy paged queries. (DB-1984)
- Move TWCS message "No compaction necessary for bucket size" to Trace level or NoSpam. (DB-2022)
- Limit max cached direct buffer on NIO to 1 MB. (DB-2028)
- Compaction strategy instantiation errors don't generate meaningful error messages, instead return only InvocationTargetException. (DB-2404)
- Non-portable syntax (MX4J bash-isms) in cassandra-env.sh broke service scripts. (DB-2123)
- nodetool describecluster incorrectly shows DseDelegateSnitch instead of the snitch configured in cassandra.yaml. (DSP-16158)
- nodetool upgradesstables fails with 20-year TTL. After upgrade to 5.1.11, take required action. (DB-2109)
- Add missing equality sign to SASI schema snapshot. (DB-2129)
- For tables using DSE Tiered Storage, nodetool cleanup places cleaned SSTables in the wrong tier. (DB-2173)
• sstableloader options assume the RPC/native (client) interface is the same as the internode (node-to-node) interface. (DB-2184)
• Audit events for CREATE ROLE and ALTER ROLE with incorrect spacing exposes PASSWORD in plain text. (DB-2285)
• Client warnings are not always propagated via LocalSessionWrapper. (DB-2304)
• Timestamps inserted with ISO 8601 format are saved with wrong millisecond value. (DB-2312)
• Compaction fails with IllegalArgumentException: null. (DB-2329)
• BulkLoader class exits without printing the stack trace for throwable error. (DB-2377)
• sstableloader does not decrypt passwords using config encryption in DSE. (DSP-13492)
• Support creating system keys before the output directory is configured in dse.yaml. (DSP-15380)
• Using geo types does not work when memtable allocation type is set to offheap_objects. (DSP-16302)
• Improved compatibility with external tables stored in the DSE Metastore in remote systems. (DSP-16561)
• Heap-size calculation is incorrect for RpcCallStatement + SearchIndexStatement. (DSP-16731)
• Non-internal users are unable to use permissions granted on CREATE. (DSP-16824)
• The -graph option for the cassandra-stress tool failed on generating the target output html in the JAR file. (DSP-17046)

5.1.11 DSE Analytics resolved issues

• A Spark application can be registered twice in rare instances. (DSP-15247)
• Java driver in Spark Connector uses daemon threads to prevent shutdown hooks from being blocked by driver thread pools. (DSP-16051)
• dse client-tool spark sql-schema --all exports definitions for solr_admin keyspace. (DSP-16073)
• DSEFS silently fails when TCP port 5599 is not open between nodes. (DSP-16101)
• cassandra nonsuperuser gets dsefs AccessDeniedException due to Insufficient permissions. (DSP-16713)
• Improved security prevents run_as runner for Spark from running a malicious program. (DSP-16093)

5.1.11 DSEFS resolved issues

• DSEFS fails to start when there is a table with duration type or other type DSEFS that can't understand. (DSP-16825)
• Under high loads, DSEFS reports temporary incorrect state for various files/directories. (DSP-17178)
• IllegalStateException during plugin shutdown causes Failed to abort request body error. (DSP-17003)

5.1.11 DSE Graph resolved issues
• Search indexes are broken for multi cardinality properties. (DSP-14802)
• Changing search index schema using a gremlin script might fail with Search index may not be modified while it is being reindexed. Please wait until reindexing has finished. message. (DSP-15831)
• Align query behavior using geo.inside() predicate for polygon search with and without search indexes. (DSP-16108)
• Classpath conflict between Lucene and SASI versions of Snowball. (DSP-16116)
• Avoid looping indefinitely when a thread making internode requests is interrupted while trying to acquire a connection. (DSP-16544)
• Setting graph.traversal_sources.g.evaluation_timeout breaks graph. (DSP-16709)
• Deleting a search index that was defined inside a graph fails. (DSP-16765)
• DSEFS Hadoop layer doesn't properly translate DSEFS exceptions to Hadoop exceptions in some methods. (DSP-16933)

5.1.11 DSE Search resolved issues

• Avoid accumulating redundant router state updates during schema disagreement. (DSP-15615)
• Servlet container shutdown (Tomcat) prematurely stops logback context. (DSP-15807)
• DSE should not start without appropriate Tomcat JAR scanning exclusions. (DSP-16841)
• Node health score of 1 is not obtainable. Search node gets stuck at 0.00 node health score after replacing a node in a cluster. (DSP-17107)

Hadoop libraries

Built-in Hadoop and Bring-Your-Own-Hadoop (BYOH) were deprecated in DataStax Enterprise (DSE) 5.0, and were removed in DSE 5.1. Hadoop removal from DSE 5.1 and later means that DSE does not allow for the startup of Hadoop services previously included in DSE, including MapReduce JobTracker and TaskTracker.

However, DSE has supported built-in Spark since DSE 4.5 and Bring-Your-Own-Spark (BYOS) since DSE 5.0, and that support continues today. Because Spark depends on certain Hadoop libraries on the server and the client, DSE continues to ship with Hadoop libraries that are required for running Spark and BYOS.

To view the included Hadoop libraries, see DataStax Enterprise 5.1.x third-party software.

dse.yaml
The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
</tbody>
</table>
Cassandra enhancements for DSE 5.1.11

DataStax Enterprise (DSE) 5.1.11 includes all changes from earlier DSE releases. These production-certified changes are enhancements to Apache Cassandra™ 3.11.0. (For Cassandra updates, see CHANGES.txt.)

- 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)
- Validate supported column type with SASI analyzer (CASSANDRA-13669)
- Remove BTree.Builder Recycler to reduce memory usage (CASSANDRA-13929)
- Reduce nodetool GC thread count (CASSANDRA-14475)
- Fix New SASI view creation during Index Redistribution (CASSANDRA-14055)
- Remove string formatting lines from BufferPool hot path (CASSANDRA-14416)
- Update metrics to 3.1.5 (CASSANDRA-12924)
- Detect OpenJDK jvm type and architecture (CASSANDRA-12793)
- Don’t use guava collections in the non-system keyspace jmx attributes (CASSANDRA-12271)
- Fix corrupted static collection deletions in 3.0 -> 2.{1,2} messages (CASSANDRA-14568)
- Fix potential IndexOutOfBoundsException with counters (CASSANDRA-14167)
- Always close RT markers returned by ReadCommand#executeLocally() (CASSANDRA-14515)
- Reverse order queries with range tombstones can cause data loss (CASSANDRA-14513)
- Fix regression of lagging commitlog flush log message (CASSANDRA-14451)
- Add Missing dependencies in pom-all (CASSANDRA-14422)
- Cleanup StartupClusterConnectivityChecker and PING Verb (CASSANDRA-14447)
- Fix deprecated repair error notifications from 3.x clusters to legacy JMX clients (CASSANDRA-13121)
- Cassandra not starting when using enhanced startup scripts in windows (CASSANDRA-14418)
- Fix progress stats and units in compactionstats (CASSANDRA-12244)
- Better handle missing partition columns in system_schema.columns (CASSANDRA-14379)
- Delay hints store excise by write timeout to avoid race with decommission (CASSANDRA-13740)
- Incorrect counting of pending messages in OutboundTcpConnection (CASSANDRA-11551)
• Fix compaction failure caused by reading un-flushed data (CASSANDRA-12743)

General upgrade advice for DSE 5.1.11

Carefully review all planning and upgrade documentation in the Upgrading DataStax Enterprise guide. The following provides general upgrade information:

PLEASE READ: MAXIMUM TTL EXPIRATION DATE NOTICE (CASSANDRA-14092)
-------------------------------------------------------------
(General upgrading instructions are available in the next section)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. By default, INSERTS with TTL exceeding the maximum supported date are rejected, but it's possible to choose a different expiration overflow policy. See CASSANDRA-14092.txt for more details.

Prior to 5.0.12 (5.0.X) and 5.1.7 (5.1.x) there was no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. Clusters in the 4.X and lower series are not subject to this when assertions are enabled. Backed up SSTables can be potentially recovered and recovery instructions can be found on the CASSANDRA-14092.txt file.

If you use or plan to use very large TTLS (10 to 20 years), read CASSANDRA-14092.txt for more information.

PLEASE READ: CVE-2017-5929 LOGBACK BEFORE 1.2.0 SERIALIZATION VULNERABILITY
-------------------------------------------------------------
QOS.ch Logback before 1.2.0 has a serialization vulnerability affecting the SocketServer and ServerSocketReceiver components.

Logback has not been upgraded to avoid breaking deployments and customizations based on older versions. If you are using vulnerable components you will need to upgrade to a newer version of Logback or stop using the vulnerable components.

GENERAL UPGRADING ADVICE FOR ANY VERSION
=================================================
Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to ALWAYS snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version \( X + 1 \) will always be able to read data files created by version \( X \), but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

3.11.4
======

Upgrading
---------

- The order of static columns in SELECT * has been fixed to match that of 2.0 and 2.1 - they are now sorted alphabetically again, by their name, just like regular columns are. If you use prepared statements and SELECT * queries, and have both simple and collection static columns in those tables, and are upgrading from an earlier 3.0 version, then you might be affected by this change. Please see CASSANDRA-14638 for details.

DSE 5.1.7
==========

Upgrading
---------

- Automatic fallback of GossipingPropertyFileSnitch to PropertyFileSnitch (cassandra-topology.properties) is disabled by default and can be enabled via the -Dcassandra.gpfs.enable_pfs_compatibility_mode=true startup flag.

DSE 5.1.6
==========

3.11.3
=====

Upgrading
---------

- Materialized view users upgrading from 3.0.15 (3.0.X series) or 3.11.1 (3.11.X series) and later that have performed range movements (join, decommission, move, etc), should run repair
on the base tables, and subsequently on the views to ensure data
affected by CASSANDRA-14251
   is correctly propagated to all replicas.
- Changes to bloom_filter_fp_chance will no longer take effect on
existing sstables when the
   node is restarted. Only compactions/upgradessstables regenerates
bloom filters and Summaries
   sstable components. See CASSANDRA-11163

Deprecation
----------
- 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.

3.11.2
-----

Upgrading
---------
- See MAXIMUM TTL EXPIRATION DATE NOTICE above.
- seed_gossip_probability setting was added to cassandra.yaml. This
   setting will pick the percentage of times
   gossip messages are sent to a seed. This improves the time it
takes for gossip changes to
   propagate across the cluster. Defaults to 100% (1.0)
- Upgrades from DSE 5.0 might have produced unnecessary schema
  migrations while
  there was at least one DSE 5.0 node in the cluster. It is therefore
highly
  recommended to upgrade from DSE 5.0 to at least DSE 5.1.6. The root
cause of
  this schema mismatch was a difference in the way how schema digests
were computed
  in DSE 5.0 and DSE 5.1. To mitigate this issue, DSE 5.1.6 and newer
announce
  DSE 5.0 compatible digests as long as there is at least one DSE 5.0
  node in the
  cluster. Once all nodes have been upgraded, the "real" schema
  version will be
  announced. Note: this fix is only necessary in DSE 5.1 and
therefore only applies
  to DSE 5.1. (DB-1477)
- DSE 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
- DSE is logging by default a heap histogram on OutOfMemoryError. To
disable that behavior
set the 'cassandra.printHeapHistogramOnOutOfMemoryError' System property to 'false'.
- Improved gossip settling added. On startup DSE waits till all nodes are seen before fully joining the cluster.
  This improves latency spikes when restarting nodes.
- LeveledCompactionStrategy SSTables will keep their existing level on nodetool refresh, nodetool move, and nodetool decommission.

Operations
-----------
- It is now possible to ALTER system_distributed tables
- New command 'nodetool abortrebuild' allows to abort a currently running rebuild operation.
  The command must be executed on the node where the rebuild operation is running. Streams may continue until they finish or timeout.
- Only MODIFY permission on base is required to update table with MV, internally it reads base data and generates updates to MV.

Metrics
-------
- New storage metrics were added:
  * TotalHintsReplayed: how many hints were successfully replayed on the _target_ node.
  * HintsOnDisk: how many hints are currently persistent on disk on this node. Metric is updated for the amount of hints contained in the hints file when hints file is written or removed.
    Values is restored on node startup.

New features
------------
- Statistics file component was added to Hint Store in order to provide information about amount of hints contained in the hints file without replaying it. Stats component is completely backwad-compatible; hint files without this component will not be counted.
  All new hint files will be created with this component. See DB-853 for more details.

**Spark Cassandra Connector changes for DSE 5.1.11**

A list of DataStax Enterprise 5.1.11 production-certified changes for the DataStax Spark Cassandra Connector.

**DSE 5.1.11:**
2.0.9
  * All updates to 1.6.12

2.0.8
* Allow non-cluster prefixed options in sqlConf (SPARKC-531)
* Change Str Literal Match to Be Greedy (SPARKC-532)
* Restore support for various timezone formats to TimestampParser (SPARKC-533)
* UDT converters optimization (SPARKC-536)

DSE 5.1.7:
2.0.7
* Adds Timestamp, Improve Conversion Perf (SPARKC-522)
* Allow setting spark.cassandra.concurrent.reads (SPARKC-520)
* Allow splitCount to be set for Dataframes (SPARKC-527)

DSE 5.1.6:
2.0.6
* Includes all patches up to 1.6.10

DSE 5.1.3:
2.0.5
* Allow IN predicates for composite partition keys and clustering keys to be pushed down to Cassandra (SPARKC-490)
* Allow 'YYYY' format LocalDate
* Add metrics for write batch Size (SPARKC-501)
* Type Converters for java.time.LocalDate (SPARKC-495)

2.0.4
* Includes patches up to 1.6.9
* Retry PoolBusy Exceptions, Throttle JWCT Calls (SPARKC-503)

DSE 5.1.2:
2.0.3
* Includes patches up to 1.6.8

DSE 5.1.1:
2.0.2
* Protect against Size Estimate Overflows (SPARKC-492)
* Add java.time classes support to converters and sparkSQL (SPARKC-491)
* Allow Writes to Static Columns and Partition Keys (SPARKC-470)

DSE 5.1.0:
2.0.1
* Refactor Custom Scan Method (SPARKC-481)

2.0.0
* Upgrade driver version for 2.0.0 Release to 3.1.4 (SPARKC-474)
* Extend SPARKC-383 to All Row Readers (SPARKC-473)

2.0.0 RC1
* Includes all patches up to 1.6.5
* Automatic adjustment of Max Connections (SPARKC-471)
* Allow for Custom Table Scan Method (SPARKC-459)
* Enable PerPartitionLimit (SPARKC-446)
* Support client certificate authentication for two-way SSL Encryption (SPARKC-359)
* Change Config Generation for Cassandra Runners (SPARKC-424)
* Remove deprecated QueryRetryDelay parameter (SPARKC-423)
* User ConnectionHostParam.default as default hosts String
* Update usages of deprecated SQLContext so that SparkSession is used instead (SPARKC-400)
* Test Reused Exchange SPARK-17673 (SPARKC-429)
* Module refactoring (SPARKC-398)
* Recognition of Java Driver Annotated Classes (SPARKC-427)
* RDD.deleteFromCassandra (SPARKC-349)
* Coalesce Pushdown to Cassandra (SPARKC-161)
* Custom Conf options in Custom Pushdowns (SPARKC-435)
* Upgrade CommonBeatUtils to 1.9.3 to Avoid SID-760 (SPARKC-457)

2.0.0 M3
* Includes all patches up to 1.6.2

2.0.0 M2
* Includes all patches up to 1.6.1

2.0.0 M1
* Added support for left outer joins with C* table (SPARKC-181)
* Removed CassandraSqlContext and underscore based options (SPARKC-399)
* Upgrade to Spark 2.0.0-preview (SPARKC-396)
  - Removed Twitter demo because there is no spark-streaming-twitter package available anymore
  - Removed Akka Actor demo because there is no support for such streams anymore
  - Bring back Kafka project and make it compile
  - Update several classes to use our Logging instead of Spark Logging because Spark Logging became private
  - Update few components and tests to make them work with Spark 2.0.0
  - Fix Spark SQL - temporarily
  - Update plugins and Scala version

TinkerPop changes for DSE 5.1.11

DataStax Enterprise (DSE) 5.1.11 includes all changes from previous releases. These production-certified changes are enhancements to TinkerPop 3.2.8:

DSE 5.1.10

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

5 June 2018

- 5.1.10 Components (page 32)
- 5.1.10 Highlights (page 32)
- 5.1.10 Changes and enhancements (page 33)
- 5.1.10 Resolved issues (page 34)
- 5.1.10 Known issues (page 35)
- 5.1.10 Cassandra enhancements (page 35)
• 5.1.10 General upgrade advice (page 36)
• 5.1.10 TinkerPop changes (page 39)

5.1.10 Components

• Apache Cassandra™ 3.11.0.2323 (updated)
• Apache Solr™ 6.0.1.0.2284 (updated)
• Apache Spark™ 2.0.2.19 (updated)
• Apache Tomcat® 8.0.47
• DataStax Spark Cassandra Connector 2.0.7
• DSE Java Driver 1.2.6 (updated)
• Netty 4.0.42.Final
• Spark Jobserver 0.6.2.238 requires compatible API (updated)
• TinkerPop 3.2.9-20180507-f6ead8b2 (updated)
• Select Hadoop libraries

5.1.10 Highlights

Executive summary highlights for DSE 5.1.10:
• DSE Analytics and DSEFS (page 32)
• DSE Graph (page 32)
• DSE Search (page 32)

The executive summary highlights are just a top-level view. Be sure to review all release notes.

5.1.10 DSE Analytics and DSEFS highlights

• Resolved an issue with reading corrupted data from DSEFS caused by incorrect handling of file offsets, if requested offset does not align exactly at the file block boundary. This critical issue was triggered by some Spark usages. (DSP-15907)
• Rare problems with multiple Spark Masters are resolved. Improved Spark Master and Spark Worker stability. (DSP-15636, DSP-15906, DSP-14405, DSP-15801)
• Resolved the missing /tmp directory in DSEFS after fresh cluster installation. (DSP-16058)
• Parquet files with partitions is improved. (DSP-16067)

5.1.10 DSE Graph highlights

• Improved performance with DSE Graph fluent API. (DSP-15686)
• Support for non-text IDs when using graph frames for bulk loading data. (DSP-15614)

5.1.10 DSE Search highlights

• Search index TTL Expiration thread loops without effect with live indexing (RT indexing). (DSP-16038)
• Solr 6.0.1 security upgrades. (DSP-15978)
5.1.10 Changes and enhancements

In addition to the (page 32), review all changes and enhancements:

- DataStax Enterprise (page 33)
- DSE Analytics (page 33)
- DSE Graph (page 33)
- DSE Search (page 33)

5.1.10 DataStax Enterprise core changes and enhancements

- CVE-2016-1000031: Security vulnerability in Apache Commons FileUpload. (DSP-15908)
- Configuration parameters for LDAP tuning (page 364) allow all connection pool options to be set. (DSP-15948)
- Solr security upgrades bundle. (DSP-15978)
  # Apache Directory API All: CVE-2015-3250
  # Apache Hadoop Common: CVE-2016-5393, CVE-2016-3086, CVE-2017-15713
  # Apache Tika parsers: CVE-2018-1339
  # Bouncy Castle Provider: CVE-2018-5382
  # Guava: Google Core Libraries for Java: CVE-2018-10237
  # Simple XML: CVE-2017-1000190
  # Xerces2-j: CVE-2013-4002
  # uimaj-core: CVE-2017-15691

5.1.10 DSE Analytics changes and enhancements

- Decreased the number of exceptions logged during master move from node to node. (DSP-14405)
- Spark Master REST API is disabled. If enabled in spark-defaults.conf, the following error is logged: ERROR Spark Master REST API is not available in DSE. (DSP-15491)
- DSEFS fetching a file from an offset returns empty content. (DSP-15907)
- In Portfolio demo, pricer is no longer required to be run with sudo. (DSP-15970)

5.1.10 DSE Graph changes and enhancements

- DseGraphFrame performance improvement reduces number of joins for count() and other id-only queries. (DSP-15554)
- Performance improvements for traversal execution with Fluent API and script-based executions. (DSP-15686)

5.1.10 DSE Search changes and enhancements
• Solr 6.0.1 security upgrades. (DSP-15978)
• Output Solr foreign filter cache warning only on classes other than DSE classes. (DSP-15625)

5.1.10 Resolved issues

Resolved issues for:

• DataStax Enterprise core (page 34)
• DSE Advanced Replication (page 34)
• DSE Analytics (page 34)
• DSE Graph (page 35)
• DSE Search (page 35)

5.1.10 DataStax Enterprise resolved issues

• CVE-2017-7525: FasterXML Jackson-databind is prone to a remote-code execution vulnerability. (DSP-14784)
• Fix legacy complex range tombstone serialization+deserialization for static and regular columns. (DSP-15878)
• Fix error in MVs referencing a function with uppercase letters on its name. (DSP-15878)
• Ignore empty Counter cells on digest calculation (DSP-16096)
• Upgrade netty to 4.0.54. Ignore log spam for unclean client shutdown. (DSP-16096)
• Avoid log spam for unclean client shutdown. (DSP-16096)
• Reusing table ID with CREATE TABLE causes failure on restart. (DSP-16096)
• Add getConcurrentCompactors to JMX to avoid loading DatabaseDescriptor to check its value in nodetool. (DSP-16096)
• Fix binding JMX to any address. (DSP-16192)

5.1.10 DSE Advanced Replication resolved issues

• dse client-tool help doesn't work if ~/.dserc file exists. (DSP-15869)

5.1.10 DSE Analytics resolved issues

• Running Spark processes as separate users (page 424) does not work. (DSP-15723)
• Multiple Spark masters can be started on the same machine. (DSP-15636)
• DSE client tool returns wrong Spark Master address. (DSP-15801)
• Unnecessary Spark Worker restarts. (DSP-15906)
• Portfolio demo does not work on package installs. (DSP-15970)
• During misconfigured cluster bootstrap, the AlwaysOn SqlServer does not start due to missing /tmp/hive directory in DSEFS. (DSP-16058)
• CassandraHiveMetastore is prevented from adding multiple partitions for File based datasources. Fixes MSCK REPAIR TABLE command. (DSP-16067)
5.1.10 DSE Graph resolved issues

- GraphSON parsing error prevents proper type detection under certain conditions. (DSP-14066)
- When using graph frames, cannot upload edges when ids for vertices are complex non-text ids. (DSP-15614)
- DseGraphFrame fails with StackOverflowError if property is meta-property. (DSP-15939)

5.1.10 DSE Search resolved issues

- A shard request timeout caused an assertion error from Lucene getNumericDocValues in the log. (DSP-14216)
- Offline sstable tools fail if DSE Search index is present on a table. (DSP-15628)
- HTTP read on solr_stress doesn't inject random data into placeholders. (DSP-15727)
- ERROR 500 on distributed http json.facet with non-zero offset. (DSP-15946)
- Search index TTL Expiration thread loops without effect with live indexing (RT indexing). (DSP-16038)

5.1.10 Known issues

DSE Analytics

- The Spark Jobserver demo has an incorrect version for the Spark Jobserver API. (DSP-15832)
  Workaround: In the demo's gradle.properties file, change the version from 0.6.2 to 0.6.2.238.

Cassandra enhancements for DSE 5.1.10

DataStax Enterprise (DSE) 5.1.10 includes all changes from earlier DSE releases. These production-certified changes are enhancements to Apache Cassandra™ 3.11.0. (For Cassandra updates, see CHANGES.txt.)

- Allow existing nodes to use all peers in shadow round (CASSANDRA-13851)
- Fix cqlsh to read connection.ssl cqlshrc option again (CASSANDRA-14299)
- Downgrade log level to trace for CommitLogSegmentManager (CASSANDRA-14370)
- CQL fromJson(null) throws NullPointerException (CASSANDRA-13891)
- Serialize empty buffer as empty string for json output format (CASSANDRA-14245)
- Deprecate background repair and probabilistic read_repair_chance table options (CASSANDRA-13910)
- Add missed CQL keywords to documentation (CASSANDRA-14359)
- Avoid deadlock when running nodetool refresh before node is fully up (CASSANDRA-14310)
- Handle all exceptions when opening sstables (CASSANDRA-14202)
- Handle incompletely written hint descriptors during startup (CASSANDRA-14080)
• Handle repeat open bound from SRP in read repair (CASSANDRA-14330)
• Fix JSON queries with IN restrictions and ORDER BY clause (CASSANDRA-14286)
• Check checksum before decompressing data (CASSANDRA-14284)

**General upgrade advice for DSE 5.1.10**

Carefully review all planning and upgrade documentation in the Upgrading DataStax Enterprise guide. The following provides general upgrade information:

Please read: Maximum TTL expiration date notice (CASSANDRA-14092)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. By default, inserts with TTL exceeding the maximum supported date are rejected, but it's possible to choose a different expiration overflow policy. See CASSANDRA-14092.txt for more details.

Prior to 5.0.12 (5.0.X) and 5.1.7 (5.1.x) there was no protection against inserts with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. Clusters in the 4.X and lower series are not subject to this when assertions are enabled. Backed up SSTables can be potentially recovered and recovery instructions can be found on the CASSANDRA-14092.txt file.

If you use or plan to use very large TTLS (10 to 20 years), read CASSANDRA-14092.txt for more information.

Please read: CVE-2017-5929 Logback before 1.2.0 serialization vulnerability

QOS.ch Logback before 1.2.0 has a serialization vulnerability affecting the SocketServer and ServerSocketReceiver components.

Logback has not been upgraded to avoid breaking deployments and customizations based on older versions. If you are using vulnerable components you will need to upgrade to a newer version of Logback or stop using the vulnerable components.
GENERAL UPGRADING ADVICE FOR ANY VERSION
========================================

Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to ALWAYS snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

DSE 5.1.7
=========

Upgrading
--------
- Automatic fallback of GossipingPropertyFileSnitch to PropertyFileSnitch (cassandra-topology.properties) is disabled by default and can be enabled via the -Dcassandra.gpfs.enable_pfs_compatibility_mode=true startup flag.

DSE 5.1.6
=========

3.11.3
=====  

Upgrading
--------
- Materialized view users upgrading from 3.0.15 (3.0.X series) or 3.11.1 (3.11.X series) and later that have performed range movements (join, decommission, move, etc), should run repair on the base tables, and subsequently on the views to ensure data affected by CASSANDRA-14251 is correctly propagated to all replicas.
- Changes to bloom_filter_fp_chance will no longer take effect on existing sstables when the node is restarted. Only compactions/upgradesstables regenerates bloom filters and Summaries sstable components. See CASSANDRA-11163

Deprecation
------------
- 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.
3.11.2
======

Upgrading
---------

- See MAXIMUM TTL EXPIRATION DATE NOTICE above.
- seed_gossip_probability setting was added to cassandra.yaml. This setting will pick the percentage of times gossip messages are sent to a seed. This improves the time it takes for gossip changes to propagate across the cluster. Defaults to 100% (1.0)
- Upgrades from DSE 5.0 might have produced unnecessary schema migrations while there was at least one DSE 5.0 node in the cluster. It is therefore highly recommended to upgrade from DSE 5.0 to at least DSE 5.1.6. The root cause of this schema mismatch was a difference in the way how schema digests were computed in DSE 5.0 and DSE 5.1. To mitigate this issue, DSE 5.1.6 and newer announce DSE 5.0 compatible digests as long as there is at least one DSE 5.0 node in the cluster. Once all nodes have been upgraded, the "real" schema version will be announced. Note: this fix is only necessary in DSE 5.1 and therefore only applies to DSE 5.1. (DB-1477)
- DSE 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
- DSE is logging by default a heap histogram on OutOfMemoryError. To disable that behavior set the 'cassandra.printHeapHistogramOnOutOfMemoryError' System property to 'false'.
- Improved gossip settling added. On startup DSE waits till all nodes are seen before fully joining the cluster. This improves latency spikes when restarting nodes.
- LeveledCompactionStrategy SSTables will keep their existing level on nodetool refresh, nodetool move, and nodetool decommission.

Operations
----------

- It is now possible to ALTER system_distributed tables
- New command 'nodetool abortrebuild' allows to abort a currently running rebuild operation.
The command must be executed on the node where the rebuild operation is running. Streams may continue until they finish or timeout.
- Only MODIFY permission on base is required to update table with MV, internally it reads base data and generates updates to MV.

Metrics
-------
- New storage metrics were added:
  * TotalHintsReplayed: how many hints were successfully replayed on the _target_ node.
  * HintsOnDisk: how many hints are currently persistent on disk on this node. Metric is updated for the amount of hints contained in the hints file when hints file is written or removed.
  
Values is restored on node startup.

New features
------------
- Statistics file component was added to Hint Store in order to provide information about amount of hints contained in the hints file without replaying it. Stats component is completely backwad-compatible; hint files without this component will not be counted. All new hint files will be created with this component. See DB-853 for more details.

**TinkerPop changes for DSE 5.1.10**

DataStax Enterprise (DSE) 5.1.10 includes all changes from previous releases. These production-certified changes are enhancements to TinkerPop 3.2.9:

- Performance enhancement to Bytecode deserialization. (TINKERPOP-1936)
- Path history isn't preserved for keys in mutations. (TINKERPOP-1947)
- Traversal construction performance enhancements (TINKERPOP-1950)
- Bump to Groovy 2.4.15 - resolves a Groovy bug preventing Lambda creation in GLVs in some cases. (TINKERPOP-1953)

**DSE 5.1.9**

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

Avoid upgrading to **DSE 5.1.9 or DSE 5.1.8** if you use TTL (time-to-live) with DSE Search live indexing (RT indexing). (DSP-16038 (page 40))

24 April 2018

- 5.1.9 Resolved issues (page 40)
5.1.9 Known issues

5.1.9 Components

- Apache Cassandra™ 3.11.0.2261
- Apache Solr™ 6.0.1.0.2224
- Apache Spark™ 2.0.2.17
- Apache Tomcat® 8.0.47
- DataStax Spark Cassandra Connector 2.0.7
- DSE Java Driver 1.2.6 (updated)
- Netty 4.0.42.Final
- Spark Jobserver 0.6.2.237 requires compatible API
- TinkerPop 3.2.8-20180327-292ccbfd
- Select Hadoop libraries

5.1.9 Resolved issue

Fix LDAP library issue. (DSP-15927)

5.1.9 Known issues

- DSE Search: Search index TTL Expiration thread loops without effect with live indexing (RT indexing). (DSP-16038)
- DSE Graph: LIMIT clause does not work in a graph traversal with search predicate TOKEN. (DSP-16292)

DSE 5.1.8

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

Avoid upgrading to **DSE 5.1.9 or DSE 5.1.8** if you use TTL (time-to-live) with DSE Search live indexing (RT indexing). ([DSP-16038](#))

5 April 2018

- 5.1.8 Components ([page 40](#))
- 5.1.8 Highlights ([page 41](#))
- 5.1.8 Changes and enhancements ([page 41](#))
- 5.1.8 Known issues ([page 42](#))
- 5.1.8 Resolved issues ([page 43](#))
- 5.1.8 Cassandra enhancements ([page 46](#))
- 5.1.8 General upgrade advice ([page 54](#))
- 5.1.8 TinkerPop changes ([page 57](#))

5.1.8 Components

- Apache Cassandra™ 3.11.0.2261 (updated)
• Apache Solr™ 6.0.1.0.2224 (updated)
• Apache Spark™ 2.0.2.17 (updated)
• Apache Tomcat® 8.0.47 (updated)
• DataStax Spark Cassandra Connector 2.0.7
• DSE Java Driver 1.2.6 (updated)
• Netty 4.0.42.Final
• Spark Jobserver 0.6.2.237 requires compatible API (updated)
• TinkerPop 3.2.8-20180327-292ccbf (updated)
• Select Hadoop libraries

5.1.8 Highlights

Executive summary highlights for DSE 5.1.8:
• DSE Advanced Replication (page 41)
• DSE Analytics and DSEFS (page 41)
• DSE Search (page 41)

The executive summary highlights are just a top-level view. Be sure to review all 5.1.8 Changes and enhancements (page 41).

5.1.8 DSE Advanced Replication highlights

• Fixed misleading warning messages about a non-replicating cluster in a multi-datacenter source cluster. (DSP-15808)

5.1.8 DSE Analytics and DSEFS highlights

• Fixed a permissions (page 428) issue affecting Spark History Server results visibility through the web UI. (DSP-15693)
• Fixed a permission issue affecting non-superusers and DSEFS. (DSP-15276)

5.1.8 DSE Search highlights

• Fixed reindexing and query performance regression for delete heavy workload. (DSP-15653, DSP-15667)

5.1.8 Changes and enhancements

In addition to the 5.1.8 Highlights (page 41), review all changes and enhancements:
• DataStax Enterprise (page 42)
• DSE Advanced Replication (page 42)
• DSE Analytics (page 42)
• DSEFS (page 42)
• DSE Graph (page 42)
• DSE Search (page 42)
5.1.8 DataStax Enterprise core changes and enhancements

- DSE demos use Jetty Runner 9.4.8. (DSP-14772)
- ANY, SUBMISSION, and WORKPOOL are unreserved keywords and can be used as keyspace, table, and column identifiers. (DSP-15353)
- Improve replace fail messages when a replace is retried before QUARANTINE_DELAY. (DSP-15824)
- Harden txn log files against exceptions when adding records and improve log messages. (DSP-15824)

5.1.8 DSE Advanced Replication changes and enhancements

- To ensure tombstones are removed often by compaction, the default value for gc_grace_seconds is reduced from 86400 (10 days) to 600 (10 minutes) for the dse_advrep.transmissions_crc table. (DSP-15749)

5.1.8 DSE Analytics changes and enhancements

- Improve logging on unsupported operation failure and remove the failed mutation from replog. (DSP-15043)
- Spark Master REST API is disabled. If enabled in spark-defaults.conf, the following error is logged: ERROR Spark Master REST API is not available in DSE. (DSP-15491)

5.1.8 DSEFS changes and enhancements

- Improved security with default file permissions -770 for event log files. Change permissions (page 428) with spark.eventLog.permissions. (DSP-15693)
- DSEFS programmatic access demos are available. (DSP-13799)

5.1.8 DSE Graph changes and enhancements

- Improved performance of anonymous traversals and bytecode-based traversals that made use of withStrategy() configurations. (DSP-15673)

5.1.8 DSE Search changes and enhancements

- Reduce the overhead of DeleteByQueryWrapper used by Solr deleteByQuery(). (DSP-15667)
- Streamline misleading Solr filter cache eviction logging. (DSP-15741)
- Support for specifying different Solr field types (page 559) for each CQL map key. (DSP-15622)

5.1.8 Known issues

- DSE Analytics: Additional configuration is required when enabling context-per-jvm in the Spark Jobserver (page 459). (DSP-15163)
- DSE Analytics: Spark Master does not launch successfully after upgrade from DSE 5.1.x to DSE 5.1.8. (DSP-15679)
To resolve the issue:

```
$dsetool sparkmaster cleanup
$dsetool sparkworker restart
```

- DSE Search: Search index TTL Expiration thread loops without effect with live indexing (RT indexing). (DSP-16038)

5.1.8 Resolved issues

Resolved issues for:

- DataStax Enterprise core (page 43)
- DSE Advanced Replication (page 43)
- DSE Analytics (page 44)
- DSEFS (page 44)
- DSE Graph (page 44)
- DSE Search (page 45)

5.1.8 DataStax Enterprise resolved issues

- Enabling and disabling dbsummary and clustersummary performance objects through dsetool does not work. (DSP-15539)
- Delay closing connection when nodes are removed to allow inflight commands to complete. (DSP-15824)
- JVM startup check not working. (DSP-15824)
- Materialized view schema file for snapshots is created as tables. (DSP-15486)
- Init timestamp with Long.MIN_VALUE instead of -1. (DSP-15486)
- AssertionError in ThrottledUnfilteredIterator due to empty UnfilteredRowIterator. (DSP-15486)
- Make sstableloader use cassandra.config.loader instead of hard-coded YamlConfigurationLoader. (DSP-15486)
- Backport CASSANDRA-9241, fix nodetool toppartitions. (DSP-15486)
- Ignore lost+found directory on startup checks. (DSP-15486)
- Protect against BigDecimals with large scale. (DSP-15486)

5.1.8 DSE Advanced Replication resolved issues

- Replog count never goes down to zero in a multi-node source cluster. (DSP-15060)
- Plugin error during shutdown: Error while fetching mutations. (DSP-15342)
- Add support again for empty quoted name (""") as selectable to select SuperColumns. (DSP-15486)
- Read connection ssl option from cqlshrc. (DSP-15486)
- SASI AND/OR semantics are incorrect for StandardAnalyzer. (DSP-15486)
- NPE Error whilst purging staled mutation files. (DSP-15502)
- Channel creation fails with NPE when using mixed case destination name. (DSP-15538)
- Unable to recover metadata from block file error due to NoSuchFileException. (DSP-15627)
- Errors during shutdown. (DSP-15637)
- advrep replog count command does not work with mixed case keyspace or table names. (DSP-15641)
- AdvRep CommitLogConsumer logging NoSuchFileException. (DSP-15753)
- Incorrect status that CDC was active when only a single advrep channel was defined in the datacenter. (DSP-15808)

5.1.8 DSE Analytics resolved issues

- JSch is susceptible to a path traversal vulnerability. (DSP-13961)
- Worker UI does not display the actual class name of driver application running in cluster mode. (DSP-15028)
- DSEFS transactions not always replayed at startup. (DSP-15462)
- Running Spark processes as separate users (page 424) does not work. (DSP-15723)

5.1.8 DSEFS resolved issues

- InvalidTypeException is thrown while running DSEFS commands on node upgraded from 5.0.x to 5.1.x. (DSP-15266)
- Timeout when trying to umount a dsefs location. (DSP-15453)
- Exception is thrown by DseFsPlugin during shutdown and is not reported. (DSP-15474)
- DSE might not shutdown properly when DSEFS encounters a problem, and exceptions are not logged. (DSP-15482)
- DSEFS programmatic access demo project is available. (DSP-13799)
- SPARK/DSEFS non-super users are unable to run SQL queries in secured DSEFS. Spark SQL applications utilize a scratch directory in DSEFS. This scratch directory is automatically created in DSE 5.1.7 and later. (DSP-15276)
- Insufficient permissions to path / error when putting a file with the dse hadoop -put command on secured DSEFS cluster. (DSP-15480)
- Small probability of duplicated predefined directories (/tmp/hive) when bootstrapping cluster with multiple datacenters and incorrect NetworkTopologyStrategy (SimpleStrategy). (DSP-15639)

5.1.8 DSE Graph resolved issues

- 0 (zero) is not treated as unlimited abort of max num errors. (DGL-307)
- Synchronization hurts graph OLAP on multi-core executors. Improve scalability of OLAP queries with remote traverses. (DSP-15068)
- Failures reported from CassandraPersistenceEngine during upgrade, especially in Graph Analytics workloads. (DSP-15130)
- DseGRaphFrame timestamp base query do not work for bot java.sql.Timestamp and String representations. (DSP-15146)
- graph solr phrase() predicate shows IndexOutOfBound error. (DSP-15408)
# Single-character tokens used in search index queries, for example with predicate
token("a") are erroneously dropped.

# Search index queries using phrase(...) predicates fail exceptionally when
processing values that end in a prefix of the search phrase.

- DseGraphFrames throws InvalidQueryException when search index is enabled.
  (DSP-15411)
- g.V().hasId([]) and g.V().has(id, []) query results are incorrect in DseGraphFrames.
  (DSP-15501)
- toJSON() does not always work with geo types. (DSP-15650)
- ObjectMapper contention for fluent API requests. (DSP-15732)

## 5.1.8 DSE Search resolved issues

- NPE during loading data with RT geonames. (DSP-12361)
- Solr resource reading failure on init after copying data from another cluster. (DSP-15419)
- Prohibit Solr timeAllowed use with partial results and allow it with deep paging.
  (DSP-15475)
- deleteById and deleteByQuery overflow prepared statement cache. (DSP-15620)
- ERROR 500 on distributed http json.facet with non-zero offset. (DSP-15633)
- Reindex with tombstones in the data performs slower than earlier DSE versions.
  (DSP-15653)

### Hadoop libraries

Built-in Hadoop and Bring-Your-Own-Hadoop (BYOH) were deprecated in DataStax
Enterprise (DSE) 5.0, and were removed in DSE 5.1. Hadoop removal from DSE 5.1 and
later means that DSE does not allow for the startup of Hadoop services previously included in
DSE, including MapReduce JobTracker and TaskTracker.

However, DSE has supported built-in Spark since DSE 4.5 and Bring-Your-Own-Spark
(BYOS) since DSE 5.0, and that support continues today. Because Spark depends on certain
Hadoop libraries on the server and the client, DSE continues to ship with Hadoop libraries
that are required for running Spark and BYOS.

To view the included Hadoop libraries, see DataStax Enterprise 5.1.x third-party software.

dse.yaml
The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>Installer-Services installations</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/dse/dse.yaml</td>
<td>/etc/dse/dse.yaml</td>
</tr>
</tbody>
</table>
Cassandra enhancements for DSE 5.1.8

DataStax Enterprise (DSE) 5.1.8 includes all changes from earlier DSE releases. These production-certified changes are enhancements to Apache Cassandra™ 3.11.0. (For Cassandra updates, see CHANGES.txt.)

### 3.11.3
- SASI tokenizer for simple delimiter based entries (CASSANDRA-14247)
- Fix Loss of digits when doing CAST from varint/bigint to decimal (CASSANDRA-14170)
- RateBasedBackPressure unnecessarily invokes a lock on the Guava RateLimiter (CASSANDRA-14163)
- Fix wildcard GROUP BY queries (CASSANDRA-14209)

#### Merged from 3.0:
- Use zero as default score in DynamicEndpointSnitch (CASSANDRA-14252)
- Respect max hint window when hinting for LWT (CASSANDRA-14215)
- Adding missing WriteType enum values to v3, v4, and v5 spec (CASSANDRA-13697)
- Don't regenerate bloomfilter and summaries on startup (CASSANDRA-11163)
- Fix NPE when performing comparison against a null frozen in LWT (CASSANDRA-14087)
- Log when SSTables are deleted (CASSANDRA-14302)
- Fix batch commitlog sync regression (CASSANDRA-14292)
- Write to pending endpoint when view replica is also base replica (CASSANDRA-14251)
- Chain commit log marker potential performance regression in batch commit mode (CASSANDRA-14194)
- Fully utilise specified compaction threads (CASSANDRA-14210)
- Pre-create deletion log records to finish compactions quicker (CASSANDRA-12763)

#### Merged from 2.2:
- Backport circleci yaml (CASSANDRA-14240)

#### Merged from 2.1:
- CVE-2017-5929 Security vulnerability in Logback warning in NEWS.txt (CASSANDRA-14183)

### 3.11.2
- Fix ReadCommandTest (CASSANDRA-14234)
- Remove trailing period from latency reports at keyspace level (CASSANDRA-14233)
- 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)
* Acquire read lock before accessing CompactionStrategyManager fields (CASSANDRA-14139)
* Split CommitLogStressTest to avoid timeout (CASSANDRA-14143)
* Avoid invalidating disk boundaries unnecessarily (CASSANDRA-14083)
* Avoid exposing compaction strategy index externally (CASSANDRA-14082)
* Fix imbalanced disks when replacing node with same address with JBOD (CASSANDRA-14084)
* Reload compaction strategies when disk boundaries are invalidated (CASSANDRA-13948)
* Remove OpenJDK log warning (CASSANDRA-13916)
* Prevent compaction strategies from looping indefinitely (CASSANDRA-14079)
* Cache disk boundaries (CASSANDRA-13215)
* Add asm jar to build.xml for maven builds (CASSANDRA-11193)
* Round buffer size to powers of 2 for the chunk cache (CASSANDRA-13897)
* Update jackson JSON jars (CASSANDRA-13949)
* Avoid locks when checking LCS fanout and if we should defrag (CASSANDRA-13930)
* Correctly count range tombstones in traces and tombstone thresholds (CASSANDRA-8527)

Merged from 3.0:
* Add MinGW uname check to start scripts (CASSANDRA-12840)
* Use the correct digest file and reload sstable metadata in nodetool verify (CASSANDRA-14217)
* Handle failure when mutating repaired status in Verifier (CASSANDRA-13933)
* Set encoding for javadoc generation (CASSANDRA-14154)
* Fix index target computation for dense composite tables with dropped compact storage (CASSANDRA-14104)
* Improve commit log chain marker updating (CASSANDRA-14108)
* Extra range tombstone bound creates double rows (CASSANDRA-14008)
* Fix SSTable ordering by max timestamp in SinglePartitionReadCommand (CASSANDRA-14010)
* Accept role names containing forward-slash (CASSANDRA-14088)
* Optimize CRC check chance probability calculations (CASSANDRA-14094)
* Fix cleanup on keyspace with no replicas (CASSANDRA-13526)
* 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)
* Fix serialized size of DataLimits (CASSANDRA-14057)
* Add flag to allow dropping oversized read repair mutations (CASSANDRA-13975)
* Fix SSTableLoader logger message (CASSANDRA-14003)
* Fix repair race that caused gossip to block (CASSANDRA-13849)
* Tracing interferes with digest requests when using RandomPartitioner (CASSANDRA-13964)
* Don't let user drop or generally break tables in system_distributed (CASSANDRA-13813)
* Provide a JMX call to sync schema with local storage (CASSANDRA-13954)
* Mishandling of cells for removed/dropped columns when reading legacy files (CASSANDRA-13939)
* Deserialise sstable metadata in nodetool verify (CASSANDRA-13922)
Merged from 2.2:
* Fix the inspectJvmOptions startup check (CASSANDRA-14112)
* Fix race that prevents submitting compaction for a table when executor is full (CASSANDRA-13801)
* Rely on the JVM to handle OutOfMemoryErrors (CASSANDRA-13006)
* Grab refs during scrub/index redistribution/cleanup (CASSANDRA-13873)

Merged from 2.1:
* Protect against overflow of local expiration time (CASSANDRA-14092)
* RPM package spec: fix permissions for installed jars and config files (CASSANDRA-14181)
* More PEP8 compliance for cqlsh (CASSANDRA-14021)

3.11.1
* Fix the computation of cdc_total_space_in_mb for exabyte filesystems (CASSANDRA-13808)
* AbstractTokenTreeBuilder#serializedSize returns wrong value when there is a single leaf and overflow collisions (CASSANDRA-13869)
* BTree.Builder memory leak (CASSANDRA-13754)
* Revert CASSANDRA-10368 of supporting non-pk column filtering due to correctness (CASSANDRA-13798)
* Add a skip read validation flag to cassandra-stress (CASSANDRA-13772)
* Fix cassandra-stress hang issues when an error during cluster connection happens (CASSANDRA-12938)
* Better bootstrap failure message when blocked by (potential) range movement (CASSANDRA-13744)
* "ignore" option is ignored in sstableloader (CASSANDRA-13721)
* Deadlock in AbstractCommitLogSegmentManager (CASSANDRA-13652)
* Duplicate the buffer before passing it to analyser in SASI operation (CASSANDRA-13512)
* Properly evict pstmts from prepared statements cache (CASSANDRA-13641)

Merged from 3.0:
* Improve TRUNCATE performance (CASSANDRA-13909)
* Implement short read protection on partition boundaries (CASSANDRA-13595)
* Fix ISE thrown by UPI.Serializer.hasNext() for some SELECT queries (CASSANDRA-13911)
* Filter header only commit logs before recovery (CASSANDRA-13918)
* AssertionWarning prepending to a list (CASSANDRA-13149)
* Fix support for SuperColumn tables (CASSANDRA-12373)
* Handle limit correctly on tables with strict liveness (CASSANDRA-13883)
* Fix missing original update in TriggerExecutor (CASSANDRA-13894)
* Remove non-rpc-ready nodes from counter leader candidates (CASSANDRA-13043)
* Improve short read protection performance (CASSANDRA-13794)
* Fix sstable reader to support range-tombstone-marker for multi-slices (CASSANDRA-13787)
* Fix short read protection for tables with no clustering columns (CASSANDRA-13880)
* Make isBuilt volatile in PartitionUpdate (CASSANDRA-13619)
* Prevent integer overflow of timestamps in CellTest and RowsTest (CASSANDRA-13866)
* Fix counter application order in short read protection (CASSANDRA-12872)
* Don't block RepairJob execution on validation futures (CASSANDRA-13797)
* Wait for all management tasks to complete before shutting down CLSM (CASSANDRA-13123)
* INSERT statement fails when Tuple type is used as clustering column with default DESC order (CASSANDRA-13717)
* Fix pending view mutations handling and cleanup batchlog when there are local and remote paired mutations (CASSANDRA-13069)
* Improve config validation and documentation on overflow and NPE (CASSANDRA-13622)
* Range deletes in a CAS batch are ignored (CASSANDRA-13655)
* Avoid assertion error when IndexSummary > 2G (CASSANDRA-12014)
* Change repair midpoint logging for tiny ranges (CASSANDRA-13603)
* Better handle corrupt final commitlog segment (CASSANDRA-11995)
* StreamingHistogram is not thread safe (CASSANDRA-13756)
* Fix pending view mutations handling and cleanup batchlog when there are local and remote paired mutations (CASSANDRA-13069)
* Improve config validation and documentation on overflow and NPE (CASSANDRA-13622)
* Range deletes in a CAS batch are ignored (CASSANDRA-13655)
* Avoid assertion error when IndexSummary > 2G (CASSANDRA-12014)
* Change repair midpoint logging for tiny ranges (CASSANDRA-13603)
* Better handle corrupt final commitlog segment (CASSANDRA-11995)
* StreamingHistogram is not thread safe (CASSANDRA-13756)
* Fix race condition in read command serialization (CASSANDRA-13363)
* Fix AssertionError in short read protection (CASSANDRA-13747)
* Don't skip corrupted sstables on startup (CASSANDRA-13620)
* Fix the merging of cells with different user type versions (CASSANDRA-13776)
* Copy session properties on cqlsh.py do_login (CASSANDRA-13640)
* Potential AssertionError during ReadRepair of range tombstone and partition deletions (CASSANDRA-13719)
* Don't let stress write warmup data if n=0 (CASSANDRA-13773)
* Gossip thread slows down when using batch commit log (CASSANDRA-12966)
* Randomize batchlog endpoint selection with only 1 or 2 racks (CASSANDRA-12884)
* Fix digest calculation for counter cells (CASSANDRA-13750)
* Fix ColumnDefinition.cellValueType() for non-frozen collection and change SSTabledump to use type.toJSONString() (CASSANDRA-13573)
* Skip materialized view addition if the base table doesn't exist (CASSANDRA-13737)
* Drop table should remove corresponding entries in dropped_columns table (CASSANDRA-13730)
* Log warn message until legacy auth tables have been migrated (CASSANDRA-13371)
* Fix incorrect [2.1 <- 3.0] serialization of counter cells created in 2.0 (CASSANDRA-13691)
* Fix invalid writetime for null cells (CASSANDRA-13711)
* Fix ALTER TABLE statement to atomically propagate changes to the table and its MVs (CASSANDRA-12952)
* Fix Digest mismatch Exception if hints file has UnknownColumnFamily (CASSANDRA-13696)
* Fixed ambiguous output of nodetool tablestats command (CASSANDRA-13722)
* Purge tombstones created by expired cells (CASSANDRA-13643)
* Make concat work with iterators that have different subsets of columns (CASSANDRA-13482)
* Set test.runners based on cores and memory size (CASSANDRA-13078)
* Allow different NUMACTL_ARGS to be passed in (CASSANDRA-13557)
DataStax Enterprise 5.1 release notes

* Allow native function calls in CQLSSTableWriter (CASSANDRA-12606)
* Fix secondary index queries on COMPACT tables (CASSANDRA-13627)
* Nodetool listsnapshots output is missing a newline, if there are no snapshots (CASSANDRA-13568)
* sstabledump reports incorrect usage for argument order (CASSANDRA-13532)

Merged from 2.2:
* Safely handle empty buffers when outputting to JSON (CASSANDRA-13868)
* Copy session properties on cqlsh.py do_login (CASSANDRA-13847)
* Fix load over calculated issue in IndexSummaryRedistribution (CASSANDRA-13738)
* Fix compaction and flush exception not captured (CASSANDRA-13833)
* Uncaught exceptions in Netty pipeline (CASSANDRA-13649)
* Prevent integer overflow on exabyte filesystems (CASSANDRA-13067)
* Fix queries with LIMIT and filtering on clustering columns (CASSANDRA-11223)
* Fix potential NPE when resume bootstrap fails (CASSANDRA-13272)
* Fix toJSONString for the UDT, tuple and collection types (CASSANDRA-13592)
* Fix nested Tuples/UDTs validation (CASSANDRA-13646)

Merged from 2.1:
* Clone HeartBeatState when building gossip messages. Make its generation/version volatile (CASSANDRA-13700)

3.11.0
* Allow native function calls in CQLSSTableWriter (CASSANDRA-12606)
* Replace string comparison with regex/number checks in MessagingService test (CASSANDRA-13216)
* Fix formatting of duration columns in CQLSH (CASSANDRA-13549)
* Fix the problem with duplicated rows when using paging with SASI (CASSANDRA-13302)
* Allow CONTAINS statements filtering on the partition key and it’s parts (CASSANDRA-13275)
* Fall back to even ranges calculation in clusters with vnodes when tokens are distributed unevenly (CASSANDRA-13229)
* Fix duration type validation to prevent overflow (CASSANDRA-13218)
* Forbid unsupported creation of SASI indexes over partition key columns (CASSANDRA-13228)
* Reject multiple values for a key in CQL grammar. (CASSANDRA-13369)
* UDA fails without input rows (CASSANDRA-13399)
* Fix compaction-stress by using daemonInitialization (CASSANDRA-13188)
* V5 protocol flags decoding broken (CASSANDRA-13443)
* Use write lock not read lock for removing sstables from compaction strategies. (CASSANDRA-13422)
* Use corePoolSize equal to maxPoolSize in JMXEnabledThreadPoolExecutors (CASSANDRA-13329)
* Avoid rebuilding SASI indexes containing no values (CASSANDRA-12962)
* Add charset to Analyser input stream (CASSANDRA-13151)
* Fix testLimitSSTables flake caused by concurrent flush (CASSANDRA-12820)
* cdc column addition strikes again (CASSANDRA-13382)
* Fix static column indexes (CASSANDRA-13277)
* DataOutputBuffer.asNewBuffer broken (CASSANDRA-13298)
* unittest CipherFactoryTest failed on MacOS (CASSANDRA-13370)
* Forbid SELECT restrictions and CREATE INDEX over non-frozen UDT columns (CASSANDRA-13247)
* Default logging we ship will incorrectly print "?:?" for "%F:%L" pattern (CASSANDRA-13171)
* Possible AssertionError in UnfilteredRowIteratorWithLowerBound (CASSANDRA-13366)
* Support unaligned memory access for AArch64 (CASSANDRA-13326)
* Improve SASI range iterator efficiency on intersection with an empty range (CASSANDRA-12915).
* Fix equality comparisons of columns using the duration type (CASSANDRA-13174)
* Obfuscate password in stress-graphs (CASSANDRA-12233)
* Move to FastThreadLocalThread and FastThreadLocal (CASSANDRA-13034)
* nodetool stopdaemon errors out (CASSANDRA-13030)
* Tables in system_distributed should not use gcgs of 0 (CASSANDRA-12954)
* Fix primary index calculation for SASI (CASSANDRA-12910)
* More fixes to the TokenAllocator (CASSANDRA-12990)
* NoReplicationTokenAllocator should work with zero replication factor (CASSANDRA-12983)
* Address message coalescing regression (CASSANDRA-12676)
* Delete illegal character from StandardTokenizerImpl.jflex (CASSANDRA-13417)
* Fix cqlsh automatic protocol downgrade regression (CASSANDRA-13307)
* Tracing payload not passed from QueryMessage to tracing session (CASSANDRA-12835)

Merged from 3.0:
* Ensure int overflow doesn't occur when calculating large partition warning size (CASSANDRA-13172)
* Ensure consistent view of partition columns between coordinator and replica in ColumnFilter (CASSANDRA-13004)
* Failed unregistering mbean during drop keyspace (CASSANDRA-13346)
* nodetool scrub/cleanup/upgradesstables exit code is wrong (CASSANDRA-13542)
* Fix the reported number of sstable data files accessed per read (CASSANDRA-13120)
* Fix schema digest mismatch during rolling upgrades from versions before 3.0.12 (CASSANDRA-13559)
* Upgrade JNA version to 4.4.0 (CASSANDRA-13072)
* Interned ColumnIdentifiers should use minimal ByteBuffers (CASSANDRA-13533)
* ReverseIndexedReader may drop rows during 2.1 to 3.0 upgrade (CASSANDRA-13525)
* Fix repair process violating start/end token limits for small ranges (CASSANDRA-13052)
* Add storage port options to sstableloader (CASSANDRA-13518)
* Properly handle quoted index names in cqlsh DESCRIBE output (CASSANDRA-12847)
* Avoid reading static row twice from old format sstables (CASSANDRA-13236)
* Fix NPE in StorageService.excise() (CASSANDRA-13163)
* Expire OutboundTcpConnection messages by a single Thread (CASSANDRA-13265)
* Fail repair if insufficient responses received (CASSANDRA-13397)
* Fix SSTableLoader fail when the loaded table contains dropped columns (CASSANDRA-13276)
* Avoid name clashes in CassandraIndexTest (CASSANDRA-13427)
* Handling partially written hint files (CASSANDRA-12728)
* Interrupt replaying hints on decommission (CASSANDRA-13308)
* Handling partially written hint files (CASSANDRA-12728)
* Fix NPE issue in StorageService (CASSANDRA-13060)
* Make reading of range tombstones more reliable (CASSANDRA-12811)
* Fix startup problems due to schema tables not completely flushed (CASSANDRA-12213)
* Fix view builder bug that can filter out data on restart (CASSANDRA-13405)
* Fix 2i page size calculation when there are no regular columns (CASSANDRA-12400)
* Fix the conversion of 2.X expired rows without regular column data (CASSANDRA-13395)
* Fix hint delivery when using ext+internal IPs with prefer_local enabled (CASSANDRA-13020)
* Fix possible NPE on upgrade to 3.0/3.X in case of IO errors (CASSANDRA-13389)
* Legacy deserializer can create empty range tombstones (CASSANDRA-13341)
* Legacy caching options can prevent 3.0 upgrade (CASSANDRA-13384)
* Use the Kernel32 library to retrieve the PID on Windows and fix startup checks (CASSANDRA-13333)
* Fix code to not exchange schema across major versions (CASSANDRA-13274)
* Dropping column results in "corrupt" SSTable (CASSANDRA-13337)
* Bug handling range tombstones in the sstable iterators (CASSANDRA-13340)
* Fix CONTAINS filtering for null collections (CASSANDRA-13246)
* Applying: Use a unique metric reservoir per test run when using Cassandra-wide metrics residing in MBeans (CASSANDRA-13216)
* Propagate row deletions in 2i tables on upgrade (CASSANDRA-13320)
* Slice.isEmpty() returns false for some empty slices (CASSANDRA-13305)
* Add formatted row output to assertEmpty in CQL Tester (CASSANDRA-13238)
* Prevent data loss on upgrade 2.1 - 3.0 by adding component separator to LogRecord absolute path (CASSANDRA-13294)
* Improve testing on macOS by eliminating sigar logging (CASSANDRA-13233)
* Cqlsh copy-from should error out when csv contains invalid data for collections (CASSANDRA-13071)
* Update c.yaml doc for offheap memtables (CASSANDRA-13179)
* Faster StreamingHistogram (CASSANDRA-13038)
* Legacy deserializer can create unexpected boundary range tombstones (CASSANDRA-13237)
* Remove unnecessary assertion from AntiCompactionTest (CASSANDRA-13070)
* Fix cqlsh COPY for dates before 1900 (CASSANDRA-13185)
* Use keyspace replication settings on system.size_estimates table (CASSANDRA-9639)
* Add vm.max_map_count StartupCheck (CASSANDRA-13008)
* Hint related logging should include the IP address of the destination in addition to host ID (CASSANDRA-13205)
* Reloading logback.xml does not work (CASSANDRA-13173)
* Lightweight transactions temporarily fail after upgrade from 2.1 to 3.0 (CASSANDRA-13109)
* Duplicate rows after upgrading from 2.1.16 to 3.0.10/3.9 (CASSANDRA-13125)
* Fix UPDATE queries with empty IN restrictions (CASSANDRA-13152)
* Fix handling of partition with partition-level deletion plus live rows in sstabledump (CASSANDRA-13177)
* Provide user workaround when system_schema.columns does not contain entries for a table that's in system_schema.tables (CASSANDRA-13180)
* Nodetool upgradesstables/scrub/compact ignores system tables (CASSANDRA-13410)
* Fix schema version calculation for rolling upgrades (CASSANDRA-13441)

Merged from 2.2:
* Nodes started with join_ring=False should be able to serve requests when authentication is enabled (CASSANDRA-11381)
* cqlsh COPY FROM: increment error count only for failures, not for attempts (CASSANDRA-13209)
* Avoid starting gossiper in RemoveTest (CASSANDRA-13407)
* Fix weightedSize() for row-cache reported by JMX and NodeTool (CASSANDRA-13393)
* Fix JVM metric names (CASSANDRA-13103)
* Honor truststore-password parameter in cassandra-stress (CASSANDRA-12773)
* Discard in-flight shadow round responses (CASSANDRA-12653)
* Don't anti-compact repaired data to avoid inconsistencies (CASSANDRA-13153)
* Wrong logger name in AnticompactionTask (CASSANDRA-13343)
* Commitlog replay may fail if last mutation is within 4 bytes of end of segment (CASSANDRA-13282)
* Fix queries updating multiple time the same list (CASSANDRA-13130)
* Fix GRANT/REVOKE when keyspace isn't specified (CASSANDRA-13053)
* Avoid race on receiver by starting streaming sender thread after sending init message (CASSANDRA-12886)
* Fix "multiple versions of ant detected..." when running ant test (CASSANDRA-13232)
* Coalescing strategy sleeps too much (CASSANDRA-13090)
* Fix flaky LongLeveledCompactionStrategyTest (CASSANDRA-12202)
* Fix failing COPY TO STDOUT (CASSANDRA-12497)
* Fix ColumnCounter::countAll behaviour for reverse queries (CASSANDRA-13222)
* Exceptions encountered calling getSeeds() breaks OTC thread (CASSANDRA-13018)
* Fix negative mean latency metric (CASSANDRA-12876)
* Use only one file pointer when creating commitlog segments (CASSANDRA-12539)

Merged from 2.1:
* Fix 2ndary index queries on partition keys for tables with static columns (CASSANDRA-13147)
* Fix ParseError unhashable type list in cqlsh copy from (CASSANDRA-13364)
* Remove unused repositories (CASSANDRA-13278)
* Log stacktrace of uncaught exceptions (CASSANDRA-13108)
* Use portable stderr for java error in startup (CASSANDRA-13211)
* Fix Thread Leak in OutboundTcpConnection (CASSANDRA-13204)
* Coalescing strategy can enter infinite loop (CASSANDRA-13159)

**General upgrade advice for DSE 5.1.8**

Carefully review all planning and upgrade documentation in the Upgrading DataStax Enterprise guide. The following provides general upgrade information:

**PLEASE READ: MAXIMUM TTL EXPIRATION DATE NOTICE (CASSANDRA-14092)**

(General upgrading instructions are available in the next section)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. By default, INSERTS with TTL exceeding the maximum supported date are rejected, but it's possible to choose a different expiration overflow policy. See CASSANDRA-14092.txt for more details.

Prior to 5.0.12 (5.0.X) and 5.1.7 (5.1.x) there was no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. Clusters in the 4.X and lower series are not subject to this when assertions are enabled. Backed up SSTables can be potentially recovered and recovery instructions can be found on the CASSANDRA-14092.txt file.

If you use or plan to use very large TTLS (10 to 20 years), read CASSANDRA-14092.txt for more information.

**PLEASE READ: CVE-2017-5929 LOGBACK BEFORE 1.2.0 SERIALIZATION VULNERABILITY**

QOS.ch Logback before 1.2.0 has a serialization vulnerability affecting the SocketServer and ServerSocketReceiver components.

Logback has not been upgraded to avoid breaking deployments and customizations.
based on older versions. If you are using vulnerable components you will need to upgrade to a newer version of Logback or stop using the vulnerable components.

GENERAL UPGRADING ADVICE FOR ANY VERSION
==============================================

Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to ALWAYS snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

DSE 5.1.7
=========  
Upgrading  
---------  
- Automatic fallback of GossipingPropertyFileSnitch to PropertyFileSnitch (cassandra-topology.properties) is disabled by default and can be enabled via the -Dcassandra.gpfs.enable_pfsCompatibility_mode=true startup flag.

DSE 5.1.6
=========  

3.11.3  
-----  
Upgrading  
---------  
- Materialized view users upgrading from 3.0.15 (3.0.X series) or 3.11.1 (3.11.X series) and later that have performed range movements (join, decommission, move, etc), should run repair on the base tables, and subsequently on the views to ensure data affected by CASSANDRA-14251 is correctly propagated to all replicas.
- Changes to bloom_filter_fp_chance will no longer take effect on existing sstables when the node is restarted. Only compactions/upgrade sstables regenerates bloom filters and Summaries sstable components. See CASSANDRA-11163

3.11.2  
-----
Upgrading
---------
- See MAXIMUM TTL EXPIRATION DATE NOTICE above.
- seed_gossip_probability setting was added to cassandra.yaml. This setting will pick the percentage of times gossip messages are sent to a seed. This improves the time it takes for gossip changes to propagate across the cluster. Defaults to 100% (1.0)
- Upgrades from DSE 5.0 might have produced unnecessary schema migrations while there was at least one DSE 5.0 node in the cluster. It is therefore highly recommended to upgrade from DSE 5.0 to at least DSE 5.1.6. The root cause of this schema mismatch was a difference in the way how schema digests were computed in DSE 5.0 and DSE 5.1. To mitigate this issue, DSE 5.1.6 and newer announce DSE 5.0 compatible digests as long as there is at least one DSE 5.0 node in the cluster. Once all nodes have been upgraded, the "real" schema version will be announced. Note: this fix is only necessary in DSE 5.1 and therefore only applies to DSE 5.1. (DB-1477)
- DSE 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
- DSE is logging by default a heap histogram on OutOfMemoryError. To disable that behavior set the 'cassandra.printHeapHistogramOnOutOfMemoryError' System property to 'false'.
- Improved gossip settling added. On startup DSE waits till all nodes are seen before fully joining the cluster. This improves latency spikes when restarting nodes.
- LeveledCompactionStrategy SSTables will keep their existing level on nodetool refresh, nodetool move, and nodetool decommission.

Operations
----------
- It is now possible to ALTER system_distributed tables
- New command 'nodetool abortrebuild' allows to abort a currently running rebuild operation.
The command must be executed on the node where the rebuild operation is running. Streams may continue until they finish or timeout.
- Only MODIFY permission on base is required to update table with MV, internally it reads base
data and generates updates to MV.

Metrics
-------
- New storage metrics were added:
  - TotalHintsReplayed: how many hints were successfully replayed on
    the _target_ node.
  - HintsOnDisk: how many hints are currently persistent on disk on
    this node. Metric is updated
    for the amount of hints contained in the hints file when hints
    file is written or removed.
    Values is restored on node startup.

New features
-------------
- Statistics file component was added to Hint Store in order to
  provide information about
  amount of hints contained in the hints file without replaying it.
Stats component is
  completely backward-compatible; hint files without this component
  will not be counted.
  All new hint files will be created with this component. See DB-853
  for more details.

**TinkerPop changes for DSE 5.1.8**

DataStax Enterprise (DSE) 5.1.8 includes all changes from previous releases. These
production-certified changes are enhancements to TinkerPop 3.2.8:

- Fixed a bug in NumberHelper that led to wrong min/max results if numbers exceeded
  the Integer limits. (TINKERPOP-1873)
- Improved error messaging for failed serialization and deserialization of request/
  response messages.
- Fixed bug in handling of Direction.BOTH in Messenger implementations to pass
  the message to the opposite side of the `StarGraph` in VertexPrograms for OLAP
  traversals. (TINKERPOP-1862)
- Fixed a bug in Gremlin Console which prevented handling of gremlin.sh flags that had
  an equal sign (=) between the flag and its arguments. (TINKERPOP-1879)
- Fixed a bug where SparkMessenger was not applying the edgeFunction`from
  MessageScope` in VertexPrograms for OLAP-based traversals. (TINKERPOP-1872)
- TinkerPop drivers prior to 3.2.4 won’t authenticate with Kerberos anymore. A long-deprecated
  option on the Gremlin Server protocol was removed.

**DSE 5.1.7**

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

15 February 2018
5.1.7 Components

- Apache Cassandra™ 3.11.0.2130 (updated)
- Apache Solr™ 6.0.1.0.2139 (updated)
- Apache Spark™ 2.0.2.16 (updated)
- Apache Tomcat® 8.0.47 (updated)
- DataStax Spark Cassandra Connector 2.0.7 (updated)
- DSE Java Driver 1.2.2
- Netty 4.0.42.Final
- Spark Jobserver 0.6.2.234 (requires compatible API)
- TinkerPop 3.2.8-20180125-cd910875 (updated)
- Select Hadoop libraries

5.1.7 Highlights

Executive summary highlights for DSE 5.1.7:

- DataStax Enterprise core (page 58)
- DSE Search (page 58)

The executive summary highlights are just a top-level view. Be sure to review all 5.1.7 Changes and enhancements (page 58).

5.1.7 DataStax Enterprise core highlights

- Fix for the possible data loss scenario caused by the TTL expiration timestamps susceptible to the year 2038 problem. (DSP-15412)

  When using a long TTL, DataStax strongly recommends upgrading to DSE 5.1.7 or later and taking required action.

5.1.7 DSE Search highlights

- Better defaults when using JTS for polygon queries. (DSP-15182)
- More responsive shutdown and index unloading while index rebuild is in progress. (DSP-12452)

5.1.7 Changes and enhancements
In addition to the 5.1.7 Highlights (page 58), review all changes and enhancements:

- DataStax Enterprise (page 59)
- DSE Search (page 59)

### 5.1.7 DataStax Enterprise core changes and enhancements

- Custom index and iTrigger implementations are not supported. Use only implementations bundled with DSE.
- Default number of threads used by performance objects is increased from 1 to 4; configure threads with new dse.yaml performance_core_threads (page 329) parameter. (DSP-14643)
- New nodetool getseeds and reloadseeds commands. (DSP-15412)

### 5.1.7 DSE Search changes and enhancements

- Wikipedia demo path error. (DSP-11327)
- DeleteById is deprecated. (DSP-13436)

### 5.1.7 Known issues

- CVE-2017-15095 jackson-databind is vulnerable to remote code execution (RCE) attacks. Applies only to workloads using --framework spark-2.0 spark-submit. (DSP-15441)
- Potential data loss for INSERTs with very large TTLs. TTL expiration timestamps are susceptible to the year 2038 problem. (DSP-15412)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. TTLs are considered "very large" when close to the maximum allowed value of 630720000 seconds (20 years), starting from 2018-01-19T03:14:06+00:00. As time progresses, the maximum supported TTL is gradually reduced as the maximum expiration date approaches. For instance, on 2028-01-19T03:14:06+00:00 with a TTL of 10 years is impacted. The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately.

**Warning:** With DSE 5.1.7 and later, DSE provides troubleshooting strategies to protect against overflow of local expiration time.

- Spark Master might not recover after upgrades from DSE 5.1.0 through 5.1.5 to DSE 5.1.6 or 5.1.7. (DSP-15679)
In some scenarios, the Spark Master might not recover directly after upgrade, and all the Spark applications must be stopped and restarted. Follow these steps to ensure Spark Master launches successfully for upgrades from any DSE 5.1.x to 5.1.8:

```
$ dsetool sparkmaster cleanup
$ dsetool sparkworker restart
```

5.1.7 Resolved issues

Resolved issues for:

- DataStax Enterprise core ([page 60](#))
- DSE Analytics ([page 61](#))
- DSEFS ([page 61](#))
- DSE Graph ([page 61](#))
- DSE Search ([page 61](#))

5.1.7 DataStax Enterprise resolved issues

- `dbsummary` does not work with default performance_core_threads. (DSP-14643)
- CVE-2017-15095 `jackson-databind` is vulnerable to remote code execution (RCE) attacks. (DSP-15096)
- Fix for possible data loss scenario caused by the TTL expiration timestamps susceptible to the [year 2038 problem](#). (DSP-15412)
  
  When using a long TTL, DataStax strongly recommends upgrading to DSE 5.1.7 or later and taking required action.

- Remove invalid path from compaction-stress script, populate data base on initial size. (DSP-15412)
- Fix infinite loop when replaying a truncated commit log file and truncation is tolerated. (DSP-15412)
- Fetch/query no columns in priming connections to avoid errors if system.local columns are changed. (DSP-15484)
- Upgrade from DSE 5.0.11 to DSE 5.1.6 fails with deserialization exception on column "workloads". (DSP-15484)
- Fix connections per host in nodetool getstreamthroughput. (DSP-15412)
- Avoid hibernate on startup for bootstrap node to avoid WTE due to not being marked alive. (DSP-15412)
- Prevent received SSTables with tombstones during repair from being compacted. (DSP-15412)
- Non-disruptive seed node list reload. (DSP-15412)
- Make `ReservedKeywords` mutable. (DSP-15412)
- Fix tpc connection being reset due to dc compression and flush socket before reset. (DSP-15412)
• Skip legacy range tombstones if only their clustering is corrupted. (DSP-15412)
• Fix AssertionError in ReadResponse$Serializer.serializedSize. (DSP-15412)
• Allow ALTER of system_distributed keyspace tables. (DSP-15412)
• Improve live-node-replacement. (DSP-15412)
• Allow skipping commit log replay does not fail on descriptor errors. (DSP-15435)

5.1.7 DSE Analytics resolved issues

• Fix for possible scenario where newly-added nodes can have a schema mismatch for system keyspaces. (DSP-11787)
• Message is not consistently displayed when SparkContext is created with different configuration. (DSP-14758)
• Spark SQL applications with DSE authentication enabled will throw errors if the DSEFS scratch directory doesn't exist. (DSP-15276)

5.1.7 DSEFS resolved issues

• DSEFS does not use ssl_native_port for internal connections between DSEFS node and Cassandra when client encryption is enabled. (DSP-15029)
• SPARK/DSEFS non-super users are unable to run sql queries in secured DSEFS. (DSP-15276)
• Rare NullPointerException during DSEFS startup. (DSP-15289)
• Occasional NoHostAvailable exceptions when shutting down DSE with DSEFS enabled. (DSP-15404)
• Setting permissions/owner on a file in DSEFS through Hadoop's interfaces does not take effect. (DSP-15255)

5.1.7 DSE Graph resolved issues

• Do not log or send back full Groovy script when the script is too large. (DSP-14410)
• Retryable failures have severity DEBUG. Only terminal failures have severity ERROR or WARN. (DSP-15045)

5.1.7 DSE Search resolved issues

• dsetool search commands should return non-zero if operation was not successful. (DSP-9631)
• Add warnings to DSE Search reload and reindex that reloads impact entire datacenter and reindex is asynchronous. (DSP-9820)
• CQL solr queries with JSON clause miss singlePass optimizations. (DSP-11407)
• Inconsistent behavior from dsetool when SSL is enabled. (DSP-15171)
• Default useJtsMulti to false to avoid performance issues with JTS multipolygon handling. (DSP-15182)
• Incorrect connection limiter scheduler shutdown order for internode transport clients. (DSP-14256)
• Avoid potentially indefinite shutdown delay with active reindexing. (DSP-12452)
Hadoop libraries

Built-in Hadoop and Bring-Your-Own-Hadoop (BYOH) were deprecated in DataStax Enterprise (DSE) 5.0, and were removed in DSE 5.1. Hadoop removal from DSE 5.1 and later means that DSE does not allow for the startup of Hadoop services previously included in DSE, including MapReduce JobTracker and TaskTracker.

However, DSE has supported built-in Spark since DSE 4.5 and Bring-Your-Own-Spark (BYOS) since DSE 5.0, and that support continues today. Because Spark depends on certain Hadoop libraries on the server and the client, DSE continues to ship with Hadoop libraries that are required for running Spark and BYOS.

To view the included Hadoop libraries, see DataStax Enterprise 5.1.x third-party software.

dse.yaml

The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>installation_location/resources/dse/conf/dse.yaml</td>
</tr>
</tbody>
</table>

Cassandra enhancements for DSE 5.1.7

DataStax Enterprise (DSE) 5.1.7 includes all changes from earlier DSE releases. These production-certified changes are enhancements to Apache Cassandra™ 3.11.0. (For Cassandra updates, see CHANGES.txt.)

- 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)
- Acquire read lock before accessing CompactionStrategyManager fields (CASSANDRA-14139)
- Split CommitLogStressTest to avoid timeout (CASSANDRA-14143)
- Set encoding for javadoc generation (CASSANDRA-14154)
- RPM package spec: fix permissions for installed jars and config files (CASSANDRA-14181)
- More PEP8 compliance for cqlsh (CASSANDRA-14021)

General upgrade advice for DSE 5.1.7

Carefully review all planning and upgrade documentation in the Upgrading DataStax Enterprise guide. The following provides general upgrade information:
Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to ALWAYS snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

4.0
===

New features
------------
- SSTableDump now supports the -l option to output each partition as it's own json object  
  See CASSANDRA-13848 for more detail
- The currentTimestamp, currentDate, currentTime and currentTimeUUID functions have been added.  
  See CASSANDRA-13132
- Support for arithmetic operations between `timestamp`/`date` and `duration` has been added.  
  See CASSANDRA-11936
- Support for arithmetic operations on number has been added. See CASSANDRA-11935
- Preview expected streaming required for a repair (nodetool repair --preview), and validate the consistency of repaired data between nodes (nodetool repair --validate). See CASSANDRA-13257
- Support for selecting Map values and Set elements has been added for SELECT queries. See CASSANDRA-7396
- The initial build of materialized views can be parallelized. The number of concurrent builder threads is specified by the property `cassandra.yaml:concurrent_materialized_view_builders`.  
  This property can be modified at runtime through both JMX and the new `setconcurrentviewbuilders` and `getconcurrentviewbuilders` nodetool commands. See CASSANDRA-12245 for more details.

Upgrading
--------
- Cassandra 4.0 removed support for COMPACT STORAGE tables. All Compact Tables have to be migrated using `ALTER ... DROP COMPACT STORAGE` statement in 3.0/3.11. Cassandra starting 4.0 will not start if flags indicate that the table is non-CQL. Syntax for creating compact tables is also deprecated.
- Support for legacy auth tables in the system_auth keyspace (users, permissions, credentials) and the migration code has been removed.

Migration of these legacy auth tables must have been completed before the upgrade to 4.0 and the legacy tables must have been removed. See the 'Upgrading' section for version 2.2 for migration instructions.

- Cassandra 4.0 removed support for the deprecated Thrift interface. Amongst other things, this imply the removal of all yaml option related to thrift ('start_rpc', rpc_port, ...).

- Cassandra 4.0 removed support for any pre-3.0 format. This means you cannot upgrade from a 2.x version to 4.0 directly, you have to upgrade to a 3.0.x/3.x version first (and run upgradesstable). In particular, this mean Cassandra 4.0 cannot load or read pre-3.0 sstables in any way: you will need to upgrade those sstable in 3.0.x/3.x first.

- Upgrades from 3.0.x or 3.x are supported since 3.0.13 or 3.11.0, previous versions will causes issues during rolling upgrades (CASSANDRA-13274).

- Cassandra will no longer allow invalid keyspace replication options, such as invalid datacenter names for NetworkTopologyStrategy. Operators MUST add new nodes to a datacenter before they can set set ALTER or CREATE keyspace replication policies using that datacenter. Existing keyspaces will continue to operate, but CREATE and ALTER will validate that all datacenters specified exist in the cluster.

- Cassandra 4.0 fixes 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, anticompaation 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.

- Config option index_interval has been 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
- The minimum value for internode message timeouts is 10ms. Previously, any positive value was allowed. See cassandra.yaml entries like read_request_timeout_in_ms for more details.
- Config option commitlog_sync_batch_window_in_ms has been deprecated as it’s documentation has been incorrect and the setting itself near useless. Batch mode remains a valid commit log mode, however.
- There is a new commit log mode, group, which 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 now required the --all flag to remove all snapshots. Previous behavior would delete all snapshots by default.

3.11.2

Upgrading

- See MAXIMUM TTL EXPIRATION DATE NOTICE above.
- 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.

3.11.0

Upgrading

- 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 (only when upgrading from DSE 5.1.1 or 5.1.2 or any version lower than DSE 5.0.10)

- 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 to not 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.
Spark Cassandra Connector changes for DSE 5.1.7

A list of DataStax Enterprise 5.1.7 production-certified changes for the DataStax Spark Cassandra Connector.

DSE 5.1.7:
2.0.7
* Adds Timestamp, Improve Conversion Perf (SPARKC-522)
* Allow setting spark.cassandra.concurrent.reads (SPARKC-520)
* Allow splitCount to be set for Dataframes (SPARKC-527)

DSE 5.1.6:
2.0.6
* Includes all patches up to 1.6.10

DSE 5.1.3:
2.0.5
* Allow IN predicates for composite partition keys and clustering keys to be pushed down to Cassandra (SPARKC-490)
* Allow 'YYYY' format LocalDate
* Add metrics for write batch Size (SPARKC-501)
* Type Converters for java.time.LocalDate (SPARKC-495)

2.0.4
* Includes patches up to 1.6.9
* Retry PoolBusy Exceptions, Throttle JWCT Calls (SPARKC-503)

DSE 5.1.2:
2.0.3
* Includes patches up to 1.6.8

DSE 5.1.1:
2.0.2
* Protect against Size Estimate Overflows (SPARKC-492)
* Add java.time classes support to converters and sparkSQL (SPARKC-491)
* Allow Writes to Static Columns and Partition Keys (SPARKC-470)

DSE 5.1.0:
2.0.1
* Refactor Custom Scan Method (SPARKC-481)

2.0.0
* Upgrade driver version for 2.0.0 Release to 3.1.4 (SPARKC-474)
* Extend SPARKC-383 to All Row Readers (SPARKC-473)

2.0.0 RC1
* Includes all patches up to 1.6.5
* Automatic adjustment of Max Connections (SPARKC-471)
* Allow for Custom Table Scan Method (SPARKC-459)
* Enable PerPartitionLimit (SPARKC-446)
DataStax Enterprise 5.1 release notes

<table>
<thead>
<tr>
<th>Feature</th>
<th>Issue-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support client certificate authentication for two-way SSL Encryption</td>
<td>SPARKC-359</td>
<td></td>
</tr>
<tr>
<td>Change Config Generation for Cassandra Runners</td>
<td>SPARKC-424</td>
<td></td>
</tr>
<tr>
<td>Remove deprecated QueryRetryDelay parameter</td>
<td>SPARKC-423</td>
<td></td>
</tr>
<tr>
<td>User ConnectionHostParam.default as default hosts String</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update usages of deprecated SQLContext so that SparkSession is used</td>
<td>SPARKC-400</td>
<td></td>
</tr>
<tr>
<td>Test Reused Exchange</td>
<td>SPARKC-429</td>
<td></td>
</tr>
<tr>
<td>Module refactoring</td>
<td>SPARKC-398</td>
<td></td>
</tr>
<tr>
<td>Recognition of Java Driver Annotated Classes</td>
<td>SPARKC-427</td>
<td></td>
</tr>
<tr>
<td>RDD.deleteFromCassandra</td>
<td>SPARKC-349</td>
<td></td>
</tr>
<tr>
<td>Coalesce Pushdown to Cassandra</td>
<td>SPARKC-161</td>
<td></td>
</tr>
<tr>
<td>Custom Conf options in Custom Pushdowns</td>
<td>SPARKC-435</td>
<td></td>
</tr>
<tr>
<td>Upgrade CommonBeatUtils to 1.9.3 to Avoid SID-760</td>
<td>SPARKC-457</td>
<td></td>
</tr>
</tbody>
</table>

2.0.0 M3
* Includes all patches up to 1.6.2

2.0.0 M2
* Includes all patches up to 1.6.1

2.0.0 M1
* Added support for left outer joins with Cassandra table              | SPARKC-181 |                                                                              |
* Removed CassandraSqlContext and underscore based options            | SPARKC-399 |                                                                              |
* Upgrade to Spark 2.0.0-preview                                      | SPARKC-396 |                                                                              |
  - Removed Twitter demo because there is no spark-streaming-twitter  |
  - Removed Akka Actor demo because there is no support for such       |
  - Bring back Kafka project and make it compile                       |
  - Update several classes to use our Logging instead of Spark Logging|
  - Update few components and tests to make them work with Spark 2.0.0 |
  - Fix Spark SQL - temporarily                                        |
  - Update plugins and Scala version                                   |

TinkerPop changes for DSE 5.1.7

DataStax Enterprise (DSE) 5.1.7 includes all changes from previous releases. These production-certified changes are enhancements to TinkerPop 3.2.7:

- Performance enhancement for OLAP: \( n^2 \) synchronous operation in OLAP
  WorkerExecutor.execute() method. (TINKERPOP-1870)
- union() can produce extra traversers. (TINKERPOP-1867)

DSE 5.1.6

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

**Attention**: TTL expiration timestamps are susceptible to the **year 2038 problem**. If the TTL value is long and an expiration date is greater than the maximum threshold
of 2038-01-19T03:14:06+00:00, the data is immediately expired and purged on the next compaction. When using a long TTL, DataStax strongly recommends upgrading to DSE 5.1.7 or later and taking required action.

22 January 2018

- 5.1.6 Components (page 69)
- 5.1.6 Highlights (page 69)
- 5.1.6 Changes and enhancements (page 70)
- 5.1.6 Resolved issues (page 72)
- 5.1.6 Known issues (page 75)
- 5.1.6 Cassandra enhancements (page 77)
- 5.1.6 General upgrade advice (page 77)

5.1.6 Components

- Apache Cassandra™ 3.11.0.2070 (updated)
- Apache Solr™ 6.0.1.0.2123 (updated)
- Apache Spark™ 2.0.2.6
- Apache Tomcat® 8.0.44
- DataStax Spark Cassandra Connector 2.0.6 (updated)
- DSE Java Driver 1.2.2
- Netty 4.0.42.Final
- Spark Jobserver 0.6.2.234 (requires compatible API)
- TinkerPop 3.2.7-20171213-77c0c764 (updated)
- Select Hadoop libraries

5.1.6 Highlights

Executive summary highlights for DSE 5.1.6:

- DataStax Enterprise core (page 69)
- DSE Advanced Replication (page 69)
- DSE Analytics (page 70)
- DSE Graph (page 70)
- DSE Search (page 70)

The executive summary highlights are just a top-level view. Be sure to review all 5.1.6 Changes and enhancements (page 70).

5.1.6 DataStax Enterprise core highlights

- Commands to support migrating from compact storage. These commands are required to upgrade to DSE 6.0. (DSP-14966)

DSE Advanced Replication highlights
• Improved handling and bug fixes in scenarios where the source cluster has multiple logical data centers. (DSP-14767, DSP-14515, DSP-15121)

5.1.6 DSE Analytics and DSEFS highlights

• Fixed a DSEFS issue that could prevent upgrades from 5.0.x to 5.1.5. (DSP-15237)
• Fixed a bug in DSEFS that in rare circumstances could cause a live lock on the server when reading files, manifesting with high CPU usage and timeouts. (DSP-15107)
• Fixed an infrequent bug where Spark worker directories could be deleted while the job is running. (DSP-15076, SPARK-22976).

5.1.6 DSE Graph highlights

• Graph loader supports GraphSON V2.
• Resolved issue of retrieving multiple edges by ID. (DSP-14580)
• Allow vertex lookup through index on id property keys. (DSP-9028)

5.1.6 DSE Search highlights

• Performance and corruption issues with encrypted indexes are addressed with a full reindex after upgrade. (DSP-14943, DSP-14485, DSP-15265).
• All installations from DSE 5.0.x or earlier versions of DSE 5.1.x should upgrade to DSE 5.1.6 to avoid potentially incorrect queries while nodes are at different versions during upgrade. (DSP-14898, DSP-14993)
• Improved protection against abusing the Solr filter cache with too many entries. (DSP-14534)
• Performance improvements with RF=(# nodes) DCs. (DSP-12962)

5.1.6 Changes and enhancements

In addition to the 5.1.6 Highlights (page 69), review all changes and enhancements:

• DataStax Enterprise (page 70)
• DSE Advanced Replication (page 71)
• DSE Analytics (page 71)
• DSE Graph (page 71)
• DSE Search (page 71)

5.1.6 DataStax Enterprise core changes and enhancements

• Generate Kerberos debug output. (DSP-12430)
• JMX SSL is supported for use with dsetool and dse advrep. See Setting up SSL for nodetool, dsetool, and dse advrep. (DSP-14200)
• New skip-read-validation flag for stress test error handling. (DSP-14775)
• Ensure that the list and set selectors elements are all of the same type. (DSP-14775)
• Do not leak body buffer in case of protocol exceptions and upgrade Netty to 4.0.52. (DSP-14775)
• Added `-Dcassandra.native_transport_startup_delay_seconds (page 363)` start-up parameter to delay startup of native transport. (DSP-14839)
• Add `nodetool rebuild` mode `reset-no-snapshot` option. (DSP-14827)
• Add `nodetool abortrebuild` command. (DSP-14827)
• Add metrics on coordination of read commands; see `type=ReadCoordination` . (DSP-14775)
• Add `cross_dc_rtt_in_ms` to cross dc requests, default 0. (DSP-14775)
• New metrics for batchlog-replays. (DSP-14839)
• New CQL ALTER TABLE `DROP COMPACT STORAGE` option to remove Thrift-compatibility from tables. (DSP-14839)
• Handle continuous paging state for empty partitions with static rows. (DSP-14959)
• Skip building views during base table streams on range movements. (DSP-14959)
• Allow DiskBoundaryManager to cache different directories. (DSP-15024)
• Do not apply read timeouts to aggregated queries and use a minimum internal page size. New `cassandra.yaml` `aggregated_request_timeout_in_ms (page 296)` setting. (DSP-15024)
• Generate LDAP debug output. (DSP-15176)

5.1.6 DSE Advanced Replication changes and enhancements

• Gremlin Console command line (page 773) options for connecting to host. (DSP-12726)
  • `--ssl-enabled true` is the same as the new `--ssl` option for JMX SSL support. See Setting up SSL for nodetool, dsetool, and dse advrep. (DSP-14200)

5.1.6 DSE Analytics changes and enhancements

• Default logging level for `org.apache.spark.rpc` has been changed to ERROR. (DSP-14651)
• Improved Spark shell startup time. (DSP-14704)
• Spark executors are not restarted if the driver port is closed or unreachable. (DSP-14824)
• Notebooks and other third-party tool integration with Spark. (DSP-14489)

5.1.6 DSE Graph changes and enhancements

• Gremlin console plugins.txt is read-only by default. (DSP-13372)
• Traversal does not timeout with the Fluent API. (DSP-13156)
• Graph traversals over a vertex-centric index with an ordering and result limit are more efficient. (DSP-15191)
• CQL Statement latency metrics. (DSP-15124)
• Improve error messaging on failed bytecode translation. Long forms of e and -i are working. (DSP-15091)

5.1.6 DSE Search changes and enhancements
• Avoid token filtering on single-node CQL solr_query. (DSP-12962)
• Maximum number of entries in SolrFilterCache is limited to 32K. (DSP-14534)
• CREATE SEARCH INDEX indexed true|false option for more performant indexes. (DSP-14364)
• Eliminate delay for scheduled snapshot collection for DSE Search performance objects. (DSP-14561)
• Added log message for filter cache evictions. (DSP-14944)
• After compact storage is dropped from a table that also has a search index, HTTP writes and deletes-by-ID on the search index are disabled. (DSP-14966)

5.1.6 Resolved issues

Resolved issues for:
• DataStax Enterprise core (page 72)
• DSE Advanced Replication (page 73)
• DSE Analytics (page 73)
• DSEFS (page 74)
• DSE Graph (page 74)
• DSE Search (page 75)

5.1.6 DataStax Enterprise resolved issues

• Audit logging does not support UNSET values from prepared statements. (DSP-13043)
• dsetool does not work with JMX SSL. To use, follow steps in Setting up SSL for nodetool, dsetool, and dse advrep. (DSP-14200)
• DataStax Installer upgrades within 5.1.x prevent Spark shell from working. (DSP-14637)
• Memory leak causes executor descriptions to accumulate in DSE process. (DSP-14868)
• Handle continuous paging state for empty partitions with static rows. (DSP-14959)
• Skip building views during base table streams on range movements. (DSP-14959)
• Add invalid-sstable-root JVM argument to all relevant test entries in build.xml. (DSP-14827)
• Do not leak body buffer in case of protocol exceptions and upgrade Netty to 4.0.52 (DSP-14775)
• Ensure that the list and set selectors elements are all of the same type. (DSP-14775)
• nodetool arguments with spaces print script errors. (DSP-14959)
• Change token allocation to use RF=1 method when RF equals rack count. (DSP-14959)
• Failed bootstrap streaming leaves auth uninitialized. (DSP-14839)
• Eliminate thread roundtrip for version handshake. (DSP-14827)
• Make nodetool assassinate more resilient to missing tokens. (DSP-14827)
• Throttle base partitions during MV repair streaming to prevent OOM. (DSP-14775)
• Register SizeEstimatesRecorder earlier and enable cleanup of invalid entries. (DSP-15024)
• Only serialize failed batchlog replay mutations to hints. (DSP-15024)
• Allow selecting static column only when querying static index. (DSP-15087)
• Require only MODIFY permission on base when updating table with MV (DSP-15087).
• Force sstableloader exit to prevent hanging due to non-daemon threads running. (DSP-15087)
• Add autoclosable to CompressionMetadata and fix leaks in SSTableMetadataViewer. (DSP-15087)
• Use all columns to calculate estimatedRowSize for aggregation internal query. (DSP-15087)
• Prevent continuous schema exchange between DSE 5.0 and DSE 5.1 nodes. (DSP-15087)
• Separate commit log replay and commit throwable inspection and policy handling. (DSP-15087)
• Fix for local DC when connections are compressed despite internode_compression: dc. (DSP-15087)
• Expanded hinted handoff instrumentation. (DSP-15087)
• Improve gossip dissemination time. (DSP-15087)
• Use more intelligent level picking for non-l0 file. (DSP-15087)
• LCS levels are not respected for nodetool refresh and replacing a node. (DSP-15087)
• Keep SSTable level for decommission, remove, and move operations. (DSP-15087)
• More quickly detect down nodes for batchlogs using the incoming connections. (DSP-15087)
• Fixes for waitForGossiper. (DSP-15087)
• Print heap histogram on OOM errors by default. (DSP-15087)
• Support frozen collection list and set in stress. (DSP-15087)
• Improved streams logging. (DSP-15087)
• Make migration-delay configurable. (DSP-15087)
• Improved schema migration logging. (DSP-15087)
• Switch RMIExporter to dynamic proxy. (DSP-15277).
• Do not fetch columns that are not in the filter fetched set. (DSP-15277)
• DataStax Enterprise will not run with Java 1.8u161 or later. (DSP-15277)

5.1.6 DSE Advanced Replication resolved issues

• Datacenter not consistently passed into TokenService causes multi-datacenter replication errors. (DSP-14767)
• Incompatibility with durable_writes=false, but no warning/error. (DSP-15205)
• CDC on a table should be disabled only when no channels are enabled for that source table. (DSP-15121)
• CDC files are left in a DC that’s not collecting. (DSP-15105)

5.1.6 DSE Analytics resolved issues

• dse client-tool configuration export/import incorrectly uses cfs as the default file system. (DSP-14535)
• Spark shuffle service fails to update secret on application re-attempts. (DSP-15038)
• Need a dedicated user to run Graph OLAP Spark Driver. (DSP-14869)
• Logs from Spark Jobserver job are missing. (DSP-14981)
• Poor handling of task notifications in Spark Driver, including possible memory leak.
  (DSP-15044)
• Cluster-deployed drivers are not cleaned up by the Spark Worker cleanup service.
  (DSP-15076)

5.1.6 DSEFS resolved issues

• "ERROR: Request body rejected, ConnectionClosedException" message is not logged in
  system.log if the client disconnects in the middle of the request. (DSP-14597)
• Added getScheme, getDefaultPort, and concat method implementations to
  DseFileSystem Hadoop API. (DSP-14605)
• Reads incorrectly show Response body rejected errors. (DSP-14615)
• DSEFS authorization is enabled when DSE authorization is enabled. DSEFS supports
  DseAuthorizer transitional mode. (DSP-14616)
• DSEFS does not retry queries. (DSP-14649)
• Incorrect return of 0 exit code for failed command execution. (DSP-14652)
• Performing cat operation on a directory is prohibited and causes a Not a regular file
  <path> message. (DSP-14696)
• User name/password was not provided warning is in the DSEFS shell log when
  security is not enabled. (DSP-14708)
• DSEFS fsck command does not fix File not found: / problem which can occur in rare
  cases after new cluster nodes are started in parallel. (DSP-15048)
• A live lock on the server when reading files manifests with high CPU usage and timeouts.
  (DSP-15107)
• DSEFS files created through Hadoop API do not properly inherit RF and block size from
  the parent directory. (DSP-15139)
• "Promise already completed" error in DSEFS connection pool. (DSP-15122)
• No check if parent element of a given target path is a directory for mkdir, put, move
  operations. (DSP-15100)

5.1.6 DSE Graph resolved issues

• Whitelist org.apache.tinkerpop.gremlin.spark.structure.Spark in sandbox so that Apache
  TinkerPop Spark-Gremlin application can be stopped programatically. (DSP-14678)
• Queries with multiple conditions using heterogeneous operators that cover the same
  property value cause an error. (DSP-14623)
• Error when retrieving multiple edges by edge IDs when the list of IDs is greater than 3.
  (DSP-14580)
• Unlabelled index queries occur even when labels were indexed by the appropriate key.
  (DSP-14579)
• graph.io read does not work with custom IDs. Limitations apply, intended for use with
  small graphs only. (DSP-14568)
• Setting a TraversalSource option from the DSE Driver isn't effective. (DSP-14713)
• QueryStrategy illegally moves HasStep condition across edge traversal. (DSP-15081)
• Date, Time, Duration, Timestamp, Blob Graph types are represented by incorrect java types in OLAP. Converters were added to have the same types as in OLTP. (DSP-15104)

5.1.6 DSE Search resolved issues

• NPE when dropping the Solr core while indexing is in progress. (DSP-13252)
• dssetool upgrade_index_files does not work with authentication enabled. (DSP-14114)
• UpdateMetrics::Latency::Mean is "unavailable" when writes are in progress. (DSP-14392)
• When executing CQL search queries with a keyspace RF=(number of nodes), then the token filter is no longer created resulting in faster queries. (DSP-14468)
• EncryptedFSDirectory#outputLengthCache corruption makes encrypted index files unreadable. (DSP-14485)
• Solr filter cache fails after restart. (DSP-14608)
• CREATE SEARCH INDEX does not have direct control over tuple and UDT fields. (DSP-14639)
• Remove code execution vulnerability: CVE-2016-6809. (DSP-14747)
• Infinite parsing loop possible with Extended DisMax (eDisMax) query parser and local parameters. (DSP-14748)
• Internal server error 500 on solr/admin/cores?action=STATUS&memory=true. (DSP-14783)
• ExtendedDismaxQParser (edismax) ignores Boolean OR when q.op=AND and mm is not explicitly set. (DSP-14799)
• Grouping by TrieDateField and DatePointField fails. (DSP-14808)
• Token filtering might be missed on mixed versions clusters. (DSP-14898)
• Support the json.facet parameter in Solr UI. (DSP-14893)
• Excessive time spent reading unencrypted segment sizes during search index (Solr core) loading. Slow startup on nodes with large encrypted indexes is resolved after upgrade to DSE 5.1.6 is completed with a full reindex for all search indexes using encryption. (DSP-14943, DSP-14485, DSP-15265)
• Shutdown order in SolrCore causes RejectedExecutionExceptions around CommitTracker. (DSP-15040)
• Cannot create core using HTTP due to missing create permission (page 639). (DSP-15046)

5.1.6 Known issues

• Potential data loss for INSERTs with very large TTLs. TTL expiration timestamps are susceptible to the year 2038 problem. (DSP-15412)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. TTLs are considered "very large" when close to the maximum allowed value of 630720000 seconds (20 years),
starting from 2018-01-19T03:14:06+00:00. As time progresses, the maximum supported TTL is gradually reduced as the maximum expiration date approaches. For instance, on 2028-01-19T03:14:06 with a TTL of 10 years is impacted. The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately.

**Warning:** Upgrade to DSE 5.1.7 or later and take required action to protect against overflow of local expiration time.

- Spark SQL applications with DSE authentication enabled will throw errors if the DSEFS scratch directory doesn’t exist. (DSP-15276)

Spark SQL applications utilize a scratch directory located in DSEFS. Make sure the dsefs://tmp/hive directory exists and that it has 733 permissions. If dsefs://tmp/hive does not exist, it must be created by a role with superuser permissions. Create the scratch directory with proper permissions:

```bash
$ dse fs 'mkdir -p -m 733 /tmp/hive'
```

- Spark Master might not recover after upgrades from DSE 5.1.0 through 5.1.5 to DSE 5.1.6 or 5.1.7. (DSP-15679)

In some scenarios, the Spark Master might not recover directly after upgrade, and all the Spark applications must be stopped and restarted. Follow these steps to ensure Spark Master launches successfully for upgrades from any DSE 5.1.x to 5.1.8:

```bash
$ dsetool sparkmaster cleanup
$ dsetool sparkworker restart
```

**Hadoop libraries**

Built-in Hadoop and Bring-Your-Own-Hadoop (BYOH) were deprecated in DataStax Enterprise (DSE) 5.0, and were removed in DSE 5.1. Hadoop removal from DSE 5.1 and later means that DSE does not allow for the startup of Hadoop services previously included in DSE, including MapReduce JobTracker and TaskTracker.

However, DSE has supported built-in Spark since DSE 4.5 and Bring-Your-Own-Spark (BYOS) since DSE 5.0, and that support continues today. Because Spark depends on certain Hadoop libraries on the server and the client, DSE continues to ship with Hadoop libraries that are required for running Spark and BYOS.

To view the included Hadoop libraries, see [DataStax Enterprise 5.1.x third-party software](http://docs.datastax.com/en/dse/5.1/index.html#third-party-software).

**dse.yaml**

The location of the `dse.yaml` file depends on the type of installation:
DataStax Enterprise 5.1 release notes

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/dse/conf/dse.yaml</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>installation_location/resources/dse/conf/dse.yaml</td>
</tr>
</tbody>
</table>

Cassandra enhancements for DSE 5.1.6

DataStax Enterprise (DSE) 5.1.6 includes all changes from earlier DSE releases. These production-certified changes are enhancements to Apache Cassandra™ 3.11.0. (For Cassandra updates, see CHANGES.txt.)

- Switch RMIExporter to dynamic proxy. (DSP-15277)
- Do not fetch columns that are not in the filter fetched set. (DSP-15277)
- Allow selecting static column only when querying static index. (DSP-15087)
- Require only MODIFY permission on base when updating table with MV. (DSP-15087)
- Force sstableloader exit to prevent hanging due to non-daemon threads running. (DSP-15087)
- Add autoclosable to CompressionMetadata and fix leaks in SSTableMetadataViewer. (DSP-15087)
- Use all columns to calculate estimatedRowSize for aggregation internal query. (DSP-15087)
- Prevent continuous schema exchange between DSE 5.0 and DSE 5.1 nodes (DSP-15087)
- Allow DiskBoundaryManager to cache different Directories (DSP-15024)
- Do not apply read timeouts to aggregated queries and use a minimum internal page size. (DSP-15024)
- Handle cont paging state for empty partitions with static rows (DSP-14959)
- Skip building views during base table streams on range movements. (DSP-14959)
- Add invalid-sstable-root JVM argument to all relevant test entries in build.xml. (DSP-14827)
- Do not leak body buffer in case of protocol exceptions and upgrade Netty to 4.0.52. (DSP-14775)
- Ensure that the list and set selectors elements are all of the same type. (DSP-14775)

General upgrade advice for DSE 5.1.6

Carefully review all planning and upgrade documentation in the Upgrading DataStax Enterprise guide. The following provides general upgrade information:

```markdown
GENERAL UPGRADING ADVICE FOR ANY VERSION
========================================

Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to ALWAYS snapshot before any upgrade, just in case you need to roll back to the previous version.
```

DSE 5.1 Developer Guide (Previous version)
(Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

DSE 5.1.6
=========

Upgrading
---------

- seed_gossip_probability setting was added to cassandra.yaml. This setting will pick the percentage of times gossip messages are sent to a seed. This improves the time it takes for gossip changes to propagate across the cluster. Defaults to 100% (1.0)
- Upgrades from DSE 5.0 might have produced unnecessary schema migrations while there was at least one DSE 5.0 node in the cluster. It is therefore highly recommended to upgrade from DSE 5.0 to at least DSE 5.1.6. The root cause of this schema mismatch was a difference in the way how schema digests were computed in DSE 5.0 and DSE 5.1. To mitigate this issue, DSE 5.1.6 and newer announce DSE 5.0 compatible digests as long as there is at least one DSE 5.0 node in the cluster. Once all nodes have been upgraded, the "real" schema version will be announced. Note: this fix is only necessary in DSE 5.1 and therefore only applies to DSE 5.1. (DB-1477)
- DSE 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
- DSE is logging by default a heap histogram on OutOfMemoryError. To disable that behavior set the 'cassandra.printHeapHistogramOnOutOfMemoryError' System property to 'false'.
- Improved gossip settling added. On startup DSE waits till all nodes are seen before fully joining the cluster. This improves latency spikes when restarting nodes.
- LeveledCompactionStrategy SSTables will keep their existing level on nodetool refresh, nodetool move, and nodetool decommission.

Operations
- New command 'nodetool abortrebuild' allows to abort a currently running rebuild operation. The command must be executed on the node where the rebuild operation is running. Streams may continue until they finish or timeout.
- Only MODIFY permission on base is required to update table with MV, internally it reads base data and generates updates to MV.

Metrics
-------
- New storage metrics were added:
  * TotalHintsReplayed: how many hints were successfully replayed on the _target_ node.
  * HintsOnDisk: how many hints are currently persistent on disk on this node. Metric is updated for the amount of hints contained in the hints file when hints file is written or removed.
  
Values is restored on node startup.

New features
-------------
- Statistics file component was added to Hint Store in order to provide information about amount of hints contained in the hints file without replaying it. Stats component is completely backwad-compatible; hint files without this component will not be counted.
  
All new hint files will be created with this component. See DB-853 for more details.

Compact Storage (only when upgrading from 5.1.5 or any version lower than 5.0.12)
---------------------
- Starting in Cassandra version 4.0, Thrift and COMPACT STORAGE will no longer be supported.
  
'ALTER ... DROP COMPACT STORAGE' statement makes Compact Tables CQL-compatible, exposing internal structure of Thrift/Compact Tables. You can find more details on exposed internal structure under:
  
http://cassandra.apache.org/doc/latest/cql/appendices.html#appendix-c-dropping-compact-storage

For uninterrupted cluster upgrades, drivers now support 'NO_COMPACT' startup option.
Supplying this flag will have same effect as 'DROP COMPACT STORAGE', but only for the current connection.

In order to upgrade, clients supporting a non-compact schema view can be rolled out
gradually. When all the clients are updated 'ALTER ... DROP COMPACT STORAGE' can be executed. After dropping compact storage, 'NO_COMPACT' option will have no effect after that.

DSE 5.1.3
=========

Upgrading
---------
- 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 table system_auth.resource_role_permissions_index is no longer used and should be dropped after all nodes are on 5.1.3. Note that upgrades from DSE 5.0 series since 5.0.10 to DSE versions before 5.1.3 are not recommended.
- Full repairs are now default if no option is specified on nodetool repair, unless incremental repair was already run on the table/keyspace being repaired, to maintain backward compatibility. Incremental repair may be run on new tables by using the --inc option.
- Full repairs will no longer run repair unless the --run-anticompaction option is specified
- Incremental repairs are no longer supported on tables with materialized views or CDC until its limitations are addressed. An incremental repair triggered on a base table or materialized view run a full repair instead. See CASSANDRA-12888 for details.

Materialized Views (only when upgrading from DSE 5.1.1 or 5.1.2 or any version lower than DSE 5.0.10)
-------------------------------------------------------------------------------------
- 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
to not be reflected in the view when repairing the base table post-upgrade. This
case 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.

3.11.2
======
Upgrading
--------
- Nothing specific to this release, but please see previous upgrading
sections.

3.11.1
======
Upgrading
--------
- 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"

3.11.0
======
Upgrading
--------
- 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_columnsUnsafe=true"

- ALTER TABLE (ADD/DROP COLUMN) operations concurrent with a read might result into data corruption (see CASSANDRA-13004 for more details). Fixing this bug required a messaging protocol version bump. By default, Cassandra 3.11 will use 3014 version for messaging.

Since Schema Migrations rely the on exact messaging protocol version match between nodes, if you need schema changes during the upgrade process, you have to start your nodes with `"-Dcassandra.force_3_0_protocol_version=true"` first, in order to temporarily force a backwards compatible protocol.

After the whole cluster is upgraded to 3.11, do a rolling restart of the cluster without setting that flag.

3.11 nodes with and without the flag set will be able to do schema migrations with other 3.x and 3.0.x releases.

While running the cluster with the flag set to true on 3.11 (in compatibility mode), avoid adding or removing any columns to/from existing tables.

If your cluster can do without schema migrations during the upgrade time, just start the cluster normally without setting aforementioned flag.

If you are upgrading from 3.0.14+ (of 3.0.x branch), you do not have to set an flag while upgrading to ensure schema migrations.
- 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).

3.10
====

New features
------------
- New `DurationType` (cql duration). See CASSANDRA-11873
- Runtime modification of concurrent_compactors is now available via nodetool
- 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.
  Compaction can now take into account overlapping tables that don't take part in the compaction to look for deleted or overwritten data in the compacted tables.
  Then such data is found, it can be safely discarded, which in turn should enable the removal of tombstones over that data.

The behavior can be engaged in two ways:
- as a "nodetool garbagecollect -g CELL/ROW" operation, which applies single-table compaction on all sstables to discard deleted data in one step.
- as a "provide_overlapping_tombstones:CELL/ROW/NONE" compaction strategy flag, which uses overlapping tables as a source of deletions/overwrites during all compactions.

The argument specifies the granularity at which deleted data is to be found:
- If ROW is specified, only whole deleted rows (or sets of rows) will be discarded.
- If CELL is specified, any columns whose value is overwritten or deleted will also be discarded.
- NONE (default) specifies the old behavior, overlapping tables are not used to decide when to discard data.

Which option to use depends on your workload, both ROW and CELL increase the disk load on compaction (especially with the size-tiered compaction strategy), with CELL being more resource-intensive. Both should lead to better read performance if deleting rows (resp. overwriting or deleting cells) is common.
- 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.

Upgrading
--------
- 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
- 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 were added to stdout. (see CASSANDRA-12004)

- The 'clientutil' jar, which has been somewhat broken on the 3.x branch, is no 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 across 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() method.
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 litterals support the ISO 8601 format. By consequence, identifiers matching that format (e.g P2Y or P1MT6H) will not be supported anymore (CASSANDRA-11873).

Spark Cassandra Connector changes for DSE 5.1.6

A list of DataStax Enterprise 5.1.6 production-certified changes for the DataStax Spark Cassandra Connector.

DSE 5.1.6:
2.0.6
* Includes all patches up to 1.6.10

DSE 5.1.3:
2.0.5
* Allow IN predicates for composite partition keys and clustering keys to be pushed down to Cassandra (SPARKC-490)
* Allow 'YYYY' format LocalDate
* Add metrics for write batch Size (SPARKC-501)
* Type Converters for java.time.localdate (SPARKC-495)

2.0.4
* Includes partches up to 1.6.9
* Retry PoolBusy Exceptions, Throttle JWCT Calls (SPARKC-503)

DSE 5.1.2:
2.0.3
* Includes patches up to 1.6.8

DSE 5.1.1:
2.0.2
* Protect against Size Estimate Overflows (SPARKC-492)
* Add java.time classes support to converters and sparkSQL(SPARKC-491)
* Allow Writes to Static Columnns and Partition Keys (SPARKC-470)

DSE 5.1.0:
2.0.1
* Refactor Custom Scan Method (SPARKC-481)
2.0.0
* Upgrade driver version for 2.0.0 Release to 3.1.4 (SPARKC-474)
* Extend SPARKC-383 to All Row Readers (SPARKC-473)

2.0.0 RC1
* Includes all patches up to 1.6.5
* Automatic adjustment of Max Connections (SPARKC-471)
* Allow for Custom Table Scan Method (SPARKC-459)
* Enable PerPartitionLimit (SPARKC-446)
* Support client certificate authentication for two-way SSL Encryption (SPARKC-359)
* Change Config Generation for Cassandra Runners (SPARKC-424)
* Remove deprecated QueryRetryDelay parameter (SPARKC-423)
* User ConnectionHostParam.default as default hosts String
* Update usages of deprecated SQLContext so that SparkSession is used instead (SPARKC-400)
* Test Reused Exchange SPARK-17673 (SPARKC-429)
* Module refactoring (SPARKC-398)
* Recognition of Java Driver Annotated Classes (SPARKC-427)
* RDD.deleteFromCassandra (SPARKC-349)
* Coalesce Pushdown to Cassandra (SPARKC-161)
* Custom Conf options in Custom Pushdowns (SPARKC-435)
* Upgrade CommonBeatUtils to 1.9.3 to Avoid SID-760 (SPARKC-457)

2.0.0 M3
* Includes all patches up to 1.6.2

2.0.0 M2
* Includes all patches up to 1.6.1

2.0.0 M1
* Added support for left outer joins with Cassandra table (SPARKC-181)
* Removed CassandraSqlContext and underscore based options (SPARKC-399)
* Upgrade to Spark 2.0.0-preview (SPARKC-396)
  - Removed Twitter demo because there is no spark-streaming-twitter package available anymore
  - Removed Akka Actor demo because there is no support for such streams anymore
    - Bring back Kafka project and make it compile
    - Update several classes to use our Logging instead of Spark Logging
  - Spark Logging became private
  - Update few components and tests to make them work with Spark 2.0.0
  - Fix Spark SQL – temporarily
  - Update plugins and Scala version

---

**TinkerPop changes for DSE 5.1.6**

DataStax Enterprise (DSE) 5.1.6 includes all changes from previous releases. These production-certified changes are enhancements to TinkerPop 3.2.7:

- Improve type-safety in Gremlin.Net methods. (TINKERPOP-1752)
- Fix for problems with hasId() fails for empty collections. (TINKERPOP-1802)
- Python supports GraphSON types g:Date, g:Timestamp and g:UUID. (TINKERPOP-1807)
- Improve error messaging on failed bytecode translation. (TINKERPOP-1811)
- Graph API removed from usage in the process test suite. (TINKERPOP-1813/TINKERPOP-1814)
- Consistent behavior of self-referencing edges. (TINKERPOP-1821)
- Improve flexibility of detachment for EventStrategy. (TINKERPOP-1829)
- Race condition in TinkerGraph index creation. (TINKERPOP-1830)
- Bug fix inTraversalHelper.replaceStep. (TINKERPOP-1832)
- API fix for DetachedEdge.Builder#setInV and setOutV doesn't return the builder. (TINKERPOP-1833)
- Long forms of e and -i are now working. (TINKERPOP-1851)

DSE 5.1.5

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

Attention: TTL expiration timestamps are susceptible to the year 2038 problem. If the TTL value is long and an expiration date is greater than the maximum threshold of 2038-01-19T03:06+00:00, the data is immediately expired and purged on the next compaction. When using a long TTL, DataStax strongly recommends upgrading to DSE 5.1.7 or later and taking required action.

19 October 2017

- 5.1.5 Components (page 88)
- 5.1.5 Highlight (page 88)
- 5.1.5 Known issues (page 89)

5.1.5 Components

- Apache Cassandra™ 3.11.0.1900
- Apache Solr™ 6.0.1.0.1984 (updated)
- Apache Spark™ 2.0.2.6
- Apache Tomcat® 8.0.44
- DataStax Spark Cassandra Connector 2.0.5 (updated)
- DSE Java Driver 1.2.2
- Netty 4.0.42.Final
- Spark Jobserver 0.6.2.234 (requires compatible API)
- TinkerPop 3.2.7-20170926-2e5c13b7
- Select Hadoop libraries

5.1.5 Highlight

A single change for DSE Search:
• Due to CVE-2017-12629, added Solr XMLParser protection from XML External Entity (XXE) attacks and removed Solr RunExecutableListener to harden security for DSE Search enabled clusters. (DSP-14618)

5.1.5 DataStax Enterprise known issues

• DataStax Enterprise will not run with Java 1.8u161 or later. (DSP-15277)
• Spark SQL applications with DSE authentication enabled will throw errors if the DSEFS scratch directory doesn't exist. (DSP-15276)

Spark SQL applications utilize a scratch directory located in DSEFS. Make sure the dsefs://tmp/hive directory exists and that it has 733 permissions. If dsefs://tmp/hive does not exist, it must be created by a role with superuser permissions. Create the scratch directory with proper permissions:

```
$ dse fs 'mkdir -p -m 733 /tmp/hive'
```

• Potential data loss for INSERTs with very large TTLs. TTL expiration timestamps are susceptible to the year 2038 problem. (DSP-15412)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. TTLs are considered "very large" when close to the maximum allowed value of 630720000 seconds (20 years), starting from 2018-01-19T03:14:06+00:00. As time progresses, the maximum supported TTL is gradually reduced as the maximum expiration date approaches. For instance, on 2028-01-19T03:14:06 with a TTL of 10 years is impacted. The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately.

**Warning:** Upgrade to DSE 5.1.7 or later and take required action to protect against overflow of local expiration time.

**DSE 5.1.4**

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

**Attention:** TTL expiration timestamps are susceptible to the year 2038 problem. If the TTL value is long and an expiration date is greater than the maximum threshold of 2038-01-19T03:14:06+00:00, the data is immediately expired and purged on the next compaction. When using a long TTL, DataStax strongly recommends upgrading to DSE 5.1.7 or later and taking required action.
12 October 2017

- 5.1.4 Components (page 90)
- 5.1.4 Highlights (page 90)
- 5.1.4 Changes and enhancements (page 91)
- 5.1.4 Resolved issues (page 92)
- 5.1.4 Known issues (page 93)
- 5.1.4 Cassandra enhancements (page 95)
- 5.1.4 General upgrade advice (page 99)

5.1.4 Components

- Apache Cassandra™ 3.11.0.1900 (updated)
- Apache Solr™ 6.0.1.0.1949 (updated)
- Apache Spark™ 2.0.2.6
- Apache Tomcat® 8.0.44
- DataStax Spark Cassandra Connector 2.0.5 (updated)
- DSE Java Driver 1.2.2
- DSEFS 5.1.2 (updated)
- Netty 4.0.42.Final
- Spark Jobserver 0.6.2.234 (requires compatible API)
- TinkerPop 3.2.7-20170926-2e5c13b7 (updated)
- Select Hadoop libraries

5.1.4 Highlights

Executive summary highlights for DSE 5.1.4:

- DSE Graph (page 90)
- DSE Search (page 90)

The executive summary highlights are just a top-level view. Be sure to review all 5.1.4 Changes and enhancements (page 91).

5.1.4 DSE Graph highlights

- Security: Graph Sandbox is enabled and configured by default. (DSP-11679)
- Vertices with custom IDs return ID components as properties. (DSP-14262)

5.1.4 DSE Search highlights

- Improved stability and performance when dealing with non-indexed fields. (DSP-6501)
  
  Full validation on all schema fields might result in validation failures after upgrade. See (page 92).

- Fixed the search performance objection regression issues. (DSP-14241)
• Fixed the memory leak issue when encrypting the index. (DSP-13826)

5.1.4 Changes and enhancements

In addition to the 5.1.4 Highlights (page 90), review all changes and enhancements:

• DataStax Enterprise (page 91)
• DSE Advanced Replication (page 91)
• DSE Graph (page 91)
• DSEFS (page 91)
• DSE Search (page 92)

5.1.4 DataStax Enterprise changes and enhancements

• scrub validates the partition key. Validation is added to schema mutation creation. (DSP-14366)
• Always define execution_profiles in cqlsh.py. (DSP-14494)
• Issue warning before running full repair when increasing replication factor. (DSP-14494)
• Add anti-compaction metrics and warn users when incremental repair is inefficient. (DSP-14494)

5.1.4 DSE Advanced Replication changes and enhancements

• Command line interface should use non-zero exit code for unknown commands. (DSP-13590)

5.1.4 DSE Graph changes and enhancements

• Enable and configure the graph sandbox by default to improve security. (DSP-11679)
• GraphFrame 0.5 fixes graph frame algorithms. (DSP-14271)
• Gremlin console uses the default plugins.txt in the DSE distribution. If a user home is specified with bin/dse gremlin-console ~/gremlin-console then extra checks are performed to ensure that plugins.txt is populated. (DSP-14286)
• Prevent multi-properties for the partition/clustering key. (DSP-14300)
• graph.tx().commit(); call is not allowed on graph.tx().commit(); graph.tx().config().option("allow_scan", true).open(); g.V().count(). Instead, use graph.tx().config().option("allow_scan", true).open(); g.V().count(). (DSP-14482)

5.1.4 DSEFS changes and enhancements

• Improved error message for DSEFS shell commands. (DSP-14157)
• Improved error messages are passed to the DSEFS clients, including DSEFS shell, if error occurs while reading a file. (DSP-14371)
• Improved error message when Spark fails to connect to DSEFS server. (DSP-14388)
• HTTP communication logging level changed from DEBUG to TRACE to improve filtering. (DSP-14400)
• Improve DSEFS stability on large workloads: DSEFS is less likely to overload Java Cassandra driver and cause BusyPoolException. Fixed edge-cases that might cause StackOverflowException and DSEFS lockup. (DSP-14408)

5.1.4 DSE Search changes and enhancements

• Full validation on all schema fields might result in validation failures after upgrade. (DSP-6501)
  # All field definitions in the schema are validated and must be DSE Search compatible, even if the fields are not indexed, have docValues applied, or used for copy-field source.
  # Tune the schema before you upgrade. All field definitions in the schema are validated and must be DSE Search compatible, even if the fields are not indexed, have docValues applied, or used for copy-field source. With the tuned index, performance gains are especially recognized for unused large blobs.

5.1.4 Resolved issues

Resolved issues for:
• DataStax Enterprise core (page 92)
• DSE Analytics (page 92)
• DSEFS (page 92)
• DSE Graph (page 93)
• DSE Search (page 93)

5.1.4 DataStax Enterprise resolved issues

• Node does not start with unable to activate HistogramInfoPlugin message after upgrade to DSE 5.1. (DSP-13301)
• Apache HttpClient directory traversal through malformed URI. (DSP-13580)
• Token create, cancel, and renew security needs tightening. (DSP-14311)
• stress-tool does not output rows. (DSP-14494)

5.1.4 DSE Analytics resolved issues

• Session management in Hive metastore is broken. (DSP-12363)
• When an application is submitted by a user without submit permission, exception message does not identify problem. (DSP-13234)
• Spark shell not usable after standalone installation with services option. (DSP-14361)
• Port setting not respected in DseCassandraConnectionFactory. (DSP-14442)
• Spark Master/Worker Web UI should bind to RPC listen address and advertise RPC broadcast address by default. (DSP-14433)

5.1.4 DSEFS resolved issues
The service dse stop command does not wait for the process to be completely stopped. (DSP-14014)

DSEFS does not support symlink for data directories. (DSP-14110)

DSEFS fsck always prints number of blocks processed, even if file system is empty. (DSP-14235)

5.1.4 DSE Graph resolved issues

- Vertex index on id property keys doesn't work. (DSP-9208)
- Unnecessary INSERT and DELETE to dse_security.digest_tokens for every graph statement executed over native protocol. (DSP-13670)
- Streamline configuration for gremlin-console connection to cluster with Kerberos authentication enabled. (DSP-14164)
- DataFrames deletes do not leverage range or partition level tombstones. (DSP-14249)
- Vertices with custom IDs do not return ID components as properties (as in g.V().properties() or g.V().values() for OLTP, OLAP, and GraphFrames. (DSP-14262)
- DseResourceManager warning message when shutting down Spark+Graph nodes. (DSP-14276)
- Graph sandbox should have org.apache.tinkerpop.gremlin.structure.io whitelisted by default. (DSP-14540)

5.1.4 DSE Search resolved issues

- Allow dynamic multi-valued fields without a corresponding CQL column. (DSP-13277)
- Non-indexed frozen map column produces unexpected results without error message. (DSP-13997)
- Non-indexed field prevents data from being indexed. (DSP-14001)
- Single-pass CQL Solr queries cannot select some data types. (DSP-14022)
- Text field does not work for group by operations; unexpected docvalues type SORTED_SET error message for text fields. (DSP-14106)
- Parsing error on cleanup of Solr secondary index with empty string in partition ID. (DSP-14234)
- Solr indexing statistics are not collected for solr_index_stats_options. (DSP-14241)
- CPU layout assertions on startup should show in log file instead of stopping startup. (DSP-14281)
- Cannot turn tracing off after running queries with tracing on. (DSP-14439)
- Indexing wiki demo fails when solrslowlog is enabled. (DSP-14521)
- Search performance objects are not working. (DSP-14241)
- Memory leak during index encryption. (DSP-13826)

5.1.4 DataStax Enterprise known issues

- DataStax Enterprise will not run with Java 1.8u161 or later. (DSP-15277)
- Spark SQL applications with DSE authentication enabled will throw errors if the DSEFS scratch directory doesn't exist. (DSP-15276)
Spark SQL applications utilize a scratch directory located in DSEFS. Make sure the dsefs://tmp/hive directory exists and that it has 733 permissions. If dsefs://tmp/hive does not exist, it must be created by a role with superuser permissions. Create the scratch directory with proper permissions:

```
$ dse fs 'mkdir -p -m 733 /tmp/hive'
```

- Potential data loss for INSERTs with very large TTLs. TTL expiration timestamps are susceptible to the year 2038 problem. (DSP-15412)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. TTLs are considered "very large" when close to the maximum allowed value of 630720000 seconds (20 years), starting from 2018-01-19T03:14:06+00:00. As time progresses, the maximum supported TTL is gradually reduced as the maximum expiration date approaches. For instance, on 2028-01-19T03:14:06 with a TTL of 10 years is impacted. The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately.

**Warning:** Upgrade to DSE 5.1.7 or later and take required action to protect against overflow of local expiration time.

Hadoop libraries

Built-in Hadoop and Bring-Your-Own-Hadoop (BYOH) were deprecated in DataStax Enterprise (DSE) 5.0, and were removed in DSE 5.1. Hadoop removal from DSE 5.1 and later means that DSE does not allow for the startup of Hadoop services previously included in DSE, including MapReduce JobTracker and TaskTracker.

However, DSE has supported built-in Spark since DSE 4.5 and Bring-Your-Own-Spark (BYOS) since DSE 5.0, and that support continues today. Because Spark depends on certain Hadoop libraries on the server and the client, DSE continues to ship with Hadoop libraries that are required for running Spark and BYOS.

To view the included Hadoop libraries, see DataStax Enterprise 5.1.x third-party software.

dse.yaml

The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
</tbody>
</table>
Cassandra enhancements for DSE 5.1.4

DataStax Enterprise (DSE) 5.1.4 includes all changes from earlier DSE releases. These production-certified changes are enhancements to Apache Cassandra™ 3.11.0. (For Cassandra updates, see CHANGES.txt.)

- Handle limit correctly on tables with strict liveness (CASSANDRA-13883)
- AbstractTokenTreeBuilder#serializedSize returns wrong value when there is a single leaf and overflow collisions (CASSANDRA-13869)
- BTree.Builder memory leak (CASSANDRA-13754)
- Revert CASSANDRA-10368 of supporting non-pk column filtering due to correctness (CASSANDRA-13798)
- Fix cassandra-stress hang issues when an error during cluster connection happens (CASSANDRA-12938)
- Better bootstrap failure message when blocked by (potential) range movement (CASSANDRA-13744)
- "ignore" option is ignored in sstableloader (CASSANDRA-13721)
- Deadlock in AbstractCommitLogSegmentManager (CASSANDRA-13652)
- Duplicate the buffer before passing it to analyser in SASI operation (CASSANDRA-13512)
- Properly evict pstmts from prepared statements cache (CASSANDRA-13641)
- Fix support for SuperColumn tables (CASSANDRA-12373)
- Remove non-rpc-ready nodes from counter leader candidates (CASSANDRA-13043)
- Improve short read protection performance (CASSANDRA-13794)
- Fix sstable reader to support range-tombstone-marker for multi-slices (CASSANDRA-13787)
- Fix short read protection for tables with no clustering columns (CASSANDRA-13880)
- Make isBuilt volatile in PartitionUpdate (CASSANDRA-13619)
- Prevent integer overflow of timestamps in CellTest and RowsTest (CASSANDRA-13866)
- Fix counter application order in short read protection (CASSANDRA-12872)
- Don't block RepairJob execution on validation futures (CASSANDRA-13797)
- Wait for all management tasks to complete before shutting down CLSM (CASSANDRA-13123)
- INSERT statement fails when Tuple type is used as clustering column with default DESC order (CASSANDRA-13717)
- Fix pending view mutations handling and cleanup batchlog when there are local and remote paired mutations (CASSANDRA-13069)
- Improve config validation and documentation on overflow and NPE (CASSANDRA-13622)
- Range deletes in a CAS batch are ignored (CASSANDRA-13655)
• Avoid assertion error when IndexSummary > 2G (CASSANDRA-12014)
• Change repair midpoint logging for tiny ranges (CASSANDRA-13603)
• Better handle corrupt final commitlog segment (CASSANDRA-11995)
• StreamingHistogram is not thread safe (CASSANDRA-13756)
• Fix MV timestamp issues (CASSANDRA-11500)
• Better tolerate improperly formatted bcrypt hashes (CASSANDRA-13626)
• Fix race condition in read command serialization (CASSANDRA-13363)
• Fix AssertionError in short read protection (CASSANDRA-13747)
• Don't skip corrupted sstables on startup (CASSANDRA-13620)
• Fix the merging of cells with different user type versions (CASSANDRA-13776)
• Copy session properties on cqlsh.py do_login (CASSANDRA-13640)
• Potential AssertionError during ReadRepair of range tombstone and partition deletions (CASSANDRA-13719)
• Don't let stress write warmup data if n=0 (CASSANDRA-13773)
• Gossip thread slows down when using batch commit log (CASSANDRA-12966)
• Randomize batchlog endpoint selection with only 1 or 2 racks (CASSANDRA-12884)
• Fix digest calculation for counter cells (CASSANDRA-13750)
• Fix ColumnDefinition.cellValueType() for non-frozen collection and change SSTabledump to use type.toJSONString() (CASSANDRA-13573)
• Skip materialized view addition if the base table doesn't exist (CASSANDRA-13737)
• Drop table should remove corresponding entries in dropped_columns table (CASSANDRA-13730)
• Log warn message until legacy auth tables have been migrated (CASSANDRA-13371)
• Fix incorrect [2.1 -> 3.0] serialization of counter cells created in 2.0 (CASSANDRA-13691)
• Fix invalid writetime for null cells (CASSANDRA-13711)
• Fix ALTER TABLE statement to atomically propagate changes to the table and its MVs (CASSANDRA-12952)
• Fix Digest mismatch Exception if hints file has UnknownColumnFamily (CASSANDRA-13696)
• Fixed ambiguous output of nodetool tablestats command (CASSANDRA-13722)
• Purge tombstones created by expired cells (CASSANDRA-13643)
• Make concat work with iterators that have different subsets of columns (CASSANDRA-13482)
• Set test.runners based on cores and memory size (CASSANDRA-13078)
• Allow different NUMACTL_ARGS to be passed in (CASSANDRA-13557)
• Fix secondary index queries on COMPACT tables (CASSANDRA-13627)
• Nodetool listsnapshots output is missing a newline, if there are no snapshots (CASSANDRA-13568)
• sstabledump reports incorrect usage for argument order (CASSANDRA-13532)
• Safely handle empty buffers when outputting to JSON (CASSANDRA-13868)
• Copy session properties on cqlsh.py do_login (CASSANDRA-13847)
- Fix load over calculated issue in IndexSummaryRedistribution (CASSANDRA-13738)
- Fix compaction and flush exception not captured (CASSANDRA-13833)
- Uncaught exceptions in Netty pipeline (CASSANDRA-13649)
- Prevent integer overflow on exabyte filesystems (CASSANDRA-13067)
- Fix queries with LIMIT and filtering on clustering columns (CASSANDRA-11223)
- Fix potential NPE when resume bootstrap fails (CASSANDRA-13272)
- Fix toJSONString for the UDT, tuple and collection types (CASSANDRA-13592)
- Fix nested Tuples/UDTs validation (CASSANDRA-13646)
- Clone HeartBeatState when building gossip messages. Make its generation/version volatile (CASSANDRA-13700)
- Allow native function calls in CQLSSTableWriter (CASSANDRA-12606)
- Replace string comparison with regex/number checks in MessagingService test (CASSANDRA-13216)
- Fix formatting of duration columns in CQLSH (CASSANDRA-13549)
- Fix the problem with duplicated rows when using paging with SASI (CASSANDRA-13302)
- Allow CONTAINS statements filtering on the partition key and it’s parts (CASSANDRA-13275)
- Fall back to even ranges calculation in clusters with vnodes when tokens are distributed unevenly (CASSANDRA-13229)
- Fix duration type validation to prevent overflow (CASSANDRA-13218)
- Forbid unsupported creation of SASI indexes over partition key columns (CASSANDRA-13228)
- Reject multiple values for a key in CQL grammar. (CASSANDRA-13369)
- UDA fails without input rows (CASSANDRA-13399)
- Fix compaction-stress by using daemonInitialization (CASSANDRA-13188)
- V5 protocol flags decoding broken (CASSANDRA-13443)
- Use write lock not read lock for removing sstables from compaction strategies. (CASSANDRA-13422)
- Use corePoolSize equal to maxPoolSize in JMXEnabledThreadPoolExecutors (CASSANDRA-13329)
- Avoid rebuilding SASI indexes containing no values (CASSANDRA-12962)
- Add charset to Analysers input stream (CASSANDRA-13151)
- Delete illegal character from StandardTokenizerImpl.jflex (CASSANDRA-13417)
- Fix cqlsh automatic protocol downgrade regression (CASSANDRA-13307)
- Tracing payload not passed from QueryMessage to tracing session (CASSANDRA-12835)
- Ensure int overflow doesn't occur when calculating large partition warning size (CASSANDRA-13172)
- Ensure consistent view of partition columns between coordinator and replica in ColumnFilter (CASSANDRA-13004)
- Failed unregistering mbean during drop keyspace (CASSANDRA-13346)
- nodetool scrub/cleanup/upgradesstables exit code is wrong (CASSANDRA-13542)
• Fix the reported number of sstable data files accessed per read (CASSANDRA-13120)
• Fix schema digest mismatch during rolling upgrades from versions before 3.0.12 (CASSANDRA-13559)
• Upgrade JNA version to 4.4.0 (CASSANDRA-13072)
• Interned ColumnIdentifiers should use minimal ByteBuffers (CASSANDRA-13533)
• ReverseIndexedReader may drop rows during 2.1 to 3.0 upgrade (CASSANDRA-13525)
• Fix repair process violating start/end token limits for small ranges (CASSANDRA-13052)
• Add storage port options to sstableloader (CASSANDRA-13518)
• Properly handle quoted index names in cqlsh DESCRIBE output (CASSANDRA-12847)
• Avoid reading static row twice from old format sstables (CASSANDRA-13236)
• Fix NPE in StorageService.excise() (CASSANDRA-13163)
• Expire OutboundTcpConnection messages by a single Thread (CASSANDRA-13265)
• Fail repair if insufficient responses received (CASSANDRA-13397)
• Fix SSTableLoader fail when the loaded table contains dropped columns (CASSANDRA-13276)
• Avoid name clashes in CassandraIndexTest (CASSANDRA-13427)
• Handling partially written hint files (CASSANDRA-12728)
• Interrupt replaying hints on decommission (CASSANDRA-13308)
• Handling partially written hint files (CASSANDRA-12728)
• Fix NPE issue in StorageService (CASSANDRA-13060)
• Make reading of range tombstones more reliable (CASSANDRA-12811)
• Fix startup problems due to schema tables not completely flushed (CASSANDRA-12213)
• Fix view builder bug that can filter out data on restart (CASSANDRA-13405)
• Fix 2i page size calculation when there are no regular columns (CASSANDRA-13400)
• Fix the conversion of 2.X expired rows without regular column data (CASSANDRA-13395)
• Fix hint delivery when using ext+internal IPs with prefer_local enabled (CASSANDRA-13020)
• Nodetool upgradesstables/scrub/compact ignores system tables (CASSANDRA-13410)
• Fix schema version calculation for rolling upgrades (CASSANDRA-13441)
• Nodes started with join_ring=False should be able to serve requests when authentication is enabled (CASSANDRA-11381)
• cqlsh COPY FROM: increment error count only for failures, not for attempts (CASSANDRA-13209)
• Avoid starting gossiper in RemoveTest (CASSANDRA-13407)
• Fix weightedSize() for row-cache reported by JMX and NodeTool (CASSANDRA-13393)
• Fix JVM metric names (CASSANDRA-13103)
• Coalescing strategy sleeps too much (CASSANDRA-13090)
• Fix 2ndary index queries on partition keys for tables with static columns (CASSANDRA-13147)
General upgrade advice for DSE 5.1.4

Carefully review all planning and upgrade documentation in the Upgrading DataStax Enterprise guide. The following provides general upgrade information:

GENERAL UPGRADE ADVICE FOR ANY VERSION
========================================

Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to ALWAYS snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

DSE 5.1.3
---------

Upgrading
--------

- 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 table system_auth.resource_role_permissions_index is no longer used and should be dropped
  after all nodes are on 5.1.3. Note that upgrades from DSE 5.0 series
  since 5.0.10 to DSE versions before 5.1.3 are not recommended.
- Full repairs are now default if no option is specified on nodetool repair, unless
  incremental repair was already run on the table/keyspace being repaired, to maintain
  backward compatibility. Incremental repair may be run on new tables
  by using the --inc option.
- Full repairs will no longer run repair unless the --run-anticompaction option is specified
- Incremental repairs are no longer supported on tables with materialized views or CDC until its limitations are addressed. An incremental repair triggered on a base table or materialized view run a full repair instead. See CASSANDRA-12888 for details.

Materialized Views (only when upgrading from DSE 5.1.1 or 5.1.2 or any version lower than DSE 5.0.10)

- 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 to not 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.

3.11.0

Upgrading

- 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
DataStax Enterprise 5.1 release notes

may still use this feature with a startup flag: 
- \texttt{Dcassandra.mv.allow_filtering_nonkey_columns_unsafe=true}

- ALTER TABLE (ADD/DROP COLUMN) operations concurrent with a read might result into data corruption (see CASSANDRA-13004 for more details).Fixing this bug required a messaging protocol version bump. By default, Cassandra 3.11 will use 3014 version for messaging.

Since Schema Migrations rely on the exact messaging protocol version match between nodes, if you need schema changes during the upgrade process, you have to start your nodes with `\texttt{Dcassandra.force_3_0_protocol_version=true}` first, in order to temporarily force a backwards compatible protocol.

After the whole cluster is upgraded to 3.11, do a rolling restart of the cluster without setting that flag.

3.11 nodes with and without the flag set will be able to do schema migrations with other 3.x and 3.0.x releases.

While running the cluster with the flag set to true on 3.11 (in compatibility mode), avoid adding or removing any columns to/from existing tables.

If your cluster can do without schema migrations during the upgrade time, just start the cluster normally without setting aforementioned flag.

If you are upgrading from 3.0.14+ (of 3.0.x branch), you do not have to set an flag while upgrading to ensure schema migrations.

- 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).
New features
------------
- New `DurationType` (cql duration). See CASSANDRA-11873
- Runtime modification of concurrent_compactors is now available via nodetool
- 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.
- Compaction can now take into account overlapping tables that don't take part in the compaction to look for deleted or overwritten data in the compacted tables.
  Then such data is found, it can be safely discarded, which in turn should enable the removal of tombstones over that data.

  The behavior can be engaged in two ways:
  - as a "nodetool garbagecollect -g CELL/ROW" operation, which applies single-table compaction on all sstables to discard deleted data in one step.
  - as a "provide_overlapping_tombstones:CELL/ROW/NONE" compaction strategy flag, which uses overlapping tables as a source of deletions/overwrites during all compactions.

  The argument specifies the granularity at which deleted data is to be found:
  - If ROW is specified, only whole deleted rows (or sets of rows) will be discarded.
  - If CELL is specified, any columns whose value is overwritten or deleted will also be discarded.
  - NONE (default) specifies the old behavior, overlapping tables are not used to decide when to discard data.
Which option to use depends on your workload, both ROW and CELL increase the
disk load on compaction (especially with the size-tiered compaction strategy),
with CELL being more resource-intensive. Both should lead to better read
performance if deleting rows (resp. overwriting or deleting cells)
is common.
- 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.

Upgrading
---------
- 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 compatiblity 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 appropiate 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).

3.8
***

New features
------------
- 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.
- 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. 

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.

Upgrading
---------
- 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).

Deprecation
------------
- 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.

3.7
===

Upgrading
---------
- 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 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.
New features
------------
- 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.
New features
-------------
- 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
  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.

Upgrading
--------
- Nothing specific to 3.4 but please see previous versions upgrading section,
  especially if you are upgrading from 2.2.

Deprecation
-----------
- The mbean interfaces
  org.apache.cassandra.authPermissionsCacheMBean and
  org.apache.cassandra.authRolesCacheMBean 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.

3.2
===

New features
-------------
- 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.
Upgrading
---------
- 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.

Deprecation
-----------
- The Thrift API is deprecated and will be removed in Cassandra 4.0.

3.1
=====

Upgrading
---------
- 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 inconditionally for that method.
- GC logging is now enabled by default (you can disable it in the jvm.options file if you prefer).

3.0
===

New features
-------------
- EACH_QUORUM is now a supported consistency level for read requests.
- Support for IN restrictions on any partition key component or clustering key as well as support for EQ and IN multicolumn restrictions has been added to UPDATE and DELETE statement.
- Support for single-column and multi-column slice restrictions (>, >=, <= and <) has been added to DELETE statements
- nodetool rebuild_index accepts the index argument without the redundant table name
- Materialized Views, which allow for server-side denormalization, is now available. Materialized views provide an alternative to secondary indexes
for non-primary key queries, and perform much better for indexing high cardinality columns.

See http://www.datastax.com/dev/blog/new-in-cassandra-3-0-materialized-views

- Hinted handoff has been completely rewritten. Hints are now stored in flat files, with less overhead for storage and more efficient dispatch. See CASSANDRA-6230 for full details.

- Option to not purge unrepaired tombstones. To avoid users having data resurrected if repair has not been run within gc_grace_seconds, an option has been added to only allow tombstones from repaired sstables to be purged. To enable, set the compaction option 'only_purge_repaired_tombstones':true but keep in mind that if you do not run repair for a long time, you will keep all tombstones around which can cause other problems.

- Enabled warning on GC taking longer than 1000ms. See cassandra.yaml:gc_warn_threshold_in_ms

Upgrading
---------

- Clients must use the native protocol version 3 when upgrading from 2.2.X as the native protocol version 4 is not compatible between 2.2.X and 3.Y. See https://www.mail-archive.com/user@cassandra.apache.org/msg45381.html for details.

- A new argument of type InetAddress has been added to IAuthenticator::newSaslNegotiator, representing the IP address of the client attempting authentication. It will be a breaking change for any custom implementations.

- token-generator tool has been removed.

- Upgrade to 3.0 is supported from Cassandra 2.1 versions greater or equal to 2.1.9, or Cassandra 2.2 versions greater or equal to 2.2.2. Upgrade from Cassandra 2.0 and older versions is not supported.

- The 'memtable_allocation_type: offheap_objects' option has been removed. It should be re-introduced in a future release and you can follow CASSANDRA-9472 to know more.

- Configuration parameter memory_allocator in cassandra.yaml has been removed.

- The native protocol versions 1 and 2 are not supported anymore.

- Max mutation size is now configurable via max_mutation_size_in_kb setting in cassandra.yaml; the default is half the size commitlog_segment_size_in_mb * 1024.

- 3.0 requires Java 8u40 or later.
- Garbage collection options were moved from cassandra-env to jvm.options file.
- New transaction log files have been introduced to replace the compactions_in_progress system table, temporary file markers (tmp and tmplink) and sstable ancestors.
  Therefore, compaction metadata no longer contains ancestors.
- Transaction log files list sstable descriptors involved in compactions and other operations such as flushing and streaming. Use the sstableutil tool to list any sstable files currently involved in operations not yet completed, which previously would have been marked as temporary.
  A transaction log file contains one sstable per line, with the prefix "add:" or "remove:".
  They also contain a special line "commit", only inserted at the end when the transaction is committed. On startup we use these files to cleanup any partial transactions that were in progress when the process exited. If the commit line is found, we keep new sstables (those with the "add" prefix) and delete the old sstables (those with the "remove" prefix), vice-versa if the commit line is missing. Should you lose or delete these log files, both old and new sstable files will be kept as live files, which will result in duplicated sstables. These files are protected by incremental checksums so you should not manually edit them. When restoring a full backup or moving sstable files, you should clean-up any left over transactions and their temporary files first. You can use this command:
  ```
  ===> sstableutil -c ks table
  ```
  See CASSANDRA-7066 for full details.
- New write stages have been added for batchlog and materialized view mutations
  you can set their size in cassandra.yaml
- User defined functions are now executed in a sandbox.
  To use UDFs and UDAs, you have to enable them in cassandra.yaml.
- New SSTable version 'la' with improved bloom-filter false-positive handling compared to previous version 'ka' used in 2.2 and 2.1. Running sstableupgrade is not necessary but recommended.
- Before upgrading to 3.0, make sure that your cluster is in complete agreement (schema versions outputted by `nodetool describecluster` are all the same).
- Schema metadata is now stored in the new `system_schema` keyspace, and legacy `system.schema_*` tables are now gone; see CASSANDRA-6717 for details.
- Pig's support has been removed.
- Hadoop BulkOutputFormat and BulkRecordWriter have been removed; use CqlBulkOutputFormat and CqlBulkRecordWriter instead.
- Hadoop ColumnFamilyInputFormat and ColumnFamilyOutputFormat have been removed;
  use CqlInputFormat and CqlOutputFormat instead.
- Hadoop ColumnFamilyRecordReader and ColumnFamilyRecordWriter have been removed;
  use CqlRecordReader and CqlRecordWriter instead.
- hinted_handoff_enabled in cassandra.yaml no longer supports a list of data centers.
  To specify a list of excluded data centers when
  hinted_handoff_enabled is set to true,
  use hinted_handoff_disabled_datacenters, see CASSANDRA-9035 for details.
- The `sstable_compression` and `chunk_length_kb` compression options have been deprecated.
  The new options are `class` and `chunk_length_in_kb`. Disabling
  compression should now
  be done by setting the new option `enabled` to `false`.
- The compression option `crc_check_chance` became a top-level table
  option, but is currently
  enforced only against tables with enabled compression.
- Only map syntax is now allowed for caching options. ALL/NONE/
  KEYS_ONLY/ROWS_ONLY syntax
  has been deprecated since 2.1.0 and is being removed in 3.0.0.
- The 'index_interval' option for 'CREATE TABLE' statements, which has been deprecated
  since 2.1 and replaced with the 'min_index_interval' and
  'max_index_interval' options,
  has now been removed.
- Batchlog entries are now stored in a new table - system.batches. The old one has been deprecated.
- JMX methods set/getCompactionStrategyClass have been removed, use
  set/getCompactionParameters or set/getCompactionParametersJson
  instead.
- SizeTieredCompactionStrategy parameter cold_reads_to_omit has been removed.
  The secondary index API has been comprehensively reworked. This will be a breaking
  change for any custom index implementations, which should now look to implement
  the new org.apache.cassandra.index.Index interface. New syntax has been added to create
  and query row-based indexes, which are not explicitly linked to a single column in the
  base table.

2.2.4
=====

Deprecation
-----------
- Pig support has been deprecated, and will be removed in 3.0. Please see CASSANDRA-10542 for more details.
- Configuration parameter memory_allocator in cassandra.yaml has been deprecated and will be removed in 3.0.0. As mentioned below for 2.2.0, jemalloc is automatically preloaded on Unix platforms.

Operations
----------
- Switching data center or racks is no longer an allowed operation on a node which has data. Instead, the node will need to be decommissioned and rebootstrapped. If moving from the SimpleSnitch, make sure that the data center and rack containing all current nodes is named "datacenter1" and "rack1". To override this behaviour use -Dcassandra.ignore_rack=true and/or -Dcassandra.ignore_dc=true.
- Reloading the configuration file of GossipingPropertyFileSnitch has been disabled.

Upgrading
---------
- The default for the inter-DC stream throughput setting (inter_dc_stream_throughput_outbound_megabits_per_sec in cassandra.yaml) is the same than the one for intra-DC one (200Mbps) instead of being unlimited. Having it unlimited was never intended and was a bug.

New features
------------
- Time windows in DTCS are now limited to 1 day by default to be able to handle bootstrap and repair in a better way. To get the old behaviour, increase max_window_size_seconds.
- DTCS option max_sstable_age_days is now deprecated and defaults to 1000 days.
- Native protocol server now allows both SSL and non-SSL connections on the same port.

2.2.3
=====
Upgrading
--------
- Nothing specific to this release, but please see 2.2 if you are upgrading from a previous version.
2.2.2
=====

Changed Defaults
----------------
- commitlog_total_space_in_mb will use the smaller of 8192, and 1/4 of the total space of the commitlog volume. (Before: always used 8192)
- The following INFO logs were reduced to DEBUG level and will now show on debug.log instead of system.log:
  - Memtable flushing actions
  - Commit log replayed files
  - Compacted sstables
  - SSTable opening (SSTableReader)

New features
------------
- Custom QueryHandlers can retrieve the column specifications for the bound variables from QueryOptions by using the hasColumnSpecifications() and getColumnSpecifications() methods.
- A new default asynchronous log appender debug.log was created in addition to the system.log appender in order to provide more detailed log debugging.
  In order to disable debug logging, you must comment-out the ASYNCDDEBUGLOG appender on conf/logback.xml. See CASSANDRA-10241 for more information.

2.2.1
=====

New features
------------
- COUNT(*) and COUNT(1) can be selected with other columns or functions

2.2
===

Upgrading
--------
- The authentication & authorization subsystems have been redesigned to support role based access control (RBAC), resulting in a change to the schema of the system_auth keyspace. See below for more detail.
For systems already using the internal auth implementations, the process for converting existing data during a rolling upgrade is straightforward.

As each node is restarted, it will attempt to convert any data in the legacy tables into the new schema. Until enough nodes to satisfy the replication strategy for the system_auth keyspace are upgraded and so have the new schema, this conversion will fail with the failure being reported in the system log.

During the upgrade, Cassandra's internal auth classes will continue to use the legacy tables, so clients experience no disruption. Issuing DCL statements during an upgrade is not supported.

Once all nodes are upgraded, an operator with superuser privileges should drop the legacy tables, system_auth.users, system_auth.credentials and system_auth.permissions. Doing so will prompt Cassandra to switch over to the new tables without requiring any further intervention. While the legacy tables are present a restarted node will re-run the data conversion and report the outcome so that operators can verify that it is safe to drop them.

New features
-------------
- The LIMIT clause applies now only to the number of rows returned to the user, not to the number of row queried. By consequence, queries using aggregates will not be impacted by the LIMIT clause anymore.
- Very large batches will now be rejected (defaults to 50kb). This can be customized by modifying batch_size_fail_threshold_in_kb.
- Selecting columns, scalar functions, UDT fields, writetime or ttl together with aggregated is now possible. The value returned for the columns, scalar functions, UDT fields, writetime and ttl will be the ones for the first row matching the query.
- Windows is now a supported platform. Powershell execution for startup scripts is highly recommended and can be enabled via an administrator command-prompt with: 'powershell set-executionpolicy unrestricted'
- It is now possible to do major compactions when using leveled compaction. Doing that will take all sstables and compact them out in levels. The levels will be non overlapping so doing this will still not be something
you want to do very often since it might cause more compactions for
a while.

It is also possible to split output when doing a major compaction
with
STCS - files will be split in sizes 50%, 25%, 12.5% etc of the total
size.
This might be a bit better than old major compactions which created
one big
file on disk.
- A new tool has been added bin/sstableverify that checks for errors/

bitrot

in all sstables. Unlike scrub, this is a non-invasive tool.
- Authentication & Authorization APIs have been updated to introduce
roles. Roles and Permissions granted to them are inherited,
supporting
role based access control. The role concept supercedes that of users
and CQL constructs such as CREATE USER are deprecated but retained
for
compatibility. The requirement to explicitly create Roles in
Cassandra
even when auth is handled by an external system has been removed, so
authentication & authorization can be delegated to such systems in
their
entirety.
- In addition to the above, Roles are also first class resources and
can be the
subject of permissions. Users (roles) can now be granted permissions
on other
roles, including CREATE, ALTER, DROP & AUTHORIZE, which removesthe
need for
superuser privileges in order to perform user/role management
operations.
- Creators of database resources (Keyspaces, Tables, Roles) are now
automatically
granted all permissions on them (if the IAuthorizer implementation
supports
this).
- SSTable file name is changed. Now you don't have Keyspace/CF name
in file name. Also, secondary index has its own directory under
parent's
directory.
- Support for user-defined functions and user-defined aggregates have
been added to CQL.

************************************************************************
IMPORTANT NOTE: user-defined functions can be used to execute
arbitrary and possibly evil code in Cassandra 2.2, and are
therefore disabled by default. To enable UDFs edit
cassandra.yaml and set enable_user_defined_functions to true.
CASSANDRA-9402 will add a security manager for UDFs in Cassandra
3.0. This will inherently be backwards-incompatible with any 2.2
UDF that perform insecure operations such as opening a socket or
writing to the filesystem.
DataStax Enterprise 5.1 release notes

- Row-cache is now fully off-heap.
- jemalloc is now automatically preloaded and used on Linux and OS-X if installed.
- Please ensure on Unix platforms that there is no libjnadispath.so installed which is accessible by Cassandra. Old versions of libjna packages (< 4.0.0) will cause problems - e.g. Debian Wheezy contains libjna version 3.2.x.
- The node now keeps up when streaming is failed during bootstrapping. You can use new `nodetool bootstrap resume` command to continue streaming after resolving an issue.
- Protocol version 4 specifies that bind variables do not require having a value when executing a statement. Bind variables without a value are called 'unset'. The 'unset' bind variable is serialized as the int value '-2' without following bytes.
  In an EXECUTE or BATCH request an unset bind value does not modify the value and does not create a tombstone, an unset bind ttl is treated as 'unlimited', an unset bind timestamp is treated as 'now', an unset bind counter operation does not change the counter value.
  Unset tuple field, UDT field and map key are not allowed.
  In a QUERY request an unset limit is treated as 'unlimited'. Unset WHERE clauses with unset partition column, clustering column or index column are not allowed.
- New `ByteType` (cql tinyint). 1-byte signed integer
- New `ShortType` (cql smallint). 2-byte signed integer
- New `SimpleDateType` (cql date). 4-byte unsigned integer
- New `TimeType` (cql time). 8-byte long
  - The `toDate(timeuuid)`, `toTimestamp(timeuuid)` and `toUnixTimestamp(timeuuid)` functions have been added to allow to convert from timeuuid into date type, timestamp type and bigint raw value.
  The functions `unixTimestampOf(timeuuid)` and `dateOf(timeuuid)` have been deprecated.
  - The `toDate(timestamp)` and `toUnixTimestamp(timestamp)` functions have been added to allow to convert from timestamp into date type and bigint raw value.
  - The `toTimestamp(date)` and `toUnixTimestamp(date)` functions have been added to allow to convert from date into timestamp type and bigint raw value.
- SizeTieredCompactionStrategy parameter `cold_reads_to_omit` has been removed.
- The default JVM flag `-XX:+PerfDisableSharedMem` will cause the following tools JVM to stop working: jps, jstack, jinfo, jmc, jcmd as well as 3rd party tools like Jolokia.
If you wish to use these tools you can comment this flag out in cassandra-env.{sh,ps1}

Upgrading
---------
- Thrift rpc is no longer being started by default. Set `start_rpc` parameter to `true` to enable it.
- Pig's CqlStorage has been removed, use CqlNativeStorage instead
- Pig's CassandraStorage has been deprecated. CassandraStorage should only be used against tables created via thrift. Use CqlNativeStorage for all other tables.
- IAuthenticator been updated to remove responsibility for user/role maintenance and is now solely responsible for validating credentials,
  This is primarily done via SASL, though an optional method exists for systems which need support for the Thrift login() method.
- IRoleManager interface has been added which takes over the maintenance functions from IAuthenticator. IAuthorizer is mainly unchanged. Auth data in systems using the stock internal implementations PasswordAuthenticator & CassandraAuthorizer will be automatically converted during upgrade, with minimal operator intervention required. Custom implementations will require modification, though these can be used in conjunction with the stock CassandraRoleManager so providing an IRoleManager implementation should not usually be necessary.
- Fat client support has been removed since we have push notifications to clients
  - cassandra-cli has been removed. Please use cqlsh instead.
  - YamlFileNetworkTopologySnitch has been removed; switch to GossipingPropertyFileSnitch instead.
  - CQL2 has been removed entirely in this release (previously deprecated in 2.0.0). Please switch to CQL3 if you haven't already done so.
- The results of CQL3 queries containing an IN restriction will be ordered in the normal order and not anymore in the order in which the column values were specified in the IN restriction.
- Some secondary index queries with restrictions on non-indexed clustering columns were not requiring ALLOW FILTERING as they should. This has been fixed, and those queries now require ALLOW FILTERING (see CASSANDRA-8418 for details).
- The SSTableSimpleWriter and SSTableSimpleUnsortedWriter classes have been
deprecated and will be removed in the next major Cassandra release. You should use the CQLSSTableWriter class instead.

- The sstable2json and json2sstable tools have been deprecated and will be removed in the next major Cassandra release. See CASSANDRA-9618 (https://issues.apache.org/jira/browse/CASSANDRA-9618) for details.
- nodetool enablehandoff will no longer support a list of data centers starting with the next major release. Two new commands will be added, enablehintsfordc and disablehintsfordc, to exclude data centers from using hinted handoff when the global status is enabled.

In cassandra.yaml, hinted_handoff_enabled will no longer support a list of data centers starting with the next major release. A new setting will be added, hinted_handoff_disabled_datacenters, to exclude data centers when the global status is enabled, see CASSANDRA-9035 for details.

2.1.13
======

New features
-------------
- New options for cqlsh COPY FROM and COPY TO, see CASSANDRA-9303 for details.

2.1.10
======

New features
-------------
- The syntax TRUNCATE TABLE X is now accepted as an alias for TRUNCATE X

2.1.9
=====

Upgrading
---------
- cqlsh will now display timestamps with a UTC timezone. Previously, timestamps were displayed with the local timezone.
- Commit log files are no longer recycled by default, due to negative performance implications. This can be enabled again with the commitlog_segment_recycling option in your cassandra.yaml
- JMX methods set/getCompactionStrategyClass have been deprecated, use set/getCompactionParameters/set/getCompactionParametersJson instead

2.1.8
=====
Upgrading
--------
- Nothing specific to this release, but please see 2.1 if you are upgrading from a previous version.

2.1.7
-----
release.

2.1.2
=====

Upgrading
---------
- Nothing specific to this release, but please see 2.1 if you are upgrading from a previous version.

2.1.1
=====

Upgrading
---------
- Nothing specific to this release, but please see 2.1 if you are upgrading from a previous version.

New features
------------
- Netty support for epoll on Linux is now enabled. If for some reason you want to disable it, pass the following system property:
  -Dcassandra.native.epoll.enabled=false

2.1
===

New features
------------
- Default data and log locations have changed. If not set in cassandra.yaml, the data file directory, commitlog directory, and saved caches directory will default to $CASSANDRA_HOME/data/
  - data,
  - $CASSANDRA_HOME/data/commitlog, and $CASSANDRA_HOME/data/
  - saved_caches,
  - respectively. The log directory now defaults to $CASSANDRA_HOME/logs.
  - If not set, $CASSANDRA_HOME, defaults to the top-level directory of the installation.
  - Note that this should only affect source checkouts and tarballs.
    - Deb and RPM packages will continue to use /var/lib/cassandra and /var/log/cassandra in cassandra.yaml.
  - SSTable data directory name is slightly changed. Each directory will have hex string appended after CF name, e.g.
    - ks/cf-5be396077b811e3a3ab9dc4b9ac088d/
  - This hex string part represents unique ColumnFamily ID.
  - Note that existing directories are used as is, so only newly created directories after upgrade have new directory name format.
  - Saved key cache files also have ColumnFamily ID in their file name.
- It is now possible to do incremental repairs, sstables that have been repaired are marked with a timestamp and not included in the next repair session. Use nodetool repair -par -inc to use this feature. A tool to manually mark/unmark sstables as repaired is available in tools/bin/sstablerepairedset. This is particularly important when using LCS, or any data not repaired in your first incremental repair will be put back in L0.
- Bootstrapping now ensures that range movements are consistent, meaning the data for the new node is taken from the node that is no longer a responsible for that range of keys. If you want the old behavior (due to a lost node perhaps) you can set the following property (-Dcassandra.consistent.rangemovement=false)
- It is now possible to use quoted identifiers in triggers' names. WARNING: if you previously used triggers with capital letters in their names, then you must quote them from now on.
- Improved stress tool (http://goo.gl/0TNqiQ)
- Incremental replacement of compacted SSTables (http://goo.gl/JfDBGW)
- The row cache can now cache only the head of partitions (http://goo.gl/6JpH6)
- Off-heap memtables (http://goo.gl/Y7znJ)
- CQL improvements and additions: User-defined types, tuple types, secondary indexing of collections, ... (http://goo.gl/kQl7GW)

Upgrading
---------
- commitlog_sync_batch_window_in_ms behavior has changed from the maximum time to wait between fsync to the minimum time. We are working on making this more user-friendly (see CASSANDRA-9533) but in the meantime, this means 2.1 needs a much smaller batch window to keep writer threads from starving. The suggested default is now 2ms.
- Rolling upgrades from anything pre-2.0.7 is not supported. Furthermore
pre-2.0 sstables are not supported. This means that before upgrading a node on 2.1, this node must be started on 2.0 and 'nodetool upgradessstables' must be run (and this even in the case of not-rolling upgrades).
- For size-tiered compaction users, Cassandra now defaults to ignoring the coldest 5% of sstables. This can be customized with the cold_reads_to_omit compaction option; 0.0 omits nothing (the old behavior) and 1.0 omits everything.
- Multithreaded compaction has been removed.
- Counters implementation has been changed, replaced by a safer one with less caveats, but different performance characteristics. You might have to change your data model to accomodate the new implementation. (See https://issues.apache.org/jira/browse/CASSANDRA-6504 and the...
blog post at http://goo.gl/qj8iQl for details).
- (per-table) index_interval parameter has been replaced with
  min_index_interval and max_index_interval parameters.
  index_interval
  has been deprecated.
- support for supercolumns has been removed from json2sstable

2.0.11
======

Upgrading
---------
- Nothing specific to this release, but refer to previous entries if
  you
  are upgrading from a previous version.

New features
------------
- DateTieredCompactionStrategy added, optimized for time series data
  and groups
  data that is written closely in time (CASSANDRA-6602 for details).
  Consider
  this experimental for now.

2.0.10
======

New features
------------
- CqlPaginRecordReader and CqlPagingInputFormat have both been
  removed.
  Use CqlInputFormat instead.
- If you are using Leveled Compaction, you can now disable doing
  size-tiered
  compaction in L0 by starting Cassandra with
  Dcassandra.disable_stcs_in_l0
  (see CASSANDRA-6621 for details).
- Shuffle and taketoken have been removed. For clusters that choose
  to
  upgrade to vnodes, creating a new datacenter with vnodes and
  migrating is
  recommended. See http://goo.gl/Sna2S1 for further information.

2.0.9
=====

Upgrading
---------
- Default values for read_repair_chance and local_read_repair_chance
  have been
swapped. Namely, default read_repair_chance is now set to 0.0, and default
local_read_repair_chance to 0.1.
- Queries selecting only CQL static columns were (mistakenly) not returning one
result per row in the partition. This has been fixed and a SELECT DISTINCT
can be used when only the static column of a partition needs to be fetch
without fetching the whole partition. But if you use static columns, please
make sure this won't affect you (see CASSANDRA-7305 for details).

2.0.8
=====

New features
-------------
- New snitches have been used for users of Google Compute Engine and of
  Cloudstack.

Upgrading
---------
- Nothing specific to this release, but please see 2.0.7 if you are upgrading
  from a previous version.

2.0.7
=====

Upgrading
---------
- Nothing specific to this release, but please see 2.0.6 if you are upgrading
  from a previous version.

2.0.6
=====

New features
-------------
- CQL now support static columns, allows to batch multiple conditional updates
  and has a new syntax for slicing over multiple clustering columns
  (http://goo.gl/B6gz4j).
- Repair can be restricted to a set of nodes using the -hosts option in nodetool.
- A new 'nodetool taketoken' command relocate tokens with vnodes.
- Hinted handoff can be enabled only for some data-centers (see hinted_handoff_enabled in cassandra.yaml)
Upgrading
---------
- Nothing specific to this release, but please see 2.0.5 if you are upgrading from a previous version.

2.0.5 =====

New features
------------
- Batchlog replay can be, and is throttled by default now. See batchlog_replay_throttle_in_kb setting in cassandra.yaml.
- Scrub can now optionally skip corrupt counter partitions. Please note that this will lead to the loss of all the counter updates in the skipped partition. See the --skip-corrupted option.

Upgrading
---------
- If your cluster began on a version before 1.2, check that your secondary index SSTables are on version 'ic' before upgrading. If not, run 'nodetool upgradesstables' if on 1.2.14 or later, or run 'nodetool upgradesstables ks cf' with the keyspace and secondary index named explicitly otherwise. If you don't do this and upgrade to 2.0.x and it refuses to start because of 'hf' version files in the secondary index, you will need to delete/move them out of the way and recreate the index when 2.0.x starts.

2.0.3 =====

New features
------------
- It's now possible to configure the maximum allowed size of the native protocol frames (native_transport_max_frame_size_in_mb in the yaml file).

Upgrading
---------
- NaN and Infinity are new valid floating point constants in CQL3 and are now reserved keywords. In the unlikely case you were using one of them as an identifier (for a
column, a keyspace or a table), you will now have to double-quote them (see http://cassandra.apache.org/doc/cql3/CQL.html#identifiers for "quoted identifiers").

- The IEndpointStateChangeSubscriber has a new method, beforeChange, that any custom implementations using the class will need to implement.

2.0.2
=====

New features
-------------

- Speculative retry defaults to 99th percentile (See blog post at http://www.datastax.com/dev/blog/rapid-read-protection-in-cassandra-2-0-2)
- Configurable metrics reporting (see conf/metrics-reporter-config-sample.yaml)
- Compaction history and stats are now saved to system keyspace (system.compaction_history table). You can access history via new 'nodetool compactionhistory' command or CQL.

Upgrading
---------

- Nodetool defaults to Sequential mode for repair operations

2.0.1
=====

Upgrading
---------

- The default memtable allocation has changed from 1/3 of heap to 1/4 of heap. Also, default (single-partition) read and write timeouts have been reduced from 10s to 5s and 2s, respectively.

2.0.0
=====

Upgrading
---------

- Java 7 is now *required*!
- Upgrading is ONLY supported from Cassandra 1.2.9 or later. This goes for sstable compatibility as well as network. When upgrading from an earlier release, upgrade to 1.2.9 first and run upgradesstables before proceeding to 2.0.
- CAS and new features in CQL such as DROP COLUMN assume that cell timestamps are microseconds-since-epoch. Do not use these features if you are using client-specified timestamps with some other source.
- Replication and strategy options do not accept unknown options anymore.
This was already the case for CQL3 in 1.2 but this is now the case for thrift too.
- auto_bootstrap of a single-token node with no initial_token will now pick a random token instead of bisecting an existing token range. We recommend upgrading to vnodes; failing that, we recommend specifying initial_token.
- reduce_cache_sizes_at, reduce_cache_capacity_to, and flush_largest_memtables_at options have been removed from cassandra.yaml.
  - CacheServiceMBean.reduceCacheSizes() has been removed. Use CacheServiceMBean.set{Key,Row}CacheCapacityInMB() instead.
  - authority option in cassandra.yaml has been deprecated since 1.2.0, but it has been completely removed in 2.0. Please use 'authorizer' option.
  - ASSUME command has been removed from cqlsh. Use CQL3 blobAsType() and typeAsBlob() conversion functions instead. See https://cassandra.apache.org/doc/cql3/CQL.html#blobFun for details.
  - Inputting blobs as string constants is now fully deprecated in favor of blob constants. Make sure to update your applications to use the new syntax while you are still on 1.2 (which supports both string and blob constants for blob input) before upgrading to 2.0.
- index_interval is now moved to ColumnFamily property. You can change value with ALTER TABLE ... WITH statement and SSTables written after that will have new value. When upgrading, Cassandra will pick up the value defined in cassandra.yaml as the default for existing ColumnFamilies, until you explicitly set the value for those.
- The deprecated native_transport_min_threads option has been removed in Cassandra.yaml.

Operations
---------
- VNodes are enabled by default in cassandra.yaml. initial_token for non-vnode deployments has been removed from the example yaml, but is still respected if specified.
- Major compactions, cleanup, scrub, and upgradesstables will interrupt any in-progress compactions (but not repair validations) when invoked.
- Disabling autocompactions by setting min/max compaction threshold to 0 has been deprecated, instead, use the nodetool commands 'disableautocompaction' and 'enableautocompaction' or set the compaction strategy option enabled = false
- ALTER TABLE DROP has been reenabled for CQL3 tables and has new semantics now.
- CAS uses gc_grace_seconds to determine how long to keep unused paxos state around for, or a minimum of three hours.
- A new hints created metric is tracked per target, replacing countPendingHints
- After performance testing for CASSANDRA-5727, the default LCS filesize has been changed from 5MB to 160MB.
- cqlsh DESCRIBE SCHEMA no longer outputs the schema of system_* keyspaces; use DESCRIBE FULL SCHEMA if you need the schema of system_* keyspaces.
- CQL2 has been deprecated, and will be removed entirely in 2.2. See CASSANDRA-5918 for details.
- Commit log archiver now assumes the client time stamp to be in microsecond precision, during restore. Please refer to commitlog_archiving.properties.

Features
--------
- Lightweight transactions (http://www.datastax.com/dev/blog/lightweight-transactions-in-cassandra-2-0)
- Alias support has been added to CQL3 SELECT statement. Refer to CQL3 documentation (http://cassandra.apache.org/doc/cql3/CQL.html) for details.
- JEMalloc support (see memory_allocator in cassandra.yaml)
- Experimental triggers support. See examples/ for how to use.
  "Experimental" means "tied closely to internal data structures; we plan to decouple this in the future, which will probably break triggers written against this initial API."
- Numerous improvements to CQL3 and a new version of the native protocol. See http://www.datastax.com/dev/blog/cql-in-cassandra-2-0 for details.

1.2.11
------

Features
--------
- Added a new consistency level, LOCAL_ONE, that forces all CL.ONE operations to execute only in the local datacenter.
- New replace_address to supplant the (now removed) replace_token and
replace_node workflows to replace a dead node in place. Works like the old options, but takes the IP address of the node to be replaced.

1.2.9
=====

Features
--------
- A history of executed nodetool commands is now captured. It can be found in ~/.cassandra/nodetool.history. Other tools output files (cli and cqlsh history, .cqlshrc) are now centralized in ~/.cassandra, as well.
- A new sstablesplit utility allows to split large sstables offline.

1.2.8
=====

Upgrading
---------
- Nothing specific to this release, but please see 1.2.7 if you are upgrading from a previous version.

1.2.7
=====

Upgrading
---------
- If you have decommissioned a node in the past 72 hours, it is imperative that you not upgrade until such time has passed, or do a full cluster restart (not rolling) before beginning the upgrade. This only applies to decommission, not removetoken.

1.2.6
=====

Upgrading
---------
- hinted_handoff_throttle_in_kb is now reduced by a factor proportional to the number of nodes in the cluster (see https://issues.apache.org/jira/browse/CASSANDRA-5272).
- CQL3 syntax for CREATE CUSTOM INDEX has been updated. See CQL3 documentation for details.
1.2.5
=====

Features
--------
- Custom secondary index support has been added to CQL3. Refer to CQL3 documentation (http://cassandra.apache.org/doc/cql3/CQL.html) for details and examples.

Upgrading
--------
- The native CQL transport is enabled by default on port 9042.

1.2.4
=====

Upgrading
--------
- 'nodetool upgradesstables' now only upgrades/rewrites sstables that are not on the current version (which is usually what you want). Use the new -a flag to recover the old behavior of rewriting all sstables.

Features
--------
- superuser setup delay (10 seconds) can now be overridden using 'cassandra.superuser_setup_delay_ms' property.

1.2.3
=====

Upgrading
--------
- CQL3 used to be case-insensitive for property map key in ALTER and CREATE statements. In other words:
  ```
  CREATE KEYSPACE test WITH replication = {'CLASS': 'SimpleStrategy',
                                    'REPLICATION_FACTOR': '1'}
  ```
  was allowed. However, this was not consistent with the fact that string literal are case sensitive in every other places and more importantly this break NetworkTopologyStrategy for which DC names are case sensitive. Those property map key are now case sensitive. So the statement above should be changed to:
  ```
  CREATE KEYSPACE test WITH replication = {'class': 'SimpleStrategy'}
  ```
1.2.2
=====

Upgrading
--------
- CQL3 type validation for constants has been fixed, which may require fixing queries that were relying on the previous loose validation. Please refer to the CQL3 documentation (http://cassandra.apache.org/doc/cql3/CQL.html) and in particular the changelog section for more details. Please note in particular that inputing blobs as strings constants is now deprecated (in favor of blob constants) and its support will be removed in a future version.

Features
-------
- Built-in CQL3-based implementations of IAuthenticator (PasswordAuthenticator) and IAuthorizer (CassandraAuthorizer) have been added. PasswordAuthenticator stores usernames and hashed passwords in system_auth.credentials table; CassandraAuthorizer stores permissions in system_auth.permissions table.
- system_auth keyspace is now alterable via ALTER KEYSPACE queries. The default is SimpleStrategy with replication_factor of 1, but it's advised to raise RF to at least 3 or 5, since CL.QUORUM is used for all auth-related queries. It's also possible to change the strategy to NTS.
- Permissions caching with time-based expiration policy has been added to reduce performance impact of authorization. Permission validity can be configured using 'permissions_validity_in_ms' setting in cassandra.yaml. The default is 2000 (2 seconds).
- SimpleAuthenticator and SimpleAuthorizer examples have been removed. Please look at CassandraAuthorizer/PasswordAuthenticator instead.

1.2.1
=====
Upgrading
---------
- In CQL3, date string are no longer accepted as timeuuid value since a date string is not a correct representation of a timeuuid. Instead, new methods (minTimeuuid, maxTimeuuid, now, dateOf, unixTimestampOf) have been introduced to make working on timeuuid from date string easy. cqlsh also does not display timeuuid as date string (since this is a lossy representation), but the new dateOf method can be used instead. Please refer to the reference documentation (http://cassandra.apache.org/doc/cql3/CQL.html) for more detail.
- For client implementors: CQL3 client using the thrift interface should use the new execute_cql3_query, prepare_cql3_query and execute_prepared_cql3_query since 1.2.0. However, Cassandra 1.2.0 was not complaining if CQL3 was set through set_cql_version but the now CQL2 only methods were used. This is now the case.
- Queries that uses unrecognized or bad compaction or replication strategy options are now refused (instead of simply logging a warning).

1.2
===

Upgrading
---------
- IAuthenticator interface has been updated to support dynamic user creation, modification and removal. Users, even when stored externally, now have to be explicitly created using CREATE USER query first. AllowAllAuthenticator and SimpleAuthenticator have been updated for the new interface, but you'll have to update your old IAuthenticator implementations for 1.2. To ease this process, a new abstract LegacyAuthenticator class has been added - subclass it in your old IAuthenticator implementation and everything should just work (this only affects users who implemented custom authenticators).
- IAuthority interface has been deprecated in favor of IAuthorizer. AllowAllAuthority and SimpleAuthority have been renamed to AllowAllAuthorizer and SimpleAuthorizer, respectively. In order to simplify the upgrade to the new interface, a new abstract LegacyAuthorizer has been added - you should subclass it in your
old IAuthority implementation and everything should just work (this only affects users who implemented custom authorities). 'authority' setting in cassandra.yaml has been renamed to 'authorizer', 'authority' is no longer recognized. This affects all upgrading users.

- 1.2 is NOT network-compatible with versions older than 1.0. That means if you want to do a rolling, zero-downtime upgrade, you'll need to upgrade first to 1.0.x or 1.1.x, and then to 1.2. 1.2 retains the ability to read data files from Cassandra versions at least back to 0.6, so a non-rolling upgrade remains possible with just one step.

- The default partitioner for new clusters is Murmur3Partitioner, which is about 10% faster for index-intensive workloads. Partitioners cannot be changed once data is in the cluster, however, so if you are switching to the 1.2 cassandra.yaml, you should change this to RandomPartitioner or whatever your old partitioner was.

- If you using counters and upgrading from a version prior to 1.1.6, you should drain existing Cassandra nodes prior to the upgrade to prevent overcount during commitlog replay (see CASSANDRA-4782). For non-counter uses, drain is not required but is a good practice to minimize restart time.

- Tables using LeveledCompactionStrategy will default to not creating a row-level bloom filter. The default in older versions of Cassandra differs; you should manually set the false positive rate to 1.0 (to disable) or 0.01 (to enable, if you make many requests for rows that do not exist).

- The hints schema was changed from 1.1 to 1.2. Cassandra automatically snapshots and then truncates the hints column family as part of starting up 1.2 for the first time. Additionally, upgraded nodes will not store new hints destined for older (pre-1.2) nodes. It is therefore recommended that you perform a cluster upgrade when all nodes are up. Because hints will be lost, a cluster-wide repair (with -pr) is recommended after upgrade of all nodes.

- The `nodetool removetoken` command (and corresponding JMX operation) have been renamed to `nodetool removenode`. This function is incompatible with the earlier `nodetool removetoken`, and attempts to remove nodes in this way with a mixed 1.1 (or lower) / 1.2 cluster, is not supported.

- The somewhat ill-conceived CollatingOrderPreservingPartitioner has been removed. Use Murmur3Partitioner (recommended) or ByteOrderedPartitioner instead.

- Global option hinted_handoff_throttle_delay_in_ms has been removed. hinted_handoff_throttle_in_kb has been added instead.

- The default bloom filter fp chance has been increased to 1%. This will save about 30% of the memory used by the old default. Existing columnfamilies will retain their old setting.
- The default partitioner (for new clusters; the partitioner cannot be changed in existing clusters) was changed from RandomPartitioner to Murmur3Partitioner which provides faster hashing as well as improved performance with secondary indexes.
- The default version of CQL (and cqlsh) is now CQL3. CQL2 is still available but you will have to use the thrift set_cql_version method (that is already supported in 1.1) to use CQL2. For cqlsh, you will need to use 'cqlsh -2'.
- CQL3 is now considered final in this release. Compared to the beta version that is part of 1.1, this final version has a few additions (collections), but also some (incompatible) changes in the syntax for the options of the create/alter keyspace/table statements. Typically, the syntax to create a keyspace is now:

```
CREATE KEYSPACE ks WITH replication = { 'class' : 'SimpleStrategy',
  'replication_factor' : 2 };
```

Also, the consistency level cannot be set in the language anymore, but is at the protocol level.
Please refer to the CQL3 documentation (http://cassandra.apache.org/doc/cql3/CQL.html) for details.
- In CQL3, the DROP behavior from ALTER TABLE has currently been removed (because it was not correctly implemented). We hope to add it back soon (Cassandra 1.2.1 or 1.2.2)

Features
--------
- Cassandra can now handle concurrent CREATE TABLE schema changes as well as other updates
- rpc_timeout has been split up to allow finer-grained control on timeouts for different operation types
- num_tokens can now be specified in cassandra.yaml. This defines the number of tokens assigned to the host on the ring (default: 1). Also specifying initial_token will override any num_tokens setting.
- disk_failure_policy allows blacklisting failed disks in JBOD configuration instead of erroring out indefinitely
- event tracing can be configured per-connection ("trace_next_query") or globally/probabilistically ("nodetool settraceprobability")
- Atomic batches are now supported server side, where Cassandra will guarantee that (at the price of pre-writing the batch to another node first), all mutations in the batch will be applied, even if the coordinator fails mid-batch.
- new IAuthorizer interface has replaced the old IAuthority. IAuthorizer allows dynamic permission management via new CQL3 statements: GRANT, REVOKE, LIST PERMISSIONS. A native implementation storing the permissions in Cassandra is being worked on and we expect to include it in 1.2.1 or 1.2.2.
- IAuthenticator interface has been updated to support dynamic user creation, modification and removal via new CQL3 statements: CREATE USER, ALTER USER, DROP USER, LIST USERS. A native implementation that stores users in Cassandra itself is being worked on and is expected to become part of 1.2.1 or 1.2.2.

1.1.5
====

Upgrading
--------
- Nothing specific to this release, but please see 1.1 if you are upgrading from a previous version.

1.1.4
====

Upgrading
--------
- Nothing specific to this release, but please see 1.1 if you are upgrading from a previous version.

1.1.3
====

Upgrading
--------
- Running "nodetool upgradesstables" after upgrading is recommended if you use Counter columnfamilies.

Features
--------
- the cqlsh COPY command can now export to CSV flat files
- added a new tools/bin/token-generator to facilitate generating evenly distributed tokens

1.1.2
====

Upgrading
If you have column families using the LeveledCompactionStrategy, you should run scrub on those column families.

Features
--------
- cqlsh has a new COPY command to load data from CSV flat files

1.1.1
-----
Upgrading
--------
- Nothing specific to this release, but please see 1.1 if you are upgrading from a previous version.

Features
--------
- Continuous commitlog archiving and point-in-time recovery.
  See conf/commitlog_archiving.properties
- Incremental repair by token range, exposed over JMX

1.1
==
Upgrading
--------
- Compression is enabled by default on newly created ColumnFamilies (and unchanged for ColumnFamilies created prior to upgrading).
- If you are running a multi datacenter setup, you should upgrade to the latest 1.0.x (or 0.8.x) release before upgrading. Versions 0.8.8 and 1.0.3-1.0.5 generate cross-dc forwarding that is incompatible with 1.1.
- EACH_QUORUM ConsistencyLevel is only supported for writes and will now throw an InvalidRequestException when used for reads. (Previous versions would silently perform a LOCAL_QUORUM read instead.)
- ANY ConsistencyLevel is only supported for writes and will now throw an InvalidRequestException when used for reads. (Previous versions would silently perform a ONE read for range queries; single-row and multi-get reads already rejected ANY.)
- The largest mutation batch accepted by the commitlog is now 128MB. (In practice, batches larger than ~10MB always caused poor performance due to load volatility and GC promotion failures.) Larger batches will continue to be accepted but will not be durable. Consider setting durable_writes=false if you really want to use such large batches.
- Make sure that global settings: key_cache_{size_in_mb, save_period} and row_cache_{size_in_mb, save_period} in conf/cassandra.yaml are used instead of per-ColumnFamily options.
- JMX methods no longer return custom Cassandra objects. Any such methods will now return standard Maps, Lists, etc.
- Hadoop input and output details are now separated. If you were previously using methods such as getRpcPort you now need to use getInputRpcPort or getOutputRpcPort depending on the circumstance.
- CQL changes:
  - Prior to 1.1, you could use KEY as the primary key name in some select statements, even if the PK was actually given a different name. In 1.1+ you must use the defined PK name.
  - The sliced_buffer_size_in_kb option has been removed from the cassandra.yaml config file (this option was a no-op since 1.0).

Features
--------
- Concurrent schema updates are now supported, with any conflicts automatically resolved. Please note that simultaneously running ‘CREATE COLUMN FAMILY’ operation on the different nodes wouldn’t be safe until version 1.2 due to the nature of ColumnFamily identifier generation, for more details see CASSANDRA-3794.
- The CQL language has undergone a major revision, CQL3, the highlights of which are covered at [1]. CQL3 is not backwards-compatible with CQL2, so we've introduced a set_cql_version Thrift method to specify which version you want. (The default remains CQL2 at least until Cassandra 1.2.) cqlsh adds a --cql3 flag to enable this.
- Row-level isolation: multi-column updates to a single row have always been *atomic* (either all will be applied, or none) thanks to the CommitLog, but until 1.1 they were not *isolated* -- a reader may see mixed old and new values while the update happens.
- Finer-grained control over data directories, allowing a ColumnFamily to be pinned to specific volume, e.g. one backed by SSD.
- The bulk loader is not longer a fat client; it can be run from an existing machine in a cluster.
- A new write survey mode has been added, similar to bootstrap (enabled via -Dcassandra.write_survey=true), but the node will not automatically join the cluster. This is useful for cases such as testing different compaction strategies with live traffic without affecting the cluster.
- Key and row caches are now global, similar to the global memtable threshold. Manual tuning of cache sizes per-columnfamily is no longer required.
- Off-heap caches no longer require JNA, and will work out of the box on Windows as well as Unix platforms.
- Streaming is now multithreaded.
- Compactions may now be aborted via JMX or nodetool.
- The stress tool is not new in 1.1, but it is newly included in binary builds as well as the source tree
- Hadoop: a new BulkOutputFormat is included which will directly
  write SSTables locally and then stream them into the cluster.
  YOU SHOULD USE BulkOutputFormat BY DEFAULT.
  ColumnFamilyOutputFormat is still around in case for some strange reason you want results
  trickling out over Thrift, but BulkOutputFormat is significantly
  more efficient.
- Hadoop: KeyRange.filter is now supported with
  ColumnFamilyInputFormat,
  allowing index expressions to be evaluated server-side to reduce
  the amount of data sent to Hadoop.
- Hadoop: ColumnFamilyRecordReader has a wide-row mode, enabled via
  a boolean parameter to setInputColumnFamily, that pages through
  data column-at-a-time instead of row-at-a-time.
- Pig: can use the wide-row Hadoop support, by setting
  PIG_WIDEROW_INPUT
to true. This will produce each row's columns in a bag.

1.0.8
=====
Upgrading
--------
- Nothing specific to 1.0.8

Other
-----
- Allow configuring socket timeout for streaming

1.0.7
=====
Upgrading
--------
- Nothing specific to 1.0.7, please report to instruction for 1.0.6

Other
-----
- Adds new setstreamthroughput to nodetool to configure streaming
  throttling
- Adds JMX property to get/set rpc_timeout_in_ms at runtime
- Allow configuring (per-CF) bloom_filter_fp_chance

1.0.6
=====
Upgrading
--------
- This release fixes an issue related to the chunk_length_kb option for compressed sstables. If you use compression on some column families, it is recommended after the upgrade to check the value for this option on these column families (the default value is 64). In case the option would not be set correctly, you should update the column family definition, setting the right value and then run scrub on the column family.
- Please report to instruction for 1.0.5 if coming from an older version.

1.0.5
=====

Upgrading
---------
- 1.0.5 comes to fix two important regression of 1.0.4. So all information concerning 1.0.4 are valid for this release, but please avoids upgrading to 1.0.4.

1.0.4
=====

Upgrading
---------
- Nothing specific to 1.0.4 but please see the 1.0 upgrading section if upgrading from a version prior to 1.0.0

Features
--------
- A new upgradesstables command has been added to nodetool. It is very similar to scrub but without the ability to discard corrupted rows (and as a consequence it does not snapshot automatically before). This new command is to be prefered to scrub in all cases where sstables should be rewritten to the current format for upgrade purposes.

JMX
---
- The path for the data, commit log and saved cache directories exposed through JMX
- The in-memory bloom filter sizes are now exposed through JMX
1.0.3
=====

Upgrading
---------
- Nothing specific to 1.0.3 but please see the 1.0 upgrading section if upgrading from a version prior to 1.0.0

Features
--------
- For non compressed sstables (compressed sstable already include more fine grained checsums), a sha1 for the full sstable is now automatically created (in a fix with suffix -Digest.sha1). It can be used to check the sstable integrity with sha1sum.

1.0.2
=====

Upgrading
---------
- Nothing specific to 1.0.2 but please see the 1.0 upgrading section if upgrading from a version prior to 1.0.0

Features
--------
- Cassandra CLI queries now have timing information

1.0.1
=====

Upgrading
---------
- If upgrading from a version prior to 1.0.0, please see the 1.0 Upgrading section
  - For running on Windows as a Service, procrun is no longer distributed with Cassandra, see README.txt for more information on how to download it if necessary.
  - The name given to snapshots directories have been improved for human readability. If you had scripts relying on it, you may need to update them.
1.0

Upgrading
--------
- Upgrading from version 0.7.1+ or 0.8.2+ can be done with a rolling restart, one node at a time. (0.8.0 or 0.8.1 are NOT network-compatible with 1.0: upgrade to the most recent 0.8 release first.) You do not need to bring down the whole cluster at once.
- After upgrading, run nodetool scrub against each node before running repair, moving nodes, or adding new ones.
- CQL inserts/updates now generate microsecond resolution timestamps by default, instead of millisecond. THIS MEANS A ROLLING UPGRADE COULD MIX milliseconds and microseconds, with clients talking to servers generating milliseconds unable to overwrite the larger microsecond timestamps. If you are using CQL and this is important for your application, you can either perform a non-rolling upgrade to 1.0, or update your application first to use explicit timestamps with the "USING timestamp=X" syntax.
- The BinaryMemtable bulk-load interface has been removed (use the sstableloader tool instead).
- The compaction_thread_priority setting has been removed from cassandra.yaml (use compaction_throughput_mb_per_sec to throttle compaction instead).
- CQL types bytea and date were renamed to blob and timestamp, respectively, to conform with SQL norms. CQL type int is now a 4-byte int, not 8 (which is still available as bigint).
- Cassandra 1.0 uses arena allocation to reduce old generation fragmentation. This means there is a minimum overhead of 1MB per ColumnFamily plus 1MB per index.
- The SimpleAuthenticator and SimpleAuthority classes have been moved to the example directory (and are thus not available from the binary distribution). They never provided actual security and in their current state are only meant as examples.

Features
--------
- SSTable compression is supported through the 'compression_options' parameter when creating/updating a column family. For instance, you can create a column family Cf using compression (through the Snappy library) in the CLI with:
create column family Cf with
compression_options={sstable_compression: SnappyCompressor}
SSTable compression is not activated by default but can be
activated or
deactivated at any time.
- Compressed SSTable blocks are checksummed to protect against bitrot
- New LevelDB-inspired compaction algorithm can be enabled by setting
the
    Columnfamily compaction_strategy=LeveledCompactionStrategy option.
Leveled compaction means you only need to keep a few MB of space
free for
    compaction instead of (in the worst case) 50%.
- Ability to use multiple threads during a single compaction. See
  multithreaded_compaction in cassandra.yaml for more details.
- Windows Service ("cassandra.bat install" to enable)
- A dead node may be replaced in a single step by starting a new node
  with -Dcassandra.replace_token=<token>. More details can be found
at
  http://wiki.apache.org/cassandra/Operations#Replacing_a_Dead_Node
- It is now possible to repair only the first range returned by the
  partitioner for a node with `nodetool repair -pr`. It makes it
easier/possible to repair a full cluster without any work
duplication by
    running this command on every node of the cluster.

New data types
-------------
- decimal

Other
-----
- Hinted Handoff has two major improvements:
  - Hint replay is much more efficient thanks to a change in the
data model
  - Hints are created for all replicas that do not ack a write.
(Formerly,
  only replicas known to be down when the write started were
hinted.)
  This means that running with read repair completely off is much
more
  viable than before, and the default read_repair_chance is reduced
from 1.0
  ("always repair") to 0.1 ("repair 10% of the time").
- The old per-ColumnFamily memtable thresholds
  (memtable_throughput_in_mb, memtable_operations_in_millions,
  memtable_flush_after_mins) are ignored, in favor of the global
  memtable_total_space_in_mb and commitlog_total_space_in_mb
settings.
  This does not affect client compatibility -- the old options are
still allowed, but have no effect. These options may be removed
entirely in a future release.
- Backlogged compactions will begin five minutes after startup. The
behavior of never starting compaction until a flush happens is usually not what is desired, but a short grace period is useful to allow caches to warm up first. - The deletion of compacted data files is not performed during Garbage Collection anymore. This means compacted files will now be deleted without delay.

0.8.5 =====

Features
--------
- SSTables copied to a data directory can be loaded by a live node through nodetool refresh (may be handy to load snapshots).
- The configured compaction throughput is exposed through JMX.

Other
-----
- The sstableloader is now bundled with the debian package.
- Repair detects when a participating node is dead and fails instead of hanging forever.

0.8.4 =====

Upgrading
---------
- Nothing specific to 0.8.4

Other
-----
- This release comes to fix a bug in counter that could lead to (important) over-count.
- It also fixes a slight upgrade regression from 0.8.3. It is thus advised to jump directly to 0.8.4 if upgrading from before 0.8.3.

0.8.3 =====

Upgrading
---------
- Token removal has been revamped. Removing tokens in a mixed cluster with 0.8.3 will not work, so the entire cluster will need to be running 0.8.3
first, except for the dead node.

Features
--------
- It is now possible to use thrift asynchronous and
  half-synchronous/half-asynchronous servers (see cassandra.yaml for
  more
details).
- It is now possible to access counter columns through Hadoop.

Other
-----
- This release fix a regression of 0.8 that can make commit log
  segment to
  be deleted even though not all data it contains has been flushed.
  Upgrades from 0.8.* is very much encouraged.

0.8.2
-----
Upgrading
--------
- 0.8.0 and 0.8.1 shipped with a bug that was setting the
  replicate_on_write option for counter column families to false
  (this
  option has no effect on non-counter column family). This is an
  unsafe
  default and 0.8.2 correct this, the default for replicate_on_write
  is
  now true. It is advised to update your counter column family
  definitions
  if replicate_on_write was uncorrectly set to false (before or after
  upgrade).

0.8.1
-----
Upgrading
--------
- 0.8.1 is backwards compatible with 0.8, upgrade can be achieved by
  a
  simple rolling restart.
- If upgrading for earlier version (0.7), please refer to the 0.8
  section
  for instructions.

Features
--------
- Numerous additions/improvements to CQL (support for counters, TTL,
  batch
  inserts/deletes, index dropping, ...).
- Add two new AbstractTypes (comparator) to support compound keys
(CompositeType and DynamicCompositeType), as well as a ReverseType to reverse the order of any existing comparator.
- New option to bypass the commit log on some keyspaces (for advanced users).

Tools
-----
- Add new data bulk loading utility (sstableloader).

0.8
===

Upgrading
----------
- Upgrading from version 0.7.1 or later can be done with a rolling restart, one node at a time. You do not need to bring down the whole cluster at once.
- After upgrading, run nodetool scrub against each node before running repair, moving nodes, or adding new ones.
- Running nodetool drain before shutting down the 0.7 node is recommended but not required. (Skipping this will result in replay of entire commitlog, so it will take longer to restart but is otherwise harmless.)
- 0.8 is fully API-compatible with 0.7. You can continue to use your 0.7 clients.
- Avro record classes used in map/reduce and Hadoop streaming code have been removed. Map/reduce can be switched to Thrift by changing org.apache.cassandra.avro in import statements to org.apache.cassandra.thrift (no class names change). Streaming support has been removed for the time being.
- The loadbalance command has been removed from nodetool. For similar behavior, decommission then rebootstrap with empty initial_token.
- Thrift unframed mode has been removed.
- The addition of key_validation_class means the cli will assume keys are bytes, instead of strings, in the absence of other information. See http://wiki.apache.org/cassandra/FAQ#cli_keys for more details.

Features
--------
- added CQL client API and JDBC/DBAPI2-compliant drivers for Java and Python, respectively (see: drivers/ subdirectory and doc/cql)
- added distributed Counters feature; see http://wiki.apache.org/cassandra/Counters
- optional intranode encryption; see comments around 'encryption_options' in cassandra.yaml
- compaction multithreading and rate-limiting; see
'concurrent_compactors' and 'compaction_throughput_mb_per_sec' in cassandra.yaml
- cassandra will limit total memtable memory usage to 1/3 of the heap by default. This can be adjusted or disabled with the memtable_total_space_in_mb option. The old per-ColumnFamily throughput, operations, and age settings are still respected but will be removed in a future major release once we are satisfied that memtable_total_space_in_mb works adequately.

Tools
-----
- stress and py_stress moved from contrib/ to tools/
- clustertool was removed (see https://issues.apache.org/jira/browse/CASSANDRA-2607 for examples of how to script nodetool across the cluster instead)

Other
-----
- In the past, sstable2json would write column names and values as hex strings, and now creates human readable values based on the comparator/validator. As a result, JSON dumps created with older versions of sstable2json are no longer compatible with json2sstable, and imports must be made with a configuration that is identical to the export.
- manually-forced compactions ("nodetool compact") will do nothing if only a single SSTable remains for a ColumnFamily. To force it to compact that anyway (which will free up space if there are a lot of expired tombstones), use the new forceUserDefinedCompaction JMX method on CompactionManager.
- most of contrib/ (which was not part of the binary releases) has been moved either to examples/ or tools/. We plan to move the rest for 0.8.1.

JMX
---
- By default, JMX now listens on port 7199.

0.7.6
-----
Upgrading
--------
- Nothing specific to 0.7.6, but see 0.7.3 Upgrading if upgrading from earlier than 0.7.1.

0.7.5
-----
Upgrading
--------
- Nothing specific to 0.7.5, but see 0.7.3 Upgrading if upgrading from earlier than 0.7.1.

Changes
-------
- system_update_column_family no longer snapshots before applying the schema change. (_update_keyspace never did. _drop_keyspace and _drop_column_family continue to snapshot.)
- added memtable_flush_queue_size option to cassandra.yaml to avoid blocking writes when multiple column families (or a column family with indexes) are flushed at the same time.
- allow overriding initial_token, storage_port and rpc_port using system properties

0.7.4
=====

Upgrading
---------
- Nothing specific to 0.7.4, but see 0.7.3 Upgrading if upgrading from earlier than 0.7.1.

Features
--------
- Output to Pig is now supported as well as input

0.7.3
=====

Upgrading
---------
- 0.7.1 and 0.7.2 shipped with a bug that caused incorrect row-level bloom filters to be generated when compacting sstables generated with earlier versions. This would manifest in IOExceptions during column name-based queries. 0.7.3 provides "nodetool scrub" to rebuild sstables with correct bloom filters, with no data lost. (If your cluster was never on 0.7.0 or earlier, you don't have to worry about this.) Note that nodetool scrub will snapshot your data files before rebuilding, just in case.

0.7.1
=====

Upgrading
---------
- 0.7.1 is completely backwards compatible with 0.7.0. Just restart each node with the new version, one at a time. (The cluster does not all need to be upgraded simultaneously.)

Features
-------
- added flush_largest_memtables_at and reduce_cache_sizes_at options to cassandra.yaml as an escape valve for memory pressure
- added option to specify -Dcassandra.join_ring=false on startup to allow "warm spare" nodes or performing JMX maintenance before joining the ring

Performance
-----------
- Disk writes and sequential scans avoid polluting page cache (requires JNA to be enabled)
- Cassandra performs writes efficiently across datacenters by sending a single copy of the mutation and having the recipient forward that to other replicas in its datacenter.
- Improved network buffering
- Reduced lock contention on memtable flush
- Optimized supercolumn deserialization
- Zero-copy reads from mmapped sstable files
- Explicitly set higher JVM new generation size
- Reduced i/o contention during saving of caches

0.7.0
=====

Features
--------
- Secondary indexes (indexes on column values) are now supported
- Row size limit increased from 2GB to 2 billion columns. rows are no longer read into memory during compaction.
- Keyspace and ColumnFamily definitions may be added and modified live
- Streaming data for repair or node movement no longer requires anticompaction step first
- NetworkTopologyStrategy (formerly DatacenterShardStrategy) is ready for use, enabling ConsistencyLevel.DCQUORUM and DCQUORUMSYNC. See comments in `cassandra.yaml`.
- Optional per-Column time-to-live field allows expiring data without have to issue explicit remove commands
- `truncate` thrift method allows clearing an entire ColumnFamily at once
- Hadoop OutputFormat and Streaming [non-jvm map/reduce via stdin/out] support
- Up to 8x faster reads from row cache
- A new ByteOrderedPartitioner supports bytes keys with arbitrary content, and orders keys by their byte value. This should be used in new deployments instead of OrderPreservingPartitioner.
- Optional round-robin scheduling between keyspaces for multitenant clusters
- Dynamic endpoint snitch mitigates the impact of impaired nodes
- New `IntegerType`, faster than LongType and allows integers of
both less and more bits than Long's 64
- A revamped authentication system that decouples authorization and allows finer-grained control of resources.

Upgrading
--------
The Thrift API has changed in incompatible ways; see below, and refer to http://wiki.apache.org/cassandra/ClientOptions for a list of higher-level clients that have been updated to support the 0.7 API.

The Cassandra inter-node protocol is incompatible with 0.6.x releases (and with 0.7 beta1), meaning you will have to bring your cluster down prior to upgrading: you cannot mix 0.6 and 0.7 nodes.

The hints schema was changed from 0.6 to 0.7. Cassandra automatically snapshots and then truncates the hints column family as part of starting up 0.7 for the first time.

Keyspace and ColumnFamily definitions are stored in the system keyspace, rather than the configuration file.

The process to upgrade is:
1) run "nodetool drain" on _each_ 0.6 node. When drain finishes (log message "Node is drained" appears), stop the process.
2) Convert your storage-conf.xml to the new cassandra.yaml using "bin/config-converter".
3) Rename any of your keyspace or column family names that do not adhere to the '^\w+' regex convention.
4) Start up your cluster with the 0.7 version.
5) Initialize your Keyspace and ColumnFamily definitions using "bin/schematool <host> <jmxport> import". _You only need to do this to one node_.

Thrift API
--------
- The Cassandra server now defaults to framed mode, rather than unframed. Unframed is obsolete and will be removed in the next major release.
- The Cassandra Thrift interface file has been updated for Thrift 0.5.

If you are compiling your own client code from the interface, you will need to upgrade the Thrift compiler to match.
- Row keys are now bytes: keys stored by versions prior to 0.7.0 will be returned as UTF-8 encoded bytes. OrderPreservingPartitioner and CollatingOrderPreservingPartitioner continue to expect that keys contain UTF-8 encoded strings, but RandomPartitioner now works on any key data.
- keyspace parameters have been replaced with the per-connection set_keyspace method.
- The return type for login() is now AccessLevel.
- The get_string_property() method has been removed.
- The get_string_list_property() method has been removed.

Configuration
-------------
- Configuration file renamed to cassandra.yaml and log4j.properties to log4j-server.properties
- PropertyFileSnitch configuration file renamed to cassandra-topology.properties
- The ThriftAddress and ThriftPort directives have been renamed to RPCAddress and RPCPort respectively.
- EndPointSnitch was renamed to RackInferringSnitch. A new SimpleSnitch has been added.
- RackUnawareStrategy and RackAwareStrategy have been renamed to SimpleStrategy and OldNetworkTopologyStrategy, respectively.
- RowWarningThresholdInMB replaced with in_memory_compaction_limit_in_mb
- GCGraceSeconds is now per-ColumnFamily instead of global
- Keyspace and column family names that do not confirm to a '^[\w]+' regex are considered illegal.
- Keyspace and column family definitions will need to be loaded via "bin/schematool <host> <jmxport> import". _You only need to do this to one node_.
- In addition to an authenticator, an authority must be configured as well. Users of SimpleAuthenticator should use SimpleAuthority for this value (the default is AllowAllAuthority, which corresponds with AllowAllAuthenticator).
- The format of access.properties has changed, see the sample configuration conf/access.properties for documentation on the new format.

JMX
---
- StreamingService moved from o.a.c.streaming to o.a.c.service
- GMFD renamed to GOSSIP_STAGE
- {Min,Mean,Max}RowCompactedSize renamed to {Min,Mean,Max}RowSize since it no longer has to wait til compaction to be computed

Other
-----
- If extending AbstractType, make sure you follow the singleton pattern followed by Cassandra core AbstractType classes: provide a public static final variable called 'instance'.

0.6.6
-----
## Upgrading

---

- As part of the cache-saving feature, a third directory (along with data and commitlog) has been added to the config file. You will need to set and create this directory when restarting your node into 0.6.6.

### 0.6.1

---

Upgrading

---

- We try to keep minor versions 100% compatible (data format, commitlog format, network format) within the major series, but we introduced a network-level incompatibility in 0.6.1. Thus, if you are upgrading from 0.6.0 to any higher version (0.6.1, 0.6.2, etc.) then you will need to restart your entire cluster with the new version, instead of being able to do a rolling restart.

### 0.6.0

---

Features

---

- row caching: configure with the RowsCached attribute in ColumnFamily definition
- Hadoop map/reduce support: see contrib/word_count for an example
- experimental authentication support, described under Authenticator in storage.conf

Configuration

---

- MemtableSizeInMB has been replaced by MemtableThroughputInMB which triggers a memtable flush when the specified amount of data has been written, including overwrites.
- MemtableObjectCountInMillions has been replaced by the MemtableOperationsInMillions directive which causes a memtable flush to occur after the specified number of operations.
- Like MemtableSizeInMB, BinaryMemtableSizeInMB has been replaced by BinaryMemtableThroughputInMB.
- Replication factor is now per-keyspace, rather than global.
- KeysCachedFraction is deprecated in favor of KeysCached
- RowWarningThresholdInMB added, to warn before very large rows get big enough to threaten node stability

Thrift API

---

- removed deprecated get_key_range method
- added batch_mutate method
- deprecated multi_get and batch_insert methods in favor of
multiget_slice and batch_mutate, respectively
- added ConsistencyLevel.ANY, for when you want write availability even when it may not be readable immediately. Unlike CL.ZERO, though, it will throw an exception if it cannot be written *somewhere*.

JMX metrics
-------------
- read and write statistics are reported as lifetime totals, instead of averages over the last minute. average-since-last requested are also available for convenience.
- cache hit rate statistics are now available from JMX under org.apache.cassandra.db.Caches
- compaction JMX metrics are moved to org.apache.cassandra.db.CompactionManager. PendingTasks is now a much better estimate of compactions remaining, and the progress of the current compaction has been added.
- commitlog JMX metrics are moved to org.apache.cassandra.db.Commitlog
- progress of data streaming during bootstrap, loadbalance, or other data migration, is available under org.apache.cassandra.streaming.StreamingService. See http://wiki.apache.org/cassandra/Streaming for details.

Installation/Upgrade
---------------------
- 0.6 network traffic is not compatible with earlier versions. You will need to shut down all your nodes at once, upgrade, then restart.

0.5.0
=====
0. The commitlog format has changed (but sstable format has not). When upgrading from 0.4, empty the commitlog either by running bin/nodeprobe flush on each machine and waiting for the flush to finish, or simply remove the commitlog directory if you only have test data. (If more writes come in after the flush command, starting 0.5 will error out; if that happens, just go back to 0.4 and flush again.) The format changed twice: from 0.4 to beta1, and from beta2 to RC1.

.5 The gossip protocol has changed, meaning 0.5 nodes cannot coexist in a cluster of 0.4 nodes or vice versa; you must upgrade your whole cluster at the same time.

1. Bootstrap, move, load balancing, and active repair have been added. See http://wiki.apache.org/cassandra/Operations. When upgrading from 0.4, leave autobootstrap set to false for the first restart of your old nodes.
2. Performance improvements across the board, especially on the write path (over 100% improvement in stress.py throughput).

3. Configuration:
   - Added "comment" field to ColumnFamily definition.
   - Added MemtableFlushAfterMinutes, a global replacement for the old per-CF FlushPeriodInMinutes setting
   - Key cache settings

4. Thrift:
   - Added get_range_slice, deprecating get_key_range

0.4.2
=====

1. Improve default garbage collector options significantly -- throughput will be 30% higher or more.

0.4.1
=====

1. SnapshotBeforeCompaction configuration option allows snapshotting before each compaction, which allows rolling back to any version of the data.

0.4.0
=====

1. On-disk data format has changed to allow billions of keys/rows per node instead of only millions. The new format is incompatible with 0.3; see 0.3 notes below for how to import data from a 0.3 install.

2. Cassandra now supports multiple keyspaces. Typically you will have one keyspace per application, allowing applications to be able to create and modify ColumnFamilies at will without worrying about collisions with others in the same cluster.


4. Removed the web interface in favor of JMX and bin/nodeprobe, which has significantly enhanced functionality.

5. Renamed configuration "<Table>" to "<Keyspace>".

6. Added commitlog fsync; see "<CommitLogSync>" in configuration.
0.3.0
=====

1. With enough and large enough keys in a ColumnFamily, Cassandra will run out of memory trying to perform compactions (data file merges). The size of what is stored in memory is \((S + 16) \times (N + M)\) where \(S\) is the size of the key (usually 2 bytes per character), \(N\) is the number of keys and \(M\), is the map overhead (which can be guestimated at around 32 bytes per key).

   So, if you have 10-character keys and 1GB of headroom in your heap space for compaction, you can expect to store about 17M keys before running into problems.

   See https://issues.apache.org/jira/browse/CASSANDRA-208

2. Because fixing #1 requires a data file format change, 0.4 will not be binary-compatible with 0.3 data files. A client-side upgrade can be done relatively easily with the following algorithm:

   ```python
   for key in old_client.get_key_range(everything):
       columns = old_client.get_slice or get_slice_super(key, all columns)
       new_client.batch_insert or batch_insert_super(key, columns)
   The inner loop can be trivially parallelized for speed.
   ```

   3. Commitlog does not fsync before reporting a write successful. Using blocking writes mitigates this to some degree, since all nodes that were part of the write quorum would have to fail before sync for data to be lost.

   See https://issues.apache.org/jira/browse/CASSANDRA-182

   Additionally, row size (that is, all the data associated with a single key in a given ColumnFamily) is limited by available memory, because compaction deserializes each row before merging.

   See https://issues.apache.org/jira/browse/CASSANDRA-16

---

**DSE 5.1.3**

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

**Attention:** TTL expiration timestamps are susceptible to the year 2038 problem. If the TTL value is long and an expiration date is greater than the maximum threshold of 2038-01-19T03:14:06+00:00, the data is immediately expired and purged on the next compaction. When using a long TTL, DataStax strongly recommends upgrading to DSE 5.1.7 or later and taking required action.

6 September 2017

- 5.1.3 Components (page 155)
- 5.1.3 Highlights (page 155)
5.1.3 Components

- Apache Cassandra™ 3.11.0.1855 (updated)
- Apache Solr™ 6.0.1.0.1833 (updated)
- Apache Spark™ 2.0.2.6
- Apache Tomcat® 8.0.44 (updated)
- DataStax Spark Cassandra Connector 2.0.5 (updated)
- DSE Java Driver 1.2.2
- DSEFS 5.1.2 (updated)
- Netty 4.0.42.Final
- Spark Jobserver 0.6.2.234 (requires compatible API)
- TinkerPop 3.2.6-20170821-ac1bcb27 (updated)
- Select Hadoop libraries

5.1.3 Highlights

Executive summary highlights for DSE 5.1.3:

- DSE Analytics (page 155)
- DSE Graph (page 156)
- DSE Search (page 156)

The executive summary highlights are just a top-level view. Be sure to review all 5.1.3 Changes and enhancements (page 156).

5.1.3 DataStax Enterprise core highlights

- Incremental repairs are no longer the default for nodetool repair. Even with nodetool repair -full or nodetool repair -pr, DSE 5.1.0-5.1.2 were run as incremental and marked sstables as repaired causing anti-compaction. (DSP-14464)

  After upgrades from DSE 5.1.0-5.1.2 to DSE 5.1.3 or later, you must follow instructions in the upgrade guide to migrate off of incremental repairs. To continue running incremental repairs, use nodetool repair -inc.

5.1.3 DSE Analytics and DSEFS highlights

- New -framework option for dse spark commands to accommodate applications that were originally written for open source Apache Spark. Specify which classpath is used, either the DSE version (default) or a similar path to open source Spark 2.0. (DSP-12954)
• DSEFS includes several important stability fixes and performance improvements. To use DSEFS in production, DataStax strongly recommends upgrading to DSE 5.1.3 to leverage these improvements.

5.1.3 DSE Graph highlights

• Significantly improved graph query performance. (DSP-11534)
• Domain specific language support. (DSP-13545)
• Graph custom id support for multiple keyed vertices. (DGL-258)

5.1.3 DSE Search highlights

• Extend TieredMergePolicy (page 518) to support automatic removal of deletes. (DSP-13626)

5.1.3 Changes and enhancements

In addition to the 5.1.3 Highlights (page 155), review all changes and enhancements:

• DataStax Enterprise (page 156)
• DSE Analytics (page 157)
• DSE Graph (page 157)
• DSEFS (page 158)
• DSE Search (page 158)

5.1.3 DataStax Enterprise changes and enhancements

• nodetool rebuildhttps://docs.datastax.com/en/cassandra/3.0/cassandra/tools/toolsRebuild.html and bootstrap improvements. (DSP-13870)
• Simplify role-permissions handling. (DSP-14159)
  The table system_auth.resource_role_permissions_index is no longer used. Drop this table after all nodes are upgraded to DSE 5.0.10. Upgrades from DSE 5.0.10+ to DSE versions earlier than 5.1.3 are not recommended. See Restrictions when upgrading to DSE 5.1.3.
• New nodetool nodetool mark_unrepaired command unifies repaired and unrepaired compaction buckets. (DSP-14255)
• Changes to nodetool repair. (DSP-14464)
  # When run without options on new tables, the default behavior is nodetool repair --full. (Earlier versions were incremental when no options were specified.)
  # When run without options on a keyspace or set of tables, nodetool repair runs incremental repair on tables previously repaired and full repair on new tables.
  # Anti-compaction is no longer run after full repairs. Use nodetool repair --run-anticompaction to restore the previous behavior.
  # Incremental repair is no longer supported on tables with MVs and CDC. An incremental repair executed on table with MVs or CDC will run full repair instead.
After upgrades from DSE 5.1.0-5.1.2 to DSE 5.1.3 or later, you must follow instructions in the upgrade guide to migrate off of incremental repairs. To continue running incremental repairs, use nodetool repair -inc.

5.1.3 DSE Analytics changes and enhancements

- Improved error on Spark:// Master URLs. (DSP-13366)
- New -framework option for dse spark commands to accommodate applications that were originally written for open source Apache Spark. Specify which classpath is used, either the DSE version (default) or a similar path to open source Spark 2.0. (DSP-12954)
- Improved error messages when no target datacenter provided for Spark application. (DSP-13236)

5.1.3 DSE Graph changes and enhancements

- Improved and simplified data batch loading of pre-formatted data. (DGL-235)
  
  Supporting changes:
  
  # Schema discovery and schema generation are deprecated. (DGL-246)
  # Standard IDs are deprecated. (DGL-247)
  # Transformations are deprecated. (DGL-248)
  # Standard vertex IDs are deprecated. Use custom vertex IDs instead. (DSP-13485)
  
  - Schema discovery and schema generation are deprecated. (DGL-246)
  - Graph custom id support for multiple keyed vertices. (DGL-258)
  - Query engine significantly improved to allow more queries to be satisfied by using indexes. In particular, AND and OR queries are now handled and translate transparently to multiple backend queries or, if possible, single search queries. (DSP-11534)
  - Allow for indexes to be used with ORDER BY clause. (DSP-11931)
  - Checking for edge connectedness no longer performs an unnecessary backend query. (DSP-12863)
  - Edge queries using between predicate now use an index, if available. (DSP-13541)
  - Improved support for domain-specific languages (DSL) in Gremlin enables the DataStax driver to specify TraversalSource. (DSP-13545)
  - cache=false at the transaction level now includes disabling AdjacencyListStoreImpl and IndexStoreImpl. (DSP-13560)
  - Vertices without multi-properties fetch all properties in a single query, rather than requesting properties one at a time. Using multi-properties as vertices is not recommended, because multiple cardinality (multi-properties) are retrieved in graph traversals more slowly than single cardinality properties. Vertices with multi-properties default to the previous behavior of requesting properties individually. (DSP-13646)
  - More Gremlin APIs are supported in DSEGraphFrames: dedup, sort, limit, filter, + as() / select(), or(). (DSP-13649)
  - Do partition deletes for the property/edge table entries if possible. (DSP-13671)
• Timeouts for graph traversals now start from the time the request is received. Earlier releases started timeouts for graph traversals at processing start time. Timeouts will appear more readily on an overloaded server. (DSP-13828)
• Numeric sack values no longer need to be explicitly typed (for example, 3.0D). You can still provide for greater specificity in the expected return type. (DSP-14026)
• Lambdas provided to the sack() step are now recognized by the LambdaRestrictionStrategy. You must disable the restrict_lambda setting to call this method. (DSP-14118)
• Support user-supplied IDs for edges (page 931) and properties (page 933). ID must be Java UUID. (DSP-12932)

5.1.3 DSEFS changes and enhancements

• Expand DSEFS repair capability. DSEFS fsck checks if data blocks exist on the remote node that claims to have them. Mixed versions during upgrades are not supported. Upgrade all nodes in the cluster before using DSEFS fsck. (DSP-13081)
• DSEFS read performance is improved. (DSP-13309)
• Launch DSEFS shell with precedence (page 483) given to the specified hosts. (DSP-14108)
• Connection reuse is improved. Closing idle connections is disabled by default. New idle_connection_timeout_ms (page 341) option in dse.yaml defines how long to wait before an idle client-server connection is closed. (DSP-14010)
• Protocol change improves efficiency of passing JSON arrays between DSEFS server and client. Mixed versions during upgrades are not supported. Upgrade all nodes in the cluster before using the DSEFS shell. (DSP-14107)

5.1.3 DSE Search changes and enhancements

• OffheapPostings is present by default in demo and auto-generated solrconfig.xml files. (DSP-10088)
• The default filter cache settings are changed. (DSP-13153)
• Streamlined autoSolrConfig.xml template for auto-generated search indexes. CQL ALTER SEARCH INDEX CONFIG, ALTER SEARCH INDEX Schema, and CREATE SEARCH INDEX shortcuts for TieredMergePolicyFactory. (DSP-13229)
• DeleteById is deprecated. (DSP-13988)
• Extend TieredMergePolicy (page 518) to support automatic removal of deletes. (DSP-13626)
• DSE Search indexing optimizes for SSDs by default. Spinning disk detection logic is removed. (DSP-13924)
• Improved error messages on invalid solr_query are more descriptive for invalid queries and syntax errors. (DSP-14003)

5.1.3 Resolved issues

Resolved issues for:
• DataStax Enterprise core (page 159)
• DSE Advanced Replication (page 159)
• DSE Analytics (page 159)
• DSEFS (page 159)
• DSE Graph (page 160)
• DSE Search (page 160)

5.1.3 DataStax Enterprise resolved issues

• Adjust and check directory ownership when starting DSE. (DSP-13245)
• CVE-2017-7957 xstream-core is vulnerable to Denial of Service (DoS) attacks. (DSP-13419)
• After restore, data cannot be queried after streaming SSTables with sstableloader to tiered storage. (DSP-14188)
• MemoryOnlyStrategy regions not immediately loaded into physical memory with new kernels. (DSP-14169)
• Make full repair default and disallow incremental repair on MV/CDC tables. (DSP-14255)
• Revert CASSANDRA-11223 behavior in AbstractReadCommandBuilder. (DSP-14135)
• Prevent marking remote SSTables shadowing compacted data as repaired. (DSP-14141)
• Rebuild logging always says 0 bytes. (DSP-13870)
• Allow aggressive expiration of fully expired sstables without timestamp/key overlap checks. (DSP-13870)
• SSTable index files can become corrupted due to StreamingHistogram bug. (DSP-14279)

5.1.3 DSE Advanced Replication resolved issues

• DataStax installer does not set up DSE Advanced Replication correctly. (DSP-13472)
• Ingestion might miss or drop data at higher insertion rates. CDC log file might be deleted even if not processed. (DSP-14043)
• DSEFS clients unnecessarily switch between remote nodes. (DSP-14108)
• Race condition under heavy load sent confusing exceptions to the log file. (DSP-14180)

5.1.3 DSE Analytics resolved issues

• Decrease logging level for RPC methods failures. (DSP-13282)
• JoinWithCassandra and SaveToCassandra blocked on adding to requests to the async execute pool. (DSP-14178)

5.1.3 DSEFS resolved issues

• DataStax installer does not set up DSEFS correctly for No Services installations. (DSP-13473)
• NullPointerException: Unexpected null value of column valid_from in <dse keyspace>.inodes while running fsck. (DSP-12615)
• Memory leak occurs with incorrect use of WebHDFS API. (DSP-13813)
• Rare client-side ParsingException. (DSP-14000)
• Incorrect FileNotFound errors when using Spark with DSEFS. (DSP-14105)

5.1.3 DSE Graph resolved issues

• -help prints help twice. (DGL-257)
• DGL prints warning excessively. (DGL-262)
• The number of vertex labels is limited to 200 per graph. (DSP-11078)
• Graph frames error if meta-property is not populated. (DSP-13063)
• Gremlin server log directory setting doesn't work if default log location is moved. Use dse-env.sh to change log locations. (DSP-13508)
• DseGraphFrame throws UnsupportedOperationException for graph with empty schema. (DSP-13858)
• DseGraphRpc.getSchemaBlob should request EXECUTE permissions instead of SELECT. (DSP-13888)
• Single cardinality edge updates work incorrectly. (DSP-14185)
• DseGraphFrames.updateVertices() requires unnecessary ID columns. (DSP-14175)
• The within predicate is not working for unindexed edges. (DSP-13209)

5.1.3 DSE Search resolved issues

• Shard request exceptions are not logged at the replica level. (DSP-12691)
• Unnecessary double segment flushing on hard commit. (DSP-13971)
• Reintroduce provisioning/dropping states for backward compatibility. Issue a warning when a graph is found. (DSP-14111)
• Search permissions cannot be managed on non-search nodes in the cluster. (DSP-14242)

5.1.3 DataStax Enterprise known issue

• DataStax Enterprise will not run with Java 1.8u161 or later. (DSP-15277)
• Potential data loss for INSERTs with very large TTLs. TTL expiration timestamps are susceptible to the year 2038 problem. (DSP-15412)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. TTLs are considered "very large" when close to the maximum allowed value of 630720000 seconds (20 years), starting from 2018-01-19T03:14:06+00:00. As time progresses, the maximum supported TTL is gradually reduced as the maximum expiration date approaches. For instance, on 2028-01-19T03:14:06 with a TTL of 10 years is impacted. The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL
expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately.

**Warning:** Upgrade to DSE 5.1.7 or later and take required action to protect against overflow of local expiration time.

### Hadoop libraries

Built-in Hadoop and Bring-Your-Own-Hadoop (BYOH) were deprecated in DataStax Enterprise (DSE) 5.0, and were removed in DSE 5.1. Hadoop removal from DSE 5.1 and later means that DSE does not allow for the startup of Hadoop services previously included in DSE, including MapReduce JobTracker and TaskTracker.

However, DSE has supported built-in Spark since DSE 4.5 and Bring-Your-Own-Spark (BYOS) since DSE 5.0, and that support continues today. Because Spark depends on certain Hadoop libraries on the server and the client, DSE continues to ship with Hadoop libraries that are required for running Spark and BYOS.

To view the included Hadoop libraries, see DataStax Enterprise 5.1.x third-party software.

### dse.yaml

The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/dse/conf/dse.yaml</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>installation_location/resources/dse/conf/dse.yaml</td>
</tr>
</tbody>
</table>

### Cassandra enhancements for DSE 5.1.3

DataStax Enterprise (DSE) 5.1.3 includes all changes from earlier DSE releases. These production-certified changes are enhancements to Apache Cassandra™ 3.11.0. (For Cassandra updates, see CHANGES.txt.)

- Fix cassandra-stress hang issues when an error during cluster connection happens (CASSANDRA-12938)
- Better bootstrap failure message when blocked by (potential) range movement (CASSANDRA-13744)
- "ignore" option is ignored in sstableloader (CASSANDRA-13721)
- Deadlock in AbstractCommitLogSegmentManager (CASSANDRA-13652)
- Duplicate the buffer before passing it to analyser in SASI operation (CASSANDRA-13512)
- Copy session properties on cqlsh.py do_login (CASSANDRA-13640)
- Potential AssertionError during ReadRepair of range tombstone and partition deletions (CASSANDRA-13719)
- Don't let stress write warmup data if n=0 (CASSANDRA-13773)
- Gossip thread slows down when using batch commit log (CASSANDRA-12966)
- Randomize batchlog endpoint selection with only 1 or 2 racks (CASSANDRA-12884)
• Fix digest calculation for counter cells (CASSANDRA-13750)
• Fix ColumnDefinition.cellValueType() for non-frozen collection and change SSTabledump to use type.toJSONString() (CASSANDRA-13573)
• Skip materialized view addition if the base table doesn't exist (CASSANDRA-13737)
• Drop table should remove corresponding entries in dropped_columns table (CASSANDRA-13730)
• Log warn message until legacy auth tables have been migrated (CASSANDRA-13371)
• Fix incorrect [2.1 <- 3.0] serialization of counter cells created in 2.0 (CASSANDRA-13691)
• Fix invalid writetime for null cells (CASSANDRA-13711)
• Fix ALTER TABLE statement to atomically propagate changes to the table and its MVs (CASSANDRA-12952)
• Fix Digest mismatch Exception if hints file has UnknownColumnFamily (CASSANDRA-13696)
• Fixed ambiguous output of nodetool tablestats command (CASSANDRA-13722)
• Purge tombstones created by expired cells (CASSANDRA-13643)
• Make concat work with iterators that have different subsets of columns (CASSANDRA-13482)
• Set test.runners based on cores and memory size (CASSANDRA-13078)
• sstabledump reports incorrect usage for argument order (CASSANDRA-13532)
• Uncaught exceptions in Netty pipeline (CASSANDRA-13649)
• Prevent integer overflow on exabyte filesystems (CASSANDRA-13067)
• Fix queries with LIMIT and filtering on clustering columns (CASSANDRA-11223)
• Fix potential NPE when resume bootstrap fails (CASSANDRA-13272)
• Fix toJSONString for the UDT, tuple and collection types (CASSANDRA-13592)
• Clone HeartBeatState when building gossip messages. Make its generation/version volatile (CASSANDRA-13700)

General upgrade advice for DSE 5.1.3

Carefully review all planning and upgrade documentation in the Upgrading DataStax Enterprise guide. The following provides general upgrade information:

GENERAL UPGRADING ADVICE FOR ANY VERSION
========================================

Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to ALWAYS snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots
using the provided 'sstableupgrade' tool.

DSE 5.1.3
=========

Upgrading
---------
- 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 table system_auth.resource_role_permissions_index is no longer used and should be dropped
  after all nodes are on 5.1.3. Note that upgrades from DSE 5.0 series since 5.0.10 to DSE
  versions before 5.1.3 are not recommended.
- Full repairs are now default if no option is specified on nodetool repair, unless
  incremental repair was already run on the table/keyspace being repaired, to maintain
  backward compatibility. Incremental repair may be run on new tables by using the -inc option.
- Full repairs will no longer run repair unless the --run-anticompaction option is specified
- Incremental repairs are no longer supported on tables with materialized views or CDC until
  its limitations are addressed. An incremental repair triggered on a base table or
  materialized view run a full repair instead. See CASSANDRA-12888 for details.

Materialized Views (only when upgrading from DSE 5.1.1 or 5.1.2 or any version lower than DSE 5.0.10)
-----------------------------------------------------
- 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
  to not 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.

3.11.0 =====

Upgrading --------

- ALTER TABLE (ADD/DROP COLUMN) operations concurrent with a read might result into data corruption (see CASSANDRA-13004 for more details). Fixing this bug required a messaging protocol version bump. By default, Cassandra 3.11 will use 3014 version for messaging.

Since Schema Migrations rely on the exact messaging protocol version match between nodes, if you need schema changes during the upgrade process, you have to start your nodes with `-Dcassandra.force_3_0_protocol_version=true` first, in order to temporarily force a backwards compatible protocol.

After the whole cluster is upgraded to 3.11, do a rolling restart of the cluster without setting that flag.

3.11 nodes with and without the flag set will be able to do schema migrations with other 3.x and 3.0.x releases.

While running the cluster with the flag set to true on 3.11 (in compatibility mode), avoid adding or removing any columns to/from existing tables.

If your cluster can do without schema migrations during the upgrade time, just start the cluster normally without setting aforementioned flag.

If you are upgrading from 3.0.14+ (of 3.0.x branch), you do not have to set an flag while upgrading to ensure schema migrations.

- 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).

3.10
====

New features
------------
- New 'DurationType' (cql duration). See CASSANDRA-11873
- Runtime modification of concurrent_compactors is now available via nodetool
- 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.
- Compaction can now take into account overlapping tables that don't take part in the compaction to look for deleted or overwritten data in the compacted tables.
  Then such data is found, it can be safely discarded, which in turn should enable the removal of tombstones over that data.
The behavior can be engaged in two ways:
- as a "nodetool garbagecollect -g CELL/ROW" operation, which applies single-table compaction on all sstables to discard deleted data in one step.
- as a "provide_overlapping_tombstones:CELL/ROW/NONE" compaction strategy flag, which uses overlapping tables as a source of deletions/overwrites during all compactions.

The argument specifies the granularity at which deleted data is to be found:
- If ROW is specified, only whole deleted rows (or sets of rows) will be discarded.
- If CELL is specified, any columns whose value is overwritten or deleted will also be discarded.
- NONE (default) specifies the old behavior, overlapping tables are not used to decide when to discard data.

Which option to use depends on your workload, both ROW and CELL increase the disk load on compaction (especially with the size-tiered compaction strategy), with CELL being more resource-intensive. Both should lead to better read performance if deleting rows (resp. overwriting or deleting cells) is common.
- 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.

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

Spark Cassandra Connector changes for DSE 5.1.3

A list of DataStax Enterprise 5.1.3 production-certified changes for the DataStax Spark
Cassandra Connector.

DSE 5.1.3:
2.0.5
* Allow IN predicates for composite partition keys and clustering keys
to be pushed down to Cassandra (SPARKC-490)
* Allow 'YYYY' format LocalDate
* Add metrics for write batch Size (SPARKC-501)
* Type Converters for java.time.localdate (SPARKC-495)

2.0.4
* Includes partches up to 1.6.9
* Retry PoolBusy Exceptions, Throttle JWCT Calls (SPARKC-503)

DSE 5.1.2:
2.0.3
* Includes patches up to 1.6.8

DSE 5.1.1:
2.0.2
* Protect against Size Estimate Overflows (SPARKC-492)
* Add java.time classes support to converters and sparkSQL (SPARKC-491)
* Allow Writes to Static Columns and Partition Keys (SPARKC-470)

DSE 5.1.0:
2.0.1
* Refactor Custom Scan Method (SPARKC-481)

2.0.0
* Upgrade driver version for 2.0.0 Release to 3.1.4 (SPARKC-474)
* Extend SPARKC-383 to All Row Readers (SPARKC-473)

2.0.0 RC1
* Includes all patches up to 1.6.5
* Automatic adjustment of Max Connections (SPARKC-471)
* Allow for Custom Table Scan Method (SPARKC-459)
* Enable PerPartitionLimit (SPARKC-446)
* Support client certificate authentication for two-way SSL Encryption (SPARKC-359)
* Change Config Generation for Cassandra Runners (SPARKC-424)
* Remove deprecated QueryRetryDelay parameter (SPARKC-423)
* User ConnectionHostParam.default as default hosts String
* Update usages of deprecated SQLContext so that SparkSession is used instead (SPARKC-400)
* Test Reused Exchange SPARK-17673 (SPARKC-429)
* Module refactoring (SPARKC-398)
* Recognition of Java Driver Annotated Classes (SPARKC-427)
* RDD.deleteFromCassandra (SPARKC-349)
* Coalesce Pushdown to Cassandra (SPARKC-161)
* Custom Conf options in Custom Pushdowns (SPARKC-435)
* Upgrade CommonBeatUtils to 1.9.3 to Avoid SID-760 (SPARKC-457)

2.0.0 M3
* Includes all patches up to 1.6.2

2.0.0 M2
* Includes all patches up to 1.6.1

2.0.0 M1
* Added support for left outer joins with Cassandra table (SPARKC-181)
* Removed CassandraSqlContext and underscore based options (SPARKC-399)
* Upgrade to Spark 2.0.0-preview (SPARKC-396)
  - Removed Twitter demo because there is no spark-streaming-twitter package available anymore
  - Removed Akka Actor demo because there is no support for such streams anymore
  - Bring back Kafka project and make it compile
  - Update several classes to use our Logging instead of Spark Logging because Spark Logging became private
  - Update few components and tests to make them work with Spark 2.0.0
  - Fix Spark SQL - temporarily
DSE 5.1.2

**Important:** DataStax recommends the latest patch release. The latest version of DataStax Enterprise 5.1 is 5.1.11. Due to Potential data loss for INSERTs with very large TTLs. (DSP-15412) *(page 203)*, DataStax does not recommend DSE 5.1.0-5.1.2 for production.

**Attention:** TTL expiration timestamps are susceptible to the year 2038 problem. If the TTL value is long and an expiration date is greater than the maximum threshold of 2038-01-19T03:14:06+00:00, the data is immediately expired and purged on the next compaction. When using a long TTL, DataStax strongly recommends upgrading to DSE 5.1.7 or later and taking required action.

18 July 2017

- 5.1.2 Components *(page 170)*
- 5.1.2 Highlights *(page 170)*
- 5.1.2 Changes and enhancements *(page 171)*
- 5.1.2 Resolved issues *(page 173)*
- 5.1.2 Known issues *(page 175)*
- 5.1.2 Cassandra enhancements *(page 176)*
- 5.1.2 General upgrade advice *(page 178)*

5.1.2 Components

- Apache Cassandra™ 3.11.0.1758 (updated)
- Apache Solr™ 6.0.1.0.1716 (updated)
- Apache Spark™ 2.0.2.6
- Apache Tomcat® 8.0.43 (updated)
- DataStax Spark Cassandra Connector 2.0.3 (updated)
- DSE Java Driver 1.2.2
- DSEFS 5.1.2 (updated)
- Netty 4.0.42.Final
- Spark Jobserver 0.6.2.234 (requires compatible API)
- TinkerPop 3.2.6-20170623-d59f0b40 (updated)
- Select Hadoop libraries

5.1.2 Highlights

Executive summary highlights for DSE 5.1.2:

- DataStax Enterprise core *(page 171)*
- DSE Analytics *(page 171)*
- DSE Graph *(page 171)*
• **DSE Search** *(page 171)*

The executive summary highlights are just a top-level view. Be sure to review all **5.1.2 Changes and enhancements** *(page 171).*

### 5.1.2 DataStax Enterprise core highlights

DataStax Enterprise 5.1.2 includes **CASSANDRA-13004** that fixes possible corruption while adding a column to a table or removing a column from a table. (DSP-13684)

This fix requires a messaging protocol version change to VERSION_3014. DataStax strongly recommends additional steps for the following upgrade paths:

<table>
<thead>
<tr>
<th>Upgrade from</th>
<th>Upgrade to</th>
<th>Upgrade steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0.0 through 5.0.8</td>
<td>5.1.2 and later</td>
<td>See the <strong>Upgrades from DSE 5.0.0 to 5.0.8 and from DSE 5.1.0 and 5.1.1 to DSE 5.1.2 only</strong> section in Upgrading from DataStax Enterprise 5.0 to 5.1.</td>
</tr>
<tr>
<td>5.1.0 through 5.1.1</td>
<td>5.1.2 and later</td>
<td>See <strong>Preparing to upgrade</strong> in <strong>Upgrades for DataStax Enterprise patch releases.</strong></td>
</tr>
</tbody>
</table>

### 5.1.2 DSE Analytics and DSEFS highlights

- DSE will not start if DSEFS is enabled (which is the default for all Analytics nodes in 5.1) and the DSEFS work directory or data directories are missing and cannot be created. In earlier releases, DSE would start but the Analytics nodes would experience hard-to-detect problems later on. (DSP-13238)
- DSEFS performance is improved when authorization is enabled. New dse.yaml advanced DSEFS options: `query_cache_size (page 341)` and `query_cache_expire_after_ms (page 341)` adjust the credential caching. (DSP-13107)

### 5.1.2 DSE Graph highlights

- Performance improvement: Gremlin script compilation. (DSP-12789)
- Significant improvement on vertex properties retrieval. (DSP-13467)
- Partitioned vertex tables (PVT) are deprecated. (DSP-13501)
- Graph Loader: Support loading geospatial data type. (DGL-225)

### 5.1.2 DSE Search highlights

- Re-indexing performance improvements. (DSP-13751), (DSP-12923)
- Fixes to solr indexing management tasks. (DSP-13778), (DSP-10088), (DSP-13793)

### 5.1.2 Changes and enhancements
In addition to the 5.1.2 Highlights (page 170), review all changes and enhancements:

- DataStax Enterprise core (page 172)
- DSE Analytics (page 172)
- DSE Graph (page 172)
- DSEFS (page 173)
- DSE Search (page 173)

5.1.2 DataStax Enterprise changes and enhancements

- Jackson Deserializer vulnerability. (DSP-13414)
- New nodetool sjk command for troubleshooting and monitoring that runs Swiss Java Knife (SJK) on the local node. (DSP-13544)
- Make o.a.c.metrics extend org.codahale.metrics to fix Metrics Reporter. (DSP-13840)
- Make sure to handle range queries while filtering. (DSP-13840).
- Allow mapping a single column to multiple SASI indexes. (DSP-13045)
- Properly evict pstmtms from prepared statements cache (DSP-13770).
- Add nodetool sequence batch functionality. (DSP-13770).
- Show correct protocol version in cqlsh (DSP-13544)
- null assertion in MemtablePostFlush. (DSP-13544)

5.1.2 DSE Analytics changes and enhancements

- When ALLOW_SPARK_HOME=true, support to specify a user-specific Spark home directory with the SPARK_HOME environment variable. (DSP-8100)
- Change lease manager log message to improve Spark Master troubleshooting. (DSP-12846)

5.1.2 DSE Graph changes and enhancements

- Specify file matching pattern for directory load. (DGL-177)
- Graph Loader: Support loading geospatial data type. (DGL-225)
- Improved error message when Spark submit has connection problems on initialization. (DSP-12632)
- Partitioned vertex tables (PVT) are deprecated. (DSP-13501)
- **A change is required** if more than 256 parameters are passed on a graph query request for TinkerPop drivers and drivers using 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 reduce script compilation times. Consider alternate methods for parameterizing scripts, like passing a single map. If the graph query request requires many arguments, pass a list. If you pass more than 256 parameters, increase the `max_query_params` (page 349) option in `dse.yaml`. (DSP-12789)
- Don't instantiate DseQueryHandler for each statement in graph. (DSP-13287)
- GraphSON 2.0 serialization performance enhancements. (DSP-13467)
• DSEFS keyspace visible in Spark SQL. (DSP-13510)
• Remove provisioning state during graph creation. Graph is either live or non-existing. (DSP-13686)
• Improve schema migration. Remove schema provisioning. (DSP-13665)

5.1.2 DSEFS changes and enhancements

• Improve authorization performance. New dse.yaml advanced DSEFS options: query_cache_size (page 341) and query_cache_expire_after_ms (page 341). (DSP-13107)
• Improve error message when DSEFS is low on storage space. (DSP-13324)
• DSEFS keyspace creation uses SimpleStrategy with replication factor of 1. After starting the cluster for the first time, you must alter the keyspace to use NetworkTopologyStrategy with proper RF. (DSP-12662)

5.1.2 DSE Search changes and enhancements

• rtOffheapPostings is present and true by default in demo and auto-generated solrconfig.xml files. (DSP-10088, DSP-13228)
• Repair-driven re-indexing is significantly faster because individual partition indexing tasks are executed in parallel. Override Cassandra's default post-repair index builder. (DSP-12923)
• The default filter cache settings are changed. (DSP-13153)
• The Tika functionality that is bundled with Apache Solr is deprecated. Instead, use the stand-alone Apache Tika project. (DSP-14002)

5.1.2 Resolved issues

Resolved issues for:
• DataStax Enterprise core (page 173)
• DSE Advanced Replication (page 173)
• DSE Analytics (page 174)
• DSEFS (page 174)
• DSE Graph (page 174)
• DSE Search (page 174)

5.1.2 DataStax Enterprise resolved issues

• CqlSlowLogPlugin can fail to determine the table name of a DropIndexStatement if the index was dropped already. (DSP-11811)
• Installer overrides for workload don’t work in No Services + Analytics. (DSP-13475)

5.1.2 DSE Advanced Replication resolved issues

None.
5.1.2 DSE Analytics resolved issues

- Default and provided Spark executor or driver JVM options could get jumbled. (DSP-12857)
- DSEFS min_free_space default value in dse.yaml is changed to 5 GB. (DSP-13178)
- Cannot interrupt Spark Shell when unable to connect to DSE and keeps retrying. (DSP-13339)
- Configuration connection for Spark applications should use a load balancing policy to choose only nodes that are running Spark in the target DC. (DSP-13325)
- When stopping Spark drivers and executors when a supervising DSE process dies, Spark executors might stay alive even after worker death due to a race condition. (DSP-13688)
- MultipleRetry policy may retry with an incorrect consistency level. (DSP-13542)

5.1.2 DSEFS resolved issues

- DSE will not start if DSEFS is enabled and fails to start due to a configuration problem. (DSP-13238)

5.1.2 DSE Graph resolved issues

- Graph loader loads entire graphshon and gryo files in to memory. (DGL-209)
- Properly parse dates from strings. (DSP-12259)
- Race condition can cause Spark Executor creation loop during DSE node shutdown. (DSP-12589)
- Order propertyKeys correctly in schema.describe(). (DSP-12761)
- Gremlin scripts taking a long time to compile. See required change (page 172) if more than 256 parameters are passed on a graph query request. (DSP-12789)
- gremlin-console isn't properly initialized when started in debug mode. (DSP-12900)
- Change ranking of indices so that Search index < Secondary Index < MV index. (DSP-13212)
- Graph profile() results should display CQL by default even in console. (DSP-13293)
- Cache empty result sets for queries that didn’t return elements. (DSP-13342)
- GraphFrames allow grouping by properties which can potentially be null. (DSP-13406)
- DseGraphFrame needs to be serializable for the spark-shell graph data export. (DSP-13427)
- Backward compatibility issue with .select().by() or local(). (DSP-13607)
- DseGraphFrame.updateEdges() insert single cardinality edges properly. (DSP-13865)
- Spark shell seems to hang indefinitely when running graph frame drop command. (DSP-13795)

5.1.2 DSE Search resolved issues

- Gremlin inside() function no longer uses search index. (DSP-13553)
- CREATE SEARCH INDEX fails with custom resources. (DSP-13778)
• Improved error message when running dse cassandra-stop when there are multiple DSE processes. (DSP-12938)
• Solr 2i invalidation deadlocks if invalidation runs with index unregistered. (DSP-13751)
• Auto-generation options need to be validated correctly. (DSP-13793)

5.1.2 DataStax Enterprise known issue

• DataStax Enterprise will not run with Java 1.8u161 or later. (DSP-15277)
• Potential data loss for INSERTs with very large TTLs. TTL expiration timestamps are susceptible to the year 2038 problem. (DSP-15412)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. TTLs are considered “very large” when close to the maximum allowed value of 630720000 seconds (20 years), starting from 2018-01-19T03:14:06+00:00. As time progresses, the maximum supported TTL is gradually reduced as the maximum expiration date approaches. For instance, on 2028-01-19T03:14:06 with a TTL of 10 years is impacted. The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately.

**Warning:** Upgrade to DSE 5.1.7 or later and take required action to protect against overflow of local expiration time.

Hadoop libraries

Built-in Hadoop and Bring-Your-Own-Hadoop (BYOH) were deprecated in DataStax Enterprise (DSE) 5.0, and were removed in DSE 5.1. Hadoop removal from DSE 5.1 and later means that DSE does not allow for the startup of Hadoop services previously included in DSE, including MapReduce JobTracker and TaskTracker.

However, DSE has supported built-in Spark since DSE 4.5 and Bring-Your-Own-Spark (BYOS) since DSE 5.0, and that support continues today. Because Spark depends on certain Hadoop libraries on the server and the client, DSE continues to ship with Hadoop libraries that are required for running Spark and BYOS.

To view the included Hadoop libraries, see [DataStax Enterprise 5.1.x third-party software](#).

dse.yaml

The location of the `dse.yaml` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
</tbody>
</table>
Cassandra enhancements for DSE 5.1.2

DataStax Enterprise (DSE) 5.1.2 includes all changes from earlier DSE releases. These production-certified changes are enhancements to Apache Cassandra™ 3.11.0. (For Cassandra updates, see CHANGES.txt.)

- Properly evict pstmts from prepared statements cache (CASSANDRA-13641)
- Allow different NUMACTL_ARGS to be passed in (CASSANDRA-13557)
- Fix secondary index queries on COMPACT tables (CASSANDRA-13627)
- Nodetool listsnapshots output is missing a newline, if there are no snapshots (CASSANDRA-13568)
- Fix toJSONArray for the UDT, tuple and collection types (CASSANDRA-13592)
- Fix nested Tuples/UDTs validation (CASSANDRA-13646)
- Replace string comparison with regex/number checks in MessagingService test (CASSANDRA-13216)
- Fix formatting of duration columns in CQLSH (CASSANDRA-13549)
- Ensure int overflow doesn't occur when calculating large partition warning size (CASSANDRA-13172)
- Ensure consistent view of partition columns between coordinator and replica in ColumnFilter (CASSANDRA-13004)
- Failed unregistering mbean during drop keyspace (CASSANDRA-13346)
- nodetool scrub/cleanup/upgradesstables exit code is wrong (CASSANDRA-13542)
- Fix the reported number of sstable data files accessed per read (CASSANDRA-13120)
- Fix schema digest mismatch during rolling upgrades from versions before 3.0.12 (CASSANDRA-13559)
- Upgrade JNA version to 4.4.0 (CASSANDRA-13072)
- Interned ColumnIdentifiers should use minimal ByteBuffers (CASSANDRA-13533)
- ReverseIndexedReader may drop rows during 2.1 to 3.0 upgrade (CASSANDRA-13525)
- Fix repair process violating start/end token limits for small ranges (CASSANDRA-13052)
- Nodes started with join_ring=False should be able to serve requests when authentication is enabled (CASSANDRA-11381)
- cqlsh COPY FROM: increment error count only for failures, not for attempts (CASSANDRA-13209)
- Fix the problem with duplicated rows when using paging with SASI (CASSANDRA-13302)
- Allow CONTAINS statements filtering on the partition key and it’s parts (CASSANDRA-13275)
- Fall back to even ranges calculation in clusters with vnodes when tokens are distributed unevenly (CASSANDRA-13229)
- Fix duration type validation to prevent overflow (CASSANDRA-13218)
Forbid unsupported creation of SASI indexes over partition key columns (CASSANDRA-13228)
Reject multiple values for a key in CQL grammar. (CASSANDRA-13369)
UDA fails without input rows (CASSANDRA-13399)
Fix compaction-stress by using daemonInitialization (CASSANDRA-13188)
V5 protocol flags decoding broken (CASSANDRA-13443)
Use write lock not read lock for removing sstables from compaction strategies. (CASSANDRA-13422)
Use corePoolSize equal to maxPoolSize in JMXEnabledThreadPoolExecutors (CASSANDRA-13329)
Avoid rebuilding SASI indexes containing no values (CASSANDRA-12962)
Add charset to Analyser input stream (CASSANDRA-13151)
Delete illegal character from StandardTokenizerImpl.jflex (CASSANDRA-13417)
Fix cqlsh automatic protocol downgrade regression (CASSANDRA-13307)
Tracing payload not passed from QueryMessage to tracing session (CASSANDRA-12835)
Add storage port options to sstableloader (CASSANDRA-13518)
Properly handle quoted index names in cqlsh DESCRIBE output (CASSANDRA-12847)
Avoid reading static row twice from old format sstables (CASSANDRA-13236)
Fix NPE in StorageService.excise() (CASSANDRA-13163)
Expire OutboundTcpConnection messages by a single Thread (CASSANDRA-13265)
Fail repair if insufficient responses received (CASSANDRA-13397)
Fix SSTableLoader fail when the loaded table contains dropped columns (CASSANDRA-13276)
Avoid name clashes in CassandraIndexTest (CASSANDRA-13427)
Handling partially written hint files (CASSANDRA-12728)
Interrupt replaying hints on decommission (CASSANDRA-13308)
Handling partially written hint files (CASSANDRA-12728)
Fix NPE issue in StorageService (CASSANDRA-13060)
Make reading of range tombstones more reliable (CASSANDRA-12811)
Fix startup problems due to schema tables not completely flushed (CASSANDRA-12213)
Fix view builder bug that can filter out data on restart (CASSANDRA-13405)
Fix 2i page size calculation when there are no regular columns (CASSANDRA-13400)
Fix the conversion of 2.X expired rows without regular column data (CASSANDRA-13395)
Fix hint delivery when using ext+internal IPs with prefer_local enabled (CASSANDRA-13020)
Nodetool upgradessstables/scrub/compact ignores system tables (CASSANDRA-13410)
Fix schema version calculation for rolling upgrades (CASSANDRA-13441)
Avoid starting gossiper in RemoveTest (CASSANDRA-13407)
Fix weightedSize() for row-cache reported by JMX and NodeTool (CASSANDRA-13393)
DataStax Enterprise 5.1 release notes

- Fix JVM metric names (CASSANDRA-13103)
- Coalescing strategy sleeps too much (CASSANDRA-13090)
- Fix 2ndary index queries on partition keys for tables with static columns (CASSANDRA-13147)
- Fix ParseError unhashable type list in cqlsh copy from (CASSANDRA-13364)

General upgrade advice for DSE 5.1.2

Carefully review all planning and upgrade documentation in the Upgrading DataStax Enterprise guide. The following provides general upgrade information:

GENERAL UPGRADING ADVICE FOR ANY VERSION
========================================

Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to ALWAYS snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

3.11.1
======

Upgrading
--------
- Nothing specific to this version but please see previous upgrading sections,
  especially if you are upgrading from 2.2.

3.11.0
======

Upgrading
--------
- ALTER TABLE (ADD/DROP COLUMN) operations concurrent with a read might result into data corruption (see CASSANDRA-13004 for more details). Fixing this bug required a messaging protocol version bump. By default, Cassandra 3.11 will use 3014 version for messaging.

Since Schema Migrations rely the on exact messaging protocol version match between nodes, if you need schema changes during the upgrade process, you have to start your nodes with `-Dcassandra.force_3_0_protocol_version=true`
first, in order to temporarily force a backwards compatible protocol.

After the whole cluster is upgraded to 3.11, do a rolling restart of the cluster without setting that flag.

3.11 nodes with and without the flag set will be able to do schema migrations with other 3.x and 3.0.x releases.

While running the cluster with the flag set to true on 3.11 (in compatibility mode), avoid adding or removing any columns to/from existing tables.

If your cluster can do without schema migrations during the upgrade time, just start the cluster normally without setting aforementioned flag.

If you are upgrading from 3.0.14+ (of 3.0.x branch), you do not have to set an flag while upgrading to ensure schema migrations.
- 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).

3.10
====

New features
------------
- New 'DurationType' (cql duration). See CASSANDRA-11873
- Runtime modification of concurrent_compactors is now available via nodetool
- 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.
- Compaction can now take into account overlapping tables that don't take part
  in the compaction to look for deleted or overwritten data in the compacted tables.
  Then such data is found, it can be safely discarded, which in turn should enable
  the removal of tombstones over that data.

The behavior can be engaged in two ways:
- as a "nodetool garbagecollect -g CELL/ROW" operation, which applies
  single-table compaction on all sstables to discard deleted data in one step.
- as a "provide_overlapping_tombstones:CELL/ROW/NONE" compaction strategy flag,
  which uses overlapping tables as a source of deletions/overwrites during all
  compactions.
  The argument specifies the granularity at which deleted data is to be found:
  - If ROW is specified, only whole deleted rows (or sets of rows) will be discarded.
  - If CELL is specified, any columns whose value is overwritten or deleted
    will also be discarded.
  - NONE (default) specifies the old behavior, overlapping tables are not used to
decide when to discard data.
Which option to use depends on your workload, both ROW and CELL increase the disk load on compaction (especially with the size-tiered compaction strategy), with CELL being more resource-intensive. Both should lead to better read performance if deleting rows (resp. overwriting or deleting cells) is common.
- 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.

Upgrading
---------
- 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 appropiate 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 litterals support the ISO 8601 format. By consequence, identifiers matching that format
DataStax Enterprise 5.1 release notes

Spark Cassandra Connector SCC-CHANGES.txt for DSE 5.1.2

A list of DataStax Enterprise 5.1.2 production-certified changes for the DataStax Spark Cassandra Connector 2.0.3.

DSE 5.1.2:
2.0.3
* Includes patches up to 1.6.8

DSE 5.1.1:
2.0.2
* Protect against Size Estimate Overflows (SPARKC-492)
* Add java.time classes support to converters and sparkSQL(SPARKC-491)
* Allow Writes to Static Columns and Partition Keys (SPARKC-470)

DSE 5.1.0:
2.0.1
* Refactor Custom Scan Method (SPARKC-481)

2.0.0
* Upgrade driver version for 2.0.0 Release to 3.1.4 (SPARKC-474)
* Extend SPARKC-383 to All Row Readers (SPARKC-473)

2.0.0 RC1
* Includes all patches up to 1.6.5
* Automatic adjustment of Max Connections (SPARKC-471)
* Allow for Custom Table Scan Method (SPARKC-459)
* Enable PerPartitionLimit (SPARKC-446)
* Support client certificate authentication for two-way SSL Encryption (SPARKC-359)
* Change Config Generation for Cassandra Runners (SPARKC-424)
* Remove deprecated QueryRetryDelay parameter (SPARKC-423)
* User ConnectionHostParam.default as default hosts String
* Update usages of deprecated SQLContext so that SparkSession is used instead (SPARKC-400)
* Test Reused Exchange SPARK-17673 (SPARKC-429)
* Module refactoring (SPARKC-398)
* Recognition of Java Driver Annotated Classes (SPARKC-427)
* RDD.deleteFromCassandra (SPARKC-349)
* Coalesce Pushdown to Cassandra (SPARKC-161)
* Custom Conf options in Custom Pushdowns (SPARKC-435)
* Upgrade CommonBeatUtils to 1.9.3 to Avoid SID-760 (SPARKC-457)

2.0.0 M3
* Includes all patches up to 1.6.2

2.0.0 M2
* Includes all patches up to 1.6.1
2.0.0 M1

* Added support for left outer joins with Cassandra table (SPARKC-181)
* Removed CassandraSqlContext and underscore based options (SPARKC-399)
* Upgrade to Spark 2.0.0-preview (SPARKC-396)
  - Removed Twitter demo because there is no spark-streaming-twitter package available anymore
  - Removed Akka Actor demo because there is no support for such streams anymore
  - Bring back Kafka project and make it compile
  - Update several classes to use our Logging instead of Spark Logging because Spark Logging became private
  - Update few components and tests to make them work with Spark 2.0.0
  - Fix Spark SQL - temporarily
  - Update plugins and Scala version

---

DSE 5.1.1

**Important:** DataStax recommends the latest patch release. The latest version of DataStax Enterprise 5.1 is 5.1.11. Due to Potential data loss for INSERTs with very large TTLs. (DSP-15412) (page 203), DataStax does not recommend DSE 5.1.0-5.1.2 for production.

**Attention:** TTL expiration timestamps are susceptible to the year 2038 problem. If the TTL value is long and an expiration date is greater than the maximum threshold of 2038-01-19T03:14:06+00:00, the data is immediately expired and purged on the next compaction. When using a long TTL, DataStax strongly recommends upgrading to DSE 5.1.7 or later and taking required action.

23 May 2017

- 5.1.1 Components (page 184)
- 5.1.1 Highlights (page 185)
- 5.1.1 Changes and enhancements (page 185)
- 5.1.1 Resolved issues (page 186)
- 5.1.1 Known issues (page 189)
- 5.1.1 Cassandra enhancements (page 190)
- 5.1.1 General upgrade advice (page 191)

5.1.1 Components

- Apache Cassandra™ 3.10.0.1695 (updated)
- Apache Solr™ 6.0.1.0.1705 (updated)
- Apache Spark™ 2.0.2.6
- Apache Tomcat® 8.0.43 (updated)
- DataStax Spark Cassandra Connector 2.0.2 (updated)
- DSE Java Driver 1.2.2
- DSEFS 5.1.26 (updated)
• Netty 4.0.42.Final
• Spark Jobserver 0.6.2.234 (requires compatible API)
• TinkerPop 3.2.5-20170321-f3032b39 (updated)
• Select Hadoop libraries

5.1.1 Highlights

Executive summary highlights for DSE 5.1.1:

• DSE Analytics *(page 185)*
• DSE Graph *(page 185)*
• DSE Search *(page 185)*

The executive summary highlights are just a top-level view. Be sure to review all release notes.

**DSE Analytics and DSEFS highlights**

DSE 5.1.1 improves the reliability of Spark workers reconnecting when the Spark Master changes to a different node. For example, if the current master node goes down. Although this scenario was rarely encountered, it would sometimes require running a command to restart the Spark workers. The affected versions are DSE 5.0.7 and 5.1.0. (DSP-11306)

**DSE Graph highlights**

DSE 5.1.1 highlights include:

• Failing OLAP queries if meta-properties were used in graph schema. (DSP-13016)
• Script synchronization to prevent multiple threads trying to compile the same Gremlin script. In multi-threaded scenarios, Gremlin scripts would hang. (DSP-12814)

**DSE Search highlights**

Skip DSE 5.1.0 and upgrade directly to DSE 5.1.1 if you:

• Use the HTTP interface. (DSP-13318), (DSP-13270)
• Have a Thrift column family backing an active Solr core. (DSP-13019)
• Use TTL to expire data. (DSP-12960)
• Use index encryption. (DSP-13155), (DSP-12620)
• Use live indexing. (DSP-12040), (DSP-12941)

5.1.1 Changes and enhancements

In addition to the **5.1.1 Highlights *(page 185)***, review all changes and enhancements:

• DataStax Enterprise core *(page 186)*
• DSE Advanced Replication *(page 186)*
• DSE Analytics *(page 186)*
• DSE Graph *(page 186)*
• DSEFS *(page 186)*
• **DSE Search** (*page 186*)

### 5.1.1 DataStax Enterprise changes and enhancements

- Security fix with commons-collections4 version 4.1 due to **CVE-2015-6420**. (DSP-13060)
- Guard mapped memory accesses with an assertion instead of causing a segmentation fault in JVM. (DSP-13344)

### 5.1.1 DSE Advanced Replication changes and enhancements

- Increased robustness of CDC processor. (DSP-12852)
- Add **audit log** (*page 1007*) compression parameter. (DSP-12949)

### 5.1.1 DSE Analytics changes and enhancements

- Spark Cassandra Connector should make DseSession compatible sessions. (DSP-12737)

### 5.1.1 DSE Graph changes and enhancements

- Make explicit parameter for setting tmp dir for mapdb and netty. (DGL-167)
- Support recursive loading of directories. (DGL-172)
- Remove double cluster client in ClusterBuilder. Instead, use a single client and configure the CL in a {{SimpleGraphStatement}} for creating the graph. (DGL-183)
- VertexInputRDD.getOrCreateVertex method performance improvement; Graph OLAP query running time reduced by ~10%. (DSP-12782)
- DseGraphFrames library is included in com.datastax.dse:dse-spark-dependencies to support application build. (DSP-13074)

### 5.1.1 DSEFS changes and enhancements

- Local node is preferred for placing new data blocks to save network bandwidth usage by DSEFS. (DSP-12746)

### 5.1.1 DSE Search changes and enhancements

- Solr demos updated to use CQL index management to create cores. (DSP-11451)
- Runtime node blacklisting for distributed search queries; the EndpointStateTracker MBean now has Blacklisted boolean attribute. (DSP-12965)
- Display reindexing progress with **dsetool core_indexing_status** (*page 1027*) --progress option. (DSP-12617)
- Support for indexing frozen sets and lists of native and user-defined (tuple/UDT) element types. Indexing frozen maps is not supported. (DSP-12983)

### 5.1.1 Resolved issues

Resolved issues for:

- **DataStax Enterprise core** (*page 187*)
5.1.1 DataStax Enterprise resolved issues

- dsetool logs clear credentials on logs. (DSP-12985)
- Plain text authentication handled incorrectly in DseAuthenticator causes performance degradation. (DSP-13201)
- Installer deletes user directories under /etc/dse/conf during upgrade to 5.1. (DSP-13296)
- SafeNet/KMIP authentication failure via LDAP. (DSP-12739)
- CVE-2012-2098 vulnerability in Apache Ant Core 1.7.0. (DSP-12925)

5.1.1 DSE Advanced Replication resolved issues

- Error while refreshing configuration. (DSP-13148)
- In flight Advanced Replication mutations are not encrypted when commitlog encryption is enabled. (DSP-12961)
- MutationFileSource fails when a transmission file is not found. (DSP-11633)
- AdvRep channel status NPE. (DSP-12522)
- AdvRep CLI metrics list output showing negative message count. (DSP-12788)
- advrep log count Serializer Not Defined Error MultiNode. (DSP-13032)

5.1.1 DSE Analytics resolved issues

- On start, Spark worker registers with master that is then changed, but doesn't reregister with new master. (DSP-11306)
- A new CQL type tinyint (page 548). (DSP-11940)
- When DSE node with Spark Master gracefully shuts down at the same time that an application is submitted or stopped, Spark Master fails to save the recovery storage information. (DSP-12795)
- Weather sensor demo website not graphing all data values. (DSP-13041)
- Extra unnecessary messages when starting Spark shell. (DSP-13239)
- The spark-submit --driver-class-path option does not place a jar only on the Driver Classpath. (DSP-13289)

5.1.1 DSEFS resolved issues

- DSEFS memory leaks. (DSP-13023)
- Cannot write file to WebHDFS REST interface with Spark. (DSP-13154)

5.1.1 DSE Graph resolved issues

- Support secondary indexes. (DGL-202)
• DGL creates duplicate edges when rerunning when using custom ids. (DGL-205)
• Properties with empty strings are skipped. New graph loader `-skip_blank_values (page 896)` option. (DGL-215)
• Tab-delimited data cannot be read correctly with File.text. (DGL-222)
• RangeStep fails when used with negative values. (DSP-11671)
• Logging level in DigestTokensManager lowered from INFO to DEBUG. (DSP-12234)
• Decimal type does not work, for both read and write, when reading a graph from Spark. (DSP-12299)
• Comparing IDs of newly created elements with normal elements causes a class cast exception. (DSP-12738)
• Allow graph.allow_scan to be set on tx level. (DSP-12794)
• Improve handling of ASM "Method code too large" exception when processing large Gremlin script. (DSP-12802)
• Many threads get stuck compiling the same script. (DSP-12814)
• Check that a new ID given to a schema element has not already been used. (DSP-12826)
• Optimize solr .within() queries correctly. (DSP-12830)
• Vertex properties without meta-properties defined in schema create invalid RDD data. (DSP-13016)
• OLAP case sensitivity for edges and meta-properties. (DSP-13085)
• Exception thrown when attempting to read IDs of vertices retrieved through a full-graph scan. (DSP-13210)
• Graph should start listening to schema updates only after DSE system keyspace is set up. (DSP-13251)
• DseGraphFrame fail with UUID as a custom id. (DSP-13302)

### 5.1.1 DSE Search resolved issues

• Remove `<dataDir>` option from solrConfig files in demo apps. (DSP-9402)
• CQL Search queries time out when a column has a colon (:) in it. Solr field name policy applies to DSE Search field names (page 510). (DSP-11296)
• Make TimeUUIDField epoch not platform-dependent. (DSP-11424)
• Term vector (TV) file epoch not platform-dependent. (DSP-12040)
• DistributedRequestException isn't created with a detail message. (DSP-12493)
• BlockCache corruption with high concurrency. (DSP-12620)
• Poor performance when searching with UDT sub-fields. (DSP-12812)
• Better TTL logging. (DSP-12885)
• Term frequency inconsistencies in RT. (DSP-12941)
• The TTL task is never de-scheduled. (DSP-12960)
• Cannot reload core after Thrift table upgrade. (DSP-13019)
• Solr listens only on port 8080 regardless of configuration. (DSP-13187)
• Solr is accepting HTTP requests before all cores have loaded. (DSP-13270)
• Excessive StatefulEncryptorAdapter usage by evicting StatefulEncryptorAdapter cache when index output gets closed. (DSP-13155)
• Upgrade Tomcat to 8.0.43 to fix CVE-2016-8735 and other security issues. (DSP-13318)

5.1.1 DataStax Enterprise known issue

• DataStax Enterprise will not run with Java 1.8u161 or later. (DSP-15277)
• Potential data loss for INSERTs with very large TTLs. TTL expiration timestamps are susceptible to the year 2038 problem. (DSP-15412)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. TTLs are considered "very large" when close to the maximum allowed value of 630720000 seconds (20 years), starting from 2018-01-19T03:14:06+00:00. As time progresses, the maximum supported TTL is gradually reduced as the maximum expiration date approaches. For instance, on 2028-01-19T03:14:06 with a TTL of 10 years is impacted. The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately.

Warning: Upgrade to DSE 5.1.7 or later and take required action to protect against overflow of local expiration time.

Hadoop libraries

Built-in Hadoop and Bring-Your-Own-Hadoop (BYOH) were deprecated in DataStax Enterprise (DSE) 5.0, and were removed in DSE 5.1. Hadoop removal from DSE 5.1 and later means that DSE does not allow for the startup of Hadoop services previously included in DSE, including MapReduce JobTracker and TaskTracker.

However, DSE has supported built-in Spark since DSE 4.5 and Bring-Your-Own-Spark (BYOS) since DSE 5.0, and that support continues today. Because Spark depends on certain Hadoop libraries on the server and the client, DSE continues to ship with Hadoop libraries that are required for running Spark and BYOS.

To view the included Hadoop libraries, see DataStax Enterprise 5.1.x third-party software.

dse.yaml
The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
</tbody>
</table>
Tarball installations
Installer-No Services installations

installation_location/
resources/dse/conf/dse.yaml

Cassandra enhancements for DSE 5.1.1

DataStax Enterprise (DSE) 5.1.1 includes all changes from earlier DSE releases. These production-certified changes are enhancements to Apache Cassandra™ 3.10.0. (For Cassandra updates, see CHANGE.txt.)

- Fix the problem with duplicated rows when using paging with SASI (CASSANDRA-13302)
- Allow CONTAINS statements filtering on the partition key and it’s parts (CASSANDRA-13275)
- Fall back to even ranges calculation in clusters with vnodes when tokens are distributed unevenly (CASSANDRA-13229)
- Fix duration type validation to prevent overflow (CASSANDRA-13218)
- Forbid unsupported creation of SASI indexes over partition key columns (CASSANDRA-13228)
- Reject multiple values for a key in CQL grammar. (CASSANDRA-13369)
- UDA fails without input rows (CASSANDRA-13399)
- Fix compaction-stress by using daemonInitialization (CASSANDRA-13188)
- V5 protocol flags decoding broken (CASSANDRA-13443)
- Use write lock not read lock for removing sstables from compaction strategies. (CASSANDRA-13422)
- Use corePoolSize equal to maxPoolSize in JMXEnabledThreadPoolExecutors (CASSANDRA-13329)
- Avoid rebuilding SASI indexes containing no values (CASSANDRA-12962)
- Add charset to Analyser input stream (CASSANDRA-13151)
- Delete illegal character from StandardTokenizerImpl.jflex (CASSANDRA-13417)
- Fix cqlsh automatic protocol downgrade regression (CASSANDRA-13307)
- Tracing payload not passed from QueryMessage to tracing session (CASSANDRA-12835)
- Add storage port options to sstableloader (CASSANDRA-13518)
- Properly handle quoted index names in cqlsh DESCRIBE output (CASSANDRA-12847)
- Avoid reading static row twice from old format sstables (CASSANDRA-13236)
- Fix NPE in StorageService.excise() (CASSANDRA-13163)
- Expire OutboundTcpConnection messages by a single Thread (CASSANDRA-13265)
- Fail repair if insufficient responses received (CASSANDRA-13397)
- Fix SSTableLoader fail when the loaded table contains dropped columns (CASSANDRA-13276)
- Avoid name clashes in CassandraIndexTest (CASSANDRA-13427)
- Handling partially written hint files (CASSANDRA-12728)
- Interrupt replaying hints on decommission (CASSANDRA-13308)
- Handling partially written hint files (CASSANDRA-12728)
• Fix NPE issue in StorageService (CASSANDRA-13060)
• Make reading of range tombstones more reliable (CASSANDRA-12811)
• Fix startup problems due to schema tables not completely flushed (CASSANDRA-12213)
• Fix view builder bug that can filter out data on restart (CASSANDRA-13405)
• Fix 2i page size calculation when there are no regular columns (CASSANDRA-13400)
• Fix the conversion of 2.X expired rows without regular column data (CASSANDRA-13395)
• Fix hint delivery when using ext+internal IPs with prefer_local enabled (CASSANDRA-13020)
• Nodetool upgradesstables/scrub/compact ignores system tables (CASSANDRA-13410)
• Fix schema version calculation for rolling upgrades (CASSANDRA-13441)
• Avoid starting gossiper in RemoveTest (CASSANDRA-13407)
• Fix weightedSize() for row-cache reported by JMX and NodeTool (CASSANDRA-13393)
• Fix JVM metric names (CASSANDRA-13103)
• Coalescing strategy sleeps too much (CASSANDRA-13090)
• Fix 2ndary index queries on partition keys for tables with static columns (CASSANDRA-13147)
• Fix ParseError unhashable type list in cqlsh copy from (CASSANDRA-13364)

General upgrade advice for DSE 5.1.1

Carefully review all planning and upgrade documentation in the Upgrading DataStax Enterprise guide. The following provides general upgrade information:

GENERAL UPGRAADING ADVICE FOR ANY VERSION
============================================
Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to ALWAYS snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

3.11.0
------

Upgrading
--------
  - 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.

3.10
====

New features
-------------
- New `DurationType` (cql duration). See CASSANDRA-11873
- Runtime modification of concurrent_compactors is now available via nodetool
- 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.
- Compaction can now take into account overlapping tables that don't take part in the compaction to look for deleted or overwritten data in the compacted tables.
  Then such data is found, it can be safely discarded, which in turn should enable the removal of tombstones over that data.

  The behavior can be engaged in two ways:
  - as a "nodetool garbagecollect -g CELL/ROW" operation, which applies
single-table compaction on all sstables to discard deleted data in one step.
- as a "provide_overlapping_tombstones:CELL/ROW/NONE" compaction strategy flag,
  which uses overlapping tables as a source of deletions/overwrites during all compactions.

The argument specifies the granularity at which deleted data is to be found:
- If ROW is specified, only whole deleted rows (or sets of rows) will be discarded.
- If CELL is specified, any columns whose value is overwritten or deleted will also be discarded.
- NONE (default) specifies the old behavior, overlapping tables are not used to decide when to discard data.

Which option to use depends on your workload, both ROW and CELL increase the disk load on compaction (especially with the size-tiered compaction strategy), with CELL being more resource-intensive. Both should lead to better read performance if deleting rows (resp. overwriting or deleting cells) is common.
- 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.

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

Spark Cassandra Connector changes for DSE 5.1.1

A list of DataStax Enterprise 5.1.1 production-certified changes for the DataStax Spark Cassandra Connector.

DSE 5.1.1:
2.0.2
  * Protect against Size Estimate Overflows (SPARKC-492)
  * Add java.time classes support to converters and sparkSQL(SPARKC-491)
  * Allow Writes to Static Columnns and Partition Keys (SPARKC-470)

DSE 5.1.0:
2.0.1
  * Refactor Custom Scan Method (SPARKC-481)

2.0.0
  * Upgrade driver version for 2.0.0 Release to 3.1.4 (SPARKC-474)
  * Extend SPARKC-383 to All Row Readers (SPARKC-473)

2.0.0 RC1
  * Includes all patches up to 1.6.5
  * Automatic adjustment of Max Connections (SPARKC-471)
  * Allow for Custom Table Scan Method (SPARKC-459)
  * Enable PerPartitionLimit (SPARKC-446)
  * Support client certificate authentication for two-way SSL Encryption (SPARKC-359)
DataStax Enterprise 5.1 release notes

* Change Config Generation for Cassandra Runners (SPARKC-424)
* Remove deprecated QueryRetryDelay parameter (SPARKC-423)
* User ConnectionHostParam.default as default hosts String
* Update usages of deprecated SQLContext so that SparkSession is used instead (SPARKC-400)
* Test Reused Exchange SPARK-17673 (SPARKC-429)
* Module refactoring (SPARKC-398)
* Recognition of Java Driver Annotated Classes (SPARKC-427)
* RDD.deleteFromCassandra (SPARKC-349)
* Coalesce Pushdown to Cassandra (SPARKC-161)
* Custom Conf options in Custom Pushdowns (SPARKC-435)
* Upgrade CommonBeatUtils to 1.9.3 to Avoid SID-760 (SPARKC-457)

2.0.0 M3
* Includes all patches up to 1.6.2

2.0.0 M2
* Includes all patches up to 1.6.1

2.0.0 M1
* Added support for left outer joins with Cassandra table (SPARKC-181)
* Removed CassandraSqlContext and underscore based options (SPARKC-399)
* Upgrade to Spark 2.0.0-preview (SPARKC-396)
  - Removed Twitter demo because there is no spark-streaming-twitter package available anymore
  - Removed Akka Actor demo because there is no support for such streams anymore
  - Bring back Kafka project and make it compile
  - Update several classes to use our Logging instead of Spark Logging because Spark Logging became private
  - Update few components and tests to make them work with Spark 2.0.0
  - Fix Spark SQL - temporarily
  - Update plugins and Scala version

DSE 5.1.0

**Important:** DataStax recommends the latest patch release. The latest version of DataStax Enterprise 5.1 is 5.1.11. Due to Potential data loss for INSERTs with very large TTLs. (DSP-15412) (page 203), DataStax does not recommend DSE 5.1.0-5.1.2 for production.

**Attention:** TTL expiration timestamps are susceptible to the year 2038 problem. If the TTL value is long and an expiration date is greater than the maximum threshold of 2038-01-19T03:14:06+00:00, the data is immediately expired and purged on the next compaction. When using a long TTL, DataStax strongly recommends upgrading to DSE 5.1.7 or later and taking required action.

18 April 2017

- 5.1.0 Components (page 197)
- 5.1.0 New features (page 197)
• 5.1.0 Experimental features (page 197)
• 5.1.0 Changes and enhancements (page 197)
• 5.1.0 Known issues (page 203)
• 5.1.0 Resolved issues (page 204)
• 5.1.0 Cassandra enhancements (page 206)
• 5.1.0 General upgrade advice (page 208)

5.1.0 Components

• Apache Cassandra™ 3.10.0.1652
• Apache Solr™ 6.0.1.0.1596
• Apache Spark™ 2.0.2.6
• Apache Tomcat® 8.0.37
• DataStax Spark Cassandra Connector (page 220) 2.0.1
• DSE Java Driver 1.2.2
• DSEFS 5.1.24
• Netty 4.0.42.Final
• Spark Jobserver 0.6.2.234 (requires compatible API)
• TinkerPop 3.2.5-20170222-de2f4034
• Select Hadoop libraries

5.1.0 New features

See DataStax Enterprise 5.1 new features (page 12).

5.1.0 Experimental features

These features are experimental. DataStax does not support these experimental features for production:

• Partitioned vertex tables (PVT) for handling supernodes in DSE Graph.

  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.

• Importing graphs using DseGraphFrame (page 926).
• The dsetool index_checks use an Apache Lucene® experimental feature.
• SASI indexes.
• Structured streaming operations to and from DSEFS use a Spark ALPHA feature.
• A DSEFS file system that spans multiple data centers.
• Labs features in OpsCenter.

5.1.0 Changes and enhancements

Changes and enhancements for:

• DataStax Enterprise core (page 198)
• DSE Advanced Replication (page 199)
5.1.0 DataStax Enterprise changes and enhancements

- Add **proxy authentication** to DSE authentication model. (DSP-3800), (DSP-8467)
- TimeWindowCompactionStrategy (**TWCS**) is set on dse_perf tables. To use TWCS on tables that were created in earlier releases, alter the tables after **upgrade to DSE 5.1**. (DSP-5560)
- **MemoryOnlyStrategy** works with compression. (DSP-6715)
- Add metrics for **dropped mutations** in Performance Object. (DSP-7936)
- DSE server startup time is improved. (DSP-9545)
- DateTieredStorageStrategy is deprecated. Use **TimeWindowStorageStrategy** instead. (DSP-9740)
- Add tab completion to cqlsh for DSE custom compaction strategies. (DSP-9864)
- Slow query log includes trace ID. (DSP-10055)
- Support for setting **row-level permissions**. Setting row-level permissions with row-level access control (RLAC) is not supported for use with DSE Search or DSE Graph. (DSP-10093)
- For G1GC the **max heap size** cap increased from 8192 MB to 32765 MB. See also **Java performance tuning**. (DSP-10459)
- Change compaction strategy used by CassandraAuditWriter. (DSP-11508)
- Implement dsetool command (**page 1057**) for printing most recent **slowest queries**. (DSP-11152)
- Improved performance and changed defaults (**page 331**) for CQL **slow query** logs. (DSP-11171)
- **Upgrades to DataStax Enterprise 5.1** are supported only from DataStax Enterprise 5.0. Upgrades from earlier versions require an interim upgrade to DSE 5.0. (DSP-11281)
- The default authenticator is DseAuthenticator and default authorizer is DseAuthorizer in cassandra.yaml. Review and adjust your security settings after upgrading to DSE 5.1. (DSP-12211)
- Authenticators other than DseAuthenticator and authorizers other than DseAuthorizer were deprecated in DSE 5.0; in DSE 5.1 some security features might not work correctly if other authenticators or authorizers are used. (DSP-12542)
- Improved help for CQL and cqlsh commands. (DSP-12845)

In cqlsh, type **help** to list all available topics. Type **help name** to find out more about the **name command**. For example, **help CAPTURE** or **help ALTER_KEYSPACE**.

- Only perform drop below RF check on decommission for non-partitioned keyspaces. (DSP-13054)
- Fix SmallInt and TinyInt serialization. (DSP-12916)
• Check for null/empty password before calling legacyAuthenticate from CassandraLoginModule. (DSP-8573)
• Allow registering user expression on SELECT statement. (DSP-12549)
• Apply request timeout in cqlsh COPY correctly, after upgrading to execution profiles. (DSP-12698)
• Update Java driver to DSE driver version 1.2.0-eap5. (DSP-11964)
• Fix AssertionError in continuous driver paging request on select count(*) query. (DSP-11964)
• Update internal DSE driver and fix formatting for Duration type. (DSP-11964)
• Replace open source Python driver with DataStax Enterprise driver. (DSP-11964)
• Fix OutOfSpaceTest. (DSP-12239)
• Allow to add index restrictions to SELECT in an immutable way. (DSP-12239)
• Allow grammar extensions to be added to cqlsh for tab completion. (DSP-12150)
• Improve compaction performance. (DSP-11695)
• Add client warning to SASI index. (DSP-11695)
• Add support for UNSET values to cqlsh COPY FROM command. (DSP-11695)
• Improve error message for incompatible authentication and authorization configuration. (DSP-11695)
• Implement optimized continuous paging. (DSP-11695)
• Added show-queries, query-log-file, and no-progress log options to cassandra-stress. (DSP-9476).
• Allow large partition generation in cassandra-stress user mode. (DSP-9476)
• Optimize variable sized integer (VIntCoding) and DataOutputStreamPlus interface using a ByteBuffer to stage writes (BufferedDataOutputStreamPlus). (DSP-9476)
• Improve metrics and reduce overhead under resource contention. (DSP-9476)
• Performance improvement: Make SinglePartitionReadCommand::queriesMulticellType() faster. (DSP-9476)
• Accept internal resource name in GRANT/REVOKE statements. (DSP-11746)
• Improve StatementRestrictions::getPartitionKeys() execution speed. (DSP-11724)
• Move responsibility for qualifying keyspace in authorization statements to IResource. (DSP-11588)
• Insert default superuser role with fixed timestamp. (DSP-11600)
• Make permissions extensible. (DSP-11600)
• Make IResource more easily extensible. (DSP-11600)
• Add method to IAuthenticator to login by user, as well as by role. (DSP-11600)
• Add private protocol version. (DSP-11535)

5.1.0 DSE Advanced Replication changes and enhancements

DSE Advanced Replication (V2) is CDC based and provides substantial improvements. CDC must be enabled in Cassandra. Migration from DSE 5.0 Advanced Replication (V1) to DSE 5.1 Advanced Replication (V2) is required.

• DSE Advanced Replication certified for use with DSE Multi-Instance. (DSP-10738)
• Support replication to multiple clusters. (DSP-8352)
• Support multi-DC edge (source) cluster configurations. (DSP-8744)
• Implement DSE Advanced Replication using Cassandra CDC (Change data capture). (DSP-9822)
• Support for setting row-level permissions. (DSP-10727)
  Row-level access control (RLAC) security on the destination cluster. (DSP-10893)
• Added support for migration. Migration from DSE 5.0 Advanced Replication (V1) to DSE 5.1 Advanced Replication (V2). (DSP-12280)
• Performance metrics (page 1005) enhancements, including gauge metric type and Transmission group metrics. (DSP-12922).

5.1.0 DSE Analytics changes and enhancements

• Implement WebHDFS REST interface on DSEFS. (DSP-2347)
• Enable optional running Spark executor (page 424) as a separate user. (DSP-4252)
• Opaquely use Solr indexes to optimize SparkSQL queries. (DSP-5028)
• DSEFS support in BYOS. (DSP-8888)
• Support SSL in the Spark Master and Worker UI. (DSP-9928)

  In dse.yaml, the spark_encryption_options are no longer valid.

  • Hive connector is removed. CassandraHive Metastore is used by Spark SQL. Hive cql/cassandra handler are removed. (DSP-10333)
  • BYOHadoop and DSE Hadoop are removed. (Deprecated in DSE 5.0) (DSP-10408)
  • Faster locking in DSEFS and support for shared locks. (DSP-11145)
  • Geo types are supported in DSE SparkSQL and represented as well known text. (DSP-11173)
  • Create CQL-based Resource Manager comm channel for Spark. (DSP-11331)
  • Analytics jobs run through dse spark-submit can take advantage of continuous paging for performance gains. See Enabling continuous paging (page 416). (DSP-11343)
  • Access DSEGraphFrame tables through SparkSQL. (DSP-11898)
  • Enable authentication for server side Spark UIs. (DSP-11955)
  • Enhanced dse client-tool spark subcommands (page 1087). (DSP-12048)
  • Programmatically setting the shuffle parameter using
    conf.set("spark.shuffle.service.port", port) is not supported. Instead, use
dse spark-submit which automatically sets the correct service port based on the
authentication state. (DSP-12471)
  • Spark Jobserver has been upgraded to 0.6.2.234. This custom version requires
applications to be recompiled using the compatible DataStax Spark Jobserver API
(recommended) or jobserver 0.7.0. (DSP-12478)

5.1.0 DSE Graph changes and enhancements

• The default number of threads used for loading vertices (load_vertex_threads (page 896)) or edges (load_edge_threads (page 896)) is changed from 1 to 0. (DGL-124)
When query fails due to timeout, state in error message which timeout was exceeded. (DSP-9393)
• Add ifExist to drop graph. (DSP-9511)
• Database errors related to graph queries go directly to drivers. (DSP-9567)
• The format of edge IDs changed. There is no user impact. (DSP-10566)
• Reject out of bounds geo data. (DSP-10748)
• Disable graph#io. (DSP-10804)
• Improve Graph and Spark integration for performance and usability with DSEGraphFrame framework for batch graph queries. (DSP-11104)
• Prevent external Solr schema changes from being overwritten by DSE Graph. (DSP-11226)
• Support Date type in Graph. (DSP-11287)
• Graph-specific MBeans moved from datastore-latencies to request-latencies category. (DSP-11521)
• Support for Solr-based fuzzy search in graph. (DSP-11273)
• DSE Graph API support for edit distance queries. (DSP-11880)
• Search regex '.' now matches all whitespace. (DSP-11952)
• Kryo version conflict. (DSP-11984)
• Add DSEG snapshot config mutator. (DSP-12072)
• Setting Spark properties (page 916) from Gremlin. (DSP-12296)
• The Geo interfaces for distance and polygon queries (page 815) are changed in the driver. (DSP-12710)
• Changes in Geo predicates. (DSP-12467)

5.1.0 DSEFS changes and enhancements

• DSEFS commands for controlling file permissions and ownership. (DSP-10582)
• Tab autocompletion is supported. (DSP-10584)
• Support for file compression. (DSP-10655)
• Enhanced local file system operations in DSEFS shell. (DSP-10933)
• Add comment (#) support in DSEFS shell. (DSP-10935)
• Expose DSEFS metrics via JMX. (DSP-11375)
• Improve DSEFS user experience: human readable sizes (-h) and single column output (-1). (DSP-11675)
• Fix recursive ls parameter (page 483) name: change -r to -R. (DSP-12016)
• Make name_id part of primary key in names table. Improved DSEFS Cassandra schema to improve recovery of all metadata from inconsistency caused by concurrent writes. Upgrades to DataStax Enterprise 5.1 require steps to get new schema. (DSP-12450)
• Although DSEFS is enabled by default in DSE 5.1.0, the dsefs.enabled setting is commented out in the new DSE 5.1.0 dse.yaml file. To enable DSEFS, uncomment the dsefs_options.enabled (page 340) setting after upgrade to DSE 5.1.0. (DSP-13310)
DSE Search in DataStax Enterprise 5.1 uses Apache Solr 6.0. (DSP-9748) This significant change requires advanced planning and specific actions before and after the upgrade.

**Important:** To upgrade DSE Search and SearchAnalytics workloads, you must follow the specific steps in upgrading to DSE 5.1.

- DataImportHandler is no longer supported. The import handler tab is removed from Solr Admin UI. Before upgrading to DSE 5.1, remove all data import handlers from solrconfig files. (DSP-6266)
- Remove the legacy netty (page 327)-based inter-node communication protocol. See https://docs.datastax.com/en/upgrade/doc/upgrade/datastax_enterprise/upgdDSE51.html#upgdDSE51__prepUpg51Search
- Timeout for non-query search requests like core creation and distributed deletes is set in the internode_messaging_options with the client_request_timeout_seconds (page 347) option. (DSP-6933)
- Automatically index both analyzed and non-analyzed versions of textual vertex properties. (DSP-7633)
- Check for index integrity (page 1040) with dsetool using lucene CheckIndex. (DSP-8875)
- New DSE Search index management commands (page 536) to manage cluster-wide search indexes. (DSP-9204)
- Lucene merge scheduling and lack of parallelism cause periods of 0 throughput. (DSP-9325)

In earlier releases, the default mergeScheduler settings in solrconfig.xml were not set appropriately. The default settings are now set automatically and appropriately, unless a custom mergeScheduler configuration is provided.

- Deprecated Solr field types require action before upgrade to DSE 5.1. (DSP-9509)
- HTTP writes are deprecated. Insert data into DSE by using CQL. (DSP-9540)
- dsetool search commands use the CQL index management commands. dsetool create_core (page 1029) no longer supports deleteAll. (DSP-9762)
- DateRangeField support with new DateRangeType data type (page 548). (DSP-10225)
- Improved asynchronous indexing performance. (DSP-10617)
- Add more checks to CassandraSolrConfig for unwanted config elements. (DSP-10677)
- LUCENE-7299 Optimized segment flushing with radix sort. (DSP-10685)
- Changes in default behavior for auto-generated schemas (page 519) to enable DocValues. (DSP-10690)
- XML correctly indented to improve readability for auto-generated resources. (DSP-10795)
- When using SpatialRecursivePrefixTreeFieldType (RPT) in search schemas, replace the units field type with distanceUnits after Upgrading to DSE 5.1. (DSP-10802)
- Optimize Solr query parser to use filter boolean queries. (DSP-10916)
- Stored=true copy fields are not supported and cause schema validation to fail. Before upgrading to 5.1, you must change the stored attribute value of a copyField directive from true to false in the schema.xml file and reload the core. (DSP-11087)
• PER PARTITION clause is not supported for DSE Search solr_query queries. (DSP-11050)
• Support limiting queries by time with the Solr timeAllowed (page 598) parameter, DSE Search differences (page 509) apply. (DSP-11165)
• Improve client-side mapping of DSE Search exceptions. (DSP-11315)
• Default batch size for the search TTL Process is changed. (DSP-11493)

When a value is not specified for ttl_index_rebuild_options.max_docs_per_batch in dse.yaml (page 326), the default is changed from 100 to 4096.

• DSE Search does not support the duration Cassandra data type. (DSP-11825)
• Improved error handling for authentication and authorization of Solr HTTP requests and Solr Admin UI. (DSP-12550)

Requests that fail due to lack of permissions return a 403 error, not a 401 error that was returned in earlier versions.

• Add support for unfrozen tuples. (DSP-12347)
• Improve default selection for dse.yaml and solrconfig.xml write path configuration. See Configuring and tuning indexing performance. (DSP-12491)

5.1.0 Known issues

Known issues for DSE core:

• Potential data loss for INSERTs with very large TTLs. TTL expiration timestamps are susceptible to the year 2038 problem. (DSP-15412)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. TTLs are considered "very large" when close to the maximum allowed value of 630720000 seconds (20 years), starting from 2018-01-19T03:14:06+00:00. As time progresses, the maximum supported TTL is gradually reduced as the maximum expiration date approaches. For instance, on 2028-01-19T03:14:06 with a TTL of 10 years is impacted. The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. There is no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately.

  Warning: Upgrade to DSE 5.1.7 or later and take required action to protect against overflow of local expiration time.

• Even with nodetool repair -full or nodetool repair -pr, DSE DSE 5.1.0-5.1.2 are run as incremental and mark sstables as repaired causing anti-compaction. (DSP-14464)
• DataStax Enterprise will not run with Java 1.8u161 or later. (DSP-15277)
• Potential data loss for INSERTs with very large TTLs, where "very large" is close to the maximum allowed value of 630720000 seconds (20 years), starting from 2018-01-19T03:14:06+00:00. As time progresses, the maximum supported TTL is gradually reduced as the maximum expiration date approaches. For instance, on 2028-01-19T03:14:06 with a TTL of 10 years is impacted. If you use very large TTLs, DataStax strongly recommends upgrading to 5.1.7 or later. (DSP-15412)

Known issue for DSE Analytics:

• The "remember me" feature used by the Shiro 1.2.4 library and also used by the Spark Job Server is vulnerable to malicious attackers. Do not enable the "remember me" feature in a custom shiro.ini file if you defined one in application.conf. DSE does not enable the "remember me" feature by default. (DSP-11072)

Known issues for DSE Search:

• DateRange parsing improperly rolls over month, day, hour, min, seconds when invalid dates in a date range are specified. (DSP-12480)

• DSE Search might miss token filtering on mixed versions clusters. Upgrade all nodes to DSE 5.1.6 or later for correct token filtering. (DSP-14998)

• Skip DSE 5.1.0 and upgrade directly to DSE 5.1.1 if you:
  # Use the HTTP interface. (DSP-13318), (DSP-13270)
  # Have a Thrift column family backing an active Solr core. (DSP-13019)
  # Use TTL to expire data. (DSP-12960)
  # Use index encryption. (DSP-13155), (DSP-12620)
  # Use live indexing. (DSP-12040), (DSP-12941)

• Solr listens only on port 8080 regardless of configuration. (DSP-13187)

• Auto generated solrconfig.xml has invalid requestHandler for JSON core creations after upgrade to 5.1.0. (DSP-13188)

If you make HTTP writes with JSON documents (deprecated) (page 202), then change the auto generated solrconfig.xml:

```xml
<requestHandler name="/update/json" class="solr.UpdateUpdateRequestHandler" startup="lazy"/>
```

to

```xml
<requestHandler name="/update/json" class="solr.UpdateRequestHandler" startup="lazy"/>
```

5.1.0 Resolved issues

Resolved issues for:

• DataStax Enterprise core (page 205)
• DSE Advanced Replication (page 205)
• DSE Analytics (page 205)
• DSE Graph (page 205)
• DSE Search (page 205)

5.1.0 DataStax Enterprise core resolved issues

• Recent worst queries for slow query log. (DSP-5088)
  New configurable `cql_slow_log_options` (page 331).
• dse lib has old metrics core version. (DSP-11389)
• `cqlsh SOURCE command` shouldn't assume `PlainTextAuthenticator`. (DSP-12773)

5.1.0 DSE Advanced Replication resolved issues

• Fix authentication and encryption settings for SSL remote cluster connections. (DSP-9470)

5.1.0 DSE Analytics resolved issues

• Make `dse client-tool` (page 1082) `sql-schema command` consistent with double-dash parameters. (DSP-10557)
• `CFS repair` (page 1063) can repair only the default file system as defined in Hadoop configuration. (DSP-12481)

5.1.0 DSE Graph resolved issues

• `Search.tokenRegex()` is case sensitive. (DSP-9425)
• Graph not working properly with Kerberos with `serializeResultToString: true`. (DSP-12201)
• Enable split-DC `graph ID` (page 349) allocation. (DSP-12516)
• `geo.distance(lng,lat,radius)` expresses radius in degrees rather than kilometers. (DSP-12415)
• Align `distance query` (page 718) behavior between vertex properties with and without search indexes. (DSP-12673)

5.1.0 DSE Search resolved issues

• Solr range facets before, after, and between return incorrect and inconsistent results on multinode clusters. (DSP-4485)
• Validate auto generated resources before writing them. (DSP-7638)
• Support for non-frozen `UDTs` (page 598). Solr `field name policy` (page 510) applies. (DSP-11412)
• Users require `SELECT permissions` on any search index that they view. Specific permissions (page 639) are required for all core operations when using the Solr Admin UI. (DSP-11910)
• `QueryUtils#getStandardVertexIdComponents` is not thread safe. (DSP-12254)
DataStax Enterprise 5.1 release notes

- Core is not correctly unloaded on restarted nodes. (DSP-12434)
- Native driver connections in dsetool aren't isolated to specified host. (DSP-12438)
- Heap is exhausted while search reindexes very wide partitions. New IndexPool MBean (page 634) attributes. (DSP-12547)
- Concurrent sorting issue with RT. (DSP-12600)
- Disable redundant, experimental, and other Solr 6 features. (DSP-13093)

Hadoop libraries

Built-in Hadoop and Bring-Your-Own-Hadoop (BYOH) were deprecated in DataStax Enterprise (DSE) 5.0, and were removed in DSE 5.1. Hadoop removal from DSE 5.1 and later means that DSE does not allow for the startup of Hadoop services previously included in DSE, including MapReduce JobTracker and TaskTracker.

However, DSE has supported built-in Spark since DSE 4.5 and Bring-Your-Own-Spark (BYOS) since DSE 5.0, and that support continues today. Because Spark depends on certain Hadoop libraries on the server and the client, DSE continues to ship with Hadoop libraries that are required for running Spark and BYOS.

To view the included Hadoop libraries, see DataStax Enterprise 5.1.x third-party software.

Cassandra enhancements for DSE 5.1.0

DataStax Enterprise (DSE) 5.1.0 includes all changes from earlier DSE releases. These production-certified changes are enhancements to Apache Cassandra™ 3.10.0. (For Cassandra updates, see CHANGES.txt.)

- Fix testLimitSSTables flake caused by concurrent flush (CASSANDRA-12820)
- cdc column addition strikes again (CASSANDRA-13382)
- Fix static column indexes (CASSANDRA-13277)
- DataOutputBuffer.asNewBuffer broken (CASSANDRA-13298)
- unittest CipherFactoryTest failed on MacOS (CASSANDRA-13370)
- Forbid SELECT restrictions and CREATE INDEX over non-frozen UDT columns (CASSANDRA-13247)
- Default logging we ship will incorrectly print "?:?" for "%F:%L" pattern (CASSANDRA-13317)
- Possible AssertionError in UnfilteredRowIteratorWithLowerBound (CASSANDRA-13366)
- Support unaligned memory access for AArch64 (CASSANDRA-13326)
- Improve SASI range iterator efficiency on intersection with an empty range (CASSANDRA-12915).
- Fix equality comparisons of columns using the duration type (CASSANDRA-13174)
- Oblfuscate password in stress-graphs (CASSANDRA-12233)
- Move to FastThreadLocalThread and FastThreadLocal (CASSANDRA-13034)
- nodetool stopdaemon errors out (CASSANDRA-13030)
- Tables in system_distributed should not use gcgs of 0 (CASSANDRA-12954)
- Fix primary index calculation for SASI (CASSANDRA-12910)
• More fixes to the TokenAllocator (CASSANDRA-12990)
• NoReplicationTokenAllocator should work with zero replication factor (CASSANDRA-12983)
• Address message coalescing regression (CASSANDRA-12676)
• Fix possible NPE on upgrade to 3.0/3.X in case of IO errors (CASSANDRA-13389)
• Legacy deserializer can create empty range tombstones (CASSANDRA-13341)
• Legacy caching options can prevent 3.0 upgrade (CASSANDRA-13384)
• Use the Kernel32 library to retrieve the PID on Windows and fix startup checks (CASSANDRA-13333)
• Fix code to not exchange schema across major versions (CASSANDRA-13274)
• Dropping column results in "corrupt" SSTable (CASSANDRA-13337)
• Bugs handling range tombstones in the sstable iterators (CASSANDRA-13340)
• Fix CONTAINS filtering for null collections (CASSANDRA-13246)
• Applying: Use a unique metric reservoir per test run when using Cassandra-wide metrics residing in MBeans (CASSANDRA-13216)
• Propagate row deletions in 2i tables on upgrade (CASSANDRA-13320)
• Slice.isEmpty() returns false for some empty slices (CASSANDRA-13305)
• Add formatted row output to assertEmpty in CQL Tester (CASSANDRA-13238)
• Prevent data loss on upgrade 2.1 - 3.0 by adding component separator to LogRecord absolute path (CASSANDRA-13294)
• Improve testing on macOS by eliminating sigar logging (CASSANDRA-13233)
• Cqlsh copy-from should error out when csv contains invalid data for collections (CASSANDRA-13071)
• Update c.yaml doc for offheap memtables (CASSANDRA-13179)
• Faster StreamingHistogram (CASSANDRA-13038)
• Legacy deserializer can create unexpected boundary range tombstones (CASSANDRA-13237)
• Remove unnecessary assertion from AntiCompactionTest (CASSANDRA-13070)
• Fix cqlsh COPY for dates before 1900 (CASSANDRA-13185)
• Use keyspace replication settings on system.size_estimates table (CASSANDRA-9639)
• Add vm.max_map_count StartupCheck (CASSANDRA-13008)
• Hint related logging should include the IP address of the destination in addition to host ID (CASSANDRA-13205)
• Reloading logback.xml does not work (CASSANDRA-13173)
• Lightweight transactions temporarily fail after upgrade from 2.1 to 3.0 (CASSANDRA-13109)
• Duplicate rows after upgrading from 2.1.16 to 3.0.10/3.9 (CASSANDRA-13125)
• Fix UPDATE queries with empty IN restrictions (CASSANDRA-13152)
• Fix handling of partition with partition-level deletion plus live rows in sstabledump (CASSANDRA-13177)
• Provide user workaround when system_schema.columns does not contain entries for a table that's in system_schema.tables (CASSANDRA-13180)
• Honor truststore-password parameter in cassandra-stress (CASSANDRA-12773)
• Discard in-flight shadow round responses (CASSANDRA-12653)
• Don't anti-compact repaired data to avoid inconsistencies (CASSANDRA-13153)
• Wrong logger name in AnticompactionTask (CASSANDRA-13343)
• Commitlog replay may fail if last mutation is within 4 bytes of end of segment (CASSANDRA-13282)
• Fix queries updating multiple time the same list (CASSANDRA-13130)
• Fix GRANT/REVOKE when keyspace isn't specified (CASSANDRA-13053)
• Avoid race on receiver by starting streaming sender thread after sending init message (CASSANDRA-12886)
• Fix "multiple versions of ant detected..." when running ant test (CASSANDRA-13232)
• Coalescing strategy sleeps too much (CASSANDRA-1309)
• Fix flaky LongLeveledCompactionStrategyTest (CASSANDRA-12202)
• Fix failing COPY TO STDOUT (CASSANDRA-12497)
• Fix ColumnCounter::countAll behaviour for reverse queries (CASSANDRA-13222)
• Exceptions encountered calling getSeeds() breaks OTC thread (CASSANDRA-13018)
• Fix negative mean latency metric (CASSANDRA-12876)
• Use only one file pointer when creating commitlog segments (CASSANDRA-12539)
• Remove unused repositories (CASSANDRA-13278)
• Log stacktrace of uncaught exceptions (CASSANDRA-13108)
• Use portable stderr for java error in startup (CASSANDRA-13211)
• Fix Thread Leak in OutboundTcpConnection (CASSANDRA-13204)
• Coalescing strategy can enter infinite loop (CASSANDRA-13159)

General upgrade advice for DSE 5.1.0

Carefully review all planning and upgrade documentation in the Upgrading DataStax Enterprise guide. The following provides general upgrade information:

GENERAL UPGRAADING ADVICE FOR ANY VERSION
===============================================

Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to ALWAYS snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

3.11.0
======
Upgrading
---------
- 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.

3.10
====

New features
------------
- New `DurationType` (cql duration). See CASSANDRA-11873
- Runtime modification of concurrent_compactors is now available via
  nodetool
- 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.
- Compaction can now take into account overlapping tables that don't
  take part
  in the compaction to look for deleted or overwritten data in the
  compacted tables.
  Then such data is found, it can be safely discarded, which in turn
  should enable
  the removal of tombstones over that data.
The behavior can be engaged in two ways:
- as a "nodetool garbagecollect -g CELL/ROW" operation, which applies single-table compaction on all sstables to discard deleted data in one step.
- as a "provide_overlapping_tombstones:CELL/ROW/NONE" compaction strategy flag, which uses overlapping tables as a source of deletions/overwrites during all compactions.

The argument specifies the granularity at which deleted data is to be found:
- If ROW is specified, only whole deleted rows (or sets of rows) will be discarded.
- If CELL is specified, any columns whose value is overwritten or deleted will also be discarded.
- NONE (default) specifies the old behavior, overlapping tables are not used to decide when to discard data.

Which option to use depends on your workload, both ROW and CELL increase the disk load on compaction (especially with the size-tiered compaction strategy), with CELL being more resource-intensive. Both should lead to better read performance if deleting rows (resp. overwriting or deleting cells) is common.
- 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.

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

3.8
===

New features
------------

- 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 ommission 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.
- 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.

Upgrading
--------
- The ReversedType behaviour has been corrected for clustering columns
of
  BYTE$ type containing empty value. Scrub should be run on the
existing
  SSTables containing a descending clustering column of BYTE$ 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).

Deprecation
----------
- 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.

3.7

Upgrading
--------
- 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 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.

3.6

New features
------------
- 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.
- 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.

3.4
===

New features
------------
- 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.

Upgrading
--------
- Nothing specific to 3.4 but please see previous versions upgrading
  section,
  especially if you are upgrading from 2.2.

Deprecation
------------
- 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.

3.2
===

New features
------------

- 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 relocatesstablestable 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.
  Note: hints compression is currently disabled by default.

Upgrading
---------

- 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

Deprecation
------------

- The Thrift API is deprecated and will be removed in Cassandra 4.0.

3.1
====

Upgrading
--------

- 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 inconditionally for that method.
- GC logging is now enabled by default (you can disable it in the jvm.options file if you prefer).

3.0
===

New features
-----------
- EACH_QUORUM is now a supported consistency level for read requests.
- Support for IN restrictions on any partition key component or clustering key
  as well as support for EQ and IN multicolumn restrictions has been added to
  UPDATE and DELETE statement.
- Support for single-column and multi-column slice restrictions (>, >=, <= and <)
  has been added to DELETE statements.
- nodetool rebuild_index accepts the index argument without the redundant table name
- Materialized Views, which allow for server-side denormalization, is now
  available. Materialized views provide an alternative to secondary
  indexes for non-primary key queries, and perform much better for indexing
  high cardinality columns.
  See http://www.datastax.com/dev/blog/new-in-cassandra-3-0-materialized-views
- Hinted handoff has been completely rewritten. Hints are now stored in flat
  files, with less overhead for storage and more efficient dispatch.
  See CASSANDRA-6230 for full details.
- Option to not purge unrepaired tombstones. To avoid users having data resurrected
  if repair has not been run within gc_grace_seconds, an option has been added to
  only allow tombstones from repaired sstables to be purged. To enable, set the
  compaction option 'only_purge_repaired_tombstones':true but keep in mind that if
  you do not run repair for a long time, you will keep all tombstones around which
  can cause other problems.
- Enabled warning on GC taking longer than 1000ms. See cassandra.yaml:gc_warn_threshold_in_ms

Upgrading
-------
- Clients must use the native protocol version 3 when upgrading from
  2.2.X as
  the native protocol version 4 is not compatible between 2.2.X and
  3.Y. See
A new argument of type InetAdress has been added to IAuthenticator::newSaslNegotiator, representing the IP address of the client attempting authentication. It will be a breaking change for any custom implementations.

- token-generator tool has been removed.
- Upgrade to 3.0 is supported from Cassandra 2.1 versions greater or equal to 2.1.9, or Cassandra 2.2 versions greater or equal to 2.2.2. Upgrade from Cassandra 2.0 and older versions is not supported.
- The 'memtable_allocation_type: offheap_objects' option has been removed. It should be re-introduced in a future release and you can follow CASSANDRA-9472 to know more.
- Configuration parameter memory_allocator in cassandra.yaml has been removed.
- The native protocol versions 1 and 2 are not supported anymore.
- Max mutation size is now configurable via max_mutation_size_in_kb setting in cassandra.yaml; the default is half the size commitlog_segment_size_in_mb * 1024.
- 3.0 requires Java 8u40 or later.
- Garbage collection options were moved from cassandra-env to jvm.options file.
- New transaction log files have been introduced to replace the compactions_in_progress system table, temporary file markers (tmp and tmplink) and sstable ancestors.

Therefore, compaction metadata no longer contains ancestors. Transaction log files list sstable descriptors involved in compactions and other operations such as flushing and streaming. Use the sstableutil tool to list any sstable files currently involved in operations not yet completed, which previously would have been marked as temporary.

A transaction log file contains one sstable per line, with the prefix "add:" or "remove:". They also contain a special line "commit", only inserted at the end when the transaction is committed. On startup we use these files to cleanup any partial transactions that were in progress when the process exited. If the commit line is found, we keep new sstables (those with the "add" prefix) and delete the old sstables (those with the "remove" prefix), vice-versa if the commit line is missing. Should you lose or delete these log files, both old and new sstables will be kept as live files, which will result in duplicated
DataStax Enterprise 5.1 release notes

sstables. These files are protected by incremental checksums so you should not manually edit them. When restoring a full backup or moving sstable files, you should clean-up any left over transactions and their temporary files first. You can use this command:

```bash
===> sstableutil -c ks table
```

See CASSANDRA-7066 for full details.

- New write stages have been added for batchlog and materialized view mutations
  you can set their size in cassandra.yaml
- User defined functions are now executed in a sandbox.
  To use UDFs and UDAs, you have to enable them in cassandra.yaml.
- New SSTable version 'la' with improved bloom-filter false-positive handling
  compared to previous version 'ka' used in 2.2 and 2.1. Running sstableupgrade is not necessary but recommended.
- Before upgrading to 3.0, make sure that your cluster is in complete agreement
  (schema versions outputted by `nodetool describecluster` are all the same).
- Schema metadata is now stored in the new `system_schema` keyspace, and legacy `system.schema_*` tables are now gone; see CASSANDRA-6717 for details.
- Pig's support has been removed.
- Hadoop BulkOutputFormat and BulkRecordWriter have been removed; use CqlBulkOutputFormat and CqlBulkRecordWriter instead.
- Hadoop ColumnFamilyInputFormat and ColumnFamilyOutputFormat have been removed;
  use CqlInputFormat and CqlOutputFormat instead.
- Hadoop ColumnFamilyRecordReader and ColumnFamilyRecordWriter have been removed;
  use CqlRecordReader and CqlRecordWriter instead.
- hinted_handoff_enabled in cassandra.yaml no longer supports a list of data centers.
  To specify a list of excluded data centers when hinted_handoff_enabled is set to true,
  use hinted_handoff_disabled_datacenters, see CASSANDRA-9035 for details.
- The `sstable_compression` and `chunk_length_kb` compression options have been deprecated.
  The new options are `class` and `chunk_length_in_kb`. Disabling compression should now be done by setting the new option `enabled` to `false`.
- The compression option `crc_check_chance` became a top-level table option, but is currently enforced only against tables with enabled compression.
- Only map syntax is now allowed for caching options. ALL/NONE/KEYS_ONLY/ROWS_ONLY syntax has been deprecated since 2.1.0 and is being removed in 3.0.0.
- The `index_interval` option for 'CREATE TABLE' statements, which has been deprecated.
DataStax Enterprise 5.1 release notes

- Batchlog entries are now stored in a new table - system.batches. The old one has been deprecated.
- JMX methods set/getCompactionStrategyClass have been removed, use set/getCompactionParameters or set/getCompactionParametersJson instead.
- SizeTieredCompactionStrategy parameter cold_reads_to_omit has been removed.
- The secondary index API has been comprehensively reworked. This will be a breaking change for any custom index implementations, which should now look to implement the new org.apache.cassandra.index.Index interface. New syntax has been added to create and query row-based indexes, which are not explicitly linked to a single column in the base table.

Spark Cassandra Connector changes for DSE 5.1.0

A list of DataStax Enterprise 5.1.0 production-certified changes for the DataStax Spark Cassandra Connector.

DSE 5.1.0: 2.0.1
* Refactor Custom Scan Method (SPARKC-481)

2.0.0
* Upgrade driver version for 2.0.0 Release to 3.1.4 (SPARKC-474)
* Extend SPARKC-383 to All Row Readers (SPARKC-473)

2.0.0 RC1
* Includes all patches up to 1.6.5
* Automatic adjustment of Max Connections (SPARKC-471)
* Allow for Custom Table Scan Method (SPARKC-459)
* Enable PerPartitionLimit (SPARKC-446)
* Support client certificate authentication for two-way SSL Encryption (SPARKC-359)
* Change Config Generation for Cassandra Runners (SPARKC-424)
* Remove deprecated QueryRetryDelay parameter (SPARKC-423)
* User ConnectionHostParam.default as default hosts String
* Update usages of deprecated SQLContext so that SparkSession is used instead (SPARKC-400)
* Test Reused Exchange SPARK-17673 (SPARKC-429)
* Module refactoring (SPARKC-398)
* Recognition of Java Driver Annotated Classes (SPARKC-427)
* RDD.deleteFromCassandra (SPARKC-349)
* Coalesce Pushdown to Cassandra (SPARKC-161)
* Custom Conf options in Custom Pushdowns (SPARKC-435)
* Upgrade CommonBeatUtils to 1.9.3 to Avoid SID-760 (SPARKC-457)
2.0.0 M3
* Includes all patches up to 1.6.2

2.0.0 M2
* Includes all patches up to 1.6.1

2.0.0 M1
* Added support for left outer joins with Cassandra table (SPARKC-181)
* Removed CassandraSqlContext and underscore based options (SPARKC-399)
  - Removed Twitter demo because there is no spark-streaming-twitter package available anymore
  - Removed Akka Actor demo because there is no support for such streams anymore
  - Bring back Kafka project and make it compile
  - Update several classes to use our Logging instead of Spark Logging because Spark Logging became private
  - Update few components and tests to make them work with Spark 2.0.0
  - Fix Spark SQL - temporarily
  - Update plugins and Scala version
Installing DataStax Enterprise 5.1

Which install method should I use?

You can install DataStax Enterprise (DSE) in several ways, depending on the purpose of the installation, the type of operating system, and the available permissions. Be sure to install on a supported platform.

<table>
<thead>
<tr>
<th>Installation method</th>
<th>About</th>
</tr>
</thead>
</table>
| Using the DataStax Installer to install (root permissions required) (page 223) | A Linux GUI-based/text installer with multiple options such as:  
• Services or No Services installation. A No Services installation is similar to a tarball installation.  
• Ability to install with or without root (page 231) permissions. (You cannot update system packages and dependencies without root permissions.)  
• Ability to install on Mac OS X (page 239) (development only).  
• Ability fully configure DSE using command line or a properties file (page 245) during the installation.  
• Set node type, such as analytics, graph, search, or transactional.  
• Configure cluster, node, and DSE settings.  
• Update some system packages and dependencies.  
• Upgrade nodes and provides a list of changes and backup files.  
• Install Developer tools such as DataStax Studio (page 1098), Javadoc, DSE Graph Loader (page 831), and DSE demos.  
• Run the Preflight check tool, which is a tool for detecting and optionally fixing many invalid or suboptimal configuration settings. |
Installing DataStax Enterprise 5.1

<table>
<thead>
<tr>
<th>Installation method</th>
<th>About</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpsCenter Lifecycle Manager</strong></td>
<td>The DataStax Lifecycle Manager (LCM) can install DSE. It is fully integrated with DSE OpsCenter. LCM provides:</td>
</tr>
<tr>
<td></td>
<td>• A web-based graphical interface for installing and configuring DSE.</td>
</tr>
<tr>
<td></td>
<td>• Performs a Services installation using RHEL or Debian packages.</td>
</tr>
<tr>
<td></td>
<td>• Ability to configure and update all DSE settings.</td>
</tr>
<tr>
<td></td>
<td>• Installs the necessary system requirements automatically, including Java.</td>
</tr>
<tr>
<td></td>
<td>• Integrates with OpsCenter Best Practice Service, which detects many suboptimal configuration settings.</td>
</tr>
<tr>
<td></td>
<td>• Automation via the LCM API.</td>
</tr>
<tr>
<td></td>
<td>To install DSE using LCM, requires installing OpsCenter, bringing your own instances on a supported platform with SSH and Python installed, and root permissions on the target nodes.</td>
</tr>
</tbody>
</table>

| Installing DataStax Enterprise 5.1 on RHEL-based systems using Yum | Installs DSE using Yum repositories on RHEL-based systems. Requires root permissions. Typically used in production environments.                                                                                               |
| Installing DataStax Enterprise 5.1 on Debian-based systems using APT | Installs DSE on Debian-based systems using APT. Requires root permissions. Typically used in production environments.                                                                                             |
| Binary installer (page 252)                                      | Installs DSE on any supported Linux-based platform.                                                                                                                                                      |

| Installing and deploying DSE on cloud platforms               | Instructions for installing and deploying DSE on CenturyLink Cloud, Google Compute Engine, Microsoft Azure, and Amazon EC2.                                                                               |

Caution: If you have installed hot fixes, be sure to manually remove the hot fix JAR files before upgrading DataStax Enterprise.

**DataStax Installers**

DataStax provides several methods for installing DataStax Enterprise from the DataStax Installer.

**Using the DataStax Installer to install (root permissions required)**

Instructions for installing or upgrading DataStax Enterprise (DSE) 5.1 using the DataStax Installer when you have root permissions.
Installing DataStax Enterprise 5.1

If you don't have root permissions or want to install in a custom directory, see Using the DataStax Installer to install (root permissions not required) (page 231) or use the binary tarball (page 252).

Warning: When DSE is installed, it creates a cassandra user in the database and runs as this user. It also creates a cassandra user in the operating system. Do not use the cassandra user in production. Using the cassandra user is a security risk. See Adding a superuser login.

Prerequisites:

- A supported platform.
- DataStax Academy registration email address and password.
- Root or sudo access.
- OpenJDK 8 or Oracle Java SE Runtime Environment 8 (JDK) (page 256) (1.8.0_40 minimum). Earlier or later versions are not supported.
  
  Attention: Although Oracle JRE/JDK 8 is supported, DataStax does more extensive testing on OpenJDK 8 starting with DSE 5.1.11. This change is due to the end of public updates for Oracle JRE/JDK 8. See Oracle Java SE Support Roadmap.

- Python 2.7.x

Table 2: Hardware requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Minimum</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB</td>
<td>24 GB</td>
</tr>
<tr>
<td>Data directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Commit log directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Saved caches directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Logs directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
</tbody>
</table>

Also see Recommended production settings (page 272) and the DataStax Enterprise Reference Architecture white paper.

About DataStax installer

The DataStax installer installs DataStax Enterprise and DataStax Agent. It does not install OpsCenter. If you select Developer Related Tools during installation, DataStax Studio (page 1098) is installed.

The installer sets some but not all cassandra.yaml parameters. It does not set dse.yaml properties. Set the remaining parameters in the following ways:
Installing DataStax Enterprise 5.1

- Manually after installation.
- To specify pre-configured cassandra.yaml and dse.yaml files, use the unattended install (page 245) with either command line (page 246) or property file (page 247).

**Note:** The latest version of DataStax Enterprise 5.1 is 5.1.11.

In a terminal window:

1. Download the installer for your computer from the DataStax download page or use the following command:

   ```
   $ curl --user dsa_email_address:password -O https://downloads.datastax.com/enterprise/DataStaxEnterprise-5.1.11-linux-x64-installer.run
   ```

   **Note:** To install earlier versions, replace the version number in the above command. To view the available versions, see the Release notes (page 18).

2. From the directory where you downloaded the install file, change the permission to executable:

   ```
   $ chmod +x DataStaxEnterprise-5.1.11-linux-x64-installer.run
   ```

3. To view the installer help:

   ```
   $ ./DataStaxEnterprise-5.1.11-linux-x64-installer.run --help
   ```

   Help displays a list of the available options and their default settings.

4. Start the installation:
   - No configuration parameters:
     ```
     $ sudo ./DataStaxEnterprise-5.1.11-linux-x64-installer.run
     ```
   - Configuration parameters:
     ```
     $ sudo ./DataStaxEnterprise-5.1.11-linux-x64-installer.run --enable_vnodes 0 ## Command line option.
     $ sudo ./DataStaxEnterprise-5.1.11-linux-x64-installer.run --optionfile ../datastax/DC4-analytics.prop ## Property file option.
     ```
For configuration parameter information, see the installer options described in Using the DataStax Installer to install using command line or properties file commands (page 245).

The installer launches.

5. After accepting the License Agreement, select the type of install. In **Service Setup**, select **Services and Utilities**.
   - **Services and Utilities**: Sets up DataStax Enterprise as a service. It installs DataStax Enterprise in system locations.
   - **No Services**: Sets up DataStax Enterprise as a standalone process. It does not require root or sudo access.

6. Select the **Install Type**:
   - **Simple**: Installs DataStax Enterprise using the default path names and options:
   - **Advanced**: Allows you to configure path names and options:

   ```
   # User Setup (page 229)
   # Ring Options (page 228)
   # Directory Locations (page 229)
   # Ports (page 230)
   ```

7. Set up the node:
### Table 3: Node Setup

| Default Interface  | Network interface for the DataStax Enterprise server.  
|                   | Single node clusters: use 127.0.0.0.  
| Cluster Name      | Name of the cluster. You must use the same cluster name for each node in the cluster.  
| Seeds             | All nodes use the seed node list for finding each other and learning the topology of the ring.  
|                   | **Single node clusters**: Set to empty.  
|                   | **Multiple node clusters**: List of seed nodes. *(Do not make all nodes seed nodes.)*  
|                   | Additional information:  
|                   | • Internode communications (gossip)  
|                   | • Initializing a single datacenter per workload type  
|                   | • Initializing multiple datacenters per workload type  
| Install developer related tools | Includes the following:  
|                   | • Javadocs  
|                   | • DataStax Enterprise demos  
|                   | • DataStax Studio *(page 1098)*  
|                   | • DSE Graph Loader *(page 831)*  

8. If an installation of DataStax Enterprise already exists, the installer displays upgrade information and installs DSE in the following locations:

- **Installer-Services installations**: `/usr/share/dse`
- **No Services installations**: Previous installation location

**Important**: Before upgrading an existing cluster, see the DataStax Upgrade Guide.

The important files to review for upgrade are `dse.yaml`, `cassandra.yaml`, and for service installs `/etc/default/dse`.

If `dse.in.sh` or `cassandra.in.sh` have been modified, make modifications to the `cassandra-env.sh` or the `dse-env.sh` instead. Changing these files is not recommended.

9. Set up the node type:
Table 4: DSE Setup

<table>
<thead>
<tr>
<th>DSE Analytics</th>
<th>DSE Analytics (page 370) includes integration with Apache Spark™.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSE Search</td>
<td>DSE Search (page 507) includes integration with Apache Solr™.</td>
</tr>
<tr>
<td>DSE Graph</td>
<td>DSE Graph (page 651) is a graph database for managing, analyzing, and searching highly-connected data.</td>
</tr>
<tr>
<td>DSE Advanced Replication</td>
<td>DSE Advanced Replication (page 958) supports configurable distributed data replication from source clusters to destination clusters bi-directionally that can experience sporadic connectivity.</td>
</tr>
</tbody>
</table>

10. Set the ring options:

Table 5: Ring Options

<table>
<thead>
<tr>
<th>Enable Virtual Nodes (vnodes)</th>
<th>Enable or disable Virtual nodes (page 367).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tokens</td>
<td>Token recommendations (page 367) for vnodes:</td>
</tr>
<tr>
<td></td>
<td>• Transactional nodes: 128</td>
</tr>
<tr>
<td></td>
<td>• Analytical nodes (Spark): 128</td>
</tr>
<tr>
<td></td>
<td>• Search nodes: 8</td>
</tr>
<tr>
<td></td>
<td>• DSE Graph: 128. When used with DSE Search: 8</td>
</tr>
<tr>
<td>Listen Address</td>
<td>cassandra.yaml: listen_address (page 280)</td>
</tr>
<tr>
<td>RPC Address</td>
<td>cassandra.yaml: rpc_address (page 284)</td>
</tr>
</tbody>
</table>

11. Set up the miscellaneous options:

Table 6: Misc Options

<table>
<thead>
<tr>
<th>Update Operating System</th>
<th>Update some system packages and dependencies. Does not upgrade or install major components such as Java.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start services</td>
<td>• Yes starts all services when the installation is complete.</td>
</tr>
<tr>
<td></td>
<td>• No, select when additional configuration is needed after installation.</td>
</tr>
<tr>
<td>OpsCenter Address</td>
<td>Optional. The server name or IP address for OpsCenter.</td>
</tr>
</tbody>
</table>

If performing a Simple installation, go to 16 (page 230).
The following steps are displayed when Advanced is selected in 5 (page 226).

12. Set up the Preflight Check (Advanced Installations only):

**Table 7: Preflight Check Options**

<table>
<thead>
<tr>
<th>Run Preflight Check</th>
<th>A <em>collection of tests</em> that can detect and fix a node's configuration. The tool can detect and fix many invalid or suboptimal configuration settings. It is not available in No Services installations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attempt to Fix Issues</td>
<td>Enable DataStax Enterprise to attempt to fix invalid or suboptimal configuration settings.</td>
</tr>
<tr>
<td>SSD Drives</td>
<td>Enter the paths to the Solid State Drives. Separate the drives by a comma.</td>
</tr>
<tr>
<td>Drives</td>
<td>Enter the paths to the hard drives. Separate the drives by a comma.</td>
</tr>
<tr>
<td>Time to run Disk Benchmarks</td>
<td>Set to simulate a normal load.</td>
</tr>
<tr>
<td>Threads per Disk Benchmark</td>
<td>Set to simulate a normal load.</td>
</tr>
</tbody>
</table>

13. Change the default user and user group (Advanced Installations only):

**Table 8: User Setup**

<table>
<thead>
<tr>
<th>OS User ID for Service</th>
<th>Default: cassandra. Because the DataStax Agent relies on user <em>cassandra</em>, DataStax does not recommend changing the default. If changed, you must manually install, update, and configure the <em>DataStax Agent</em>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS User Group for Service</td>
<td>Default: cassandra. See above.</td>
</tr>
</tbody>
</table>

14. Change the default directory locations (Advanced Installations only):

**Table 9: Directory Locations**

<table>
<thead>
<tr>
<th>Data Directory</th>
<th><em>cassandra.yaml</em> <code>data_file_directories</code> (page 281)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commitlog Directory</td>
<td><em>cassandra.yaml:</em> <code>commitlog_directory</code> (page 281)</td>
</tr>
<tr>
<td>Saved Caches Directory</td>
<td><em>cassandra.yaml:</em> <code>saved_caches_directory</code> (page 281)</td>
</tr>
<tr>
<td>Hints Directory</td>
<td><em>cassandra.yaml: hints_directory</em> (page 302)</td>
</tr>
</tbody>
</table>
15. Change the default ports (Advanced Installations only):

<table>
<thead>
<tr>
<th>Ports</th>
<th>File Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage Port</td>
<td>cassandra.yaml: storage_port (page 294)</td>
</tr>
<tr>
<td>SSL Storage Port</td>
<td>cassandra.yaml: ssl_storage_port (page 310)</td>
</tr>
<tr>
<td>RPC Port</td>
<td>cassandra.yaml: rpc_port (page 299)</td>
</tr>
</tbody>
</table>

16. In **System Configuration**, review any warnings about potential issues.

17. After the installation completes, review the installation logs to verify the installation.

   **Note:** If you have closed the logs, see Services (page 264) or No-Services (page 269) installer locations.

DataStax Enterprise is ready for additional configuration (page 231).

18. Single-node cluster installations only:

   a. If DataStax Enterprise is not already running:
      
      - Package and Installer-Services installations:
        
        ```
        $ sudo service dse start
        ```
        
        For more start options, see Starting DataStax Enterprise as a service (page 1090).

      - Tarball and Installer-No Services installations:
        
        ```
        $ installation_location/bin/dse cassandra
        ```
        
        where the `installation_location` is either:
        
        ```
        # /usr/share/dse
        # the directory where you installed DataStax Enterprise.
        ```
        
        For more start options, see Starting DataStax Enterprise as a stand-alone process (page 1093)

   b. Verify that DataStax Enterprise is running:
Installing DataStax Enterprise 5.1

- Package and Installer-Services installations:
  
  $ nodetool status

- Tarball and Installer-No Services installations:
  
  $ installation_location/bin/nodetool status

Results using vnodes:

<table>
<thead>
<tr>
<th>Datacenter: Cassandra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status=Up/Down</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>-- Address Load Tokens Owns Host ID</td>
</tr>
<tr>
<td>Rack</td>
</tr>
<tr>
<td>UN 127.0.0.1 82.43 KB 128 ? 40725dc8-7843-43ae-9c98-7c532bf51fe rack1</td>
</tr>
</tbody>
</table>

Results not using vnodes:

<table>
<thead>
<tr>
<th>Datacenter: Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status=Up/Down</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>-- Address Load Owns Host ID</td>
</tr>
<tr>
<td>Token Rack</td>
</tr>
<tr>
<td>UN 172.16.222.136 103.24 KB ? 3c1d0657-0990-4f78-a3c0-3e0c37fc3a06 1647352612226902707 rack1</td>
</tr>
</tbody>
</table>

What's next:

- Configuration and log file locations (page 262) - Services and package installations.
- Configuration and log file locations (page 267) - No Services and tarball installations.
- Starting and stopping DataStax Enterprise (page 1090).
- Planning and testing DSE cluster deployments.
- DataStax Studio (page 1098) documentation.

Using the DataStax Installer to install (root permissions not required)

Instructions for installing or upgrading DataStax Enterprise 5.1 on any Linux-based platform using the DataStax Installer when you do not have root permissions or want to install in a custom directory. If you have root permissions, use Using the DataStax Installer to install (root permissions required) (page 223).

Warning: When DSE is installed, it creates a cassandra user in the database and runs as this user. It also creates a cassandra user in the operating system. Do not
use the `cassandra` user in production. Using the `cassandra` user is a security risk. See Adding a superuser login.

Prerequisites:

- A supported platform.
- DataStax Academy registration email address and password.
- OpenJDK 8 or Oracle Java SE Runtime Environment 8 (JDK) *(page 256)* (1.8.0_40 minimum). Earlier or later versions are not supported.

  **Attention:** Although Oracle JRE/JDK 8 is supported, DataStax does more extensive testing on OpenJDK 8 starting with DSE 5.1.11. This change is due to the end of public updates for Oracle JRE/JDK 8. See Oracle Java SE Support Roadmap.

- Python 2.7.x

**Table 11: Hardware requirements**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Minimum</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB</td>
<td>24 GB</td>
</tr>
<tr>
<td>Data directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Commit log directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Saved caches directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Logs directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
</tbody>
</table>

Also see Recommended production settings *(page 272)* and the DataStax Enterprise Reference Architecture white paper.

About DataStax installer

The DataStax installer installs DataStax Enterprise and DataStax Agent. It does not install OpsCenter. If you select Developer Related Tools during installation, DataStax Studio *(page 1098)* is installed.

The installer sets some but not all cassandra.yaml parameters. It does not set dse.yaml properties. Set the remaining parameters in the following ways:

- Manually after installation.
- To specify pre-configured cassandra.yaml and dse.yaml files, use the unattended install *(page 245)* with either command line *(page 246)* or property file *(page 247).*

**Note:** The latest version of DataStax Enterprise 5.1 is 5.1.11.

In a terminal window:
1. Download the installer for your computer from the DataStax download page or use the following command:

   $ curl --user dsa_email_address:password -O https://downloads.datastax.com/enterprise/DataStaxEnterprise-5.1.11-linux-x64-installer.run

   **Note:** To install earlier versions, replace the version number in the above command. To view the available versions, see the Release notes (page 18).

2. From the directory where you downloaded the install file, change the permission to executable:

   $ chmod +x DataStaxEnterprise-5.1.11-linux-x64-installer.run

3. To view the installer help:

   $ ./DataStaxEnterprise-5.1.11-linux-x64-installer.run --help

   Help displays a list of the available options and their default settings.

4. Start the installation:

   - No configuration parameters:

     $ ./DataStaxEnterprise-5.1.11-linux-x64-installer.run
     $ ./DataStaxEnterprise-5.1.11-linux-x64-installer.run --mode text

   - Configuration parameters:

     $ ./DataStaxEnterprise-5.1.11-linux-x64-installer.run --enable_vnodes 0 ## Command line option.
     $ ./DataStaxEnterprise-5.1.11-linux-x64-installer.run --optionfile ../datastax/DC4-analytics.prop ## Property file option.

   For configuration parameter information, see the installer options described in Using the DataStax Installer to install using command line or properties file commands (page 245).

   The installer launches.
5. After accepting the License Agreement, select the type of install. In Service Setup, select **No Services**.
   - **Services and Utilities**: Sets up DataStax Enterprise as a service. It installs DataStax Enterprise in system locations.
   - **No Services**: Sets up DataStax Enterprise as a standalone process. It does not require root or sudo access.

6. Select the **Install Type**:
   - **Simple**: Installs DataStax Enterprise using the default path names and options:
   - **Advanced**: Allows you to configure path names and options:
     - User Setup *(page 229)*
     - Ring Options *(page 228)*
     - Directory Locations *(page 228)*
     - Ports *(page 230)*

7. Set the installation directory.

8. Set up the node:
Table 12: Node Setup

| Default Interface | Network interface for the DataStax Enterprise server.  
Single node clusters: use 127.0.0.0. |
| Cluster Name | Name of the cluster. You must use the same cluster name for each node in the cluster. |
| Seeds | All nodes use the seed node list for finding each other and learning the topology of the ring.  
**Single node clusters**: Set to empty.  
**Multiple node clusters**: List of seed nodes. *(Do not make all nodes seed nodes.)*  
Additional information:  
• Internode communications (gossip)  
• Initializing a single datacenter per workload type  
• Initializing multiple datacenters per workload type |
| Install developer related tools | Includes the following:  
• Javadocs  
• DataStax Enterprise demos  
• DataStax Studio *(page 1098)*  
• DSE Graph Loader *(page 831)* |

9. Set up the node type:

Table 13: DSE Setup

| DSE Analytics | DSE Analytics *(page 370)* includes integration with Apache Spark™. |
| DSE Search | DSE Search *(page 507)* includes integration with Apache Solr™. |
| DSE Graph | DSE Graph *(page 651)* is a graph database for managing, analyzing, and searching highly-connected data. |
| DSE Advanced Replication | DSE Advanced Replication *(page 958)* supports configurable distributed data replication from source clusters to destination clusters bi-directionally that can experience sporadic connectivity. |

10. If an installation of DataStax Enterprise already exists, the installer displays upgrade information and installs DSE in the following locations:  
• Installer-Services installations: */usr/share/dse*
• No Services installations: Previous installation location

**Important:** Before upgrading an existing cluster, see the DataStax Upgrade Guide.

The important files to review for upgrade are dse.yaml, cassandra.yaml, and for service installs /etc/default/dse.

If dse.in.sh or cassandra.in.sh have been modified, make modifications to the cassandra-env.sh or the dse-env.sh instead. Changing these files is not recommended.

11. Set the ring options:

**Table 14: Ring Options**

<table>
<thead>
<tr>
<th>Enable Virtual Nodes (vnodes)</th>
<th>Enable or disable Virtual nodes (&lt;page 367&gt;).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tokens</td>
<td>Token recommendations (&lt;page 367&gt;) for vnodes:</td>
</tr>
<tr>
<td></td>
<td>• Transactional nodes: 128</td>
</tr>
<tr>
<td></td>
<td>• Analytical nodes (Spark): 128</td>
</tr>
<tr>
<td></td>
<td>• Search nodes: 8</td>
</tr>
<tr>
<td></td>
<td>• DSE Graph: 128. When used with DSE Search: 8</td>
</tr>
</tbody>
</table>

| Listen Address                | cassandra.yaml: listen_address (<page 280>) |
| RPC Address                   | cassandra.yaml: rpc_address (<page 284>) |

12. Set up the miscellaneous options:

**Table 15: Misc Options**

| Update Operating System       | Update some system packages and dependencies. Does not upgrade or install major components such as Java. |
| Start services                | • **Yes** starts all services when the installation is complete. |
|                               | • **No**, select when additional configuration is needed after installation. |
| OpsCenter Address             | Optional. The server name or IP address for OpsCenter. |

If performing a Simple installation, go to 16 (<page 237>).

The following steps are displayed when Advanced is selected in 5 (<page 234>).

13. Set up the Preflight Check (Advanced Installations only):
Table 16: Preflight Check Options

<table>
<thead>
<tr>
<th>Run Preflight Check</th>
<th>A collection of tests that can detect and fix a node's configuration. The tool can detect and fix many invalid or suboptimal configuration settings. It is not available in No Services installations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attempt to Fix Issues</td>
<td>Enable DataStax Enterprise to attempt to fix invalid or suboptimal configuration settings.</td>
</tr>
<tr>
<td>SSD Drives</td>
<td>Enter the paths to the Solid State Drives. Separate the drives by a comma.</td>
</tr>
<tr>
<td>Drives</td>
<td>Enter the paths to the hard drives. Separate the drives by a comma.</td>
</tr>
<tr>
<td>Time to run Disk Benchmarks</td>
<td>Set to simulate a normal load.</td>
</tr>
<tr>
<td>Threads per Disk Benchmark</td>
<td>Set to simulate a normal load.</td>
</tr>
</tbody>
</table>

14. Change the default directory locations (Advanced Installations only):

Table 17: Directory Locations

<table>
<thead>
<tr>
<th>Directory Locations</th>
<th>cassandra.yaml:commitlog_directory (page 281)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commitlog Directory</td>
<td>cassandra.yaml:commitlog_directory (page 281)</td>
</tr>
<tr>
<td>Saved Caches Directory</td>
<td>cassandra.yaml: saved_caches_directory (page 281)</td>
</tr>
<tr>
<td>Hints Directory</td>
<td>cassandra.yaml:hints_directory (page 302)</td>
</tr>
<tr>
<td>Logs Directory</td>
<td>Log data for the database, Spark, and Tomcat. See the default file locations for Installer-No Services and tarball installations (page 267).</td>
</tr>
</tbody>
</table>

15. Change the default ports (Advanced Installations only):

Table 18: Ports

<table>
<thead>
<tr>
<th>Ports</th>
<th>cassandra.yaml:storage_port (page 294)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage Port</td>
<td>cassandra.yaml:storage_port (page 294)</td>
</tr>
<tr>
<td>SSL Storage Port</td>
<td>cassandra.yaml:ssl_storage_port (page 310)</td>
</tr>
<tr>
<td>RPC Port</td>
<td>cassandra.yaml:rpc_port (page 299)</td>
</tr>
</tbody>
</table>

16. In System Configuration, review any warnings about potential issues.

17. After the installation completes, review the installation logs to verify the installation.
Note: If you have closed the logs, see Services (page 264) or No-Services (page 269) installer locations.

DataStax Enterprise is ready for additional configuration (page 231).

18. Single-node cluster installations only:

a. Start DataStax Enterprise from the installation directory:

```
$ bin/dse cassandra
```

where the installation directory is either:

- /usr/share/dse
- the directory where you installed DataStax Enterprise.

Note: For other start options, see Starting DataStax Enterprise as a stand-alone process (page 1093).

b. Verify that DataStax Enterprise is running. From the installation directory:

```
$ bin/nodetool status
```

Results using vnodes:

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 ?
40725dc8-7843-43ae-9c98-7c532b1f517e rack1

Results not using vnodes:

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 rack1

What's next:
Installing DataStax Enterprise 5.1

- Configuration and log file locations (page 262) - Services and package installations.
- Configuration and log file locations (page 267) - No Services and tarball installations.
- Starting and stopping DataStax Enterprise (page 1090).
- Planning and testing DSE cluster deployments.
- DataStax Studio (page 1098) documentation.

Using the DataStax Installer to install on Mac OS X

Use these instructions for installing DataStax Enterprise 5.1 using the DataStax Installer on Mac OS X.

Prerequisites:
- Mac OS X is supported for development only.
- DataStax Academy registration email address and password.
- Latest version of Oracle Java 8 is recommended.
- On some versions of Mac OS X, you may need to install readline: easy_install readline.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Minimum</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB</td>
<td>24 GB</td>
</tr>
<tr>
<td>Data directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Commit log directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Saved caches directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Logs directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
</tbody>
</table>

Also see Recommended production settings (page 272) and the DataStax Enterprise Reference Architecture white paper.

About DataStax installer

The DataStax installer installs DataStax Enterprise and DataStax Agent. It does not install OpsCenter. If you select Developer Related Tools during installation, DataStax Studio (page 1098) is installed.

The installer sets some but not all cassandra.yaml parameters. It does not set dse.yaml properties. Set the remaining parameters in the following ways:

- Manually after installation.
- To specify pre-configured cassandra.yaml and dse.yaml files, use the unattended install (page 245) with either command line (page 246) or property file (page 247).
Installing DataStax Enterprise 5.1

**Note:** The latest version of DataStax Enterprise 5.1 is 5.1.11.

In a terminal window:

1. Download the .dmg file for DataStax Enterprise from the DataStax download page or use the following command:
   
   ```
   $ curl --user dsa_email_address:password -O https://downloads.datastax.com/enterprise/DataStaxEnterprise-5.1.11-osx-installer.dmg
   
   **Note:** To install earlier versions, replace the version number in the above command. To view the available versions, see the Release notes (page 18).
   
   2. From the directory where you downloaded the install file, click the DataStaxEnterprise-5.1.11-osx-installer.dmg file.

   3. Double-click the DataStax Enterprise installer.

   ![DataStaxEnterprise-5.1.11-osx-installer.png](DataStaxEnterprise-5.1.11-osx-installer.png)

   The installer launches.

   ![Setup - DataStax Enterprise.png](Setup - DataStax Enterprise.png)

   **Setup - DataStax Enterprise**

   Welcome to DataStax Enterprise Server. This install includes the 5.1.0 version of the server along with management, monitoring, and development tools.

   4. After accepting the License Agreement, select the type of installation:

   **Service Setup**

   - **No Services:** This installation sets up the server as a standalone process.
• **Services and Utilities**: This installation sets up the server as a service. It sets up a property (.plist) file to start the service on login.

  # Simple: Installs DataStax Enterprise using the default path names and options:
  # Advanced: Allows you to configure path names and options:
  
  # User Setup (page 229)
  # Ring Options (page 228)
  # Directory Locations (page 229)
  # Ports (page 230)

5. Set the installation directory.

6. Set up the node:

<table>
<thead>
<tr>
<th>Table 20: Node Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Interface</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Cluster Name</strong></td>
</tr>
<tr>
<td><strong>Seeds</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Install developer related tools</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

7. If an installation of DataStax Enterprise already exists, the installer displays upgrade information and installs DSE in the previous installation location.
Important: Before upgrading an existing cluster, see the DataStax Upgrade Guide.

The important files to review for upgrade are dse.yaml, cassandra.yaml, and for service installs /etc/default/dse.

If dse.in.sh or cassandra.in.sh have been modified, make modifications to the cassandra-env.sh or the dse-env.sh instead. Changing these files is not recommended.

8. Set the ring options:

Table 21: Ring Options

<table>
<thead>
<tr>
<th>Enable Virtual Nodes (vnodes)</th>
<th>Enable or disable Virtual nodes (page 367).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tokens</td>
<td>Token recommendations (page 367) for vnodes:</td>
</tr>
<tr>
<td></td>
<td>• Transactional nodes: 128</td>
</tr>
<tr>
<td></td>
<td>• Analytical nodes (Spark): 128</td>
</tr>
<tr>
<td></td>
<td>• Search nodes: 8</td>
</tr>
<tr>
<td></td>
<td>• DSE Graph: 128. When used with DSE Search: 8</td>
</tr>
<tr>
<td>Listen Address</td>
<td>cassandra.yaml: listen_address (page 280)</td>
</tr>
<tr>
<td>RPC Address</td>
<td>cassandra.yaml: rpc_address (page 284)</td>
</tr>
</tbody>
</table>

9. Set up the node type:

Table 22: DSE Setup

| DSE Analytics                  | DSE Analytics (page 370) includes integration with Apache Spark™. |
| DSE Search                     | DSE Search (page 507) includes integration with Apache Solr™. |
| DSE Graph                      | DSE Graph (page 651) is a graph database for managing, analyzing, and searching highly-connected data. |
| DSE Advanced Replication       | DSE Advanced Replication (page 958) supports configurable distributed data replication from source clusters to destination clusters bi-directionally that can experience sporadic connectivity. |

10. Set up the miscellaneous options:
Table 23: Misc Options

<table>
<thead>
<tr>
<th>Start services</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>starts all services when the installation is complete.</td>
<td>select when additional configuration is needed after installation.</td>
</tr>
</tbody>
</table>

| OpsCenter Address | Optional. The server name or IP address for OpsCenter. |

If performing a Simple installation, go to 16 (page 230).

The following steps are displayed when Advanced is selected in 4 (page 240).

11. Change the default directory locations (Advanced Installations only):

Table 24: Directory Locations

<table>
<thead>
<tr>
<th>Directory Locations</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Directory</td>
<td>cassandra.yaml data_file_directories (page 281)</td>
</tr>
<tr>
<td>Commitlog Directory</td>
<td>cassandra.yaml:commitlog_directory (page 281)</td>
</tr>
<tr>
<td>Saved Caches Directory</td>
<td>cassandra.yaml:saved_caches_directory (page 281)</td>
</tr>
<tr>
<td>Hints Directory</td>
<td>cassandra.yaml:hints_directory (page 302)</td>
</tr>
<tr>
<td>Logs Directory</td>
<td>Log data for the database, Spark, and Tomcat. See the default file locations for Installer-Services and package installations (page 262) or Installer-No Services and tarball installations (page 267).</td>
</tr>
</tbody>
</table>

12. Change the default ports (Advanced Installations only):

Table 25: Ports

<table>
<thead>
<tr>
<th>Ports</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage Port</td>
<td>cassandra.yaml:storage_port (page 294)</td>
</tr>
<tr>
<td>SSL Storage Port</td>
<td>cassandra.yaml:ssl_storage_port (page 310)</td>
</tr>
<tr>
<td>RPC Port</td>
<td>cassandra.yaml:rpc_port (page 299)</td>
</tr>
</tbody>
</table>

13. In System Configuration, review any warnings about potential issues.

14. After the installation completes, review the installation logs to verify the installation.

**Note:** If you have closed the logs, see Services (page 264) or No-Services (page 269) installer locations.

DataStax Enterprise is ready for additional configuration (page 231).
15. Single-node cluster installations only:

   a. If DataStax Enterprise is not running, start it from the installation directory:

   ```
   $ bin/dse cassandra
   ```

   where the installation directory is either:

   - `/usr/share/dse`
   - the directory where you installed DataStax Enterprise.

   **Note:** For other start options, see Starting DataStax Enterprise as a stand-alone process *(page 1093).*

   b. Verify that DataStax Enterprise is running from the installation directory:

   ```
   $ bin/nodetool status
   ```

   **Results using vnodes:**
   
<table>
<thead>
<tr>
<th>Datacenter: Cassandra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status=Up/Down</td>
</tr>
<tr>
<td>/ State=Normal/Leaving/Joining/Moving</td>
</tr>
<tr>
<td>-- Address  Load  Tokens  Owns  Host ID</td>
</tr>
<tr>
<td>Rack</td>
</tr>
<tr>
<td>UN 127.0.0.1 82.43 KB 128 ?</td>
</tr>
<tr>
<td>40725dc8-7843-43ae-9c98-7c532b1f517e rack1</td>
</tr>
</tbody>
</table>

   **Results not using vnodes:**

<table>
<thead>
<tr>
<th>Datacenter: Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status=Up/Down</td>
</tr>
<tr>
<td>/ State=Normal/Leaving/Joining/Moving</td>
</tr>
<tr>
<td>-- Address  Load  Owns  Host ID</td>
</tr>
<tr>
<td>Token  Rack</td>
</tr>
<tr>
<td>UN 172.16.222.136 103.24 KB ? 3c1d0657-0990-4f78-a3c0-3e0c37fc3a06 1647352612226902707</td>
</tr>
</tbody>
</table>

   **What's next:**

   - Configuration and log file locations *(page 262)* - Services and package installations.
   - Configuration and log file locations *(page 267)* - No Services and tarball installations.
   - Starting and stopping DataStax Enterprise *(page 1090).*
   - Planning and testing DSE cluster deployments.
   - DataStax Studio *(page 1098)* documentation
cassandra.yaml
The location of the cassandra.yaml file on Mac OS X is:

```
installation_location/resources/cassandra/conf/cassandra.yaml
```

dse.yaml
The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>File Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/dse/conf/dse.yaml</td>
</tr>
</tbody>
</table>

Using the DataStax Installer to install using command line or properties file commands

Instructions for installing DataStax Enterprise 5.1 using the DataStax Installer using command line or properties file commands.

**Warning:** When DSE is installed, it creates a cassandra user in the database and runs as this user. It also creates a cassandra user in the operating system. Do not use the cassandra user in production. Using the cassandra user is a security risk. See Adding a superuser login.

**Prerequisites:**
- A supported platform.
- DataStax Academy registration email address and password.
- Root or sudo access.
- OpenJDK 8 or Oracle Java SE Runtime Environment 8 (JDK) (page 256) (1.8.0_40 minimum). Earlier or later versions are not supported.
  
  **Attention:** Although Oracle JRE/JDK 8 is supported, DataStax does more extensive testing on OpenJDK 8 starting with DSE 5.1.11. This change is due to the end of public updates for Oracle JRE/JDK 8. See Oracle Java SE Support Roadmap.

- Python 2.7.x

**Table 26: Hardware requirements**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Minimum</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB</td>
<td>24 GB</td>
</tr>
<tr>
<td>Data directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Commit log directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
</tbody>
</table>
Installing DataStax Enterprise 5.1

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Minimum</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saved caches directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Logs directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
</tbody>
</table>

Also see [Recommended production settings (page 272)](#) and the [DataStax Enterprise Reference Architecture white paper](#).

About the installer

The DataStax installer installs DataStax Enterprise and the DataStax Agent. If `developer_install` is specified, it installs DataStax Studio ([page 1098](#)). It does not install OpsCenter.

You can use this installer to set `cassandra.yaml` and `dse.yaml` properties with the following template files:

- `cassandra_yaml_template`
- `dse_yaml_template`

The DataStax Enterprise download page provides a `sample_install_5.1.prop` file.

**Note:** The latest version of DataStax Enterprise 5.1 is 5.1.11.

In a terminal window:

1. Download the installer for your computer from the [DataStax download page](#) or use the following command:

   ```
   $ curl --user dsa_email_address:password -O https://downloads.datastax.com/enterprise/DataStaxEnterprise-5.1.11-linux-x64-installer.run
   ```

   **Note:** To install earlier versions, replace the version number in the above command. To view the available versions, see the [Release notes (page 18)](#).

2. From the directory where you downloaded the install file, change the permission to executable:

   ```
   $ chmod +x DataStaxEnterprise-5.1.11-linux-x64-installer.run
   ```

3. You can either use the command line or a properties file:

   - **Command line:**
Installing DataStax Enterprise 5.1

s

For available options, see the table below (page 247). Be sure to add "--" to the option. For example:

The installer uses the default value for any unspecified option.

• Properties file:

where \texttt{option\_file\_name} is the name of the file containing the installation options. For example:

Properties file format: \texttt{option=argument}. For example:

The properties file options are the same as the command line options without the --.

Custom installation location example:

$ sudo ./DataStaxEnterprise-5.1.11-linux-x64-installer.run --optionfile ../datastax/custom_location.prop --mode unattended

system\_install=no\_services
prefix=/home/dse

\begin{table}[h]
\centering
\caption{Unattended install options}
\begin{tabular}{|c|c|l|}
\hline
Option & Argument & Description \\
\hline
Install options & & \\
\hline
\end{tabular}
\end{table}
### Installing DataStax Enterprise 5.1

<table>
<thead>
<tr>
<th>Option</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>installation_location</td>
<td>Set installation location available only for No Services installations. For upgrade installations, No Services installations use the previous installation location.</td>
</tr>
<tr>
<td>cassandra_yaml_template</td>
<td>file_name</td>
<td>Template for cassandra.yaml file.</td>
</tr>
<tr>
<td>dse_yaml_template</td>
<td>file_name</td>
<td>Template for dse.yaml file.</td>
</tr>
<tr>
<td>logs_dir</td>
<td>log_location</td>
<td>Set the log directory for the database and Spark data.</td>
</tr>
<tr>
<td>developer_install</td>
<td>developer_install</td>
<td>Install demos and Javadoc.</td>
</tr>
<tr>
<td>do_drain</td>
<td>0 (no) or 1 (yes)</td>
<td>Drain existing node before installing. Default: 1</td>
</tr>
<tr>
<td>start_services</td>
<td>0 (no) or 1 (yes)</td>
<td>Start services after installation. Default: 1</td>
</tr>
<tr>
<td>update_system</td>
<td>0 (no) or 1 (yes)</td>
<td>Update some system packages and dependencies. Does not upgrade or install major components such as Java. Default: 1 for root user, 0 for others.</td>
</tr>
</tbody>
</table>

#### Set up node type options

<table>
<thead>
<tr>
<th>install_type</th>
<th>simple or advanced</th>
<th>Set install type:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>simple</td>
<td>• <em>simple</em> - installs DataStax Enterprise using the default path names and options.</td>
</tr>
<tr>
<td></td>
<td>advanced</td>
<td>• <em>advanced</em> - configure path names and options. See Install Type (page 226).</td>
</tr>
</tbody>
</table>

Default: *simple*
<table>
<thead>
<tr>
<th>Option</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| system_install   | Use one of the following:  
• no_services  
• services_and_utilities | Set up DataStax Enterprise as a service or standalone process. If installed as a service, installs DataStax Enterprise in system locations. Default: services_and_utilities for root user, no_services for others.  
For Services installations, the installer puts DataStax Enterprise in system locations and /usr/share/dse.  
For upgrade installations, No Services installations use the previous installation location. For Services installations, the installer puts DataStax Enterprise in system locations and /usr/share/dse. |
<p>| enable_analytics | 0 (no) or 1 (yes) | Enable or disable DSE Analytics. Default: 0                                                                                                                                                                    |
| enable_search    | 0 (no) or 1 (yes) | DSE Search (page 507) includes integration with Apache Solr™. Default: 0                                                                                                                                      |
| enable_graph     | 0 (no) or 1 (yes) | DSE Graph (page 651) is a graph database for managing, analyzing, and searching highly-connected data. Default: 0                                                                                           |
| enable_advrepl   | 0 (no) or 1 (yes) | DSE Advanced Replication (page 958) supports configurable distributed data replication from source clusters to destination clusters bi-directionally that can experience sporadic connectivity. Default: 0 |
| opscenter_address| opscenter_address | Optional. The server name or IP address for OpsCenter.                                                                                                                                                       |
| Preflight check options |                      |                                                                                                                                                                                                               |
| run_pfc          | 0 (no) or 1 (yes) | Preflight tool is a collection of tests that can be run on a node to detect and fix a configuration. The tool can detect and fix many invalid or suboptimal configuration settings. Not available for No Services installations. |
| pfc_fix_issues   | 0 (no) or 1 (yes) | Attempt to fix invalid or suboptimal configuration settings. Not available for No Services installations.                                                                                                      |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pfc_ssd</strong></td>
<td><strong>paths_to_SSDs</strong></td>
<td>Paths to the Solid State Drives. Separate the drives by a comma. Not available for No Services installations.</td>
</tr>
<tr>
<td>(Services installations only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>pfc_devices</strong></td>
<td><strong>paths_to_hard_drives</strong></td>
<td>Paths to your hard drives. Separate the drives by a comma. Not available for No Services installations.</td>
</tr>
<tr>
<td>(Services installations only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>pfc_disk_duration</strong></td>
<td></td>
<td>Set to simulate a normal load. Not available for No Services installations.</td>
</tr>
<tr>
<td>(Services installations only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>pfc_disk_threads</strong></td>
<td></td>
<td>Set to simulate a normal load. Not available for No Services installations.</td>
</tr>
<tr>
<td>(Services installations only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>User and user group options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cassandra_user</strong></td>
<td><strong>user_name</strong></td>
<td>User name for running service. Because the DataStax Agent relies on user cassandra, DataStax does not recommend changing the default. If changed, you must manually install, update, and configure the DataStax Agent.</td>
</tr>
<tr>
<td><strong>cassandra_group</strong></td>
<td><strong>group_name</strong></td>
<td>Group name for running service. See above.</td>
</tr>
<tr>
<td><strong>cassandra.yaml options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>These values override options set in the <strong>cassandra_yaml_template (page 248)</strong>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cassandra_commitlog_dir</strong></td>
<td>directory</td>
<td>commitlog_dir (page 281)</td>
</tr>
<tr>
<td><strong>cassandra_data_dir</strong></td>
<td>directory</td>
<td>data_file_directories (page 281)</td>
</tr>
<tr>
<td><strong>cassandra_hints_dir</strong></td>
<td>directory</td>
<td>hints_directory (page 302)</td>
</tr>
<tr>
<td><strong>cassandra_saved_caches_dir</strong></td>
<td>directory</td>
<td>saved_caches_directory (page 281)</td>
</tr>
<tr>
<td><strong>enable_vnodes</strong></td>
<td>0 (no) or 1 (yes)</td>
<td>Enable or disable virtual nodes (vnodes). Default: 1 for transactional nodes, 0 for others.</td>
</tr>
<tr>
<td><strong>interface</strong></td>
<td><strong>IP_address</strong></td>
<td>Default interface to use for listening on all services.</td>
</tr>
<tr>
<td><strong>listen_address</strong></td>
<td><strong>IP_address</strong></td>
<td>listen_address (page 280)</td>
</tr>
</tbody>
</table>
Installing DataStax Enterprise 5.1

<table>
<thead>
<tr>
<th>Option</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_tokens</td>
<td>number_of_tokens</td>
<td>num_token (page 293)</td>
</tr>
<tr>
<td>ring_name</td>
<td>name</td>
<td>Name of ring.</td>
</tr>
<tr>
<td>rpc_address</td>
<td>IP_address</td>
<td>rpc_address (page 284)</td>
</tr>
<tr>
<td>rpc_port</td>
<td>port_number</td>
<td>rpc_port (page 299)</td>
</tr>
<tr>
<td>seeds</td>
<td></td>
<td>Comma separated list of seed IP_addresses</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Seed list for this node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do not make all nodes seed nodes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Internode communications (gossip).</td>
</tr>
<tr>
<td>ssl_storage_port</td>
<td>port_number</td>
<td>ssl_storage_port (page 310)</td>
</tr>
<tr>
<td>storage_port</td>
<td>port_number</td>
<td>storage_port (page 294)</td>
</tr>
</tbody>
</table>

dse.yaml
The location of the dse.yaml file depends on the type of installation:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
<tr>
<td>Installer-Services</td>
<td></td>
</tr>
<tr>
<td>installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services</td>
<td>resources/dse/conf/dse.yaml</td>
</tr>
<tr>
<td>installations</td>
<td></td>
</tr>
</tbody>
</table>


cassandra.yaml
The location of the cassandra.yaml file depends on the type of installation:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/dse/cassandra/cassandra.yaml</td>
</tr>
<tr>
<td>Installer-Services</td>
<td></td>
</tr>
<tr>
<td>installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services</td>
<td>resources/cassandra/conf/cassandra.yaml</td>
</tr>
<tr>
<td>installations</td>
<td></td>
</tr>
</tbody>
</table>

DataStax Enterprise is ready for additional configuration (page 231).

What’s next:
- Configuration and log file locations (page 262) - Services and package installations.
- Configuration and log file locations (page 267) - No Services and tarball installations.
- Starting and stopping DataStax Enterprise (page 1090).
- Planning and testing DSE cluster deployments.
Installing DataStax Enterprise 5.1 using the binary tarball

Use these instructions for installing DataStax Enterprise (DSE) 5.1 on Linux-based platforms using a binary tarball.

Some things to know about installing DSE

- The latest version of DataStax Enterprise 5.1 is 5.1.11.
- When DSE is installed, it creates a cassandra user in the database. Do not use the cassandra user in production. See Creating superuser accounts.
- When installed from the binary tarball, DataStax Enterprise runs as a stand-alone process.
- This procedure installs DataStax Enterprise 5.1 and the developer related tools: Javadoc, DataStax Enterprise demos, DataStax Studio, and the DSE Graph Loader.

It does not install OpsCenter, DataStax Agent, Studio (page 1098), or Graph Loader (page 831). After installing, you must configure and start DataStax Enterprise.

- After installing, you must configure and start DataStax Enterprise.
- When DSE is installed, it creates a cassandra user in the database. Do not use the cassandra user in production. See Creating superuser accounts.

Prerequisites:

- A supported platform.
- DataStax Academy registration email address and password.
- OpenJDK 8 or Oracle Java SE Runtime Environment 8 (JDK) (page 256) (1.8.0_40 minimum). Earlier or later versions are not supported.

**Attention:** Although Oracle JRE/JDK 8 is supported, DataStax does more extensive testing on OpenJDK 8 starting with DSE 5.1.11. This change is due to the end of public updates for Oracle JRE/JDK 8. See Oracle Java SE Support Roadmap.

- RedHat-compatible distributions require EPEL (Extra Packages for Enterprise Linux).
- Python 2.7.x

Table 28: Hardware requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Minimum</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB</td>
<td>24 GB</td>
</tr>
<tr>
<td>Data directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
<tr>
<td>Commit log directory</td>
<td>20 GB</td>
<td>200 GB</td>
</tr>
</tbody>
</table>
1. Verify that a required version of Java is installed:

   $ java -version

   If not OpenJDK 8 or Oracle Java 8, see Installing supporting software (page 256).

   **Important:**
   - Although Oracle JRE/JDK 8 is supported, DataStax does more extensive testing on OpenJDK 8 starting with DSE 5.1.11. This change is due to the end of public updates for Oracle JRE/JDK 8.
   - Package management tools do not install OpenJDK 8 or Oracle Java.

2. When installing from the binary tarball, you can either download the tarball and then extract the files, or use curl.

   - Download and extract the tarball specifying the version:

     **Note:** The latest version is 5.1.11. To view the available versions, see the Release notes (page 18).

     a. Using your Your DataStax Academy registration Profile Name and Downloads Key or email address and password, download the tarball from Download DataStax Enterprise.

     b. Extract the files:

        $ tar -xzvf dse-5.1.11-bin.tar.gz

   - Using curl to download and extract the tarball:

     **Caution:** If you choose to run the curl command, your password is retained in the shell history. To avoid this DataStax recommends using curl with the `--netrc` or `--netrc-file` option. Alternately, download the tarball from DataStax downloads.

        $ curl --user DSA_profile_name:downloads_key -L \  
        https://downloads.datastax.com/enterprise/dse-5.1.11-bin.tar.gz | \  
        tar xz
where `DSA_profile_name` and `downloads_key` are your DataStax Academy Profile Name and My Downloads Key. Depending on your environment, you might need to replace `@` in your email address with `%40` and escape any character in your password that is used in your operating system's command line. Examples: `!` and `\`.

For backward compatibility, you can use your DataStax Academy email address and password instead of your Profile Name and Downloads Key.

The files are downloaded and extracted into the `dse-version` directory.

3. You can use either the default data and logging directory locations or define your locations:
   - To use the default data and logging directory locations, create and change ownership for the following:

     ```sh
     $ sudo mkdir -p /var/lib/cassandra; sudo chown -R $USER:$GROUP /var/lib/cassandra &&
     sudo mkdir -p /var/log/cassandra; sudo chown -R $USER:$GROUP /var/log/cassandra &&
     sudo mkdir -p /var/lib/dsefs; sudo chown -R $USER:$GROUP /var/lib/dsefs &&
     sudo mkdir -p /var/lib/spark; sudo chown -R $USER:$GROUP /var/lib/spark &&
     sudo mkdir -p /var/log/spark; sudo chown -R $USER:$GROUP /var/log/spark &&
     sudo mkdir -p /var/lib/spark/rdd; sudo chown -R $USER:$GROUP /var/lib/spark/rdd &&
     sudo mkdir -p /var/lib/spark/worker; sudo chown -R $USER:$GROUP /var/lib/spark/worker
     ```

   - To define your own data and logging directory locations:
     a. In the `installation_location`, make the directories for data and logging directories. For example:

        ```sh
        $ mkdir installation_location/dse-data &&
        cd dse-data &&
        mkdir commitlog &&
        mkdir saved_caches &&
        mkdir hints &&
        mkdir cdc_raw
        ```

     b. Go the directory containing the `cassandra.yaml` file:

        ```sh
        $ cd installation_location/resources/cassandra/conf
        ```

     c. Edit the following lines in the `cassandra.yaml` file:
Installing DataStax Enterprise 5.1

- **Optional:** Define your own Spark directories:
  
  a. Make the directories for the Spark `lib` and `log` directories.
  
  b. Edit the `spark-env.sh` file to match the locations of your Spark `lib` and `log` directories, as described in Configuring Spark nodes (page 410).
  
  c. Make a directory for the DSEFS data directory and set its location in `dsefs_options` (page 339).

**spark-env.sh**

The default location of the `spark-env.sh` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/spark/spark-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/spark/spark-env.sh</td>
</tr>
<tr>
<td>Tarball installations</td>
<td><code>installation_location/</code></td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td><code>resources/spark/conf/spark-env.sh</code></td>
</tr>
</tbody>
</table>

DataStax Enterprise is ready for additional configuration (page 231).

4. Single-node cluster installations only:

   a. Start DataStax Enterprise from the installation directory:

      ```
      $ bin/dse cassandra
      ```

      **Note:** For other start options, see Starting DataStax Enterprise as a stand-alone process (page 1093).

   b. From the *installation directory*, verify that DataStax Enterprise is running:

      ```
      $ bin/nodetool status
      ```

      Results using vnodes:

      ```
      Datacenter: Cassandra
      ```
Installing DataStax Enterprise 5.1

Results not using vnodes:

Datacenter: Analytics

<table>
<thead>
<tr>
<th>Status=Up/Down</th>
<th>State=Normal/Leaving/Joining/Moving</th>
<th>--</th>
<th>Address</th>
<th>Load</th>
<th>Tokens</th>
<th>Owns</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>UN</td>
<td>127.0.0.1</td>
<td>82.43 KB</td>
<td>128</td>
<td>?</td>
<td>40725dc8-7843-43ae-9c98-7c532b1f517e</td>
</tr>
</tbody>
</table>

What's next:

- Configuration and log file locations (page 262) - Services and package installations.
- Configuration and log file locations (page 267) - No Services and tarball installations.
- Starting and stopping DataStax Enterprise (page 1090).
- Planning and testing DSE cluster deployments.
- DataStax Studio (page 1098) documentation.

Installing on Docker

Use DataStax Docker images to create DataStax Enterprise (DSE) server, DSE OpsCenter, and DataStax Studio containers in non-production environments.

See the DataStax Academy Quick Downloads page for information on downloading and using the DataStax images for Docker.

Note: A valid DataStax Academy account is required to access the images and documentation.

Installing supporting software

Installing Oracle JDK on Debian or Ubuntu Systems

Configure your operating system to use the latest version of Oracle Java Platform, Standard Edition 8.
The Oracle Java Platform, Standard Edition (JDK) has been removed from the official software repositories of Ubuntu and only provides a binary (.bin) version. You can get the JDK from the [Java SE Downloads](https://www.oracle.com/java/technologies/javase-downloads.html).

1. Check which version of the JDK your system is using:

   ```bash
   $ java -version
   
   If Oracle Java, the results should look like:
   
   java version "1.8.0_65"
   Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
   Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)
   
2. If necessary, go to Oracle Java SE Downloads, accept the license agreement, and download the installer for your distribution.

   **Note:** If installing the Oracle JDK in a cloud environment, accept the license agreement, download the installer to your local client, and then use `scp` (secure copy) to transfer the file to your cloud machines.

3. Make a directory for the JDK:

   ```bash
   $ sudo mkdir -p /usr/lib/jvm
   
4. Unpack the tarball and install the JDK:

   ```bash
   $ sudo tar zxfv jdk-8u65-linux-x64.tar.gz -C /usr/lib/jvm
   
   The JDK files are installed into a directory called `/usr/lib/jvm/jdk-8u_version`.

5. Tell the system that there's a new Java version available:

   ```bash
   $ sudo update-alternatives --install "/usr/bin/java" "java" "/usr/lib/jvm/jdk1.8.0_version/bin/java" 1
   
   If updating from a previous version that was removed manually, you may need to execute the above command twice, because you'll get an error message the first time.

6. Set the new JDK as the default using the following command:

   ```bash
   $ sudo update-alternatives --config java
   
7. Make sure your system is using the correct JDK:
Installing Oracle JDK on RHEL-based Systems

Configure your operating system to use the latest version of Oracle Java Platform, Standard Edition 8.

1. Check which version of the JDK your system is using:

   $ java -version

   If Oracle Java, the results should look like:

   ```
   java version "1.8.0_65"
   Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
   Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)
   ```

2. If necessary, go to Oracle Java SE Downloads, accept the license agreement, and download the installer for your distribution.

   **Note:** If installing the Oracle JDK in a cloud environment, accept the license agreement, download the installer to your local client, and then use `scp` (secure copy) to transfer the file to your cloud machines.

3. From the directory where you downloaded the package, run the install:

   ```
   $ sudo rpm -ivh jdk-8u_version-linux-x64.rpm
   ```

   The RPM installs the JDK into the `/usr/java/` directory.

4. Set your system to use the Oracle JDK:

   ```
   $ sudo alternatives --install /usr/bin/java java /usr/java/jdk1.8.0_version/bin/java 200000
   ```

5. Use the `alternatives` command to switch to the Oracle JDK.

   ```
   $ sudo alternatives --config java
   ```

   **Note:** If you have trouble, you may need to set JAVA_HOME and PATH in your profile, such as `.bash_profile`. 
The following examples assume that the JDK is in `/usr/java` and which `java` shows `/usr/bin/java`:

- **Shell or bash:**
  ```bash
  $ export JAVA_HOME=/usr/java/latest
  $ export PATH=$JAVA_HOME/bin:$PATH
  ```

- **C shell (csh):**
  ```csh
  $ setenv JAVA_HOME "/usr/java/latest"
  $ setenv PATH $JAVA_HOME/bin:$PATH
  ```

6. **Make sure your system is using the correct JDK:**

   ```bash
   java -version
   ```

   For example, Java SE Runtime Environment 1.0.8_65:

   ```
   java version "1.8.0_65"
   Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
   Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)
   ```

**Installing OpenJDK on Debian-based Systems**

Configure your operating system to use the OpenJDK 8.

In a terminal:

1. **Install the OpenJDK 8 from a PPA repository:**

   ```bash
   $ sudo add-apt-repository ppa:openjdk-r/ppa
   ```

2. **Update the system package cache and install:**

   ```bash
   $ sudo apt-get update
   $ sudo apt-get install openjdk-8-jdk
   ```

3. **If you have more than one Java version installed on your system use the following command to switch versions:**

   ```bash
   $ sudo update-alternatives --config java
   ```
4. Make sure your system is using the correct JDK:

```
$ java -version

openjdk version "1.8.0_72-internal"
OpenJDK Runtime Environment (build 1.8.0_72-internal-b05)
OpenJDK 64-Bit Server VM (build 25.72-b05, mixed mode)
```

### Installing OpenJDK on RHEL-based Systems

Configure your operating system to use the OpenJDK 8.

In a terminal:

1. Install the OpenJDK 8:

```
$ su -c "yum install java-1.8.0-openjdk"
```

2. If you have more than one Java version installed on your system use the following command to switch versions:

```
$ sudo alternatives --config java
```

3. Make sure your system is using the correct JDK:

```
$ java -version

openjdk version "1.8.0_71"
OpenJDK Runtime Environment (build 1.8.0_71-b15)
OpenJDK 64-Bit Server VM (build 25.71-b15, mixed mode)
```

### Uninstalling DataStax Enterprise 5.1

Select the uninstall method for your type of installation.

#### Uninstalling from the DataStax Installer

Use this method when you have installed DataStax Enterprise from the DataStax Installer (page 223).

1. Go to the server installation directory:
   - If installed on Linux with root permissions: $ /usr/share/dse
   - If installed on Linux without root permissions: $ installation_location/dse
   - If installed on Mac OS X $ installation_location/dse
2. Launch the uninstaller:

   - **Linux:** $ ./uninstall ## Run the uninstaller as root or sudo if needed
   - **Mac OS X:** Double-click uninstaller.

3. Select the type of uninstall and follow the instructions on the uninstaller.
   
   **Note:** If you are going to reinstall DataStax Enterprise with the existing data files, be sure to drain the node and move the files somewhere else before uninstalling.

Using the Unattended Uninstaller

To use this method, you must have installed DataStax Enterprise from the DataStax Installer (page 223).

1. Create a configuration file called `uninstall.property` in the same directory as the uninstaller. For example:

   ```bash
   /usr/share/dse/uninstall.property
   ```

2. In the `uninstall.property` file, set the required properties:
   
   - `do_drain=1|0` - drains the node before uninstalling
   - `full_uninstall=1|0` - uninstalls all components
     
     where 1=yes and 0=no.

3. From the directory containing the uninstaller:

   ```bash
   $ sudo ./uninstall --mode unattended
   ```

Uninstalling Debian- and RHEL-based packages

Use this method when you have installed DataStax Enterprise using APT or Yum.

1. Stop the DataStax Enterprise service:

   ```bash
   $ nodetool drain
   $ sudo service dse stop
   ```

2. Make sure all services are stopped:

   ```bash
   $ ps auwx | grep dse
   ```

3. If services are still running, use the PID to kill the service:
$ bin/dse cassandra-stop -p dse_pid

4. Remove the installation directories:

**RHEL-based packages:**

$ sudo yum remove "dse-*" "datastax-*"

**Debian-based packages:**

$ sudo apt-get purge "dse-*" "datastax-*"

Uninstalling the binary tarball

Use this method when you have installed DataStax Enterprise using the binary tarball (page 252).

1. Stop the node:

   $ bin/dse cassandra-stop ## Use sudo if needed

2. Make sure all services are stopped:

   $ ps auwx | grep dse

3. If services are still running, use the PID to kill the service:

   $ bin/dse cassandra-stop -p dse_pid

4. Remove the installation directory.

   $ rm -r installation_directory

**Default file locations**

**Default file locations for package and Installer-Services installations**

The default location of the files depend on how DataStax Enterprise is installed:

- The files are located in the same locations for Package installations and when the Services is selected using the DataStax Installer.
- The files are located in the same locations for Tarball and when No Services is selected using the DataStax Installer.
Default directories for cassandra.yaml and dse.yaml

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/dse/cassandra/cassandra.yaml</td>
<td>cassandra.yaml (page 278) is the main configuration file for the DataStax Enterprise database with default configuration for all nodes.</td>
</tr>
<tr>
<td>/etc/dse/dse.yaml</td>
<td>dse.yaml (page 312) is the main configuration file for DataStax Enterprise.</td>
</tr>
</tbody>
</table>

Default database directories

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/var/lib/cassandra/data</td>
<td>commitlog, data, hints, saved_caches directories</td>
</tr>
</tbody>
</table>
| /var/log/cassandra | Log files, including:  
  - audit directory  
  - debug.log  
  - gremlin.log  
  - solrvalidation.log  
  - system.log  
You can change logging locations. |
| /var/run/cassandra | Database process ID (pid) directory |
| /usr/share/dse/cassandra/tools | Tools for testing, starting, using SSTables, plus YAML examples. |
| /etc/dse/cassandra | Property files and cqlshrc samples including:  
  - cassandra-env.sh  
  - cassandra-rackdc.properties  
  - cassandra-topology.properties  
  - cassandra-topology.yaml  
  - commitlog_archiving.properties  
  - cqlshrc.sample  
  - logback.xml |
| /etc/init.d | Set node type and other server configuration |

Default DSEFS data directory

The default location for the DSEFS data directory is /var/lib/dsefs.
### Default DataStax Enterprise Installer directories, install logs, and uninstaller

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
</table>
| /usr/share/dse/backups/log_file_dir | Backup and log files:  
  - bitrock_installer.log  
  - copied_config_files.log  
  - install_dependencies.log  
  - pfc_results.txt  
  - Backup files from previous releases |
| /usr/share/dse/uninstall | Uninstall DataStax Enterprise (page 260) |

### Default DSE Graph directories

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/dse/graph/gremlin-console/conf/remote.yaml</td>
<td>Gremlin console configuration for connection to the Gremlin Server, including Kerberos authentication and SSL encryption.</td>
</tr>
<tr>
<td>/etc/dse/graph/logback-gremlin-server.xml</td>
<td>GremlinServerFileAppender</td>
</tr>
</tbody>
</table>

### Default DSE Search directories

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/share/dse/resources/solr/conf</td>
<td>Solr configuration</td>
</tr>
<tr>
<td>/usr/share/dse/demos/wikipedia</td>
<td>Search - Wikipedia demo</td>
</tr>
<tr>
<td>/usr/share/dse/solr/web/demos/wikipedia</td>
<td>Search - Wikipedia demo with Tomcat</td>
</tr>
<tr>
<td>/var/log/cassandra</td>
<td>Search log messages are in the system.log file</td>
</tr>
</tbody>
</table>

### Default Spark directories

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/dse/spark/</td>
<td>spark-env.sh (page 410), spark-defaults.conf, spark-daemon-defaults.conf</td>
</tr>
<tr>
<td>/usr/share/dse/spark/lib</td>
<td>Spark library</td>
</tr>
<tr>
<td>/var/log/spark</td>
<td>Spark Master and Worker logs</td>
</tr>
<tr>
<td>/usr/share/dse/spark/spark-jobserver</td>
<td>Spark Jobserver</td>
</tr>
</tbody>
</table>
Directories

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/share/dse/demos/</td>
<td>Spark Portfolio Manager demo</td>
</tr>
<tr>
<td>portfolio_manager</td>
<td></td>
</tr>
<tr>
<td>/var/lib/dsefs</td>
<td>The default directory to store the DSE File</td>
</tr>
<tr>
<td>System data.</td>
<td></td>
</tr>
</tbody>
</table>

Default location for the logback configuration file

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/dse/cassandra/logback.xml</td>
<td>logback.xml is the logback configuration file</td>
</tr>
</tbody>
</table>

Default location audit logs

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/dse/tomcat/conf/server.xml</td>
<td>Default location for Tomcat server logs for DSE</td>
</tr>
<tr>
<td>/etc/dse/tomcat/conf/server.xml</td>
<td>Search.</td>
</tr>
<tr>
<td>/var/log/cassandra/dropped_audit_events.log</td>
<td>Default location for dropped events logs.</td>
</tr>
</tbody>
</table>

Default OpsCenter directories

See the [OpsCenter documentation](https://docs.datastax.com/en/dse/5.1/opscenter/).  

Default DSE Multi-Instance configuration files

With DSE Multi-Instance, multiple DataStax Enterprise nodes reside on a single host machine. To segregate the configuration for each DataStax Enterprise node, node-specific directory structures are used to store configuration and operational files. For example, in addition to `/etc/dse/dse.yaml`, the DSE Multi-Instance `dse.yaml` files are stored in `/etc/dse-nodeId/dse.yaml` locations. The `server_id` option is generated in DSE Multi-Instance `/etc/dse-nodeId/dse.yaml` files to uniquely identify the physical server on which multiple instances are running and is unique for each database instance. See [DSE Multi-Instance server_id (page 347)](https://docs.datastax.com/en/dse/5.1/multi-instance/tomcat/index.html).

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/dse/dse.yaml</td>
<td>/etc/dse/dse.yaml is the primary configuration</td>
</tr>
<tr>
<td></td>
<td>file for DataStax Enterprise</td>
</tr>
<tr>
<td>/etc/dse-node1/dse.yaml</td>
<td>/etc/dse-node1/dse.yaml is the configuration</td>
</tr>
<tr>
<td></td>
<td>file for the DataStax Enterprise node in the</td>
</tr>
<tr>
<td></td>
<td>dse-node1 directory</td>
</tr>
<tr>
<td>/etc/dse-node2/dse.yaml</td>
<td>/etc/dse-node2/dse.yaml is the configuration</td>
</tr>
<tr>
<td></td>
<td>file for the DataStax Enterprise node in the</td>
</tr>
<tr>
<td></td>
<td>dse-node2 directory</td>
</tr>
</tbody>
</table>

For a comprehensive list of file locations in a DSE Multi-Instance cluster, see [directories for DSE Multi-Instance (page 266)](https://docs.datastax.com/en/dse/5.1/multi-instance/tomcat/index.html).
Default DSE Multi-Instance generated directories

With DSE Multi-Instance, these directories are created on the host machine for each node.

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSE Multi-Instance root directory</td>
<td><code>/etc/defaults</code></td>
<td>Each DSE Multi-Instance host machine has the <code>/etc/defaults</code> root directory. This default location is not configurable.</td>
</tr>
<tr>
<td>DataStax Enterprise node type</td>
<td><code>/etc/defaults/dse-nodeId</code></td>
<td>Defines the node type (transactional, search, analytics, graph, and so on).</td>
</tr>
<tr>
<td>DataStax Enterprise configuration file</td>
<td><code>/etc/dse-nodeId/dse.yaml</code></td>
<td>The <code>dse.yaml</code> (page 312) configuration file for each node.</td>
</tr>
<tr>
<td>DataStax Enterprise process ID (pid) directory</td>
<td><code>/var/run/dse-nodeId.dse-nodeId.pid</code></td>
<td>The default DataStax Enterprise process ID (pid) directory for each node.</td>
</tr>
<tr>
<td>Database configuration</td>
<td><code>/etc/dse-nodeId/cassandra/cassandra.yaml</code></td>
<td>The <code>cassandra.yaml</code> (page 278) configuration file for each node.</td>
</tr>
<tr>
<td>Database data directory</td>
<td><code>/var/lib/dse-nodeId/data</code></td>
<td>The root directory for storing data on each node. Define with <code>dse add-node --data-directory-directory ...</code></td>
</tr>
<tr>
<td>Database log files</td>
<td><code>/var/log/dse-nodeId/cassandra</code></td>
<td>The default directory where the audit.log, output.log, solrvalidation.log, and system.log log files are stored for each node. Define with <code>dse add-node --logs-directory ...</code></td>
</tr>
<tr>
<td>Database pid directory</td>
<td><code>/var/run/dse-nodeId</code></td>
<td>Database process ID (pid) directory for each node.</td>
</tr>
<tr>
<td>Caches directory</td>
<td><code>/var/lib/dse-nodeId/saved_caches</code></td>
<td>The table key and row caches directory for each node. Define with <code>dse add-node --saved-caches-directory-directory ...</code></td>
</tr>
<tr>
<td>Commit log files</td>
<td><code>/var/lib/dse-nodeId/commitlog</code></td>
<td>The commit log directory for each node. Define with <code>dse add-node --commit-directory-directory ...</code></td>
</tr>
<tr>
<td>Purpose</td>
<td>Directories</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Hints directory</td>
<td>/var/lib/dse-nodeId/ hints</td>
<td>The hints directory for each node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define with <code>dse add-node --hints-directory=directory ...</code>.</td>
</tr>
<tr>
<td>Spark configuration file</td>
<td>/etc/dse-nodeId/spark/spark-env.sh</td>
<td>Spark configuration file <code>spark-env.sh (page 410)</code> for each node.</td>
</tr>
<tr>
<td>Spark Worker data directory</td>
<td>/var/lib/dse-nodeId/spark/worker</td>
<td>The data directory for Spark Worker for each node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define with <code>dse add-node --spark-worker-directory=directory ...</code>.</td>
</tr>
<tr>
<td>Spark Worker local node</td>
<td>/var/lib/dse-nodeId/spark/rdd</td>
<td>The local directory for Spark Worker for each node.</td>
</tr>
<tr>
<td>directory</td>
<td></td>
<td>Define with <code>dse add-node --spark-local-directory=directory ...</code>.</td>
</tr>
<tr>
<td>Spark logs directory</td>
<td>/var/log/dse-nodeId/spark</td>
<td>The Spark logs directory for each node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define with <code>dse add-node --spark-log-directory=directory ...</code>.</td>
</tr>
<tr>
<td>Logback configuration</td>
<td>/etc/dse-nodeId/cassandra/logback.xml</td>
<td>Logback configuration file for each node.</td>
</tr>
<tr>
<td>Solr configuration</td>
<td>/etc/dse-nodeId/solr</td>
<td>Solr configuration files for each node.</td>
</tr>
<tr>
<td>Tomcat log files</td>
<td>/var/log/dse-nodeId/tomcat</td>
<td>The directory for Tomcat server logs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define with <code>dse add-node --tomcat-logs=directory ...</code>.</td>
</tr>
</tbody>
</table>

**Licenses and other documents**

The default location is `installation_location`. Also see DataStax Enterprise third-party software.

**Default file locations for tarball and Installer-No Services installations**

The default location of the files depend on how DataStax Enterprise is installed:

- The files are located in the same locations for Package installations and when the Services is selected using the DataStax Installer.
• The files are located in the same locations for Tarball and when No Services is selected using the DataStax Installer.

Default installation location

The default installation_location depends on whether you installed DataStax Enterprise by using the DataStax Installer or from the binary tarball:

• **Installer-No Services (page 223):** `/usr/share/dse`
• **Tarball installation (page 252):** The location where you extracted DataStax Enterprise.

Default directories for cassandra.yaml and dse.yaml

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>installation_location/resources/cassandra/conf/cassandra.yaml</code></td>
<td><strong>cassandra.yaml</strong> <em>(page 278)</em> is the main configuration file for the database.</td>
</tr>
<tr>
<td><code>installation_location/resources/cassandra/conf/installation_location/resources/dse/conf/dse.yaml</code></td>
<td><strong>dse.yaml</strong> <em>(page 312)</em> is the main configuration file for DataStax Enterprise.</td>
</tr>
</tbody>
</table>

Default database directories

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>installation_location/resources/cassandra/bin</code></td>
<td>Commands and utilities, such as nodetool, cqlsh, sstabledump, and sstableloader</td>
</tr>
<tr>
<td><code>installation_location/resources/cassandra/conf</code></td>
<td>Property files and cqlshrc samples including:</td>
</tr>
<tr>
<td></td>
<td>• cassandra-env.sh</td>
</tr>
<tr>
<td></td>
<td>• cassandra-rackdc.properties</td>
</tr>
<tr>
<td></td>
<td>• cassandra-topology.properties</td>
</tr>
<tr>
<td></td>
<td>• cassandra-topology.yaml</td>
</tr>
<tr>
<td></td>
<td>• commitlog_archiving.properties</td>
</tr>
<tr>
<td></td>
<td>• cqlshrc.sample</td>
</tr>
<tr>
<td></td>
<td>• logback.xml</td>
</tr>
<tr>
<td><code>/var/lib/cassandra</code> or <code>installation_location</code></td>
<td>commitlog, data, hints, saved_caches directories</td>
</tr>
</tbody>
</table>
### Directories

<table>
<thead>
<tr>
<th><strong>Directories</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/var/log/cassandra</code></td>
<td>Log files, including:</td>
</tr>
<tr>
<td></td>
<td>• audit directory</td>
</tr>
<tr>
<td></td>
<td>• debug.log</td>
</tr>
<tr>
<td></td>
<td>• gremlin.log</td>
</tr>
<tr>
<td></td>
<td>• solrvalidation.log</td>
</tr>
<tr>
<td></td>
<td>• system.log</td>
</tr>
<tr>
<td></td>
<td>You can <a href="#">change logging locations</a>.</td>
</tr>
</tbody>
</table>

**Default DSEFS data directory**

The default location for the DSEFS data directory is `/var/lib/dsefs`.

**Default DataStax Enterprise Installer and install log directories**

<table>
<thead>
<tr>
<th><strong>Directories</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/installation_location/dse/backups/log_file_dir/copied_config_files.log</code></td>
<td>Show Config File Overwrites</td>
</tr>
<tr>
<td><code>/installation_location/dse/backups/log_file_dir/bitrock_installer.log</code></td>
<td>View Installation Log</td>
</tr>
<tr>
<td><code>/installation_location/dse/backups/log_file_dir/install_dependencies.log</code></td>
<td>View Dependency Installation Log</td>
</tr>
<tr>
<td><code>/installation_location/dse/README.md</code></td>
<td>View README</td>
</tr>
<tr>
<td><code>/installation_location/dse/uninstall</code></td>
<td>Uninstall DataStax Enterprise</td>
</tr>
</tbody>
</table>

**Default DSE Graph directories**

<table>
<thead>
<tr>
<th><strong>Directories</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/installation_location/resources/graph/gremlin-console/conf/remote.yaml</code></td>
<td>Gremlin console configuration for connection to the Gremlin Server, including Kerberos authentication and SSL encryption.</td>
</tr>
<tr>
<td><code>/installation_location/resources/graph/conf/logback-gremlin-server.xml</code></td>
<td>GremlinServerFileAppender</td>
</tr>
</tbody>
</table>

**Default DSE Search directories**

<table>
<thead>
<tr>
<th><strong>Directories</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/installation_location/resources/solr/conf</code></td>
<td>Solr configuration</td>
</tr>
<tr>
<td><code>/installation_location/resources/solr/lib</code></td>
<td>Solr driver</td>
</tr>
</tbody>
</table>
### Directories

| installation_location/demos/wikipedia | Search - Wikipedia demo |
| /var/log/cassandra | Search log messages are in the system.log file. |

#### Default Spark directories

| installation_location/resources/spark/conf | spark-env.sh (page 410), spark-defaults.conf, spark-daemon-defaults.conf |
| /var/lib/spark | Spark library |
| /var/log/spark | Spark Master and Worker logs |
| installation_location/resources/spark/spark-jobserver | Spark Jobserver |
| installation_location/demos/portfolio_manager | Spark Portfolio Manager demo |
| /var/lib/dsefs | The default directory to store the DSE File System data. |

#### Default Logback-appender directories

| installation_location/resources/cassandra/conf/logback.xml | logback.xml is the logback configuration file |

#### Default location audit logs

| /var/log/cassandra/dropped_audit_events.log | Default location for dropped events logs. |

#### Default OpsCenter directories

See the [OpsCenter documentation](#).
Default directory for Token-generator tool

<table>
<thead>
<tr>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>installation_location/resources/cassandra/tools/bin/token-generator</td>
<td>For manually Generating tokens.</td>
</tr>
</tbody>
</table>

Licenses and other documents

The default location is installation_location. Also see DataStax Enterprise third-party software.
Recommended production settings

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

- Use the latest Java Virtual Machine (page 272)
- Synchronize clocks (page 272)
- TCP settings (page 273)
- Disable CPU frequency scaling (page 274)
- Make sure that new settings persist after reboot (page 274)
- Optimize SSDs (page 274)
- Use the optimum --setra setting for RAID on SSD (page 275)
- Disable zone_reclaim_mode on NUMA systems (page 275)
- Set user resource limits (page 275)
- Disable swap (page 276)
- Check the Java Hugepages setting (page 277)
- Set the heap size for optional Java garbage collection in DataStax Enterprise (page 277)
- Determining the heap size when using Concurrent-Mark-Sweep (CMS) garbage collection in DataStax Enterprise (page 278)
- Set the heap size for optimal Java garbage collection (page 278)
- Apply optimum blockdev --setra settings for RAID on spinning disks (page 278)

Use the latest Java Virtual Machine

Use the latest 64-bit version of Oracle Java Platform, Standard Edition 8 (JDK) or OpenJDK 8.

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 \nnet.ipv4.tcp_keepalive_probes=3 \nnet.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 Firewall idle connection timeout causing nodes to lose communication during low traffic times on Linux.

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

```bash
$ sudo sysctl -w \
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
```

To persist the TCP settings:

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

   ```bash
   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:

   ```bash
   $ 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:

```bash
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:

   ```
   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 8KB, the same as it is for non-RAID SSDs. For details, see Optimizing SSDs (page 274).

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 and Installer-Services installations:

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

   ```
   <cassandra_user> - memlock unlimited
   <cassandra_user> - nofile 100000
   <cassandra_user> - nproc 32768
   <cassandra_user> - as unlimited
   ```

   Tarball and Installer-No Services 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 100000
<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 100000
  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:

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

To confirm the limits are applied to the DataStax Enterprise process, run the following command where <pid> is the process ID of the currently running DataStax Enterprise process:

$ cat /proc/<pid>/limits

For more information, see Insufficient user resource limits errors.

Disable swap

Failure to disable swap entirely can severely lower performance. 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.
If you insist on using swap, you can set `vm.swappiness=1`. This allows the kernel swap out the absolute least used parts.

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

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:

```bash
echo never | sudo tee /sys/kernel/mm/transparent_hugepage/defrag
```

For more information, including a temporary fix, see [No DSE processing but high CPU usage](#).

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

The default JVM garbage collection (GC) for DataStax Enterprise 5.1 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 ¼ and ½ 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.
Configuration

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 resources 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 274).

cassandra-env.sh

The location of the cassandra-env.sh file depends on the type of installation:

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/dse/cassandra/cassandra-env.sh</td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/cassandra/conf/cassandra-env.sh</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td></td>
</tr>
</tbody>
</table>

YAML and configuration properties

cassandra.yaml configuration file

The cassandra.yaml file is the main configuration file for DataStax Enterprise. The dse.yaml 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. The file is located in the following directories:
• Package installations and Installer-Services: /etc/dse/cassandra
• Tarball installations and Installer-No Services: installation_location/resources/cassandra/conf

Syntax

For the properties in each section, the main setting has zero spaces, and at least two spaces are required before each entry in that section. Adhere to the YAML syntax.

Organization

The configuration properties are grouped into the following sections:

• Quick start (page 279)
  The minimal properties needed for configuring a cluster.

• Commonly used (page 281)
  Properties most frequently used when configuring DataStax Enterprise.

• Performance tuning (page 287)
  Tuning performance and system resource utilization, including commit log, compaction, memory, disk I/O, CPU, reads, and writes.

• Advanced (page 291)
  Properties for advanced users or properties that are less commonly used.

• Security (page 304)
  DSE Unified Authentication provides authentication, authorization, and role management. Enabling DSE Unified Authentication requires additional configuration in dse.yaml, see Configuring DSE Unified Authentication.

• Continuous paging (page 310) Properties configure memory, threads, and duration when pushing pages continuously to the client.

  **Note:** Values with note mark default values that are defined internally, missing, or commented out, or whose implementation depends on other properties in the cassandra.yaml file. Additionally, some commented-out values may not match the actual default values. These are recommended alternatives to the default values.

Quick start properties

The minimal properties needed for configuring a cluster.

Related information: Initializing a single datacenter per workload type and Initializing multiple datacenters per workload type.

cluster_name
Default: Test Cluster 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.

**listen_address**

Default: localhost 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.

Correct settings for various use cases:

- **Single-node installations:** do one of the following:
  
  # Comment this property out. If the node is properly configured (host name, name resolution, and so on), the database uses InetAddress.getLocalHost() to get the local address from the system.
  
  # Leave set to the default, localhost.

- **Node in a multi-node installations:** set this property to the node's IP address or hostname, or set listen_interface (page 280).

- **Node in a multi-network or multi-Datacenter installation, within an EC2 environment that supports automatic switching between public and private interfaces:** set listen_address to the node’s IP address or hostname, or set listen_interface (page 280).

- **Node with two physical network interfaces in a multi-datacenter installation or cluster deployed across multiple Amazon EC2 regions using the Ec2MultiRegionSnitch:**
  
  1. Set listen_address to this node's private IP or hostname, or set listen_interface (page 280) (for communication within the local datacenter).

  2. Set broadcast_address (page 292) to the second IP or hostname (for communication between datacenters).

  3. Set listen_on_broadcast_address (page 292) to true.

  4. If this node is a seed node, add the node’s public IP address or hostname to the seeds (page 285) list.

- Open the storage_port (page 294) or ssl_storage_port (page 310) on the public IP firewall.

**listen_interface**

Default: eth0. note (page 279) 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, do not set both.

*Default directories*

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

**cdc_raw_directory**

The directory where the CDC log is stored. Default locations:

- Package installations and Installer-Services: /var/lib/cassandra/cdc_raw
- Tarball installations and Installer-No Services: /var/lib/cassandra/cdc_raw

The directory where Change Data Capture logs are stored.

**commitlog_directory**

The directory where the commit log is stored. Default location: /var/lib/cassandra/commitlog

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

**data_file_directories**

The directory location where table data is stored (in SSTables). The database distributes data evenly across the location, subject to the granularity of the configured compaction strategy. Default locations: /var/lib/cassandra/data.

For production, DataStax recommends RAID 0 and SSDs.

**saved_caches_directory**

The directory location where table key and row caches are stored. For all installations, the default location of the saved_caches directory is /var/lib/cassandra/saved_caches.

**Commonly used properties**

Properties most frequently used when configuring DataStax Enterprise.

Before starting a node for the first time, you should carefully evaluate your requirements.

**Common initialization properties**

**Note:** Be sure to set the properties in the Quick start section (page 279) as well.

**commit_failure_policy**

Default: stop. Policy for commit disk failures:

- **die**
Shut down gossip and Thrift and kill the JVM, so the node can be replaced.

- **stop**
  
  Shut down gossip and Thrift, 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.

**prepared_statements_cache_size_mb**

Default: auto, which is \( \frac{1}{256} \)th of the heap or 10 MB, whichever is greater. Maximum size of the native protocol prepared statement cache.

**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.

**thrift_prepared_statements_cache_size_mb**

Default: auto, which is \( \frac{1}{256} \)th of the heap or 10 MB, whichever is greater. Maximum size of the Thrift prepared statement cache. Leave empty if you do not use Thrift.

**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.

**disk_optimization_strategy**

Default: disabled. \(^\text{note} (page\ 279)\) The strategy for optimizing disk reads. Possible values: ssd or spinning.

**disk_failure_policy**

Default: stop. Sets how the database responds to disk failure. Recommend settings: **stop** or **best_effort**. Valid values:

- **die**

  Shut down gossip and Thrift and kill the JVM for any file system errors or single SSTable errors, so the node can be replaced.

- **stop_paranoid**

  Shut down gossip and Thrift even for single SSTable errors.

- **stop**
Shut down gossip and Thrift, 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 allows obsolete data at consistency level of ONE.

- **ignore**
  Ignore fatal errors and let the requests fail; all file system errors are logged but otherwise ignored.

Related information: Handling Disk Failures In Cassandra 1.2 blog and Recovering from a single disk failure using JBOD.

**endpoint_snitch**
Default: DseSimpleSnitch. Set to 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**
  Only appropriate 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.

cassandra-rackdc.properties
The location of the cassandra-rackdc.properties file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/cassandra/cassandra-rackdc.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/cassandra/conf/cassandra-rackdc.properties</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td></td>
</tr>
</tbody>
</table>

cassandra-topology.properties
The location of the cassandra-topology.properties file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/cassandra/cassandra-topology.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
</tbody>
</table>
### Configuration

**Tarball installations**

| Installer-No Services | installation_location/resources/cassandra/conf/cassandra-topology.properties |

- **PropertyFileSnitch**

  Determines proximity by rack and datacenter, which 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 and the availability zone as the rack and uses only private IP addresses. For this reason, it does not work across multiple regions.

- **Ec2MultiRegionSnitch**

  Uses the public IP as the broadcast_address (page 292) to allow cross-region connectivity. This means you must also set seed (page 285) addresses to the public IP and open the storage_port (page 294) or ssl_storage_port (page 310) 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.

**Related information:** Snitches

**rpc_address**

Default: localhost. The listen address for client connections (Thrift RPC service and native transport). Valid values:

- **unset**:
Resolves the address using the configured hostname configuration of the node. If left unset, the hostname resolves to the IP address of this node using /etc/hostname, /etc/hosts, or DNS.

- 0.0.0.0:
  Listens on all configured interfaces. You must set the broadcast_rpc_address (page 299) to a value other than 0.0.0.0.

- IP address
- hostname

Related information: Network

rpc_interface
Default: eth1. note (page 279) The listen address for client connections. Interface must correspond to a single address, IP aliasing is not supported. See rpc_address (page 284).

rpc_interface_prefer_ipv6
Default: false. 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.

seed_provider
The addresses of hosts 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 (Default: org.apache.cassandra.locator.SimpleSeedProvider)
  The class that handles the seed logic. It can be customized, but this is typically not required.

- -seeds (Default: 127.0.0.1)
  A comma-delimited list of IP addresses 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.

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).
Related information: Initializing a single datacenter per workload type and Initializing multiple datacenters per workload type.

**enable_user_defined_functions**
Default: false. User defined functions (UDFs) present a security risk, since they are executed on the server side. UDFs are executed in a sandbox to contain the execution of malicious code. They are disabled by default.

**enable_scripted_user_defined_functions**
Default: false. Java UDFs are always enabled, if enable_user_defined_functions is true. Enable this option to use UDFs with language javascript or any custom JSR-223 provider. This option has no effect if enable_user_defined_functions is false.

**Common compaction settings**

**compaction_throughput_mb_per_sec**
Default: 16. Throttles compaction to the specified Mb/second across the instance. The faster the database inserts data, the faster the system must compact in order to keep the SSTable count down. The recommended value is 16 to 32 times the rate of write throughput (in Mb/second). Setting the value to 0 disables compaction throttling.

Related information: Configuring compaction

**compaction_large_partition_warning_threshold_mb**
Default: 100. The database logs a warning when compacting partitions larger than the set value.

**Common memtable settings**

**memtable_heap_space_in_mb**
Default: 1/4 of heap size. note (page 279) 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. For details, see memtable_cleanup_threshold (page 290).

Related information: Tuning the Java heap

**memtable_offheap_space_in_mb**
Default: 1/4 of heap size. note (page 279) Sets the total 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. For details, see memtable_cleanup_threshold (page 290).

Related information: Tuning the Java heap

**Common disk settings**

**concurrent_reads**
Workloads with more data than can fit in memory encounter a bottleneck in fetching data from disk during reads. Setting `concurrent_reads` to `(16 \times \text{number_of_drives})` allows operations to queue low enough in the stack so that the OS and drives can reorder them. The default setting applies to both logical volume managed (LVM) and RAID drives.

**concurrent_writes**

Default: 32.  
*note* *(page 279)* Writes in DataStax Enterprise are rarely I/O bound, so the ideal number of concurrent writes depends on the number of CPU cores on the node. The recommended value is $8 \times \text{number_of_cpu_cores}$.

**concurrent_counter_writes**

Default: 32.  
*note* *(page 279)* Counter writes read the current values before incrementing and writing them back. The recommended value is $(16 \times \text{number_of_drives})$.

**concurrent_materialized_view_writes**

Default: 32. Limit on the number of concurrent materialized view writes. Set this to the lesser of concurrent reads or concurrent writes, because there is a read involved in each materialized view write.

**Common automatic backup settings**

**incremental_backups**

Default: false. Backs up data updated since the last snapshot was taken. When enabled, the database creates a hard link to each SSTable flushed or streamed locally in a `backups` subdirectory of the keyspace data. Removing these links is the operator's responsibility.

Related information: [Enabling incremental backups](#)

**snapshot_before_compaction**

Default: false. Enables or disables taking a snapshot before each compaction. A snapshot is useful to back up data when there is a data format change. Be careful using this option: the database does not clean up older snapshots automatically.

Related information: [Configuring compaction](#)

**Common fault detection setting**

**phi_convict_threshold**

Default: 8.  
*note* *(page 279)* Adjusts the sensitivity of the failure detector on an exponential scale. Generally this setting does not need adjusting.

Related information: [About failure detection and recovery](#)

**Performance tuning properties**

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

**Commit log settings**

**commitlog_sync** *(periodic|batch)*
Configuration

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

commitlog_sync_period_in_ms
Default: 10000. Use with commitlog_sync: periodic. Controls how often the commit log is synchronized to disk. Periodic syncs are acknowledged immediately.

commitlog_sync_batch_window_in_ms
Default: disabled. Use with commitlog_sync:batch. Note (page 279) The maximum length of time that queries may be batched together.

commitlog_segment_size_in_mb
Default: 32. 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 for most commitlog archiving, but if you want a finer granularity, 8 or 16 MB is reasonable.

By default, the max_mutation_size_in_kb (page 288) is set to half of the commitlog_segment_size_in_kb.

Related information: Commit log archive configuration

max_mutation_size_in_kb
Default: ½ of commitlog_segment_size_in_mb (page 288).

If a mutation's size exceeds this value, 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.

Restriction: If you set max_mutation_size_in_kb explicitly, then you must set commitlog_segment_size_in_mb to at least twice the size of max_mutation_size_in_kb / 1024.

See commitlog_segment_size_in_mb (page 288) above.

commitlog_compression
Default: disabled. The compressor to use if commit log is compressed. Valid values: LZ4, Snappy or Deflate. If no value is set for this property, the commit log is written uncompressed.

cdc_total_space_in_mb
Default: 4096 and 1/8th of the total space of the drive where the cdc_raw_directory resides.) Note (page 279) If space gets above this value, the database throws WriteTimeoutException on Mutations including tables with CDC enabled. A CDCCompactor (a consumer) is responsible for parsing the raw CDC logs and deleting them when parsing is completed.

cdc_free_space_check_interval_ms
Default: 250. Note When the cdc_raw limit is hit and the CDCCompactor is either running behind or experiencing backpressure, this interval is checked to see if any new space for cdc-tracked tables has been made available.

commitlog_total_space_in_mb
Configuration

Default: 32 for 32-bit JVMs, 8192 for 64-bit JVMs. *note (page 279) Total space used for commit logs. If the total space used by all commit logs goes above this value, 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 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.

Related information: Configuring memtable thresholds

---

**gc_log_threshold_in_ms**

Default: 200. The threshold for log messages at the INFO level. Adjust to minimize logging.

*Compaction settings*

Related information: Configuring compaction

**concurrent_compactors**

Defaults to the smaller of number of disks or number of cores, with a minimum of 2 and a maximum of 8 per CPU core. *note (page 279) 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 286)` first.

*Note: 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.*

**sstable_preemptive_open_interval_in_mb**

Default: 50. 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 setting helps to smoothly transfer reads between the SSTables by reducing page cache churn and keeps hot rows hot.

*Memtable settings*

**memtable_allocation_type**

Default: heap_buffers. The method the database uses to allocate and manage memtable memory.

- **heap_buffers**
  
  On heap NIO (non-blocking I/O) buffers.

- **offheap_buffers**
Configuration

Off heap (direct) NIO buffers.

- **offheap_objects**
  
  Native memory, eliminating NIO buffer heap overhead.

**memtable_cleanup_threshold (deprecated)**

Default: \(1/(\text{memtable\_flush\_writers (page 290)} + 1)\).\(^{\text{note (page 279)}}\) Ratio used for automatic memtable flush. The database adds \(\text{memtable\_heap\_space\_in\_mb (page 286)}\) to \(\text{memtable\_offheap\_space\_in\_mb (page 286)}\) and multiplies the total by \(\text{memtable\_cleanup\_threshold}\) to get a space amount in MB. When the total amount of memory used by all non-flushing memtables exceeds this amount, the database flushes the largest memtable to disk.

For example, consider a node where the total of \(\text{memtable\_heap\_space\_in\_mb}\) and \(\text{memtable\_offheap\_space\_in\_mb}\) is 1000, and \(\text{memtable\_cleanup\_threshold}\) is 0.50. The \(\text{memtable\_cleanup}\) amount is 500MB. This node has two memtables: Memtable A (150MB) and Memtable B (350MB). When either memtable increases, the total space they use exceeds 500MB and the database flushes the Memtable B to disk.

A larger value for \(\text{memtable\_cleanup\_threshold}\) means larger flushes, less frequent flushes and potentially less compaction activity, but also less concurrent flush activity, which can make it difficult to keep your disks saturated under heavy write load.

This section documents the formula used to calculate the ratio based on the number of \(\text{memtable\_flush\_writers (page 290)}\). The default value in \texttt{cassandra.yaml} is 0.11, which works if the node has many disks or if you set the node’s \(\text{memtable\_flush\_writers}\) to 8. As another example, if the node uses a single SSD, the value for \(\text{memtable\_cleanup\_threshold}\) computes to 0.33, based on the minimum \(\text{memtable\_flush\_writers}\) value of 2.

**file\_cache\_size\_in\_mb**

Default: Smaller of 1/4 heap or 512. Total memory to use for SSTable-reading buffers.

**buffer\_pool\_use\_heap\_if\_exhausted**

Default: disabled.\(^{\text{note (page 279)}}\) Indicates whether the database allocates on-heap or off-heap memory when the SSTable buffer pool is exhausted (when the buffer pool has exceeded the maximum memory \(\text{file\_cache\_size\_in\_mb (page 290)}\)), beyond this amount, the database stops caching buffers, but allocates on request.

**memtable\_flush\_writers**

Default: Smaller of number of disks or number of cores with a minimum of 2 and a maximum of 8.\(^{\text{note (page 279)}}\) The number of memtable flush writer threads. These threads are blocked by disk I/O, and each one holds a memtable in memory.
while blocked. If your data directories are backed by SSDs, increase this setting to the number of cores.

**Cache and index settings**

**column_index_size_in_kb**
- Default: 64. Granularity of the index of rows within a partition. For huge rows, decrease this setting to improve seek time. If you use key cache, be careful not to make this setting too large because key cache will be overwhelmed. If you're unsure of the size of the rows, it's best to use the default setting.

**index_summary_capacity_in_mb**
- Default: 5% of the heap size [empty]. \(^\text{note} (\text{page 279})\) Fixed memory pool size in MB for SSTable index summaries. If the memory usage of all index summaries exceeds this limit, any SSTables with low read rates shrink their index summaries to meet this limit. This is a best-effort process. In extreme conditions, the database may use more than this amount of memory.

**index_summary_resize_interval_in_minutes**
- Default: 60. How frequently index summaries should be re-sampled. Re-sampling is done periodically to redistribute memory from the fixed-size pool to SSTables proportional their recent read rates. To disable, set to -1. This setting leaves existing index summaries at their current sampling level.

**Disks settings**

**stream_throughput_outbound_megabits_per_sec**
- Default: 200. \(^\text{note} (\text{page 279})\) 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. This can saturate the network connection and degrade client (RPC) performance.

**inter_dc_stream_throughput_outbound_megabits_per_sec**
- Default: 200. \(^\text{note} (\text{page 279})\) Throttle for all streaming file transfers between datacenters, and for network stream traffic as configured with stream_throughput_outbound_megabits_per_sec (\text{page 291}).

**streaming_keep_alive_period_in_secs**
- Default: disabled. \(^\text{note} (\text{page 279})\) Specifies for node to send keep-alive message at this interval. The stream session fails when a keep-alive message is not received for 2 keep-alive cycles.

**trickle_fsync**
- Default: false. 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.

**trickle_fsync_interval_in_kb**
- Default: 10240. The size of the fsync in kilobytes.

**Advanced properties**

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

**auto_bootstrap**
Default: true. This setting has been removed from default configuration. It causes new (non-seed) nodes migrate the right data to themselves automatically. When initializing a fresh cluster without data, add **auto_bootstrap**: false.

Related information: Initializing a single datacenter per workload type and Initializing multiple datacenters per workload type.

**batch_size_warn_threshold_in_kb**
Default: 64. Log a warning message when any multiple-partition batch size exceeds this value.

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

**batch_size_fail_threshold_in_kb**
Default: 640. Fails any batch whose size exceeds this setting. The default value is 10X the value of **batch_size_warn_threshold_in_kb**.

**unlogged_batch_across_partitions_warn_threshold**
Default: 10. Causes the database to log a WARN message on any batches not of type LOGGED that span across more partitions than this limit. The default value is 10 partitions.

**cdc_enabled**
Default: false. Enable or disable change data capture (CDC) functionality on a per-node basis. This modifies the logic used for write path allocation rejection (standard: never reject. cdc: reject Mutation containing a CDC-enabled table if at space limit in cdc_raw_directory).

**Important:** Do not enable CDC on a mixed-version cluster. Upgrade all nodes to DataStax Enterprise 5.1 before enabling and restarting the cluster.

**broadcast_address**
Default: listen_address. **Note** (page 279) 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 listen_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 280).

**listen_on_broadcast_address**
Default: false. If this node uses multiple physical network interfaces, set a unique IP address for broadcast_address (page 292) and set listen_on_broadcast_address to true. This enables the node to communicate on both interfaces.
Set this property to false if the node is on a network that automatically routes between public and private networks, as Amazon EC2 does.

For configuration details, see the instructions for `listen_address` (page 280).

**initial_token**
Default: 1 (disabled). 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 293).
If your installation is not using `vnodes` or this node’s `num_tokens` (page 293) is set it 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` (vnodes) in special cases such as Restoring from a snapshot.

**num_tokens**
(Default: 1 disabled) Set this property for virtual node token architecture. Determines the number of token ranges to assign to this virtual node (page 367) (vnode). Use a number between 1 and 128, where 1 disables vnodes. 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.

- **Random selection algorithm:**
  
  **Note:** Over time loads in a datacenter using this algorithm become uneven distributed. The random selection algorithm method is not recommended by DataStax. Instead, use the allocation algorithm.

  Assign token ranges randomly. A higher num_token value increases the probability that the data and workload are evenly distributed. Use 8 when randomly assigning token ranges.

- **Allocation algorithm:** Assign token ranges using the allocation algorithm which optimizes the workload balance using the target keyspace replication factor. Enabled when the allocate_tokens_for_local_replication_factor (page 293) is set. DataStax recommends setting the number of tokens to 8 to distribute the workload with ~10% variance between nodes.

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**
When adding a vnode to an existing cluster or setting up nodes in a new datacenter, set to the target replication factor (RF) of keyspaces in the datacenter. Triggers the recommended algorithmic allocation for the RF and num_tokens for this node. 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.

See Virtual node (vnode) configuration (page 367), and for set up instructions see Adding vnodes to an existing cluster or Adding a datacenter to a cluster.

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

**partitioner**
Default: org.apache.cassandra.dht.Murmur3Partitioner. Sets 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 backwards compatibility:
- RandomPartitioner
- ByteOrderedPartitioner (deprecated)
- OrderPreservingPartitioner (deprecated)

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

**Related information:** Partitioners

**storage_port**
Default: 7000. The port for inter-node communication.

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

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

**Advanced automatic backup setting**

**auto_snapshot**
Default: true. Enables or disables whether the database takes a snapshot 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.

**Key caches and global row properties**

When creating or modifying tables, you can enable or disable the key cache (partition key cache) or row cache for that table by setting the caching parameter. Other row and key 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.

Related information: Configuring caches

**key_cache_keys_to_save**

Default: disabled. All keys are saved. Note (page 279) Number of keys from the key cache to save.

**key_cache_save_period**

Default: 14400. (4 hours) Duration in seconds that keys are kept in cache. Caches are saved to saved_caches_directory (page 281). Saved caches greatly improve cold-start speeds and have relatively little effect on I/O.

**key_cache_size_in_mb**

Default: empty. A global cache setting for the maximum size of the key cache in memory (for all tables). If no value is set, the cache is set to the smaller of 5% of the available heap, or 100MB. To disable set to 0.

Related information: nodetool setcachecapacity, Enabling and configuring caching.

**column_index_cache_size_in_kb**

Default: 2. A 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. This limitation prevents index entries from large partitions from taking up all the space in the partition key cache (which is controlled by key_cache_size_in_mb).

**row_cache_class_name**

Default: disabled. Note (page 279) The classname of the row cache provider to use. Valid values: OHCProvider (fully off-heap) or SerializingCacheProvider (partially off-heap). Important: Use only row cache provider implementations bundled with DSE.

**row_cache_keys_to_save**

Default: disabled. All keys are saved. Note (page 279) Number of keys from the row cache to save.

**row_cache_size_in_mb**

Default: 0. To disable, set to 0. Maximum size of the row cache in memory. The row cache can save more time than key_cache_size_in_mb (page 295), 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.

**row_cache_save_period**

Default: 0. To disable, set to 0. The number of seconds that rows are kept in cache. Caches are saved to saved_caches_directory (page 281). This setting has limited use as described in row_cache_size_in_mb.

Counter caches properties
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**

Default value: empty. *note (page 279)* 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.

**counter_cache_save_period**

Default: 7200. (2 hours) the amount of time after which the database saves the counter cache (keys only). The database saves caches to *saved_caches_directory* (page 281).

**counter_cache_keys_to_save**

Default value: disabled. *note (page 279)* Number of keys from the counter cache to save. When this property is disabled, the database saves all keys.

**Tombstone settings**

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. See *Cassandra anti-patterns: Queues and queue-like datasets*. 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.

**tombstone_warn_threshold**

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

**tombstone_failure_threshold**

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

**Network timeout settings**

**aggregated_request_timeout_in_ms**

Number of milliseconds that the coordinator waits for aggregated read operations to complete. For example, *SELECT COUNT(*)*, *MIN(x)*. Default: 120000.

**cas_contention_timeout_in_ms**
Default: 1000. The number of milliseconds during which 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.

count_write_request_timeout_in_ms
Default: 5000. The number of milliseconds that the coordinator waits for counter writes to complete before timing it out.

range_request_timeout_in_ms
Default: 10000. The number of milliseconds that the coordinator waits for sequential or index scans to complete before timing it out.

read_request_timeout_in_ms
Default: 5000. The number of milliseconds that the coordinator waits for read operations to complete before timing it out.

request_timeout_in_ms
Default: 10000. The default timeout value for other miscellaneous operations.

Related information: Hinted handoff: repair during write path.

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.

truncate_request_timeout_in_ms
Default: 60000. The number of milliseconds that 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 294) is disabled (not recommended), you can reduce this time.

write_request_timeout_in_ms
Default: 2000. The number of milliseconds that the coordinator waits for a write operations to complete before timing it out for requests with at least one node in the local datacenter.

Related information: Hinted handoff: repair during write path.

cross_dc_rtt_in_ms
Default: 0. Increases the cross-datacenter timeout \((\text{write\_request\_timeout\_in\_ms} + \text{cross\_dc\_rtt\_in\_ms})\) for requests that only involve nodes in a remote datacenter. This setting is intended to reduce hint pressure.

Tip: DataStax recommends using \text{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 \text{QUORUM}.

Inter-node settings

cross_node_timeout
Default: false. Enables or disables 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.

**internode_send_buff_size_in_bytes**
Default: empty. note (page 279) The sending socket buffer size in bytes for internode calls.

The buffer size set by this parameter and **internode_recv_buff_size_in_bytes** (page 298) 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`

Related information: TCP settings (page 273)

**internode_recv_buff_size_in_bytes**
Default: empty. note (page 279) The receiving socket buffer size in bytes for internode calls.

**internode_compression**
Default: dc. Controls whether traffic between nodes is compressed. Valid values:

- **all**
  Compresses all traffic.

- **dc**
  Compresses traffic between datacenters only.

- **none**
  No compression.

**inter_dc_tcp_nodelay**
Default: false. Enable this property or disable tcp_nodelay for inter-datacenter communication. If this property is 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.

Native transport (CQL Binary Protocol)
**start_native_transport**
Default: true. Enables or disables the native transport server. This server uses the same address as the `rpc_address` (page 284), but the port it uses is different from `rpc_port` (page 299). See `native_transport_port` (page 299).

**native_transport_port**
Default: 9042. The port where the CQL native transport listens for clients.

**native_transport_max_threads**
Default: 128. **note** (page 279) The maximum number of thread handling requests. Similar to `rpc_max_threads` (page 299), but this property differs as follows:

- The default for `native_transport_max_threads` is 128; the default for `rpc_max_threads` is unlimited.
- There is no corresponding `native_transport_min_threads`.
- The database stops idle native transport threads after 30 seconds.

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

**native_transport_max_concurrent_connections**
Default: -1. The maximum number of concurrent client connections. The default value of -1 means unlimited.

**native_transport_max_concurrent_connections_per_ip**
Default: -1. The maximum number of concurrent client connections per source IP address. The default value of -1 means unlimited.

**RPC (remote procedure call) settings**
Settings for configuring and tuning client connections.

**broadcast_rpc_address**
Default: empty. **note** (page 279) The RPC address for broadcast to drivers and other nodes. This cannot be set to 0.0.0.0. If left blank, the database uses the `rpc_address` (page 284) or `rpc_interface` (page 299). If `rpc_address` or `rpc_interface` is set to 0.0.0.0, this property must be set.

**rpc_port**
Default: 9160. Thrift port for client connections.

**start_rpc**
Default: true. Enables or disables the Thrift RPC server.

**rpc_keepalive**
Default: true. Enables or disables keepalive on client connections (RPC or native).

**rpc_max_threads**
Default: unlimited. **note** (page 279) Regardless of your choice of RPC server (`rpc_server_type` (page 300)), `rpc_max_threads` dictates the maximum number of concurrent requests in the RPC thread pool. If you are using the parameter `sync` (see `rpc_server_type` (page 300)) it also dictates the number of clients that can be connected. A high number of client connections could cause excessive memory usage for the thread stack. Connection pooling on the client side is highly recommended. Setting a `rpc_max_threads` acts as a safeguard against
misbehaving clients. If the number of threads reaches the maximum, the database blocks additional connections until a client disconnects.

**rpc_min_threads**
Default: unlimited. The minimum thread pool size for remote procedure calls.

**rpc_recv_buff_size_in_bytes**
Default: empty. The receiving socket buffer size for remote procedure calls.

**rpc_send_buff_size_in_bytes**
Default: empty. The sending socket buffer size in bytes for remote procedure calls.

**rpc_server_type**
Default: sync. The database provides three options for the RPC server. sync and hsha performance is about the same, but hsha uses less memory.

- **sync**: (Default: one thread per Thrift connection.)

  For a very large number of clients, memory is the limiting factor. On a 64-bit JVM, 180 KB is the minimum stack size per thread and corresponds to your use of virtual memory. Physical memory may be limited depending on use of stack space.

- **hsha**:

  Half synchronous, half asynchronous. All Thrift clients are handled asynchronously using a small number of threads that does not vary with the number of clients. This mechanism scales well to many clients. The RPC requests are synchronous (one thread per active request).

  **Note:** If you select this option, you must change the default value (unlimited) of rpc_max_threads (page 299).

- Your own RPC server

  You must provide a fully-qualified class name of an o.a.c.t.TServerFactory that can create a server instance.

---

**Advanced fault detection settings**

Settings to handle poorly performing or failing components.

**gc_warn_threshold_in_ms**
Default: 1000. 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.)

Additional information: Configuring logging.

**otc_coalescing_strategy**
Default: DISABLED. Supported strategies are: FIXED, MOVINGAVERAGE, TIMEHORIZON, and DISABLED. Suitable for VMs, but not noticeably performant in other environments. The OutboundTcpConnection (otc) strategy to:

- Increase message throughput (doubling or more).
- Process multiple messages with one trip to read from a socket.
- Perform all the task submission work at the same time.
- Reduce context switching.
- Increase cache friendliness of network message processing.

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

**otc_coalescing_window_us**

Default: disabled. *(page 279)* How many microseconds to wait for coalescing. For fixed strategy, the amount of time after the first message is received before it is sent with any accompanying messages. For moving average, this is the maximum wait time and the interval that messages must arrive on average to enable coalescing.

**otc_coalescing_enough_coalesced_messages**

Default: disabled. *(page 279)* The threshold for the number of messages. Do not coalesce messages when this value is exceeded. Should be more than 2 and less than 128.

**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%)

**back_pressure_enabled**

Default: false. Enable for the coordinator to apply the specified back pressure strategy to each mutation that is sent to replicas.

**back_pressure_strategy**

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

Default: RateBasedBackPressure. 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.

- **RateBasedBackPressure**
  
  Ratio between incoming mutation responses and outgoing mutation requests.

- **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.

- **factor**
  A number between 1 and 10. Increases or decreases rate limiting.

- **flow**
  Default: FAST. 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.

**max_value_size_in_mb**
Default: 256. The maximum size of any value in SSTables. It detects SSTable corruption and marks the SSTables as corrupted when the threshold is exceeded.

**dynamic_snitch_badness_threshold**
Default: 0.1. 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. A value of zero to 1.0 for the read_repair_chance table property maximizes cache capacity across the nodes.

**dynamic_snitch_reset_interval_in_ms**
Default: 600000. Time interval after which the database resets all node scores. This allows a bad node to recover.

**dynamic_snitch_update_interval_in_ms**
Default: 100. The number of milliseconds between the database’s calculation of node scores. Because score calculation is CPU intensive, be careful when reducing this interval.

**hints_flush_period_in_ms**
Default: 10000. The number of milliseconds the database waits before flushing hints from internal buffers to disk.

**hints_directory**
Default: $CASSANDRA_HOME/data/hints. The directory in which hints are stored.

**hinted_handoff_enabled**
Default: true. Enables or disables hinted handoff. To enable per datacenter, add a list of datacenters. For example: hinted_handoff_enabled: DC1, DC2. 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.

Related information: Hinted handoff: repair during write path

**hinted_handoff_disabled_datacenters**
Default: empty. A blacklist of datacenters that will not perform hinted handoffs. To disable hinted handoff on a certain datacenter, add its name to this list. For example:

```
 hinted_handoff_disabled_datacenters: - DC1 - DC2
```

Related information: Hinted handoff: repair during write path

**hinted_handoff_throttle_in_kb**

Default: 1024. 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 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 database computes the hint transmission rate based on the uncompressed hint size, even if internode_compression (page 298) or hints_compression (page 303) is enabled.

**max_hint_window_in_ms**

Default: 10800000. (3 hours) 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.

Related information: About failure detection and recovery

**maxhints_delivery_threads**

Default: 2. Number of threads the database uses to deliver hints. In multiple datacenter deployments, consider increasing this number because cross datacenter handoff is generally slower.

**maxhints_file_size_in_mb**

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

**hints_compression**

Default: LZ4Compressor. The compressor for hint files. Supported compressors: LZ, Snappy, and Deflate. If you do not specify a compressor, the database does not compress hints files.

**batchlog_replay_throttle_in_kb**

(Definition: 1024 kilobytes per second) Total maximum throttle for replaying failed logged batches. Throttling is reduced proportionally to the number of nodes in the cluster.

**Request scheduler properties**

Settings to handle incoming client requests according to a defined policy. If your nodes are overloaded and dropping requests, DataStax recommends that you add more nodes rather than use these properties to prioritize requests.

**Note:** The properties in this section apply only to the Thrift transport. They have no effect on the use of CQL over the native protocol.
**request_scheduler**

Default: org.apache.cassandra.scheduler.NoScheduler. The scheduler to handle incoming client requests according to a defined policy. This scheduler is useful for throttling client requests in single clusters containing multiple keyspaces. This parameter is specifically for requests from the client and does not affect inter-node communication. Valid values:

- org.apache.cassandra.scheduler.NoScheduler
  The database does no scheduling.

- org.apache.cassandra.scheduler.RoundRobinScheduler
  The database uses a round robin of client requests to a node with a separate queue for each request_scheduler_id (page 304) property.

- The database uses a Java class that implements the RequestScheduler interface.

**request_scheduler_id**

Default: keyspace. Note (page 279) The scope of the scheduler’s activity. Currently the only valid value is keyspace.

**request_scheduler_options**

Default: NoScheduler. A list of properties that define configuration options for request_scheduler (page 304).

RoundRobin:

A round robin of client requests to a node with a separate queue for each request_scheduler_id (page 304).

- throttle_limit - The number of in-flight requests per client. Requests that exceed this limit are queued up until running requests complete. Recommended value is \((\text{concurrent}\_\text{reads} + \text{concurrent}\_\text{writes}) \times 2\).

- default_weight - Default: 1. Note (page 279) How many requests the scheduler handles during each turn of the round robin.

- weights - A list of keyspaces with assigned weights.

**Thrift interface properties**

Legacy API for older clients. CQL is a simpler and better API for the database.

**thrift_framed_transport_size_in_mb**

Default: 15. Frame size (maximum field length) for Thrift. The frame is the row or part of the row that the application is inserting.

**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
Configuration

Enabling DSE Unified Authentication requires additional configuration in dse.yaml, see Configuring DSE Unified Authentication.

**authenticator**

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

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

**internode_authenticator**

Default: enabled. {note (page 279) Internode authentication backend. It implements org.apache.cassandra.auth.AllowAllInternodeAuthenticator to allows or disallow connections from peer nodes.

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

**authorizer**

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

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

**role_manager**

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

**roles_validity_in_ms**

Default: 2000. Validity period for roles cache; set to 0 to disable. 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.

**roles_update_interval_in_ms**
**Configuration**

Default: 2000. Enable to refresh interval for roles cache. Defaults to the same value as `roles_validity_in_ms`. 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 must be also.

**credentials_validity_in_ms**  
Default: 2000. How many milliseconds credentials in the cache remain valid. This cache is tightly coupled to the provided PasswordAuthenticator implementation of `IAuthenticator (page 305)`. If another `IAuthenticator` implementation is configured, the database does not use this cache, and these settings have no effect.

*Note:* Credentials are cached in encrypted form. This may cause a performance penalty that offsets the reduction in latency gained by caching.

*Caution:* Cache credentials and permissions are not automatically invalidated after issuing a `REVOKE` statement.

This setting is disabled when set to 0.

**credentials_update_interval_in_ms**  
Default: same value as `credentials_validity_in_ms`. After this interval, cache entries become eligible for refresh. The next time the cache is accessed, the system schedules an asynchronous reload of the cache. Until this cache reload is complete, the cache returns the old values.

If `credentials_validity_in_ms` is nonzero, this property must also be nonzero.

**permissions_validity_in_ms**  
Default: 2000. Fetching permissions can be resource intensive. Define how many milliseconds permissions in cache remain valid to manage performance impact of permissions queries. 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.

*Caution:* Cache credentials and permissions are not automatically invalidated after issuing a `REVOKE` statement.

This setting is disabled when set to 0.

**permissions_update_interval_in_ms**  
Default: same value as `permissions_validity_in_ms` (*page 306*). 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 nonzero, `roles_update_interval_in_ms` must also be non-zero.

**permissions_cache_max_entries**  
Default: 1000. 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
Configuration

applies to both caches. To size the permissions cache for use with Setting row-level permissions, use this formula:

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

If this option is not present in cassandra.yaml, manually enter it. See Enabling DSE Unified Authentication.

server_encryption_options

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

- **internode_encryption**: Default: none. Enables or disables encryption 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. Available inter-node options:

  # 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).

- **keystore**: Default: conf/.keystore.

  The location of a 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.

- **keystore_password**: Default: cassandra.

  Password for the keystore.

- **truststore**: Default: conf/.truststore.

  Location of the truststore containing the trusted certificate for authenticating remote servers.

- **truststore_password**: Default: cassandra.
Password for the truststore.

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

Advanced settings:

- **protocol**: Default: TLS.
- **algorithm**: Default: SunX509.
- **store_type**: Default: 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

- **require_client_auth**: Default: false.
  Enables or disables certificate authentication.

- **require_endpoint_verification**: Default: false.
  Enables or disables host name verification.

Related information: Securing internal transactional node connections

**client_encryption_options**

Enables or disables 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 currently enabled for DataStax Enterprise. Available options:

- **enabled**: Default: false.
  To enable client encryption, set to true.

- **optional**: Default: false.
  Allow unsecured connections when client encryption is enabled.

- **keystore**: Default: conf/.keystore.
  The location of a 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.

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

- **require_client_auth**: Default: false.
  Enables or disables certificate authentication.

- **truststore**: Default: conf/.truststore.
  Set this property if require_client_auth is true.

- **truststore_password**: Default: cassandra
  Set if require_client_auth is true.

Advanced settings:

- **protocol**: Default: TLS.
- **algorithm**: Default: SunX509.
- **store_type**: Default: 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
  ```

Related information: [Securing client to cluster connections](#)

**transparent_data_encryption_options**
DataStax Enterprise only supports this option for backwards compatibility. When using DSE, configure data encryption options (page 321) in the dse.yaml; see Transparent data encryption.

TDE properties:

cassandra.yaml
The location of the cassandra.yaml file depends on the type of installation:

| Package installations | /etc/dse/cassandra/
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>cassandra.yaml</td>
</tr>
</tbody>
</table>
Configuration

<table>
<thead>
<tr>
<th>Tarball installations</th>
<th>installation_location/</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-No Services installations</td>
<td>resources/cassandra/conf/</td>
</tr>
<tr>
<td></td>
<td>cassandra.yaml</td>
</tr>
</tbody>
</table>

- **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_storage_port**
  - Default: 7001. The SSL port for encrypted communication. Unused unless enabled in encryption_options.

- **native_transport_port_ssl**
  - Default: 9142. If client encryption is enabled and native_transport_port_ssl is disabled, the native_transport_port (default: 9042) will encrypt all traffic. To use both unencrypted and encrypted traffic, enable native_transport_port_ssl.

Continuous paging

- **continuous_paging**
  - Pushes pages continuously to the client when requested by the client, parameters control:
    - Maximum memory used. Default: 60 # 4 # 8 = 1920 MB
      (max_concurrent_sessions # max_session_pages # max_page_size_mb).
    - Maximum number of threads.
    - Maximum duration for local queries.

  **Guidance**:  
  - If the client is not reading from the socket, the producer thread is blocked after it has prepared max_session_pages, up to max_client_wait_time_ms.
• 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.

Maximum period of time = max_client_wait_time_ms + max_local_query_time_ms.

• Consider adjusting max_local_query_time_ms and max_client_wait_time_ms when high write workloads exist on tables that have continuous paging requests.

• If fewer threads exist than sessions (max_threads < max_concurrent_sessions), a session cannot execute until another one is swapped out.

• Distributed queries (CL > ONE or non-local data) are swapped out after every page, while local queries at CL = ONE are swapped out after max_local_query_time_ms.

• If the client is slow in reading pages, try increasing the delay by adjusting max_client_wait_time_ms.

Parameters:

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

• max_session_pages
  Default: 4. The maximum number of pages that can be buffered for each session.

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

• max_client_wait_time_ms
  Default: 20000. The maximum time for the server to wait for the client to read from the socket. If exceeded, the session is aborted and the client receives an error. Setting max_client_wait_time_ms to a value too low may result in client side errors.

• max_local_query_time_ms
  Default: 5000. The maximum time for a local continuous query to run. When 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 max_threads < max_concurrent_sessions.
Configuration

- **max_threads**

  Default: 24. The number of threads dedicated to continuous paging sessions.

**cassandra-env.sh**

The location of the `cassandra-env.sh` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td><code>/etc/dse/cassandra/cassandra-env.sh</code></td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td><code>installation_location/resources/cassandra/conf/cassandra-env.sh</code></td>
</tr>
<tr>
<td>Tarball installations</td>
<td></td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td><code>installation_location/resources/dse/conf/cassandra-env.sh</code></td>
</tr>
</tbody>
</table>

**dse.yaml**

The location of the `dse.yaml` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td><code>/etc/dse/dse.yaml</code></td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td><code>installation_location/resources/dse/conf/dse.yaml</code></td>
</tr>
<tr>
<td>Tarball installations</td>
<td></td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td><code>installation_location/resources/dse/conf/dse.yaml</code></td>
</tr>
</tbody>
</table>

**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, restart the node for the changes to take effect.

**cassandra.yaml** (page 278)

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

**Syntax**

For the options in each section, the main setting has zero spaces, and at least two spaces are required before each entry in that section. For example, in the `node_health_options` section, at least two spaces are required before `refresh_rate_ms`, `uptime_ramp_up_period_seconds`, and `dropped_mutation_window_minutes`:

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

Adhere to the YAML syntax. The default values are shown for each section.
Organization

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

- Security and authentication options (page 313)
- DSE In-Memory (page 324)
- Node health (page 324)
- Health-based routing (page 324)
- Lease metrics (page 325)
- DSE Search options (page 325)
- DSE Analytics options (page 336)
- Performance Service options (page 329)
- Audit logging (page 342)
- DSE Tiered Storage (page 344)
- DSE Advanced Replication (page 345)
- Inter-node messaging (page 346)
- DSE Multi-Instance (page 347)
- DSE Graph options (page 347)

Security and authentication options

- Authentication options (page 313)
- Role management options (page 315)
- Authorization options (page 315)
- Kerberos options (page 316)
- LDAP options (page 317)
- Encrypt sensitive system resources (page 320)
- Encrypted configuration properties settings (page 321)
- KMIP encryption options (page 322)
- DSE Search index encryption settings (page 323)

Authentication options

Authentication options for the DSE Authenticator, which allows you to use multiple schemes for authentication in a DSE cluster. Additional configuration is required in the cassandra.yaml configuration file (page 278) file.

Note: Internal and LDAP schemes can also be used for role management, see role_management_options (page 315).

Default values:

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

**warn**

### transitional_mode

**disabled**

### configuration_options

Options for the DSE Authenticator to authenticate connections. Authenticators other than DSE Authenticator are not supported.

#### enabled

**Default:** *false*. Enables user authentication. When false, the DSE Authenticator allows all connections.

#### 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.

#### 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

Only enable (true) when using multiple schemes for authentication. Prevents unintentional role assignment that might occur if user or group names overlap in the authentication service. When true every role requires permissions to a scheme in order to be assigned, see Binding a role to an authentication scheme.

#### 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. In analytics clusters, set to true when using with Spark jobs.

#### 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. Default.
- **allow** - Allow the request without any warning.

#### transitional_mode

For temporary use during authentication setup in an already established environment. Allows access to the database using the anonymous role, which has all permissions except AUTHORIZE.

To enable, use one of the following options:

- **permissive** - Allow all connections that provide credentials. Maps authenticated superusers to their role AND maps all other users to anonymous.
- **normal** - Allow all connections that provide credentials. Maps all authenticated users to their role AND maps all other connections to anonymous.
Configuration

- **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.

  When set to **disabled**, all connections must provide valid credentials and map to a login enabled role.

Role management options
Default values:

```yaml
role_management_options:
  mode: internal
```

**role_management_options**
Options for the DSE Role Manager. To enable role manager, set `authorization_options` (page 313) enabled to true and `role_manager` (page 305) in `cassandra.yaml` to `com.datastax.bdp.cassandra.auth.DseRoleManager`, see Managing roles. When `scheme_permissions` (page 314) 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. Default.
- **ldap** - Scheme that assigns roles by looking up the user name in LDAP and mapping the group attribute (`ldap_options` (page 317)) to an internal role name. To configure an LDAP scheme, complete the steps in Defining an LDAP scheme.

  **Note:** Nested roles are not supported for LDAP.

Authorization options
Default values:

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

**authorization_options**
Options for the DSE Authorizer.

**enabled**
Enables the use of DSE Authorizer for role-based access control (RBAC).

**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:
Configuration

- **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.

**allow_row_level_security**
 Default: false. True enables row-level access control (RLAC) permissions; use the same setting on all nodes.

Kerberos options
Default values:

```yaml
kerberos_options:
  keytab: path_to_keytab/dse.keytab
  service_principal: dse_user/_HOST@REALM
  http_principal: HTTP/_HOST@REALM
  qop: auth
```

**kerberos_options**
Configure security for a DataStax Enterprise cluster using Kerberos. See Kerberos guidelines.

**keytab**
The keytab file must contain the credentials for both of the fully resolved principal names, which replace _HOST with the Fully Qualified Domain Name (FQDN) of the host in the service_principal and http_principal settings. The UNIX user running DSE must also have read permissions on the 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:
- Package and Installer-Services installations: cassandra
- Package installations: the name of the UNIX user that starts the service

where:
- _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.

The service_principal must be consistent everywhere: in the dse.yaml file, present in the keytab, and in the cqlshrc file (where service_principal is separated into service/hostname).

**http_principal**
The http_principal is used by the Tomcat application container to run DSE Search. The Tomcat web server uses GSS-API 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. Default.
- **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. See [Enabling DSE Unified Authentication](#).

**Default values:**

```yaml
ldap_options:
  server_host: localhost ## Appropriate only for development and testing on a single node.
  server_port: 389
  search_dn: uid=Admin
  search_password: secret
  use_ssl: false
  use_tls: false
  truststore_path: path/to/truststore
  truststore_password: passwordToTruststore
  truststore_type: jks
  user_search_base: ou=users,dc=example,dc=com
  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:**

```yaml
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
```
```yaml
use_ssl: false
use_tls: false
truststore_path: path/to/truststore
truststore_password: passwordToTruststore
truststore_type: jks
user_search_base: cn=users,dc=win2012domain,dc=mycompany,dc=lan
user_search_filter: (sAMAccountName={0})
user_memberof_attribute: memberOf
group_search_type: directory_search
group_search_type: memberof_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
```

**ldap_options**

Options to configure LDAP security. See Defining an LDAP scheme.

**server_host**

The host name of the LDAP server.

*Note:* Only install LDAP on the same host (localhost) in single node test or development environments.

**server_port**

The port on which the LDAP server listens. Default: 389

**search_dn**

Distinguished name (DN) of an account with read access to the user_search_base and group_search_base. Comment out to use an anonymous bind. 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, which has access to all database objects using the consistency level QUOROM.

**search_password**

The password of the search_dn account.

**use_ssl**

Set to true to enable SSL connections to the LDAP server. If set to true, change server_port to the SSL port of the LDAP server. Default: false

**use_tls**

Set to true to enable TLS connections to the LDAP server. If set to true, change the server_port to the TLS port of the LDAP server. Default: false

**truststore_path**

The path to the truststore for SSL certificates.

**truststore_password**
The password to access the trust store.

**truststore_type**
The type of truststore. Default: jks

**user_search_base**
The search base for your domain, used to look up users. Set the ou and dc elements for your LDAP domain. Typically this is set to `ou=users,dc=domain,dc=top_level_domain`. For example, `ou=users,dc=example,dc=com`.

Active Directory uses a different search base, typically `CN=search,CN=Users,DC=ActDir_domname,DC=internal`. For example, `CN=search,CN=Users,DC=example-sales,DC=internal`.

**user_search_filter**
The search filter for looking up user names. Set the LDAP attribute name of the user identifier equal to `{0}`. For example AD (Microsoft Active Directory), is typically `samAccountName={0}`. Default: `uid={0}`

**user_memberof_attribute**
The attribute on the user entry that contains group membership information. Required when managing roles using `group_search_type: memberof_search` with LDAP (`role_manager.mode:ldap`).

**group_search_type**
Required when managing roles with LDAP (`role_manager.mode: ldap`). Defines how group membership is looked up for a user. Choose from one of the following values:

- **directory_search** - Filters the results by doing a subtree search of `group_search_base` to find groups that contain the user name in the attribute defined in the `group_search_filter`. (Default)

- **memberof_search** - Get groups from the user attribute defined in `user_memberof_attribute`. The directory server must have `memberof` support, which is a default user attribute in Microsoft Active Directory (AD).

**group_search_base**
The unique distinguished name (DN) of the group record from which to start the group membership search on.

**group_search_filter**
Set to any valid LDAP filter.

**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. Default: `cn`

**credentials_validity_in_ms**
The duration period in milliseconds for the credential cache. Default: 0

**search_validity_in_seconds**
The duration period in seconds for the search cache. Default: 0

**connection_pool**
The configuration settings for the connection pool for making LDAP requests.
Configuration

- **max_active** - The maximum number of active connections to the LDAP server.
  Default: 8
- **max_idle** - The maximum number of idle connections in the pool awaiting requests. Default: 8

Encrypt sensitive system resources

The `system_info_encryption` section that controls encryption of sensitive system resources using either a local encryption key or remote KMIP key.

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

Default values:

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

**system_info_encryption**

Controls encryption of sensitive system resources using either a local encryption key or remote KMIP key.

**enabled**

Set to `true` to enable encryption of system resources that might contain sensitive information, including the `system.batchlog` and `system.paxos` tables, hint files, and the database commit log. After enabling system resource encryption in an environment that already has data, encrypt the existing SSTables by running `nodetool upgradesstables -a system.batchlog system.paxos`

**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**

Default: AES. The name of the JCE cipher algorithm used to encrypt system resources.

**Table 29: 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>
</tbody>
</table>
### Configuration

<table>
<thead>
<tr>
<th>cipher_algorithm</th>
<th>secret_key_strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blowfish</td>
<td>32-448</td>
</tr>
<tr>
<td>RC2</td>
<td>40-128</td>
</tr>
</tbody>
</table>

**secret_key_strength**

Default: 128. Length of key to use for the system resources. See Table 1 (page 320).

*Note:* DSE uses a matching local key or request the key type from the KMIP server. For KMIP, if an existing key does not match the KMIP server automatically generates a new key.

**chunk_length_kb**

Default: 64. 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 -a system batchlog paxos` on all nodes in the cluster.

**key_provider**

Set to `KmipKeyProviderFactory` to encrypt sensitive system data with a KMIP key. Comment out this property if using a local encryption key.

Default: none

**kmip_host**

Set to the `kmip_group_name` that defines the KMIP host in `kmip_hosts` (page 322) section. DSE requests a key from the KMIP host and uses the key generated by the KMIP provider. Default: none

#### Encrypted configuration properties settings

Settings for using encrypted passwords in sensitive configuration file properties.

```plaintext
system_key_directory: /etc/dse/conf
config_encryption_active: false
config_encryption_key_name: (key_filename | KMIP_key_URL )
```

**system_key_directory**

Path to the directory where local encryption 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. Default: `/etc/dse/conf`

See Setting up local encryption keys.

*Note:* This directory is not used for KMIP keys.

**config_encryption_active**

Default: false. Set to true to enable decryption of configuration property values using the specified `config_encryption_key_name` (page 322). When enabled, encrypt values for following properties:

- dse.yaml LDAP values:
Configuration

**Restriction:** Use plain text for the KMIP keystore or truststore passwords.

- cassandra.yaml SSL values:

  server_encryption_options.keystore_password
  server_encryption_options.truststore_password
  client_encryption_options.keystore_password
  client_encryption_options.truststore_password

**Tip:** dsetool encryptconfigvalue *(page 1034)* returns encrypts values using the config_encryption_key_name *(page 322)* key

**config_encryption_key_name**

Default: system_key. The default name is not configurable.

Set to the local encryption key filename or KMIP key URL to use for configuration file property value decryption.

**Note:** Use dsetool encryptconfigvalue *(page 1034)* to generate encrypted values for the configuration file properties.

**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.

Default values:

- **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
    - key_cache_millis: 300000
    - timeout: 1000

**kmip_hosts**

Connection settings for key servers that support the KMIP protocol.

**kmip_groupname**
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.

**hosts**
A comma-separated list of KMIP hosts using the Fully Qualified Domain Name (FQDN). DSE queries the host in the listed order.

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. For example: `/etc/dse/conf/KMIP_keystore.jks`

**keystore_type**
The type of key store. The default value is jks.

**keystore_password**
The password to access the key store.

**truststore_path**
The path to a Java truststore created using the KMIP root certificate. For example: `/etc/dse/conf/KMIP_truststore.jks`

**truststore_type**
The type of truststore. The default value is jks.

**truststore_password**
The password to access the truststore.

**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. Default: 300000. DataStax recommends using the default value.

**timeout**
Socket timeout in milliseconds. Default: 1000.

DSE Search index encryption settings

Default values:

```yaml
solr_encryption_options:
  decryption_cache_offheap_allocation: true
  decryption_cache_size_in_mb: 256
```

**solr_encryption_options**
Specify settings to tune encryption of search indexes.

**decryption_cache_offheap_allocation**
Specify whether to allocate shared DSE Search decryption cache off JVM heap. Default: true
**decryption_cache_size_in_mb**

Sets the maximum size of shared DSE Search decryption cache, in megabytes (MB). Default: 256

---

**DSE In-Memory options**

<table>
<thead>
<tr>
<th>max_memory_to_lock_mb:</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_memory_to_lock_fraction: 0.20</td>
</tr>
<tr>
<td># max_memory_to_lock_mb: 10240</td>
</tr>
</tbody>
</table>

**max_memory_to_lock_mb**

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

- **max_memory_to_lock_fraction**
  
  Specify a fraction of the system memory. The default value of 0.20 specifies to use up to 20% of system memory.

- **max_memory_to_lock_mb**

  Specify a maximum amount of memory in megabytes (MB).

---

**Node health options**

<table>
<thead>
<tr>
<th>node_health_options:</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_rate_ms: 50000</td>
</tr>
<tr>
<td>uptime_ramp_up_period_seconds: 10800</td>
</tr>
<tr>
<td>dropped_mutation_window_minutes: 30</td>
</tr>
</tbody>
</table>

**node_health_options**

Node health options are always enabled for all nodes. Node health is a score-based representation of how fit a node is to handle search queries.

- **refresh_rate_ms**

  Default: 60000

- **uptime_ramp_up_period_seconds**

  Default: 10800 (3 hours). 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.

- **dropped_mutation_window_minutes**

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

---

**Health-based routing**

| enable_health_based_routing: true |

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

Lease metrics

Default values:

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

**lease_metrics_options**

Lease holder statistics help monitor the lease subsystem for automatic management (page 418) 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 325)
- Reindexing of bootstrapped data (page 326)
- CQL Solr paging (page 326)
- Solr CQL query options (page 326)
- DSE Search resource upload limit (page 327)
- Shard transport options (page 327)
- DSE Search indexing settings (page 327)

Scheduler settings for DSE Search indexes

Default values:

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

**ttl_index_rebuild_options**

To ensure that records with TTLs are purged from search indexes when they expire, the search indexes are periodically checked for expired documents. The `ttl_index_rebuild_options` settings control the schedulers in charge of querying for and removing expired records, and the execution of the checks.
**fixed_rate_period**
Schedules how often to check for expired data in seconds. Default: 300

**initial_delay**
Speeds startup time by delaying the first TTL checks in seconds. Default: 20

**max_docs_per_batch**
Sets the maximum number of documents to check and delete per batch by the TTL rebuild thread. Default: 4096

**thread_pool_size**
To manage system resource consumption and prevent many search cores from executing simultaneous TTL deletes, defines the maximum number of cores that can execute TTL cleanup concurrently. Default: 1

Reindexing of bootstrapped data

async_bootstrap_reindex: 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.

CQL Solr paging

Options to specify the paging behavior.

**cql_solr_query_paging**
Options to specify the paging behavior.
- **off** - Default. Paging is off. Ignore driver paging settings for CQL Solr 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 (page 579).

- **driver** - Respects driver paging settings. Specifies to use Solr pagination (page 582) (cursors) only when the driver uses pagination. Enabled automatically for DSE SearchAnalytics workloads.

Solr CQL query options

Default value:
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: 10000 (10 seconds).

DSE Search resource upload limit

Default value:

solr_resource_upload_limit_mb: 10

solr_resource_upload_limit_mb
Default: 10. You can configure the maximum resource file size or disable resource
upload Sets the maximum DSE Search resource upload size limit in megabytes
(MB). Set to 0 to disable resource uploading.

Shard transport options

This shard transport option for inter-node communication between DSE Search nodes
controls timeout behavior during distributed queries.

Default values:

shard_transport_options:
  netty_client_request_timeout: 60000

shard_transport_options
For inter-node communication between DSE Search nodes.

netty_client_request_timeout
Default: 60000. The client request timeout is the maximum cumulative time (in
milliseconds) that a distributed search request will wait idly for shard responses.
Defines timeout behavior during distributed queries.

DSE Search indexing settings

DSE Search implements multi-threaded indexing to improve performance on multi-core
machines. All index updates are internally dispatched to a per-core indexing thread pool and
executed asynchronously, which allows for greater concurrency and parallelism. However,
index requests can return a response before the indexing operation is executed.

Default values:

max_solr_concurrency_per_core: 2
# enable_back_pressure_adaptive_nrt_commit: true
# back_pressure_threshold_per_core: 2000
# 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
max_solr_concurrency_per_core

Configures the maximum number of concurrent asynchronous indexing threads per DSE Search index. Default: number_of_available_CPU_cores.

If set to 1, DSE Search reverts to using synchronous indexing behavior, where data is synchronously written to the database in a single thread and indexed for DSE Search.

To achieve optimal performance, assign this value to number of available CPU cores divided by the number of search cores. For example, with 16 CPU cores and 4 search cores, the suggested value is 4. Also see Configuring and tuning indexing performance.

To prevent writes from overwhelming reads, reduce this value and adjust parallelDeleteTasks (page 519) in the search index config.

Note: Dynamic switching to search concurrency level at 1 is disallowed.

disable_back_pressure_adaptive_nrt_commit

Allows back pressure system to adapt max auto soft commit time (defined per search index config) to the actual load. Setting is respected only for NRT (near real time) cores. When DSE search cores have real-time (RT) live indexing, adaptive commits are disabled regardless of this property value. See live indexing with RT. Default: true

back_pressure_threshold_per_core

The total number of queued asynchronous indexing requests per search core. When this number is exceeded, back pressure prevents excessive resource consumption by throttling new incoming requests. DataStax recommends using a back_pressure_threshold_per_core value of 1000 * max_solr_concurrency_per_core (page 328).

Default: 2000

flush_max_time_per_core

The maximum time, in minutes, to wait for the flushing of asynchronous index updates, which 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 to ensure 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: 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, expressed. This advanced option should be changed only if exceptions happen during core loading.

Default: 5 (if not specified)

enable_index_disk_failure_policy
DSE Search activates the configured disk failure policy if IOExceptions occur during index update operations.
Default: false

**solr_data_dir**
The directory to store index data. By default, each DSE Search index is saved in solrconfig_data_dir/keystore_name.table_name, or as specified by the dse.solr.data.dir system property. See Managing the location of DSE Search data.

**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 schema.xml file. Then reload the core and reindex. The default value is false. To override false, set useFieldCache=true in the request.

### Performance Service options

- Global Performance Service options ([page 329](#))
- CQL Performance Service options ([page 329](#))
- Spark Performance Service options ([page 335](#))
- Spark Performance Service options ([page 335](#))

#### 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.

Default values:

```plaintext
performance_core_threads: 4
performance_max_threads: (cassandra.concurrent_writes)
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. Limited to the value of concurrent_writes ([page 287](#)) in the cassandra.yaml file. Default: The number of cassandra.concurrent_writes.

- **performance_queue_capacity**
  The number of queued tasks in the backlog when the number of performance_max_threads are busy. Default: 32000

#### CQL 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 1057) subcommands.

Default values:

```yaml
graph_events:
  ttl_seconds: 600

cql_slow_log_options:
  enabled: true
  threshold: 200.0
  minimum_samples: 100
  ttl_seconds: 259200
  skip_writing_to_db: true
  num_slowest_queries: 5

cql_system_info_options:
  enabled: false
  refresh_rate_ms: 10000

resource_level_latency_tracking_options:
  enabled: false
  refresh_rate_ms: 10000

db_summary_stats_options:
  enabled: false
  refresh_rate_ms: 10000

cluster_summary_stats_options:
  enabled: false
  refresh_rate_ms: 10000

spark_cluster_info_options:
  enabled: false
  refresh_rate_ms: 10000

histogram_data_options:
  enabled: false
  refresh_rate_ms: 10000
  retention_count: 3

user_level_latency_tracking_options:
  enabled: false
  refresh_rate_ms: 10000
  top_stats_limit: 100
  quantiles: false
```

**graph_events**
Graph event information.

**ttl_seconds**
Defines the TTL in milliseconds. Default: 600
cql_slow_log_options

Report distributed sub-queries for search (query executions on individual shards) that take longer than a specified period of time. See Collecting slow queries.

enabled
Enables (true) or disables (false) log entries for slow queries. Default: true

threshold
Defines the threshold (in milliseconds or as a percentile). Default: 200.0

• 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.

minimum_samples
Defines the initial number of queries before activating the percentile filter. Default: 100

ttl_seconds
Defines the time, in milliseconds, to keep the slow query log entries. Default: 259200

skip_writing_to_db
Keeps (true) slow queries in-memory only and does not write data to database. Default: true

Note: When false, the threshold must be >= 2000 ms to prevent a high load on database.

num_slowest_queries
The number of slow queries to keep in-memory. Default: 5

cql_system_info_options

CQL system information tables settings See Collecting system level diagnostics.

cql_system_info_options:
  enabled: false
  refresh_rate_ms: 10000

enabled
Default: false

refresh_rate_ms
Default: 10000

resource_level_latency_tracking_options

Data resource latency tracking settings. See Collecting system level diagnostics.

resource_level_latency_tracking_options:
  enabled: false
  refresh_rate_ms: 10000

enabled
Configuration

Default: false

**refresh_rate_ms**
Default: 10000

**db_summary_stats_options**

Database summary statistics settings. See Collecting database summary diagnostics.

```yaml
db_summary_stats_options:
  enabled: false
  refresh_rate_ms: 10000
```

**enabled**
Default: false

**refresh_rate_ms**
Default: 10000

**cluster_summary_stats_options**

Cluster summary statistics settings. See Collecting cluster summary diagnostics.

```yaml
cluster_summary_stats_options:
  enabled: false
  refresh_rate_ms: 10000
```

**enabled**
Default: false

**refresh_rate_ms**
Default: 10000

**spark_cluster_info_options**

See Monitoring Spark with Spark Performance Objects.

```yaml
spark_cluster_info_options:
  enabled: false
  refresh_rate_ms: 10000
```

**histogram_data_options**

Histogram data tables settings. See Collecting histogram diagnostics.

**enabled**
When true, the dropped mutation metrics are stored in the dropped_messages table in the dse_perf keyspace. Default: false

**refresh_rate_ms**
Default: 10000

**retention_count**
Default: 3

**user_level_latency_tracking_options**

User-resource latency tracking settings. See Collecting user activity diagnostics.

**enabled**
Default: false
### refresh_rate_ms
Default: 10000

### top_stats_limit
Default: 100

### quantiles
Default: false

#### DSE Search Performance Service options

These settings are used by the Performance Service. See [DSE Performance Service](#).

**Default values:**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Enabled</th>
<th>Ttl Seconds</th>
<th>Refresh Rate ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>solr_indexing_error_log_options</td>
<td>false</td>
<td>604800</td>
<td></td>
</tr>
<tr>
<td>solr_slow_sub_query_log_options</td>
<td>false</td>
<td>604800</td>
<td></td>
</tr>
<tr>
<td>solr_update_handler_metrics_options</td>
<td>false</td>
<td>604800</td>
<td></td>
</tr>
<tr>
<td>solr_request_handler_metrics_options</td>
<td>false</td>
<td>604800</td>
<td></td>
</tr>
<tr>
<td>solr_index_stats_options</td>
<td>false</td>
<td>604800</td>
<td></td>
</tr>
<tr>
<td>solr_cache_stats_options</td>
<td>false</td>
<td>604800</td>
<td></td>
</tr>
<tr>
<td>solr_latency_snapshot_options</td>
<td>false</td>
<td>604800</td>
<td></td>
</tr>
</tbody>
</table>

**solr_indexing_error_log_options**
Enable to collect record errors that occur during document indexing.

**enabled**
Default: false

**ttl_seconds**
Default: 604800

**async_writers**
Defines the number of server threads dedicated to writing in the log. More than one server thread might degrade performance. Default: 1

**solr_slow_sub_query_log_options**
See Collecting slow search queries.

**enabled**
Default: false

**ttl_seconds**
Default: 604800

**async_writers**
Defines the number of server threads dedicated to writing in the log. More than one server thread might degrade performance. Default: 1

**threshold_ms**
Default: 100

**solr_update_handler_metrics_options**
See Collecting handler statistics.

**enabled**
Determines whether the object is enabled at startup.

**ttl_seconds**
How many seconds a record survives before it is expired from the performance object.

**refresh_rate_ms**
Period (in milliseconds) between sample recordings for periodically updating statistics like the `solr_result_cache_stats`.

**solr_request_handler_metrics_options**
Records core-specific direct and request update handler statistics over time.

**enabled**
Default: false

**ttl_seconds**
Default: 604800

**refresh_rate_ms**
Default: 60000

**solr_index_stats_options**
See Collecting index statistics.

**enabled**
Default: false

**ttl_seconds**
Default: 604800

**refresh_rate_ms**
**solr_cache_stats_options**
See Collecting cache statistics.

- **enabled**
  Default: false
- **ttl_seconds**
  Default: 604800
- **refresh_rate_ms**
  Default: 60000

**solr_latency_snapshot_options**
See Collecting Apache Solr performance statistics.

- **enabled**
  Default: false
- **ttl_seconds**
  Default: 604800
- **refresh_rate_ms**
  Default: 60000

**Spark Performance Service options**

Default values:

```json
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**
    Default: false
  - **refresh_rate_ms**
    Default: 10000 milliseconds
  - **driver**
    Enables collection of the metrics by the Spark Driver.
  - **sink**
    Enables writing of the metrics collected from the Spark Driver. Default: false
  - **connectorSource**
    Enables writing of the Spark Cassandra Connector metrics at the Spark Driver. Default: false
  - **jvmSource**
Configuration

Enables JVM heap and GC metrics at the Spark Driver. Default: false

**stateSource**
Enables application state metrics. Default: false

**executor**
Enables collection of the metrics collected at Spark executors. Default: false

**sink**
Enables writing of the metrics collected at Spark executors. Default: false

**connectorSource**
Enables writing of the Spark Cassandra Connector metrics at Spark executors. Default: false

**jvmSource**
Enables JVM heap and GC metrics at Spark executors. Default: false

DSE Analytics options

- Spark (page 336)
- Starting Spark drivers and executors (page 338)
- DSE File System (DSEFS) options (page 339)
- Spark Performance Service (page 335)

Spark memory and Spark encryption options

Default values:

```
initial_spark_worker_resources: 0.7
spark_shared_secret_bit_length: 256
spark_security_enabled: false
spark_security_encryption_enabled: false

spark_daemon_readiness_assertion_interval: 1000

spark_ui_options:
  encryption: inherit
  encryption_options:
    enabled: false
    keystore: .keystore
    key_password: cassandra
    require_client_auth: false
    truststore: .truststore
    truststore_password: cassandra
  # Advanced settings
  # protocol: TLS
  # algorithm: SunX509
  # store_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]
```

**initial_spark_worker_resources**
DataStax Enterprise can control the memory and cores offered by particular Spark Workers in semi-automatic fashion. Specify the fraction of system resources that are made available to the Spark Worker.
The available resources are calculated in the following way:

- Spark Worker memory = initial_spark_worker_resources * (total system memory - memory assigned to DataStax Enterprise)
- Spark Worker cores = initial_spark_worker_resources * total system cores

The lowest values that you can assign to Spark Worker memory is 64 MB. 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. The range of the initial_spark_worker_resources value is 0.01 to 1. If the range is not specified, the default value 0.7 is used.

This mechanism is used by default to set the Spark Worker memory and cores. To override the default, uncomment and edit one or both SPARK_WORKER_MEMORY and SPARK_WORKER_CORES options in the spark-env.sh file.

spark-env.sh
The default location of the spark-env.sh file depends on the type of installation:

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>Default Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/dse/spark/spark-env.sh</td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/spark/conf/spark-</td>
</tr>
<tr>
<td></td>
<td>env.sh</td>
</tr>
</tbody>
</table>

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

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.

encryption
- inherit - inherit the SSL settings from the client encryption options. Default.
- custom - use the following encryption_options (page 338).
**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**
Enable (true) or disable (false) 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: .keystore

**keystore_password**
The password to access the key store. Default: cassandra

**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`.

**truststore_password**
The password to access the truststore. Default: cassandra

**protocol**
Defines the encryption protocol. Default: TLS

**algorithm**
Defines the key manager algorithm. Default: SunX509

**store_type**
Defines the keystore type. Default: JKS

**cipher_suites**
Defines the cipher suites for Spark encryption. Default:

- 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

Starting Spark drivers and executors

Options to configure how Spark driver and executor processes are created and managed.

Default values:

```yaml
spark_process_runner:
  runner_type: default
  run_as_runner_options:
    user_slots:
      - slot1
      - slot2
```

**runner_type**
• default
• run_as - Use the run_as_runner_options options. See Running Spark processes as separate users (page 424).

run_as_runner_options
Define the slot users for separating Spark processes users from the DSE service user. See Running Spark processes as separate users (page 424).

DSE File System (DSEFS) options
Properties to enable and configure the DSE file system (DSEFS (page 475)).

Note: DSEFS replaces the Cassandra File System (CFS).

Default values:

dsefs_options:
  enabled: false
  keyspace_name: dsefs
  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

# Advanced properties for DSEFS
# service_startup_timeout_ms: 30000
# service_close_timeout_ms: 600000
# server_close_timeout_ms: 600000
# compression_frame_max_size: 1048576
# query_cache_size: 2048
# query_cache_expire_after_ms: 2000
# gossip_options:
#   round_delay_ms: 5000
#   startup_delay_ms: 5000
#   shutdown_delay_ms: 30000
# rest_options:
#   request_timeout_ms: 330000
#   connection_open_timeout_ms: 55000
#   client_close_timeout_ms: 60000
#   server_request_timeout_ms: 300000
#   idle_connection_timeout_ms: 0
#   internode_idle_connection_timeout_ms: 120000
#   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

dsefs_options
DSE File System (DSEFS) (page 475) options determine whether DSEFS should be enabled on this node.

**enabled**
Enable or disable DSEFS. This parameter takes one of the following values:

- true - enables DSEFS on this node, regardless of the workload.
- false - disables DSEFS on this node, regardless of the workload. Default.
- blank or commented out (#) - DSEFS will start only if the node is configured to run analytics workloads.

**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: 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.

**public_port**
The public port on which DSEFS listens for clients. 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 RPC (page 284) address. Default: 5598

**private_port**
The private port for DSEFS inter-node communication. Do not open this port to firewalls; this private port must be not visible from outside of the cluster. Default: 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 different than the devices that are used for DataStax Enterprise. Using multiple directories on JBOD improves performance and capacity. Default: /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: 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: 5368709120

**Advanced properties for DSEFS**

**service_start_timeout_ms**
Wait time, in milliseconds, before the DSEFS server times out while waiting for services to bootstrap. Default: 30000

**service_close_timeout_ms**
Wait time, in milliseconds, before the DSEFS server times out while waiting for services to close. Default: 600000

**server_close_timeout_ms**
Wait time, in milliseconds, that the DSEFS server waits during shutdown before closing all pending connections.

**compression_frame_max_size**
The maximum accepted size of a compression frame defined during file upload. Default: 1048576

**query_cache_size**
Maximum number of elements in a single DSEFS Server query cache. Default: 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: 2000

**gossip options**
Options to configure DSEFS gossip rounds.

**round_delay_ms**
The delay, in milliseconds, between gossip rounds. Default: 5000

**startup_delay_ms**
The delay time, in milliseconds, between registering the location and reading back all other locations from the database. Default: 5000

**shutdown_delay_ms**
The delay time, in milliseconds, between announcing shutdown and shutting down the node. Default: 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: 330000

**connection_open_timeout_ms**
The time, in milliseconds, that the client waits to establish a new connection. Default: 55000

**client_close_timeout_ms**
The time, in milliseconds, that the client waits for pending transfer to complete before closing a connection. Default: 60000

**server_request_timeout_ms**
The time, in milliseconds, to wait for the server rest call to complete. Default: 30000

**idle_connection_timeout_ms**
The time, in milliseconds, to wait before closing an idle connection. Closing idle connections is disabled by default. Default: 0 (disabled)

**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.
Configuration

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

**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

Audit logging options

Default values:

```yaml
audit_logging_options:
  enabled: false
  logger: SLF4JAuditWriter
  retention_time: 0
```

**audit_logging_options**
To get the maximum information from data auditing, turn on data auditing on every node. See [Enabling data auditing in DataStax Enterprise](#) and [Configuring audit logging](#).

**enabled**
Default: false

**logger**
Default: SLF4JAuditWriter

- **SLF4JAuditWriter** - Logs audit information to the SLF4JAuditWriter logger. Audit logging configuration settings are in the logback.xml file.

  **logback.xml**
The location of the logback.xml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>Installer-Services installations</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/dse/cassandra/</td>
<td></td>
</tr>
<tr>
<td>logback.xml</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tarball installations</th>
<th>Installer-No Services installations</th>
</tr>
</thead>
<tbody>
<tr>
<td>installation_location/</td>
<td></td>
</tr>
<tr>
<td>resources/cassandra/conf/</td>
<td></td>
</tr>
<tr>
<td>logback.xml</td>
<td></td>
</tr>
</tbody>
</table>

- **CassandraAuditWriter** - Logs audit information to the dse_audit.audit_log database table. This logger can be run synchronously or asynchronously. See
related cassandra_audit_writer_options (page 343) configuration entries and
Configuring audit logging to a database table.

**included_categories or excluded_categories**
The default is to include all categories. Specify either included or excluded
categories.

Comma separated list of audit event categories to include or exclude from the audit
log. Categories are: QUERY, DML, DDL, DCL, AUTH, ERROR.

- **included_categories**: comma_separated_list

  or

- **excluded_categories**: comma_separated_list

**included_keyspaces or excluded_keyspaces**
The default is to include all keyspaces. Specify either included or excluded
keyspaces. Specifying both is an error.

Use a regular expression to filter keyspaces, or use a comma separated list of
keyspaces to be included or excluded from the audit log.

- **included_categories**: comma_separated_list

  or

- **excluded_categories**: comma_separated_list

**retention_time**
The amount of time, in hours, that audit events are retained by supporting loggers.
Only CassandraAuditWriter supports retention time. Values of 0 or less retain
events forever. Default: 0

**cassandra_audit_writer_options**
Configuration options for CassandraAuditWriter.

```
cassandra_audit_writer_options:
  mode: sync
  batch_size: 50
  flush_time: 500
  num_writers: 10
  queue_size: 10000
  write_consistency: QUORUM
  dropped_event_log: /var/log/cassandra/dropped_audit_events.log
```

**mode**
Sets the mode the writer runs in. Default: sync

- **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. While this 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.

**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 292) in cassandra.yaml. Default: 50

**flush_time**
- Available only when mode: async.
- 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

**num_writers**
- Available only when mode: async.
- The number of worker threads asynchronously logging events to the CassandraAuditWriter. Default: 10

**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: 10000

**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. Default: /var/log/cassandra/dropped_audit_events.log

**day_partition_millis**
- To spread audit log information across multiple nodes, specify the interval, in milliseconds, between changing nodes. For example, specify 43200000 milliseconds to change the target node every 12 hours. Default: 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 in CREATE or ALTER table statements.

Default values:

```yaml
tiered_storage_options:
```
strategy1:
  tiers:
  - paths:
    - /mnt1
    - /mnt2
  - paths:
    - /mnt3
    - /mnt4
  - paths:
    - /mnt5
    - /mnt6

Note: To manage compaction options, use the compaction option in CREATE TABLE or ALTER TABLE.

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
Specific file paths to 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.

Default values:

```yaml
#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 DSE Advanced Replication.

**enabled**
- Set `enabled:true` on an edge node to collect data in the replication log. Default: `false`.

**conf_driver_password_encryption_enabled**
- Enable or disable encryption of driver passwords. When enabled, the stored driver password is expected to be encrypted with the system key. After you create the system key, you must copy the same system key to every node in the cluster.

**advanced_replication_directory**
- Set the directory for storing advanced replication CDC logs. Default is `/var/lib/cassandra/advrep`. A directory `replication_logs` will be created within the specified directory.

**security_base_path**
- The base path to prepend to paths in the Advanced Replication configuration locations, including locations to SSL keystore, SSL truststore, and so on. Default: `/base/path/to/advrep/security/files/`

Inter-node messaging options

Configuration for the internal messaging service used by several components of DataStax Enterprise. For 5.0 and later, all internode messaging requests use this service.

```
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. Default: 256

**server_acceptor_threads**
- The number of server acceptor threads. Default: the number of available processors.

**server_worker_threads**
- The number of server worker threads. Default: the number of available processors * 8.

**client_max_connections**
- The maximum number of client connections. Default: 100

**client_worker_threads**
The number of client worker threads. Default: the number of available processors * 8.

**handshake_timeout_seconds**
Timeout for communication handshake process. Default: 10

**client_request_timeout_seconds**
Timeout for non-query search requests like core creation and distributed deletes. Default: 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 347)
- DSE Graph id assignment and partitioning strategy options (page 349)
- DSE Graph listener options (page 351)
- DSE Graph messaging options (page 351)
- DSE Graph event observers options (page 352)
- DSE Graph shared data options (page 353)
- DSE Graph Gremlin Server options (page 353)

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.

Default values:

```yaml
graph:
  adjacency_cache_clean_rate: 1024
  adjacency_cache_max_entry_size_in_mb: 0
  adjacency_cache_size_in_mb: 128
  analytic_evaluation_timeout_in_minutes: 10080
  gremlin_server_enabled: true
  index_cache_clean_rate: 1024
  index_cache_max_entry_size_in_mb: 0
  index_cache_size_in_mb: 128
  max_query_queue: 10000
  #max_query_threads:
  realtime_evaluation_timeout_in_seconds: 30
  schema_agreement_timeout_in_ms: 10000
  schema_mode: Production
  system_evaluation_timeout_in_seconds: 180
  window_size: 100000
```
Configuration

max_query_params: 256

**graph**
These graph options are system-level configuration options and options that are shared between graph instances.

**adjacency_cache_clean_rate**
The number of stale rows per second to clean from each graph's adjacency cache. Default: 1024.

**adjacency_cache_max_entry_size_in_mb**
The maximum entry size in each graph's adjacency cache. When set to zero, the default is calculated based on the cache size and the number of CPUs. Entries that exceed this size are quietly dropped by the cache without producing an explicit error or log message. Default: 0.

**adjacency_cache_size_in_mb**
The amount of RAM to allocate to each graph's adjacency (edge and property) cache. Default: 128.

**analytic_evaluation_timeout_in_minutes**
Maximum time to wait for an analytic (Spark) traversal to evaluate. Default: 10080 (7 days).

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.

**gremlin_server_enabled**
Enables or disables Gremlin Server. Default: true.

**index_cache_clean_rate**
The number of stale entries per second to clean from the adjacency cache. Default: 1024.

**index_cache_max_entry_size_in_mb**
The maximum entry size in the index adjacency cache. When set to zero, the default is based on the cache size and the number of CPUs. Value: integer. + #
default is calculated based on the cache size and the number of CPUs. Entries that exceed this size are quietly dropped by the cache without producing an explicit error or log message. Default: 0.

**index_cache_size_in_mb**
The amount of RAM to allocate to the index cache. Default: 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. Default: 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:
- If gremlinPool is present and nonzero:
  10 * the gremlinPool setting
- If gremlinPool is not present in this file or set to zero:
The number of available CPU cores

See gremlinPool.

**realtime_evaluation_timeout_in_seconds**
Maximum time to wait for a real-time traversal to evaluate. Default: 30 seconds.

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.

**schema_agreement_timeout_in_ms**
Maximum time to wait for cassandra to agree on schema versions before timing out. Default: 10000

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.

**schema_mode**
Controls the way that the schemas are handled. Valid values:

- 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.

**system_evaluation_timeout_in_seconds**
Maximum time to wait for a system-based request to execute. Default: 180 (3 minutes).

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.

**window_size**
The number of samples to keep when aggregating log events. Only a small subset of graph's log events use this system. Modifying this setting is rarely necessary or helpful. Default: 100000.

**max_query_params**
The maximum number of parameters that can be passed on a graph query request for TinkerPop drivers and drivers using 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: 256

DSE Graph id assignment and partitioning strategy options

Default values:

```
ids:
```
block_renew: 0.8  
community_reuse: 28  
consistency_mode: GLOBAL  
# datacenter_id: integer unique per DC when consistency_mode:  
DC_LOCAL  
id_hash_modulus: 20  
member_block_size: 512

ids  
DSE Graph configuration options for standard vertex ID assignment and partitioning strategies.

block_renew  
The graph standard vertex ID allocator operates on blocks of contiguous IDs. Each block is allocated using a database lightweight transaction that requires coordination latency. To hide the cost of allocating a standard ID block, the allocator begins asynchronously buffering a replacement block whenever a current block is nearly empty. This block_renew parameter defines "nearly empty" as a floating point number between 0 and 1. The value is how much of a standard ID block can be used before graph starts asynchronously allocating its replacement. This setting has no effect on custom IDs. Value must be between 0 and 1. Default: 0.8.

community_reuse  
For graphs using standard vertex IDs, if a transaction creates multiple vertices, the allocator attempts to assign vertex IDs that colocate vertices on the same database replicas. If an especially large vertex cohort is created, the allocator chunks the vertex creation and assigns a random target location to avoid load hot spotting. This setting controls the vertex chunk size and has no effect on custom IDs. Default: 28.

consistency_mode  
Must be set to DC_LOCAL or GLOBAL.  
  • DC_LOCAL - The node uses LOCAL_QUORUM when allocating an ID for a graph vertex. The datacenter_id option must be correctly configured on every node in the cluster.  
  • GLOBAL - (Default) The node uses QUORUM when allocating an ID for a graph vertex. The datacenter_id option is ignored.  

This option must have the same value on every node in the cluster. Its value can only be changed when the entire cluster is stopped. This setting has no effect on custom IDs.

datacenter_id  
Applies only when consistency_mode is DC_LOCAL. Set to an arbitrary value between 1 and 127, inclusive. This setting has no effect on custom IDs.  

Warning: Each datacenter in the cluster must have a unique datacenter_id. Violating this constraint will corrupt the graph database without warning.  

This setting has no effect on custom IDs. Default: no explicit default value.

id_hash_modulus  
An integer between 1 and 2^24 (both inclusive) that affects maximum ID capacity and the maximum storage space used by ID allocations. Lower values reduce the storage space consumed and the lightweight transaction overhead imposed at
Configuration

startup. Lower values also reduce the total number of IDs that can be allocated over the life of a graph, because this parameter is proportional to the allocatable ID space. However, the proportion coefficient is Long.MAX_VALUE (2^63-1), so ID headroom should be sufficient, practically speaking, even if this is set to 1. This setting has no effect on custom IDs. Default: 20.

**member_block_size**

The graph standard vertex ID allocator claims uniformly-sized blocks of contiguous IDs using lightweight transactions on the database. This setting controls the size of each block. This setting has no effect on custom IDs. Default: 512.

DSE Graph listener options

Default values:

```
listener:
  listener_name: string
    black_types: # This list is empty by default
    interval_in_seconds: 3600
    type: slf4j
    white_types: # This list is empty by default
```

**listener**

Options that contain all registered state listeners identified by their name.

**listener_name**

Replace `listener_name` with a string that identifies the listener. The string must begin with a lower case letter and can be composed of lowercase letters, numbers, and underscores.

**black_types**

The names of state types that are ignored. All state types but those given are listened to. Default: (empty).

**interval_in_seconds**

The interval in which the state values are logged. Default: 3600

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.

**type**

The type of the state listener. Must be one of the following values: slf4j. Default: slf4j.

**white_types**

The names of state types that should be listened. Only those state types are listened to and all others ignored. Default: (empty).

DSE Graph messaging options

Default values:

```
msg:
  graph_msg_timeout_in_ms: 5000
```

**msg**
Options to configure DSE Graph internal query and lightweight messaging system.

**graph_msg_timeout_in_ms**
Graph messages must be acknowledged within this interval, or else the message is assumed dropped/failed. Graph retries the message or fails the responsible request if the retry limit is exceeded. Default: 5000

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.

DSE Graph event observers options

Default values:

```yaml
observer:
  observer_name: string
    black_types: # This list is empty by default
    observed_graphs: # This list is empty by default
    slow_threshold_in_ms: 300000
    type: slf4j
    white_types: # This list is empty by default
```

**observer**
Options to configure all registered event observers identified by their name.

**observer_name**
Replace `observer_name` with a string that identifies the event observers. This string is the names of event types that are ignored. All event types but those given are observed. The string must begin with a lower case letter and can be composed of lowercase letters, numbers, and underscores. Value: YAML-formatted list of strings.

**black_types**
The names of event types that are ignored. All event types but those given are observed. Value: YAML-formatted list of strings. Default: (empty).

**observed_graphs**
The names of the graphs for which events are observed. Value: YAML-formatted list of strings. Default: (empty).

**slow_tx_graphs**
The names of the graphs for which slow transactions are monitored. Default: (empty).

**slow_threshold_in_ms**
Threshold at which slow queries get reported. Default: 300000

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.

**type**
The type of the event observer. Must be one of the following values: slf4j, slow_request. Default: slf4j.

**white_types**
The names of event types that should be observed. Only those event types are observed and all others ignored. Value: YAML-formatted list of strings. Default: (empty).

DSE Graph shared data options

Default values:

```
shared_data:
    refresh_interval_in_ms: 60000
```

**shared_data**
Options for shared data in DSE Graph.

**refresh_interval_in_ms**
The interval between refreshes in which the graph schema is reread from the database tables. Note that schema is also immediately updated when schema changes occur, so this parameter is a fail safe to poll for schema changes periodically. Default: 60000

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.

DSE Graph Gremlin Server options

The Gremlin Server is configured using Apache TinkerPop specifications.

Default values:

```
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 `port` value identifies the available communications port for Gremlin Server. Default: 8182
threadPoolWorker
The number of worker threads that handle 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. Default: 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. Default: 8

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 Configuring the Gremlin console for Gremlin Server in the remote.yaml file.
cassandra.yaml
The location of the cassandra.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/cassandra/cassandra.yaml</td>
</tr>
</tbody>
</table>
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 355)
- Ec2Snitch (page 357)
- Ec2MultiRegionSnitch (page 358)

This topic also includes instructions for migrating (page 355) 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-rackdc.properties
The location of the `cassandra-rackdc.properties` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/cassandra/cassandra-rackdc.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>installation_location/resources/cassandra/conf/cassandra-rackdc.properties</td>
</tr>
</tbody>
</table>

**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 355) snitch is recommended for production. See Migrating from the PropertyFileSnitch to the GossipingPropertyFileSnitch (page 355).

**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.

```
# 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
```
cassandra-topology.properties
The location of the cassandra-topology.properties file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/cassandra/cassandra-topology.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>installation_location/resources/cassandra/conf/cassandra-topology.properties</td>
</tr>
</tbody>
</table>

### Configuring snitches for cloud providers

**Ec2Snitch**

Use the Ec2Snitch for simple cluster deployments on Amazon EC2 where all nodes in the cluster are within a single region.

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.) Because private IPs are used, this snitch does not work across multiple regions.

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.

**Keyspace strategy options**

When defining your keyspace strategy options, use the EC2 region name, such as `us-east`, as your datacenter name.

cassandra-rackdc.properties

The location of the `cassandra-rackdc.properties` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Type</th>
<th>File Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/dse/cassandra/cassandra-rackdc.properties</td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/cassandra/cassandra-rackdc.properties</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/cassandra/conf/cassandra-rackdc.properties</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>installation_location/resources/cassandra/conf/cassandra-rackdc.properties</td>
</tr>
</tbody>
</table>

**Ec2MultiRegionSnitch**

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 \texttt{broadcast_address} to allow cross-region connectivity. Configure each node as follows:

1. In the cassandra.yaml, set the \texttt{listen_address (page 280)} to the private IP address of the node, and the \texttt{broadcast_address (page 292)} 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:

   
   \begin{verbatim}
   seeds: 50.34.16.33, 60.247.70.52
   \end{verbatim}

   To find the public IP address, from each of the seed nodes in EC2:

   \begin{verbatim}
   $ curl http://instance-data/latest/meta-data/public-ipv4
   \end{verbatim}

   \textbf{Note:} Do not make all nodes seeds, see Internode communications (gossip).

3. Be sure that the \texttt{storage_port (page 294)} or \texttt{ssl_storage_port (page 310)} 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 \texttt{us-east-1} region, \texttt{us-east} is the datacenter name and \texttt{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 \texttt{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.)
### Configuration

#### Region: us-east

Node and datacenter:
- **node0**
  \( dc\textunderscore suffix=_1\textunderscore transactional \)
- **node1**
  \( dc\textunderscore suffix=_1\textunderscore transactional \)
- **node2**
  \( dc\textunderscore suffix=_2\textunderscore transactional \)
- **node3**
  \( dc\textunderscore suffix=_2\textunderscore transactional \)
- **node4**
  \( dc\textunderscore suffix=_1\textunderscore analytics \)
- **node5**
  \( dc\textunderscore suffix=_1\textunderscore search \)

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

Node and datacenter:
- **node0**
  \( dc\textunderscore suffix=_1\textunderscore transactional \)
- **node1**
  \( dc\textunderscore suffix=_1\textunderscore transactional \)
- **node2**
  \( dc\textunderscore suffix=_2\textunderscore transactional \)
- **node3**
  \( dc\textunderscore suffix=_2\textunderscore transactional \)
- **node4**
  \( dc\textunderscore suffix=_1\textunderscore analytics \)
- **node5**
  \( dc\textunderscore suffix=_1\textunderscore search \)

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.

**cassandra-rackdc.properties**

The location of the **cassandra-rackdc.properties** file depends on the type of installation:

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>File Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td><code>/etc/dse/cassandra/cassandra-rackdc.properties</code></td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td><code>/etc/dse/cassandra/cassandra-rackdc.properties</code></td>
</tr>
<tr>
<td>Tarball installations</td>
<td><code>installation_location/resources/cassandra/conf/cassandra-rackdc.properties</code></td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td><code>installation_location/resources/cassandra/conf/cassandra-rackdc.properties</code></td>
</tr>
</tbody>
</table>

**cassandra.yaml**

The location of the **cassandra.yaml** file depends on the type of installation:
GoogleCloudSnitch

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.

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:

Note: Datacenter names are case-sensitive.

<table>
<thead>
<tr>
<th>Node</th>
<th>dc_suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>node0</td>
<td>dc_suffix=_a_transactional</td>
</tr>
<tr>
<td>node1</td>
<td>dc_suffix=_a_transactional</td>
</tr>
<tr>
<td>node2</td>
<td>dc_suffix=_a_transactional</td>
</tr>
<tr>
<td>node3</td>
<td>dc_suffix=_a_transactional</td>
</tr>
<tr>
<td>node4</td>
<td>dc_suffix=_a_analytics</td>
</tr>
<tr>
<td>node5</td>
<td>dc_suffix=_a_search</td>
</tr>
</tbody>
</table>

cassandra-rackdc.properties
The location of the cassandra-rackdc.properties file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/cassandra/cassandra-rackdc.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/cassandra/cassandra-rackdc.properties</td>
</tr>
</tbody>
</table>
Configuration

<table>
<thead>
<tr>
<th>Tarball installations</th>
<th>installation_location/resources/cassandra/conf/cassandra-rackdc.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-No Services installations</td>
<td>installation_location/resources/cassandra/conf/cassandra-rackdc.properties</td>
</tr>
</tbody>
</table>

**CloudstackSnitch**

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.

**DataStax Enterprise start-up parameters**

You can run DataStax Enterprise (DSE) with start-up parameters by adding them to the `jvm.options` file or at the command line when starting up a installation.

**Tip:** You can also add options such as maximum and minimum heap size to the `jvm.options` file to pass them to the Java virtual machine at start up, rather than setting them in the environment.

**Usage**

Add a start-up parameter to the `jvm.options` file as follows:

```
-Dparameter_name
```

**jvm.options**

The location of the `jvm.options` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/cassandra/jvm.options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>installation_location/resources/cassandra/conf/jvm.options</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/cassandra/conf/jvm.options</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>installation_location/resources/cassandra/conf/jvm.options</td>
</tr>
</tbody>
</table>

When starting up Tarball or Installer-No Services installations, you can set parameters from the command line:

```
installation_location/bin/dse cassandra [-Dparameter_name -Dparameter_name ...]
```

See the Example section *(page 365).*

**Start-up parameters**

Use the `-D` option before the start-up parameter name on the command line or in the `jvm.options` file.

**-Dcassandra.auto_bootstrap=false**

`false` on initial set-up of the cluster. The next time you start the cluster, you do not need to change the `cassandra.yaml` file on each node to revert to `true`. default setting in the `cassandra.yaml` file.

Default: `true`.

**-Dcassandra.available_processors=number_of_processors**
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.

-Dcassandra.config=directory
  Sets the directory location of the cassandra.yaml file. The default location depends on the type of installation.

-Dcassandra.consistent.rangemovement=true
  Set to true, makes bootstrapping behavior effective.

-Dcassandra.disable.auth_caches_remote_configuration=true
  Disables 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.

-Dcassandra.expiration_date_overflow_policy=POLICY
  Set the policy for TTL (time to live) timestamps that exceed 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.
  Default: REJECT.

-Dcassandra.force_default_indexing_page_size=true
  Disable dynamic calculation of the page size used when indexing an entire partition (during initial index build/rebuild). If set to true, the page size will be fixed to the default of 10000 rows per page.

-Dcassandra.ignore_dc=true | false
  When set to true, ignores the datacenter name change on startup. Applies only when using DseSimpleSnitch. (Default: false)

-Dcassandra.initial_token=token
  Use when DSE is not using virtual nodes (vnodes). Sets the initial partitioner token for a node the first time the node is started. (Default: disabled)
  Note: Vnodes automatically select tokens.

-Dcassandra.join_ring=true | false
  When set to false, prevents the node from joining a ring on startup. (Default: true)
  You can add the node to the ring afterwards using nodetool join and a JMX call.

-Dcassandra.load_ring_state=true | false
  When set to false, clears all gossip state for the node on restart. (Default: true)

-Dcassandra.metricsReporterConfigFile=file
  Enables pluggable metrics reporter.

-Dcassandra.native_transport_port=port
  Sets the port on which the CQL native transport listens for clients. (Default: 9042)

-Dcassandra.native_transport_startup_delay_seconds=seconds
  Delays the startup of native transport server for the number of seconds. (Default: 0)

-Dcassandra.partitioner=partitioner
Sets the partitioner. (Default: org.apache.cassandra.dht.Murmur3Partitioner)

-Dcassandra.partition_sstables_by_token_range=true | false
Whether to disable JBOD SSTable partitioning by token range to multiple data_file_directories. (Default: true). Set to false only as directed by DataStax Support.

-Dcassandra.replace_address=listen_address_of_dead_node|broadcast_address_of_dead_node
To replace a node, restart a new node in its place specifying the listen_address (page 280) or broadcast_address (page 292) that the new node is assuming. 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 Ec2MultiRegionSnitch (page 358).

-Dcassandra.replay_list=table
Allows restoring specific tables from an archived commit log.

-Dcassandra.ring_delay_ms=ms
Defines the amount of time a node waits to hear from other nodes before formally joining the ring. (Default: 30000ms)

-Dcassandra.rpc_port=port
Sets the port for the Thrift RPC service, which is used for client connections. (Default: 9160).

-Dcassandra.ssl_storage_port=port
Sets the SSL port for encrypted communication. (Default: 7001)

-Dcassandra.start_native_transport=true | false
Enables or disables the native transport server. See start_native_transport (page 299) in cassandra.yaml. (Default: true)

-Dcassandra.start_rpc=true | false
Enables or disables the Thrift RPC server. (Default: true)

-Dcassandra.storage_port=port
Sets the port for inter-node communication. (Default: 7000)

-Dcassandra.triggers_dir=directory - Deprecated
Sets the default location for the triggers JARs.

-Dcassandra.write_survey=true
Enables 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.

LDAP tuning switches

-Ddse.search.client.timeout.secs=seconds
Native driver search core management calls using the dsetool search-specific commands use the default request timeout of 600 seconds (10 minutes).

-Ddse.ldap.connection.timeout.ms
The number of milliseconds before the connection times out.

-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.

-Ddse.ldap.pool.exhausted.action
Determines what the pool does when it is full. It can be one of:

- **fail** - the pool will throw an exception
- **block** - the pool will block for max wait ms (default)
- **grow** - the pool will just keep growing (not recommended)

-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.

-Ddse.ldap.pool.test.borrow
Tests a connection when it is borrowed from the pool.

-Ddse.ldap.pool.test.return
Tests a connection returned to the pool.

-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.

-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.

-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.

-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.

-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`.

### triggers
The location of the `triggers` directory depends on the type of installation:

<table>
<thead>
<tr>
<th>Type of Installation</th>
<th>Directory Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td><code>/etc/dse/cassandra/triggers</code></td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td><code>/etc/dse/cassandra/triggers</code></td>
</tr>
<tr>
<td>Tarball installations</td>
<td><code>installation_location/resources/cassandra/conf/triggers</code></td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td><code>installation_location/resources/cassandra/conf/triggers</code></td>
</tr>
</tbody>
</table>

**Starting a node without joining the ring:**

- Command line:

  ```bash
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

Choosing a compaction strategy

To implement the chosen compaction strategy:

1. To understand how compaction and compaction strategies work, read How is data maintained?

2. Review your application’s requirements use this information to answer the questions below.

3. Configure the table (page 367) to use the most appropriate strategy.

4. Test the compaction strategies (page 367) against your data.

Which compaction strategy is best?

The following questions are based on the experiences of developers and users with the strategies.

Does your table process time series data?
If so, your best choice is Compaction strategiesTWCS. If not, the following questions introduce other considerations to guide your choice.

Does your table handle more reads than writes, or more writes than reads? 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 properties.

You can start compaction manually using the `nodetool compact` 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` and measure the results.
- Set up a node on your existing cluster and use the `write survey mode` to sample live data.

Configuring Virtual Nodes

**Virtual node (vnode) configuration**

Virtual nodes simplify many tasks in DataStax Enterprise, 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.

DataStax Enterprise 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 analytics datacenter (no vnodes).
- A search datacenter with vnodes.
Guidelines for using virtual nodes

Whether virtual nodes (vnodes) are enabled or disabled depends on the initial cassandra.yaml settings. There are two methods of distributing token ranges. DataStax recommends using the allocation algorithm. Use the same method on all systems in the datacenter.

**Allocation algorithm**

- Optimizes token range distribution between nodes and racks in the datacenter based on the keyspace replication factor (*allocate_tokens_for_local_replication_factor (page 293)*) of the datacenter.
- Distributes the token ranges proportionately using the *num_tokens (page 293)* settings. 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.

**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.

DataStax recommends using 8 vnodes (tokens). This distributes the workload between systems with a ~10% variance and has minimal impact on performance. Set the number of vnode tokens (*num_tokens*) based on the workload distribution requirements of the datacenter:

<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>

Enabling vnodes

In the cassandra.yaml file:

1. Uncomment *num_tokens (page 293)* and set the required number of tokens.

2. (Recommended) To use the allocation algorithm uncomment *allocate_tokens_for_local_replication_factor (page 293)* 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 293) 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 278) 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 293) and allocate_tokens_for_local_replication_factor (page 293).

   b. Uncomment the initial_token (page 293) 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 Advance 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

SparkR

DataStax Enterprise supports SparkR for R analytic processing.

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 475))

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.

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:

- cfs
- cfs_archive
- dse_leases
- dsefs
- "HiveMetaStore"
- spark_system

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:

```
ALTER KEYSPACE dse_leases
WITH REPLICATION = {
  '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
```
Using DataStax Enterprise advanced functionality

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 search queries (page 572). 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 469).

**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 (page 582).

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.

dse.yaml
The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/dse/conf/dse.yaml</td>
</tr>
</tbody>
</table>

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 519). Fields that are not defined in the search index schema columns not defined are excluded in the results returned from Spark queries.

**Using predicate push down in Spark SQL**

Solr predicate push down allows queries in SearchAnalytics datacenters to use Solr-indexed columns in Spark SQL queries. To enable Solr predicate push down, set the spark.sql.dse.solr.enable_optimization property to true either on a global or per-table or per-dataset basis.

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
```
Using DataStax Enterprise advanced functionality

To create a temporary table in Spark SQL with Solr predicate push down enabled:

```scala
CREATE TEMPORARY TABLE temp USING org.apache.spark.sql.cassandra OPTIONS (table "tab", keyspace "ks", spark.sql.dse.solr.enable_optimization "true");
```

Set the `spark.sql.dse.solr.enable_optimization` property globally by adding it to the server configuration file (page 417).

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 423).

```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.
Using DataStax Enterprise advanced functionality

Figure 1: Traditional and DSE Analytics Solo deployments

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.

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.
Using DataStax Enterprise advanced functionality

For information on creating a DSE Analytics Solo datacenter, see Creating a DSE Analytics Solo datacenter (page 433).

Analyzing data using Spark

Spark is the default mode when you start an analytics node in a packaged installation.

About Spark

Spark is the default mode when you start an analytics node in a packaged installation. 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 405) for accessing data stores in DSE
- DSE Resource Manager for managing (page 410) Spark components in a DSE cluster
- Spark Job Server (page 459)
- Spark SQL (page 439) support
- Spark SQL Thrift Server (page 447)
- Spark streaming (page 437)
- DataFrames (page 446) API to manipulate data within Spark
- Using SparkR with DataStax Enterprise (page 454)

Spark architecture

The software components for a single DataStax Enterprise analytics node are:

- Spark Worker
- DataStax Enterprise File System (DSEFS)
- Cassandra File System (CFS), deprecated as of DSE 5.1
- 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
Using DataStax Enterprise advanced functionality

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 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 379).
Figure 2: Spark integration with DataStax Enterprise
As you run Spark, you can access data in the Hadoop Distributed File System (HDFS), the Cassandra File System (CFS), 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 spark_system 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 master-address
```

DataStax Enterprise provides Automatic Spark Master management (page 418).

Unsupported Spark 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 for CQL execution 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 and Installer-Services 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 1090), the node is launched as a Spark node. You can enable additional components.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Option in <code>/etc/default/dse</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td><code>SPARK_ENABLED=1</code></td>
<td>Start the node in Spark mode.</td>
</tr>
<tr>
<td>SearchAnalytics mode</td>
<td><code>SPARK_ENABLED=1 SEARCH_ENABLED=1</code></td>
<td>SearchAnalytics mode requires testing in your environment before it is used in production clusters. In <code>dse.yaml</code>, <code>cql_solr_query_paging: driver</code> (page 326) is required.</td>
</tr>
</tbody>
</table>

**Tarball and Installer-No Services 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 <code>dse.yaml</code>, <code>cql_solr_query_paging: driver</code> (page 326) 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
```

<table>
<thead>
<tr>
<th>Address</th>
<th>DC</th>
<th>State</th>
<th>Load</th>
<th>Rack</th>
<th>Workload</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>Status</td>
<td>State</td>
<td>Load</td>
<td>Owns</td>
<td>Workload</td>
<td>Token</td>
</tr>
<tr>
<td>Health [0,1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Role</th>
<th>Rack</th>
<th>Keyspace</th>
<th>Status</th>
<th>CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.200.175.149</td>
<td>Analytics</td>
<td>rack1</td>
<td>Analytics(SM)</td>
<td>normal</td>
<td>185 KiB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.90</td>
</tr>
<tr>
<td>10.200.175.148</td>
<td>Analytics</td>
<td>rack1</td>
<td>Analytics(SW)</td>
<td>normal</td>
<td>194.5 KiB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.90</td>
</tr>
</tbody>
</table>

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 and Installer-Services installations: `$ dse spark`

Tarball and Installer-No Services installations: `installation_location/bin/dse spark`

You can use Cassandra specific properties (page 430) to start Spark. Spark binds to the `listen_address` that is specified in `cassandra.yaml`.

### cassandra.yaml

The location of the `cassandra.yaml` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Type of Installation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td><code>/etc/dse/cassandra/cassandra.yaml</code></td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td><code>/etc/dse/cassandra/cassandra.yaml</code></td>
</tr>
<tr>
<td>Tarball installations</td>
<td><code>installation_location/resources/cassandra/conf/cassandra.yaml</code></td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td><code>installation_location/resources/cassandra/conf/cassandra.yaml</code></td>
</tr>
</tbody>
</table>

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 and Installer-Services installations: `$ dse spark`
- Tarball and Installer-No Services 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 and Installer-Services installations is:
Using DataStax Enterprise advanced functionality

$ 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:

$ 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 493).

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:

$ dse spark-submit --deploy-mode cluster --supervise --class com.datastax.HttpSparkStream target/HttpSparkStream.jar -d $NUM_SPARK_NODES

Unlike the standard behavior for the Spark `status` and `kill` options, in DSE deployments these options do not require the Spark Master IP address:

- `dse spark-submit -kill driver_id [--master master_ip_address]`
- `dse spark-submit -status driver_id [--master master_ip_address]`

For example, to kill a driver of a Spark application running in the DSE cluster:

$ dse spark-submit --kill driver-20180726160353-0019

To get the status of a Spark application running in the DSE cluster:

$ dse spark-submit --status driver-20180726160353-0019

**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 1082) commands.
**Note:** DataStax recommends using the environment variables instead of passing user credentials on the command line.

Authentication credentials can be provided in several ways, see Connecting to authentication enabled clusters.

### dse.yaml

The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/dse/conf/</td>
</tr>
<tr>
<td></td>
<td>dse.yaml</td>
</tr>
</tbody>
</table>

### 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. 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:

```bash
$ 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:

   ```bash
   $ dse client-tool configuration export dse-config.jar
   ```

   b. Copy the exported JAR to the client nodes.

   ```bash
   $ scp dse-config.jar user@clientnode1.example.com:
   ```

   c. On the client node:

   ```bash
   $ dse client-tool configuration import dse-config.jar
   ```

2. Run the Spark command against the remote node.

   ```bash
   $ 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.
Using DataStax Enterprise advanced functionality

$ 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 461).

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
Spark session available as 'spark'.
Welcome to

   ____              __
  / __/__  ___ _____/ /__
 / _ \ _ \// _ `/ __/  '_/
/_/\_\_\_\_\_,_/_/ /_/
     \/       \\  /_/_/\_
     version 2.0.0.1

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_91)
Type in expressions to have them evaluated.
Type :help for more information.

scala>

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:
Using DataStax Enterprise advanced functionality

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.

val newSpark = spark.newSession()

Building a Spark session using the Builder API

The Builder API allows you to create a Spark session manually.

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.

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.

```java
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 386](#)). 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`.

```scala
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.

   ```sql
   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 379](#)).

2. Load data into the `words` table.
Using DataStax Enterprise advanced functionality

```
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

================================================
Keyspace: test
================================================
Table: words
================================================
- word   : String              (partition key column)
- count  : Int
```

```
scala> :showSchema test
```

```
```

```
```
5. Get information about only the test keyspace.

```
showSchema test
```

```
Keystore: test
Table: words
- word : String (partition key column)
- count : Int
```

6. Get information about the words table.

```
showSchema test words
```

```
Keystore: 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.

```
val rdd = sc.cassandraTable("test", "words")
```

```
rdd:
  CassandraRow] = CassandraRDD[0] at RDD at CassandraRDD.scala:47
```

The RDD is returned in the rdd value. To read the table, use this command.

```
rdd.toArray.foreach(println)
```

```
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.
Using DataStax Enterprise advanced functionality

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
   firstRow: com.datastax.spark.connector.CassandraRow = CassandraRow(word: foo, count: 10)
   ```

2. Get the column names.

   ```scala
   rdd.columnNames
   res3: com.datastax.spark.connector.ColumnSelector = AllColumns
   ```

3. Use a generic get to query the table by passing the return type directly.

   ```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")
   ```
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.

   ```
   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.

   ```
   $ bin/dse spark
   ```

   The Welcome to Spark output and prompt appears.

3. Define a CassandraRDD[CassandraRow] to access the collection set.

   ```
   val row = sc.cassandraTable("test", "users").toArray.apply(0)
   ```

4. Query the collection set from Spark.

   ```
   row.getList[String]("emails")
   res2: Vector[String] = Vector(s@email.com, someone@email.com)
   row.getList[List[String]]("emails")
   res3: List[String] = List(s@email.com, someone@email.com)
   row.getList[Seq[String]]("emails")
   res4: Seq[String] = List(s@email.com, someone@email.com)
   row.getList[IndexedSeq[String]]("emails")
   ```
Using DataStax Enterprise advanced functionality

```scala
res5: IndexedSeq[String] = Vector(s@email.com, someone@email.com)

row.get[Set[String]]("emails")

res6: Set[String] = Set(s@email.com, someone@email.com)

row.get[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

```

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

scala> 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 31: Recommended 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>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 32: 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")
```

```scala
table:
```
Using DataStax Enterprise advanced functionality

```scala
total: Double = 30.0
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.
   - Index the color column.
   - Insert some data into the table

2. Run the test_cars.cql file using cqlsh or DataStax Studio. For example using cqlsh:
   ```
   $ cqlsh -f test_cars.cql
   ```

3. Filter the rows using Spark:
   ```scala
   sc.cassandraTable("test", "cars").select("id", "model").where("color = ?", "black").toArray.foreach(println)
   ``
   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)
   sc.cassandraTable("test", "cars").select("id", "model").where("color = ?", "silver").toArray.foreach(println)
**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.

```
$ 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.

```scala
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()
    .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:

```scala
import com.datastax.spark.connector.DseConfiguration._
new SparkConf().enableDseSupport()
```

In Java:

```java
SparkConf rawConf = new SparkConf();
```
Using DataStax Enterprise advanced functionality

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 446) to manipulate data within Spark.

Saving a collection of tuples
The following example shows how to save a collection of tuples to the database.
scala> val collection = sc.parallelize(Seq(("cat", 30), ("fox", 40)))
collection: org.apache.spark.rdd.RDD[(String, Int)] =
ParallelCollectionRDD[6] at parallelize at console:22
scala> collection.saveToCassandra("test", "words", SomeColumns("word",
"count"))
scala>

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
fox |
40
(4 rows)

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)
defined class WordCount
scala> val collection = sc.parallelize(Seq(WordCount("dog", 50),
WordCount("cow", 60)))

Page 396

DSE 5.1 Developer Guide (Previous version)


collection: org.apache.spark.rdd.RDD[WordCount] = 
ParallelCollectionRDD[0] at parallelize at console:24

scala> collection.saveToCassandra("test", "words", SomeColumns("word", 
"count"))

scala>

In cqlsh, query the words table to select all the contents.

```
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:

```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 + ":" + _count
}
object WordCount {
  implicit object Mapper extends JavaBeanColumnMapper[WordCount]
}

Enter CTRL D to exit paste mode. The output is:

// Exiting paste mode, now interpreting.
import com.datastax.spark.connector.mapper.JavaBeanColumnMapper
defined class WordCount
defined module WordCount

Query the WordCount object.

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> :paste
// 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.

// Exiting paste mode, now interpreting.
import com.datastax.spark.connector.mapper.DefaultColumnMapper
defined class WordCount
defined module WordCount

Continue entering these commands:

scala> sc.cassandraTable[WordCount]("test", "words").toArray
res21: Array[WordCount] = Array(WordCount(cow,60), WordCount(bar,20),
WordCount(foo,10), WordCount(cat,30), WordCount(fox,40),
WordCount(dog,50))

scala>
sc.parallelize(Seq(WordCount("bar",20),WordCount("foo",40))).saveToCassandra("test",
"words", SomeColumns("word", "count"))

scala>

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 33: 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>
</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.

   ```
   $ dse hadoop fs -mkdir webhdfs://hadoopNode.example.com:50070/user/guest/data
   $ dse 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`.

<table>
<thead>
<tr>
<th>CQL Type</th>
<th>Scala Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeuuid</td>
<td>java.util.UUID</td>
</tr>
<tr>
<td>uuid</td>
<td>java.util.UUID</td>
</tr>
<tr>
<td>varint</td>
<td>BigInt, java.math.BigInteger</td>
</tr>
<tr>
<td>nullable values</td>
<td>Option</td>
</tr>
</tbody>
</table>
4. Ensure that we can access the HDFS data by interacting with the data using dse hadoop fs.

The following command counts the number of lines of HDFS data.

```bash
$ dse 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:
Using DataStax Enterprise advanced functionality

```scala
case class Tmax(location: String, year: Int, month: Int, day: Int, tmax: Double, datestring: String)
```

Read the data into an RDD.

```scala
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 `Tmax` case class.

```scala
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.

```scala
tmax_c10.count
res11: Long = 3606
```

Save the case class instances to the database.

```scala
tmax_c10.saveToCassandra("spark_ex2", "sftmax")
```

8. Verify the records match by counting the rows using CQL.

```scala
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 `Tmin` case class to store the minimum temperature sensor data.

```scala
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 `PairRDDs`, with the first element in the pair being the join key.

```scala
val tmin_pair = tmin_c10.map(x=>(x.datestring,x))
val tmax_pair = tmax_c10.map(x=>(x.datestring,x))
```

Create a `THiLoDelta` case class to store the difference between the maximum and minimum temperatures.

```scala
```

Page 402
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 284)* of any Spark node in a browser followed by port number 7080. Starting in DSE 5.1, all Spark nodes within an Analytics datacenter will redirect to the current Spark Master.
- To change the port, modify the `spark-env.sh` configuration file *(page 410)*. If you change the port number, set it to the same port number on every node in the datacenter.

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.

```
spark-daemon-defaults.conf
The default location of the `spark-daemon-defaults.conf` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Type of Installation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td><code>/etc/dse/spark/spark-daemon-defaults.conf</code></td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td><code>installation_location/resources/spark/conf/spark-daemon-defaults.conf</code></td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td></td>
</tr>
</tbody>
</table>
```

DSE SSL encryption and authentication only apply to the Spark Master and Worker UIs, not the Spark Driver UI. To use encryption and authentication with the Driver UI, refer to the Spark security documentation.

Authorization is not supported in the Spark web UI. Any authenticated user can monitor and control any Spark applications within the UI.

**Spark Master at spark://10.200.175.149:7077**

<table>
<thead>
<tr>
<th>Worker Id</th>
<th>Address</th>
<th>State</th>
<th>Cores</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>worker-01611130193565-10.200.175.149-52114</td>
<td>10.200.175.149:52114</td>
<td>ALIVE</td>
<td>1 (1 Used)</td>
<td>4.1 GB (1024.0 MB Used)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application ID</th>
<th>Name</th>
<th>Cores</th>
<th>Memory per Node</th>
<th>Submitted Time</th>
<th>User</th>
<th>State</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>app-20161208184429-0010</td>
<td>Spark shell</td>
<td>1</td>
<td>1024.0 MB</td>
<td>2015/12/08 18:44:29</td>
<td>automation</td>
<td>RUNNING</td>
<td>15 s</td>
</tr>
</tbody>
</table>

See the Spark documentation for information on using the Spark web UI.

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.
spark-env.sh
The default location of the spark-env.sh file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/spark/spark-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/spark/conf/spark-</td>
</tr>
<tr>
<td></td>
<td>env.sh</td>
</tr>
</tbody>
</table>

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.ui.confidentialKeys filter is configured as a comma separated list of regular expressions that by default includes all properties that contain the string "token" or "password". To modify the filter, edit the spark.ui.confidentialKeys property in spark-defaults.conf in the Spark configuration directory.

spark-defaults.conf
The default location of the spark-defaults.conf file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/spark/spark-defaults.conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>spark/conf/spark-defaults.conf</td>
</tr>
</tbody>
</table>

Getting started with the Spark Cassandra Connector Java API

The Spark Cassandra Connector Java API 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 18) 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
libraryDependencies ++= Seq(
  "com.datastax.dse" % "dse-spark-dependencies" % dseVersion %
  "provided" excludeAll (ExclusionRule("com.datastax.dse", "dse-java-driver-core"),
       ExclusionRule("org.apache.solr", "solr-solrj")
  ),
  "com.datastax.dse" % "dse-java-driver-core" % "1.2.3",
  "org.apache.solr" % "solr-solrj" % "6.0.1"
)```

libraryDependencies ++= Seq(
  "com.datastax.dse" % "dse-spark-dependencies" % dseVersion %
  "provided" excludeAll (ExclusionRule("com.datastax.dse", "dse-java-driver-core"),
       ExclusionRule("org.apache.solr", "solr-solrj")
  ),
  "com.datastax.dse" % "dse-java-driver-core" % "1.2.3",
  "org.apache.solr" % "solr-solrj" % "6.0.1"
)
Using DataStax Enterprise advanced functionality

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>
  <exclusions>
    <exclusion>
      <groupId>com.datastax.dse</groupId>
      <artifactId>dse-java-driver-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.apache.solr</groupId>
      <artifactId>solr-solrj</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

Then add the DataStax repository:

```xml
<repositories>
  <repository>
    <id>DataStax-Repo</id>
  </repository>
</repositories>
```

For example project templates, see 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.
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.

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.

SparkConf conf = new SparkConf()
  .setAppName( "My application");SparkContext sc = new SparkContext(conf);

dse-spark-version.jar
The default location of the dse-spark-version.jar file depends on the type of installation:

<table>
<thead>
<tr>
<th>Type of Installation</th>
<th>Default Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/usr/share/dse/dse-spark-version.jar</td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td>/usr/share/dse/dse-spark-version.jar</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/lib/dse-spark-version.jar</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>installation_location/lib/dse-spark-version.jar</td>
</tr>
</tbody>
</table>

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<>()` instance and calling the `mapColumnTo()` method and passing in the type. Then call the `select()` method to set the column name.

```java
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.

```java
JavaRDD<Person> personRdd = CassandraJavaUtil.javaFunctions(sc) .cassandraTable("my_keyspace", "my_table", mapRowTo(Person.class));
```

CassandraJavaPairRDD<>() 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.

```java
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:

```java
CassandraJavaPairRDD<Integer, Person> idPersonRdd = CassandraJavaUtil.javaFunctions(sc) .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)
```
Using DSE Spark with third party tools and integrations

The `dse exec` command sets the required environment variables required to run third-party tools that integrate with Spark.

```
$dse exec command
```

**Note:** The `dse exec` command was introduced in DSE 5.1.6.

**Jupyter integration**

Download and install Jupyter notebook on a DSE node.

To launch Jupyter notebook:

```
$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 [https://github.com/cloudera/livy](https://github.com/cloudera/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:

```
livy.spark.master = dse:///
livy.repl.enable-hive-context = true
```

To launch Livy:

```
$dse exec livy-server
```

**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.
Using DataStax Enterprise advanced functionality

RStudio integration

Download and install R (page 454) 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 the Spark SQL Thriftserver (page 447) and JDBC (page 450) connections rather than SparkR.

## Configuring Spark

Configuring Spark for DataStax Enterprise includes:

### Configuring Spark nodes

Modify the settings for Spark nodes security, performance, and logging.

**hive-site.xml**

For use with Spark, the default location of the `hive-site.xml` file is:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/spark/hive-site.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/spark/conf/hive-</td>
</tr>
<tr>
<td></td>
<td>site.xml</td>
</tr>
</tbody>
</table>

To manage Spark performance and operations:

- Set environment variables (page 410)
- Protect Spark directories (page 411)
- Grant access to default Spark directories (page 412)
- Secure Spark nodes (page 412)
- Configure Spark memory and cores (page 414)
- Configure Spark logging options (page 423)

### 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 336) 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.

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
```

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 \
-DSpark.worker.cleanup.enabled=true \ 
-DSpark.worker.cleanup.interval=1800 \ 
-DSpark.worker.cleanup.appDataTtl=604800"
```

dse.yaml
The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/dse/conf/dse.yaml</td>
</tr>
</tbody>
</table>

spark-env.sh
The default location of the spark-env.sh file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/spark/spark-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/spark/conf/spark-env.sh</td>
</tr>
</tbody>
</table>

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.
Using DataStax Enterprise advanced functionality

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_LOCAL_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 336). 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.
Note: DSE 5.1.4 (page 93), DSE 5.1.5 (page 89), and 5.1.6 (page 75) users should refer to the release notes for information on using Spark SQL applications and DSE authentication.

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
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:

| Installer-Services and Package installations | /etc/dse/spark/hive-site.xml |
| Installer-No Services and 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.

Using authorization with Spark
There are two kinds of authorization permissions which apply to Spark. Work pool permissions control the ability to submit a Spark application to DSE. Submission permissions control the ability to manage a particular application. All the following instructions assume you are issuing the CQL commands as a database superuser.

Use GRANT CREATE ON ANY WORKPOOL TO role to grant permission to submit a Spark application to any Analytics datacenter.

Use GRANT CREATE ON WORKPOOL datacenter_name TO role to grant permission to submit a Spark application to a particular Analytics datacenter.

There are similar revoke commands:
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REVOKE CREATE ON ANY WORKPOOL FROM role</td>
<td>Grant permission to manage and remove an application for a specific role.</td>
</tr>
<tr>
<td>REVOKE CREATE ON WORKPOOL datacenter_name FROM role</td>
<td>Grant permission to create an application in a specific datacenter for a specific role.</td>
</tr>
</tbody>
</table>

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 **GRANT MODIFY ON ANY SUBMISSION TO role** to grant permission to manage any submission in any work pool to the specified role.

Use **GRANT MODIFY ON ANY SUBMISSION IN WORKPOOL datacenter_name TO role** to grant permission to manage any submission in a specified datacenter.

Use **GRANT MODIFY ON SUBMISSION id IN WORKPOOL datacenter_name TO role** to grant permission to manage a submission identified by the provided id in a given datacenter.

There are similar revoke commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REVOKE MODIFY ON ANY SUBMISSION FROM role</td>
<td>Revoke permission to manage an application for a specific role.</td>
</tr>
<tr>
<td>REVOKE MODIFY ON ANY SUBMISSION IN WORKPOOL datacenter_name FROM role</td>
<td>Revoke permission to create an application in a specific datacenter for a specific role.</td>
</tr>
<tr>
<td>REVOKE MODIFY ON SUBMISSION id IN WORKPOOL datacenter_name FROM role</td>
<td>Revoke permission to manage a submission identified by the provided id in a given datacenter.</td>
</tr>
</tbody>
</table>

In order to issue these commands as a regular database user, the user needs to have permission to use the DSE resource manager RPC:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT ALL ON REMOTE OBJECT DseResourceManager TO role</td>
<td>Grant permission to use the DSE resource manager RPC.</td>
</tr>
</tbody>
</table>

Each DSE Analytics user needs to have permission to use the client tools RPC:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT ALL ON REMOTE OBJECT DseClientTool TO role</td>
<td>Grant permission to use the client tools RPC.</td>
</tr>
</tbody>
</table>

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 **SPARK_WORKER_MEMORY** option configures the total amount of memory that you can assign to all executors that are run by a single Spark Worker on the particular node.

**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 1.6.2 Available Properties", or the **--executor-memory** `mem` argument to the **dse spark** command *(page 430)*.

### 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 2.0.2 Available Properties. For example:

```bash
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 **SPARK_WORKER_CORES** option configures the number of cores offered by Spark Worker for executors. 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** command *(page 430)*.

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 **initial_spark_worker_resources** parameter in the **dse.yaml** file specifies the fraction of system resources that are made available to the Spark Worker. The available resources are calculated in the following way:

- Spark Worker memory = **initial_spark_worker_resources** * (total system memory - memory assigned to DSE)
- Spark Worker cores = **initial_spark_worker_resources** * total system cores

The lowest values you can assign to Spark Worker 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 range of the **initial_spark_worker_resources** value is 0.01 to 1. If the range is not specified, the default value 0.7 is used.

This mechanism is used by default to set the Spark Worker memory and cores. To override the default, uncomment and edit one or both **SPARK_WORKER_MEMORY** and **SPARK_WORKER_CORES** options in the **spark-env.sh** file.
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:

```
# 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.

cassandra-rackdc.properties

The location of the cassandra-rackdc.properties file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/cassandra/cassandra-rackdc.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/cassandra/conf/cassandra-rackdc.properties</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td></td>
</tr>
</tbody>
</table>

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 (page 430), or in spark-defaults.conf.

spark-defaults.conf

The default location of the spark-defaults.conf file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/spark/spark-defaults.conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/spark/conf/spark-defaults.conf</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td></td>
</tr>
</tbody>
</table>

Enabling 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 disabled. To enable it, set the spark.dse.continuous_paging_enabled setting to true when starting the Spark SQL shell or in spark-defaults.conf. For example:
Using DataStax Enterprise advanced functionality

$ dse spark-sql --conf spark.dse.continuous_paging_enabled=true

**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:

   ```
   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 and Installer-Services:** `/usr/share/dse/spark/lib`
- **Tarball installations and Installer-No Services:** `/var/lib/spark`

To add the Graphite JARs to Spark in a package installation, copy them to the Spark lib directory:

```
$ cp metrics-graphite-3.1.2.jar /usr/share/dse/spark/lib/ &&
cp metrics-json-3.1.2.jar /usr/share/dse/spark/lib/
```

**Spark server configuration**

The `spark-daemon-defaults.conf` file configures DSE Spark Masters and Workers.
Using DataStax Enterprise advanced functionality

Table 34: 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.

```plaintext
spark.executor.logs.rolling.maxRetainedFiles 3
spark.executor.logs.rolling.strategy size
spark.executor.logs.rolling.maxSize 50000
```

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 1022).

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 383) 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:
Using DataStax Enterprise advanced functionality

$ 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 376) are flagged as Workload: Analytics(SM).

$ dsetool ring

<table>
<thead>
<tr>
<th>Address</th>
<th>DC</th>
<th>Rack</th>
<th>Workload</th>
<th>Health [0,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.200.181.62</td>
<td>Analytics</td>
<td>rack1</td>
<td>Analytics(SM)</td>
<td>?</td>
</tr>
<tr>
<td>-9223372036854775808</td>
<td>Up</td>
<td>Normal</td>
<td>111.91 KiB</td>
<td>0.10</td>
</tr>
</tbody>
</table>

• Query the dseleases.leases table to list all the masters from each data center with Analytics nodes:

select * from dseleases.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/5.1</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/5.1</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/5.1</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 dseleases keyspace

If the dseleases 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 dseleases keyspace for the new DSE Analytics datacenter. If DataStax Enterprise or Spark security options are enabled on
Using DataStax Enterprise advanced functionality

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 an 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 371).

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 325) 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>new_holder</td>
<td>old_holder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-------</td>
<td>---------------</td>
<td>----</td>
</tr>
</tbody>
</table>
+-------------------+-------+---------------+----|
Leader/master/5.1  | dc1   | 10.200.180.44 | 2018-05-17 00:45:02.971000+0000 |
|                   |       | 10.200.180.44 |    |
Leader/master/5.1  | dc1   | 10.200.180.49 | 2018-05-17 02:37:07.381000+0000 |
|                   |       | 10.200.180.49 |    |
+-------------------+-------+---------------+----|

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:

```sql
select * from dse_perf.leases;
```

| name              | dc   | monitor
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>acquire_average_latency_ms</td>
<td>acquire_latency99ms</td>
<td></td>
</tr>
<tr>
<td>acquire_max_latency_ms</td>
<td>acquire_rate15</td>
<td></td>
</tr>
<tr>
<td>disable_average_latency_ms</td>
<td>disable_latency99ms</td>
<td></td>
</tr>
<tr>
<td>disable_max_latency_ms</td>
<td>disable_rate15</td>
<td></td>
</tr>
<tr>
<td>renew_average_latency_ms</td>
<td>renew_latency99ms</td>
<td></td>
</tr>
<tr>
<td>renew_max_latency_ms</td>
<td>renew_rate15</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:

```xml
<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>Graph Status</th>
<th>State</th>
<th>Load</th>
<th>Rack</th>
<th>Workload owns</th>
<th>Health [0,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Token</td>
<td>DC</td>
<td>Status</td>
<td>State</td>
<td>Load</td>
<td>Rack</td>
<td>Workload owns</td>
<td>Health [0,1]</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 371).

Use cqlsh commands to verify the replication factor of the analytics keyspaces

1. Describe the dse_leases keyspace:

```cql
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:

```cql
ALTER KEYSPACE dse_leases WITH replication =
{'class': 'NetworkTopologyStrategy', 'Analytics1': '3',
'Analytics2': '3'};
```

3. Run nodetool repair.

dse.yaml
The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tarball installations</th>
<th>installation_location/ resources/dse/conf/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-No Services installations</td>
<td></td>
</tr>
</tbody>
</table>

logback.xml
The location of the logback.xml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/cassandra/logback.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
</tbody>
</table>
Configuring Spark logging options

You can configure Spark logging options for the Spark logs.

Log directories

logback.xml

The location of the logback.xml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/dse/cassandra/logback.xml</td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/cassandra/logback.xml</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/cassandra/conf/logback.xml</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>installation_location/resources/cassandra/conf/logback.xml</td>
</tr>
</tbody>
</table>

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 410) as \$HOME/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>
</table>

DSE 5.1 Developer Guide (Previous version)
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Spark Master</th>
<th>logback.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark Worker</td>
<td>logback-spark-server.xml</td>
</tr>
<tr>
<td>Spark Driver (Spark Shell, Spark applications)</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.

```text
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 and Installer-Services: /etc/dse/spark/
- Tarball installations and Installer-No Services: 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:** Authentication credentials can be provided in several ways, see Connecting to authentication enabled clusters.

**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`.

dse.yaml
The location of the `dse.yaml` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td><code>/etc/dse/dse.yaml</code></td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td><code>/etc/dse/dse.yaml</code></td>
</tr>
<tr>
<td>Tarball installations</td>
<td><code>installation_location/ resources/dse/conf/dse.yaml</code></td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td><code>installation_location/ resources/dse/conf/dse.yaml</code></td>
</tr>
</tbody>
</table>

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
Using DataStax Enterprise advanced functionality

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.

```
spark_process_runner:
  # Allowed options are: default, run_as
  runner_type: run_as

  run_as_runner_options:
    user_slots:
    - slot1
    - slot2
```

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.

   $ 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.

   $ 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:

   $ 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.

   | 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.

   # 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
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:

   ```
   $ dse hadoop fs -mkdir /spark
   $ dse hadoop fs -mkdir /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
   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 http://node hostname:18080.

**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.

### Enabling Spark apps in cluster mode when authentication is enabled

You must enable Spark applications in cluster mode when JAR files are on the Cassandra File System (CFS) and authentication is enabled. When the application is submitted in cluster mode and the JAR files are on CFS, the Spark Worker process is responsible for obtaining the required JAR file. When authentication is required, the Spark Worker process requires the authentication credentials to CFS. The Spark Worker will start executors for unrelated Spark jobs, so giving the Spark Worker process credentials enables all future Spark jobs to pull JAR files from CFS for their dependencies. Credentials that are granted to the Spark Worker must be considered "shared" among all submitted applications, regardless of the submitting user. Shared credentials do not apply to accessing CFS from the application code.

**spark-env.sh**

The default location of the `spark-env.sh` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/spark/spark-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/spark/spark-env.sh</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/ resources/spark/conf/spark-env.sh</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>installation_location/ resources/spark/conf/spark-env.sh</td>
</tr>
</tbody>
</table>

1. To enable Spark applications in cluster mode when JAR files are on CFS and authentication is enabled, do one of the following:

   - Add this statement to the spark-env.sh on every DataStax Enterprise node:
Before you start the DataStax Enterprise server process, set the 
**SPARK_WORKER_OPTS** environment variable in a way that guarantees 
visibility to DataStax Enterprise server processes.

This environment variable does not need to be passed to applications that are submitted with the **dse spark** or **dse spark-submit** commands.

2. Follow these best practices:

   - Create a unique user with privileges only on CFS (access to related CFS keyspace), and then use the unique user credentials for the Spark Worker authentication. This best practice limits the amount of protected information in the database that is accessible through user Spark Jobs without explicit permission.
   - Create a distinct CFS directory and limit the directory access privileges to read only.

### 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:

- **--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**.

  **spark-defaults.conf**
The default location of the `spark-defaults.conf` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Type of Installation</th>
<th>File Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td><code>/etc/dse/spark/spark-defaults.conf</code></td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td><code>installation_location/resources/spark/conf/spark-defaults.conf</code></td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td></td>
</tr>
</tbody>
</table>

- `--executor-memory memory`

  How much memory to allocate on each machine for the application. You can provide the memory argument in JVM format using either the k, m, or g suffix.

- `--total-executor-cores cores`

  The total number of cores the application uses

- `--conf name=value`

  An arbitrary Spark option to the Spark configuration prefixed by `spark`.

- `--help`

  Shows a help message that displays all options except DataStax Enterprise Spark shell options.

- `--jars <additional-jars>`

  A comma-separated list of paths to additional JAR files.

- `--verbose`

  Displays which arguments are recognized as Spark configuration options and which arguments are forwarded to the Spark shell.

Spark shell application arguments:

- `--class class_name`

  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 430)` and these additional `spark-submit submission_arguments`:

- `--class class_name`
Using DataStax Enterprise advanced functionality

The full name of the application main class.

- **--name name**
  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 `rpc_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 383) using the **--master master** 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_ONE. Consistency level to use when reading.

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 = 5. 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.

cassandra.yaml
The location of the `cassandra.yaml` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Type of Installation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/dse/cassandra/</td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td>cassandra.yaml</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/resources/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

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`. 
Using DataStax Enterprise advanced functionality

- Make sure all keyspaces in the DC1 datacenter use NetworkTopologyStrategy. If necessary, alter the keyspace.

```sql
ALTER KEYSPACE mykeyspace
WITH REPLICAION = { 'class' = 'NetworkTopologyStrategy', 'DC1' : 3 };
```

- Add nodes to a new datacenter named DC2, then enable Analytics on those nodes (page 1090).

- Configure the dse_leases and spark_system keyspaces to replicate to both DC1 and DC2. For example:

```sql
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.

```bash
dse spark-submit --master "dse://?connection.local_dc=DC2"
--class com.datastax.dse.demo.loss.Spark10DayLoss --conf "spark.cassandra.connection.local_dc=DC1" portfoio.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 10.0.0.2.

To create an RDD from the transactional cluster's data:

```java
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","10.0.0.2"))

  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.

- `SPARK_WORKER_MEMORY` in `spark-env.sh` is the maximum amount of memory to give all executors for all applications running on a particular node.
- `initial_spark_worker_resources` in `dse.yaml` is used to automatically calculate `SPARK_WORKER_MEMORY` if it is commented out (as it is by default). It uses the following formula:

  ```
  initial_spark_worker_resources * (total system memory - memory assigned to DataStax Enterprise)
  ```
Using DataStax Enterprise advanced functionality

- `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 `SPARK_WORKER_MEMORY`. 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`.

**spark-env.sh**
The default location of the `spark-env.sh` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>Installer-Services installations</th>
<th>/etc/dse/spark/spark-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>Installer-No Services installations</td>
<td>installation_location/resources/spark/conf/spark-env.sh</td>
</tr>
</tbody>
</table>

**dse.yaml**
The location of the `dse.yaml` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>Installer-Services installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>Installer-No Services installations</td>
<td>installation_location/resources/dse/conf/dse.yaml</td>
</tr>
</tbody>
</table>

**cassandra-env.sh**
The location of the `cassandra-env.sh` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>Installer-Services installations</th>
<th>/etc/dse/cassandra/cassandra-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>Installer-No Services installations</td>
<td>installation_location/resources/cassandra/conf/cassandra-env.sh</td>
</tr>
</tbody>
</table>

**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:

- `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.
Using DataStax Enterprise advanced functionality

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 `org.apache.spark.streaming.StreamingContext` 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 272).

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 = ....
   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.

```java
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.

```scala
val lines = ssc.socketTextStream(server IP address, server port number)
```

5. Count the words in each batch and save the data to the table.

```scala
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.

```scala
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.

```sql
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:

```bash
$ nc -lk 9999
one two two three three three four four four four someword
```

In a different terminal start a Spark shell.

```bash
$ 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

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 472). See the Spark Streaming Programming Guide for more information, API documentation, and examples on Spark Streaming.

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`s 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.

DataFrames are Spark Dataset 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 447) to handle the underlying JDBC connections.

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.

```scala
val results = spark.sql("SELECT * from my_keyspace_name.my_table")
```

3. Use the returned data.

```scala
results.show()
```

| +--------------------+-----------+ |
| | id            | description |
| +-------------------+-----------+ |
| de2d0de1-4d70-11e... | thing     |
| db7e4191-4d70-11e... | another   |
| d576ad50-4d70-11e... | yet another |

### 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.

**dse-spark-version.jar**

The default location of the `dse-spark-version.jar` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/usr/share/dse/dse-spark-version.jar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>installation_location/lib/dse-spark-version.jar</td>
</tr>
</tbody>
</table>

The Spark session object is used to connect to DataStax Enterprise.

Create the Spark session instance using the `builder interface`:

```scala
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.

```scala
DataFrame employees = spark.sql("SELECT * FROM company.employees"); 
employees.registerTempTable("employees");
```
Using DataStax Enterprise advanced functionality

```
DataFrame managers = spark.sql("SELECT name FROM employees WHERE role = 'Manager' ");
```

The returned DataFrame object supports the standard Spark operations.

```
employees.collect();
```

**Querying DSE Graph vertices and edges with Spark SQL**

Spark SQL can query DSE Graph vertex and edge tables. The com.datastax.bdp.graph.spark.sql.vertex and com.datastax.bdp.graph.spark.sql.edge data sources are used to specify vertex and edge tables in Spark SQL.

```
spark-sql> CREATE DATABASE graph_example;
spark-sql> USE graph_example;
spark-sql> CREATE TABLE vertices USING com.datastax.bdp.graph.spark.sql.vertex OPTIONS (graph 'example');
spark-sql> CREATE TABLE edges USING com.datastax.bdp.graph.spark.sql.edge OPTIONS (graph 'example');
```

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 1090).*

1. Start the Spark SQL shell.

```
$ dse spark-sql
```

2. Register the vertex and edge tables for your graph using CREATE TABLE.

```
CREATE DATABASE graph_gods;
USE graph_gods;
CREATE TABLE vertices USING com.datastax.bdp.graph.spark.sql.vertex OPTIONS (graph 'gods');
CREATE TABLE edges USING com.datastax.bdp.graph.spark.sql.edge OPTIONS (graph 'gods');
```

3. Query the vertices and edges using SELECT statements.

```
SELECT * FROM vertices where name = 'Zeus';
```

4. Join the vertices and edges in a query.
Vertices are identified by `id` columns. Edges 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 edges.dst FROM vertices JOIN edges ON vertices.id = 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 450)/ODBC (page 451) driver and the Spark SQL Thrift Server (page 447).

**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 [(columns)]
```
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`
- `INSERT`
- `INTO`
- `IS`
- `JOIN`
- `LEFT`
- `LIMIT`
- `MAX`
- `MIN`
- `NOT`
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 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 `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' =
```
Using DataStax Enterprise advanced functionality

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.

```sql
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.

```scala
scala> 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.

```python
dse pyspark

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 DataStax Enterprise advanced functionality

Using the DSE Spark console, the following Scala example shows how to create a DataFrame object from one table and save it to another.

```scala
$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`:

```scala
table1.write.format("org.apache.spark.sql.cassandra")
  .options(Map("table" -> "othertable", "keyspace" -> "test"))
  .save()
```

### Using the Spark SQL Thrift server

The Spark SQL Thrift server uses JDBC and ODBC interfaces for client connections to the database.

When reading or writing large amounts of data, DataStax recommends using DataFrames to enable the use of the Spark Cassandra Connector and the benefits of the tuning parameters that come with it.

**hive-site.xml**

For use with Spark, the default location of the `hive-site.xml` file is:

<table>
<thead>
<tr>
<th>Package installations</th>
<th><code>/etc/dse/spark/hive-site.xml</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td><code>installation_location/resources/spark/conf/hive-site.xml</code></td>
</tr>
</tbody>
</table>

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>
</property>
```
Using DataStax Enterprise advanced functionality

Ensure that you use the `hive-site.xml` file in the Spark directory:

| Installer-Services and Package installations | /etc/dse/spark/hive-site.xml |
| Installer-No Services and Tarball installations | `installation_location/resources/spark/conf/hive-site.xml` |

2. Start DataStax Enterprise with Spark enabled as a service (page 1090) or in a standalone (page 1093) 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. You can configure the port and bind address in `resources/spark/conf/spark-env.sh`: HIVE_SERVER2_THRIFT_PORT, HIVE_SERVER2_THRIFT_BIND_HOST.

   ```
   $ dse spark-sql-thriftserver start --hiveconf hive.server2.thrift.port=10001
   ```

   You can specify general Spark configuration settings by using the `--conf` option.

   ```
   $ dse spark-sql-thrift-server 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 \texttt{dse spark-sql-thriftserver stop} command.

\texttt{$ dse \ spark-sql-thriftserver \ stop$}

\textbf{What's next:}

You can now connect your application by using JDBC to the server at the URI: \texttt{jdbc:hive2://hostname:port number}, using ODBC, or use \texttt{dse spark-beeline (page 453)}.

\textbf{Enabling SSL for the Spark SQL Thrift Server}

Communication between the JDBC driver and Spark SQL Thrift Server can be encrypted using SSL.

The following instructions give an example of how to set up SSL with a self-signed keystore and truststore.

\texttt{hive-site.xml}

For use with Spark, the default location of the \texttt{hive-site.xml} file is:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>Installer-Services installations</th>
<th>/etc/dse/spark/hive-site.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>Installer-No Services installations</td>
<td>\texttt{installation_location/resources/spark/conf/hive-site.xml}</td>
</tr>
</tbody>
</table>

1. Create the keystore and truststore using the \texttt{keytool} command.

2. Add the required settings to enable SSL to the \texttt{hive-site.xml} configuration file.

\begin{verbatim}
<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>
<property>
    <name>hive.server2.keystore.password</name>
    <value>keystore password</value>
</property>
\end{verbatim}
Using DataStax Enterprise advanced functionality

3. Start or restart the Spark SQL Thrift server.
   
   **Note:** Changes in the `hive-site.xml` configuration file only require a restart of Spark SQL Thriftserver service, not DSE.

   
   ```
   $ dse spark-sql-thriftserver 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
   ```

### Accessing the Spark SQL Thrift Server with the Simba JDBC driver

The Simba JDBC Driver for Spark provides a standard JDBC interface to the information stored in DataStax Enterprise with the Spark SQL Thrift Server running.

Your DSE license includes a license to use the Simba drivers.

**Prerequisites:**

You must have a running DataStax Analytics cluster with Spark enabled (page 1090), and one node in the cluster running the Spark SQL Thrift Server (page 448).

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.
Using DataStax Enterprise advanced functionality

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 the Spark SQL Thrift Server is running, and `<port>` is the port number on which the Spark SQL Thrift Server is listening.

```
jdbc:spark://node1.example.com:10000
```

4. For more details, refer to the included documentation in the Simba driver download ZIP.

**Simba ODBC Driver for Apache Spark (Windows)**

The Simba ODBC Driver for Spark provides Windows users access to the information stored in DataStax Enterprise clusters with a running Spark SQL Thrift Server. This driver allows you to access the data stored on your DataStax Enterprise Spark nodes using business intelligence (BI) tools, such as Tableau and Microsoft Excel. 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.

**Prerequisites:**

To use the Simba ODBC Driver for Spark you must have:

- One of the following operating systems:
  - Windows 7 SP1
  - Windows 8 or 8.1
  - Windows Server 2008 R2 SP1
  - Windows Server 2012 and Windows Server 2012 R2
- A running DSE Analytics cluster with Spark enabled *(page 1090)*, and one node in the cluster running the Spark SQL Thrift Server *(page 448)*

1. Download the appropriate 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.
Using DataStax Enterprise advanced functionality

3. Refer to the *Simba ODBC Driver for Spark Installation Guide* which is installed at 
   Start > Program Files > Simba Spark ODBC Driver.

**Configuring the Spark ODBC Driver (Windows)**

Configure an ODBC data source for ODBC applications, including business intelligence (BI) tools like Tableau or Microsoft Excel.

1. Choose either the 32 bit or 64 bit ODBC driver.
   
   a. For the 32-bit driver, click Start > Program Files > Simba Spark ODBC Driver > 32 bit ODBC Data Source Administrator.

   b. For the 64-bit driver, click Start > Program Files > Simba Spark ODBC Driver > 64 bit ODBC Data Source Administrator.

2. Click the Drivers tab to verify that the Simba Spark ODBC Driver is present.

3. Create either a User or System DSN (data source name) for your ODBC tool connection.

   a. a. Click the User DSN or System DSN tab.

   b. b. Click Add > Simba Spark ODBC Driver > Finish.

   c. In Simba Spark ODBC Driver DSN Setup, enter the following:

<table>
<thead>
<tr>
<th>Data Source Name</th>
<th>The name for your DSN.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Optional longer description of your DSN.</td>
</tr>
<tr>
<td>Spark Server Type</td>
<td>SparkThriftServer (Spark 1.1 and later)</td>
</tr>
<tr>
<td>Host(s)</td>
<td>IP or hostname of your Spark SQL Thrift Server.</td>
</tr>
<tr>
<td>Port</td>
<td>Listening port for the Spark SQL Thrift Server (default 10000)</td>
</tr>
<tr>
<td>Database</td>
<td>Specify default to load all tables into the default database. Or pick a specific keyspace.</td>
</tr>
<tr>
<td>Auth Mechanism</td>
<td>User Name</td>
</tr>
<tr>
<td>User Name</td>
<td>leave blank</td>
</tr>
</tbody>
</table>

   d. Click Test.

   The test results should indicate a successful connection.
4. For advanced configuration options, refer to the Simba ODBC Driver for Spark Installation Guide which is installed at Start > Program Files > Simba Spark ODBC Driver.

**What's next:** Use the newly created data source in ODBC applications like Tableau and Microsoft Excel.

After the ODBC query is transmitted to the Spark SQL Thrift server, the appropriate Spark jobs are executed, then the data is returned via ODBC to the application.

To troubleshoot or understand the queries being executed at the Spark SQL Thrift server, open a web browser to the Spark Master web interface (http://node name:4040) on the DSE cluster, click on the Thrift server application, then view the SQL tab.

**Simba ODBC Driver for Apache Spark (Linux)**

The Simba ODBC Driver for Spark provides Linux users access to the information stored in DataStax Enterprise clusters with a running Spark SQL Thrift Server. 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.

**Prerequisites:**

To use the Simba ODBC Driver for Spark you must have:

- A running DSE Analytics cluster with Spark enabled (page 1090), and one node in the cluster running the Spark SQL Thrift Server (page 448)

1. Download the appropriate Simba ODBC Driver for Apache Spark (Linux 32- or 64-bit) from the DataStax Drivers Download page.

2. Expand the downloaded file into a suitable location.

   ```
   $ mkdir simba-odbc
   $ cd simba-odbc
   $ tar xvf version.tar.gz
   ```

3. Refer to the included Spark ODBC Install and Configuration Guide (PDF format) for detailed usage and configuration information.

**Connecting to the Spark SQL Thrift server using Beeline**

You can use Shark Beeline to test the Spark SQL Thrift server (page 447).

1. Start DataStax Enterprise with Spark enabled as a service (page 1090) or in a standalone (page 1093) installation.
2. 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. You can configure the port and bind address in `resources/spark/conf/spark-env.sh`: `HIVE_SERVER2_THRIFT_PORT`, `HIVE_SERVER2_THRIFT_BIND_HOST`.

```
$ dse spark-sql-thriftserver start --hiveconf hive.server2.thrift.port=10001
```

You can specify general Spark configuration settings by using the `--conf` option.

```
$ dse spark-sql-thriftserver start --conf spark.cores.max=4
```

3. Start the Beeline shell.

```
$ dse spark-beeline
```

4. Connect to the server using the JDBC URI for your server.

```
beeline> !connect jdbc:hive2://localhost:10000
```

5. 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;
```

**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:

```bash
$ 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:

```bash
$ sudo yum install R
```

Starting SparkR

Start the SparkR shell using the `dse` command to automatically set the Spark session within R.

1. Start the R shell using the `dse` command.
   ```bash
   $ dse sparkR
   ```

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 5.1 release notes (page 18)) 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 1082) 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 456).

Connecting to DataStax Enterprise using the Spark shell on an external Spark cluster

Use the generated byos.properties configuration file (page 455) 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.

   $ 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 (page 1082).

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.

   clients
   The default location of the clients directory depends on the type of installation:
Package installations
Installer-Services installations

Installer-No Services installations

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/usr/share/dse/clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>/usr/share/dse/clients</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/clients</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>installation_location/clients</td>
</tr>
</tbody>
</table>

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.

```bash
$ spark-shell --jars byos-5.0.jar --properties-file byos.properties
```

**Generating Spark SQL schema files**

Spark SQL can import schema files generated by DataStax Enterprise.

1. Export the schema file using `dse client-tool`.

```bash
$ dse client-tool --use-server-config spark sql-schema --all > output.sql
```

2. Copy the schema to an external Spark node.

```bash
$ scp output.sql user@sparknode1.example.com:
```

3. On a Spark node, import the schema using Spark.

```bash
$ 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
Using DataStax Enterprise advanced functionality

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.
   
   ```
   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-thriftsparkconf.conf byos.properties > custom-sparkconf.conf
   ```

5. Start Spark SQL Thrift Server with the custom configuration file and byos-version.jar.
   
   ```
   $ /usr/hdp/2.4.2.0-258/spark/sbin/start-thriftserver.sh --jars byos-version.jar --properties-file custom-sparkconf.conf
   ```

   
   ```
   $ beeline -u 'jdbc:hive2://hostname:port/default;principal=hive/_HOST@REALM'
   ```

**What's next:**

Generated SQL schema (page 457) 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.

**Accessing HDFS or CFS resources using Kerberos authentication**

If you are using Kerberos authentication and need to access HDFS or CFS data from BYOS nodes, follow these steps to configure DSE and Spark.

1. Copy hdfs-site.xml from your Hadoop configuration directory to the DSE Hadoop configuration directory
Using DataStax Enterprise advanced functionality

$ scp hdfs-site.xml admin@dsenode:/etc/dse/hadoop2-client/conf/

2. Pass a comma separate list of HDFS or CFS root directories with the 
   spark.dse.access.namenodes parameter when using DSE Spark commands.

   The spark.dse.access.namenodes parameters have the same effect as
   spark.yarn.access.namenodes from stand-alone Spark.

   The Spark application must have access to the nodes and Kerberos must be
   properly configured to be able to access them. They must either be in the same
   realm or in a trusted realm.

   DSE Spark acquires security tokens for each of the nodes so the Spark application
   can access those remote HDFS or CFS clusters.

   dse spark --conf spark.dse.access.namenodes=cfs://node1/,hdfs://
   node2/,webhdfs://node3:50070

3. Pass a comma separate list of HDFS or CFS root directories with the
   spark.yarn.access.namenodes parameter when using stand-alone Spark
   commands.

   $ spark-shell --master yarn-client --jars dse-
   byos*.jar --properties-file merged.conf --conf
   spark.yarn.access.namenodes=cfs://node1/,hdfs://
   node2/,webhdfs://node3:50070

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 18) in the release notes to find the
version of the Spark Jobserver included in this version of DSE.

Valid spark-submit options (page 430) are supported and can be applied to the Spark
Jobserver. To use the Jobserver:

- Start the job server:
  
  $ dse spark-jobserver start [any_spark_submit_options]

- Stop the job server:
  
  $ dse spark-jobserver stop

The default location of the Spark Jobserver depends on the type of installation:

- Package installations and Installer-Services: /usr/share/dse/spark/spark-
  jobserver
• Tarball installations and Installer-No Services: \texttt{installation\_location/resources/spark/spark\_jobserver}

All the uploaded JARs, temporary files, and log files are created in the user's $\texttt{HOME/.spark\_jobserver}$ 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.

\textbf{Note:}

Running multiple \texttt{SparkContext} instances in a single JVM is not recommended. Therefore it is not recommended to create a new \texttt{SparkContext} for each submitted job in a single Spark Jobserver instance. We recommend one of the two following Spark Jobserver usages.

• \textbf{Persistent Context Mode}: a single pre-created \texttt{SparkContext} shared by all jobs.
• \textbf{Context per JVM}: each job has its own \texttt{SparkContext} in a separate JVM. See the Spark Jobserver docs for details.

\textbf{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:

\begin{verbatim}
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>]
\end{verbatim}

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 \texttt{spark\_jobserver} demo included with DSE.

The default location of the \texttt{demos} directory depends on the type of installation:

• Package installations and Installer-Services: \texttt{/usr/share/dse/demos}
• Tarball installations and Installer-No Services: \texttt{installation\_location/demos}

\textbf{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 \texttt{dse.conf} file in the Spark Jobserver directory.

\begin{verbatim}
spray.can.server {
  ssl-encryption = on
  keystore = "path to keystore"
  keystorePW = "keystore password"
}
\end{verbatim}
The default location of the Spark Jobserver depends on the type of installation:

- **Package installations and Installer-Services**: /usr/share/dse/spark/spark-jobserver
- **Tarball installations and Installer-No Services**: 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 either LDAP or internal authorization (username/password) enabled.

1. Install a single Demo node using the DataStax Installer in **GUI or Text (page 223)** mode with the following settings:
   - **Install Options** page - **Default Interface**: 127.0.0.1 (You must use this IP for the demo.)
   - **Node Setup** page - **Node Type**: Analytics

2. Start DataStax Enterprise if you haven't already:
   - **Package and Installer-Services installations**:
     $ sudo service dse start
   - **Tarball and Installer-No Services installations**:
Using DataStax Enterprise advanced functionality

$ installation_location/bin/dse cassandra -k ## Starts node in Spark mode

The default installation_location is /usr/share/dse.

3. Go to the Portfolio Manager demo directory.

The default location of the Portfolio Manager demo depends on the type of installation:

- Package installations and Installer-Services: /usr/share/dse/demos/portfolio_manager
- Tarball installations and Installer-No Services: 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.

   The Spark application takes several minutes to run.

   $ ./10-day-loss.sh

10. Run the equivalent Spark Java job in the `10-day-loss-java.sh` script.

    $ ./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 either 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
  
  $ sudo apt-get install python2.7-dev

  # RedHat or CentOS
  
  $ sudo yum install python27

  # Mac OS X already has Python 2.7 installed.

- **pip** installer tool:
  
  # Debian and Ubuntu
  
  $ sudo apt-get install python-pip

  # RedHat or CentOS
  
  $ sudo yum install python-pip

  # Mac OS X
  
  $ sudo easy_install pip

- **The `libsasl2-dev` package:**
  
  # Debian and Ubuntu
  
  $ sudo apt-get install libsasl2-dev

  # RedHat or CentOS
  
  $ sudo yum install cyrus-sasl-lib

- **The required Python packages:**
  
  # All platforms
Using DataStax Enterprise advanced functionality

```bash
sudo pip install pyhs2 six flask cassandra-driver
```

If you installed DataStax Enterprise using a tarball or the GUI-no services option, set the `PATH` environment variable to the DataStax Enterprise installation `/bin` directory.

```
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 379).

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 and Installer-Services: `/usr/share/dse/demos`
   - Tarball installations and Installer-No Services: `installation_location/demos`

   The output confirms that the script imported the data into CQL and copied files to DSEFS.

   ```
   ... 10 rows imported in 0.019 seconds.
   2590 rows imported in 2.211 seconds.
   76790 rows imported in 33.522 seconds.
   + echo 'Copy csv files to Hadoop...'
   Copy csv files to Hadoop...
   + dse hadoop fs -mkdir /datastax/demos/weather_sensors/
   ```

If an error occurs, set the `PATH` as described in Prerequisites (page 465), 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
   ```
Using DataStax Enterprise advanced functionality

```
$ 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:

   ```
   $ cd installation_location/demos/weather_sensors
   $ python web/weather.py
   ```

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:

   ![Weather sensors app](image)

   **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.

   ```
   app.run(host='0.0.0.0', port=8984, threaded=True, debug=True)
   ```

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.
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:

```
$ echo "DROP KEYSPACE weathercql;" | cqlsh
```

**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:**
Using DataStax Enterprise advanced functionality

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.
   - Packages/Services: See Starting DataStax Enterprise as a service (page 1090).
   - Tarball/No Services: See Starting DataStax Enterprise as a stand-alone
     process (page 1093).

2. Ensure that the cluster is running correctly by running dsetool ring. The node type
   should be SearchAnalytics.
   - Package and Installer-Services installations: dsetool ring
   - Tarball and Installer-No Services 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 and Installer-Services: /usr/share/dse/demos/
       wikipedia
     - Tarball installations and Installer-No Services: 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")
8. Run a query using the title Solr index and collect the results.

```scala
scala> val result =
    table.select("id","title").where("solr_query='title:Boroph*'").collect
```

Equivalent JSON query:

```json
where("solr_query='{"q": "title:Boroph*"}'
```

Result:

```
result:
  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 Search index filter syntax (page 572).

**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.

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 and Installer-Services installations: See Starting DataStax Enterprise as a service (page 1090).
   - Tarball and Installer-No Services installations: See Starting DataStax Enterprise as a stand-alone process (page 1093).

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 and Installer-Services: `/usr/share/dse/demos/` `portfolio_manager`
   - Tarball installations and Installer-No Services: `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-mlib/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 and Installer-Services: `/usr/share/dse/demos`
- Tarball installations and Installer-No Services: `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 CFS (Cassandra File System)-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

   ```sql
   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 CFS. For example, on Linux:

   ```bash
   $ bin/dse hadoop fs -put mypath/normalfill.csv /
   ```

5. Start the Spark shell.

6. Verify that Spark can access the `int_ks` keyspace:

   ```scala
   scala> :showSchema int_ks
   Keyspace: int_ks
   Table: int_compound
   - pkey : Int (partition key column)
   - ckey1 : Int (clustering column)
   - data1 : Int
   ```
Using DataStax Enterprise advanced functionality

int_ks appears in the list of keyspaces.

7. Read in the file from CFS, splitting it on the comma delimiter. Transform each element into an Integer.

```scala
scala> val normalfill = sc.textFile("/normalfill.csv").map(line =>
          line.split(",").map(_.toInt));
```

Alternatively, read in the file from the local file system.

```scala
scala> val file = sc.textFile("file:///
          local-path/normalfill.csv")
```

8. Check that Spark can find and read the CSV file.

```scala
scala> normalfill.take(1);
res2: Array[Array[Int]] = Array(Array(6, 7, 8))
```

9. Save the new data to the database.

```scala
scala> 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;
pkey | ckey1 | data1
-------+--------+--------
  5 |   1   |   2   
  1 |   2   |   3   
  8 |   6   |   7   
  2 |   3   |   4   
  4 |   5   |   1   
  7 |   8   |   6   
  6 |   7   |   8   
  3 |   4   |   5   
```
Using DataStax Enterprise advanced functionality

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](https://github.com/datastax/SparkBuildExamples.git).

   ```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 [Connecting to authentication enabled clusters](#).

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 file path, 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 481) 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 480) 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-analytic 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.

dse.yaml

The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>Installer-Services installations</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/dse/dse.yaml</td>
<td></td>
</tr>
</tbody>
</table>
Prerequisites: DataStax Enterprise 5.0 or later is required.

On each node:

1. In the dse.yaml file, set the properties for the DSE File System options:

   ```yaml
   dsefs_options:
     enabled: false
     keyspace_name: dsefs
     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
   
   a. Enable DSEFS:

      ```yaml
      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:

      ```yaml
      keyspace_name: dsefs
      
      You can optionally configure multiple DSEFS file systems (page 481) 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:

      ```yaml
      work_dir: /var/lib/dsefs
      
   d. Define the public port on which DSEFS listens for clients:

      ```yaml
      public_port: 5598
      
      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 RPC (page 284) address.
      
   e. Define the private port for DSEFS inter-node communication:
Using DataStax Enterprise advanced functionality

```
private_port: 5599
```

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.

```
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. 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).

```
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:

```
# Advanced properties for DSEFS
```
5. Continue with using DSEFS *(page 480).*

**Disabling DSEFS**

To disable DSEFS and remove metadata and data:

1. Remove all directories and files from the DSEFS file system:

   ```bash
   $ 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
   ```

   The location of the `dse.yaml` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>Installer-Services installations</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/dse/dse.yaml</td>
<td>/etc/dse/dse.yaml</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Tarball installations</th>
<th>installation_location/ resources/dse/conf/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-No Services installations</td>
<td>resources/dse/conf/</td>
</tr>
</tbody>
</table>

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 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 476) multiple DSEFS file systems in a datacenter, and perform other functions, including setting the Kafka log retention.

dse.yaml
   The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/dse/conf/dse.yaml</td>
</tr>
</tbody>
</table>

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.

   **Note:** DSEFS keyspace creation uses SimpleStrategy with replication factor of 1. After starting the cluster for the first time, you must alter the keyspace to use NetworkTopologyStrategy with proper RF.

   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:

      ```
      ALTER KEYSPACE dsefs
      WITH REPLICATION = {
        'class': 'NetworkTopologyStrategy',
        'Analytics': '3'};
      ```

      **Note:** Datacenter names are case sensitive. Verify the case of the using utility, using a command like dsetool status.

   b. Run `nodetool repair` on the DSEFS keyspace.

      ```
      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:

      ```
      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:
Using DataStax Enterprise advanced functionality

```java
dsefs_options:
    ...
    keyspace_name: dsefs1
```

On each node in DC2:

```java
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 KEYSPACE dsefs1
WITH REPLICATION = {
    'class': 'NetworkTopologyStrategy',
    'DC1': '3'};
```

On DC2:

```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 493).

You can interact with the DSEFS file system in several modes: interactive command line shell, as part of dse commands, or with a REST API.

Interactive DSEFS command line shell

To use the interactive DSEFS command line shell:

<table>
<thead>
<tr>
<th>Action</th>
<th>Command line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launch DSEFS shell</td>
<td><code>$ dse fs</code></td>
</tr>
<tr>
<td></td>
<td><code>dsefs / &gt;</code></td>
</tr>
<tr>
<td></td>
<td>The DSEFS prompt shows the current working directory on DSEFS.</td>
</tr>
<tr>
<td></td>
<td>The current local working directory that you launch DSEFS from is the default directory that is used for searching local files.</td>
</tr>
<tr>
<td>Launch DSEFS shell with precedence given to the specified hosts</td>
<td><code>dse fs --prefer-contact-points -h 10.0.0.2,10.0.0.5</code></td>
</tr>
<tr>
<td></td>
<td>The --prefer-contact-points is used in conjunction with the --h option to 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.</td>
</tr>
<tr>
<td></td>
<td>Without the --prefer-contact-points option, DSEFS will switch to the closest available DSEFS node automatically, even if the --h option is used to specify contact points.</td>
</tr>
<tr>
<td>View entire DSEFS command list</td>
<td><code>dsefs / &gt; help</code></td>
</tr>
<tr>
<td>View help for any DSEFS command</td>
<td><code>dsefs / &gt; help dsefs_command</code></td>
</tr>
<tr>
<td>Add a comment to a DSEFS shell command</td>
<td><code>dsefs / &gt; get archive.tgz local_archive.tgz #retrieve the archive</code></td>
</tr>
<tr>
<td></td>
<td>Use the # character. Everything after the # character will be ignored.</td>
</tr>
</tbody>
</table>
### 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`:

```
$ 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
conf src target build.sbt
dsefs file:/home/user1/path/to/local/files > cd ..
dsefs file:/home/user1/path/to/local /
```

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
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
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

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 Connecting to authentication enabled clusters. For authentication with DSEFS, see DSEFS authentication (page 493).

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'

DSEFS command options

The following DSEFS commands and arguments are supported:
<table>
<thead>
<tr>
<th>DSEFS command</th>
<th>Description and command arguments</th>
</tr>
</thead>
</table>
| **append** source destination | Append a local file to a remote file.  
  • *source* is the path to the local file to read data from.  
  • *destination* is the path to the remote file to append the file to. |
| **cat** file_or_files | Concatenate files and print on the standard output.  
  • *file_or_files* is the file or files in DSEFS to print to standard output. Separate files with a space. |
| **cd** directory | Change the remote working directory in DSEFS.  
  • *directory* is the remote directory to change to.  
  • .. is the parent directory.  
  The DSEFS prompt identifies the current working directory in DSEFS:  
  • dsefs / > is the default directory  
  • dsefs /dir2 > is the current working directory dir2 |
| chgrp [options] group path | Change file or directory group ownership.  
  • -r, -R recursively changes the file and directory group ownership.  
  • -v explains in more detail what is being done.  
  • *group* the new group name.  
  • *path* the file or directory whose group will be changed. |
| chmod [options] octal permission mode path | Change the permissions of a file or directory.  
  • -r, -R recursively changes the file and directory permissions.  
  • -v explains in more detail what is being done.  
  • *octal permission mode* octal representation of permission mode for owner, group, and others.  
  • *path* the file or directory whose permissions will be changed |
| chown [options] path | Change files or directories ownership and/or group ownership.  
  • -r, -R recursively changes the file and directory ownership.  
  • -v explains in more detail what is being done.  
  • -u, --user *username* the new owner username.  
  • -g, --group *group* the new group owner name.  
  • *path* the file or directory whose ownership will be changed. |
<table>
<thead>
<tr>
<th>DSEFS command</th>
<th>Description and command arguments</th>
</tr>
</thead>
</table>
| **cp [options] source destination** | Copies a file within a file system or between two file systems. If the destination path points to a different file system than DSEFS, the block size and redundancy options are ignored.  
  - `-o, --overwrite` overwrite the destination file if it exists.  
  - `-b, --block-size value` The preferred block size in bytes.  
  - `-n, --redundancy-factor num_nodes` is how many replicas of the file data to create in DSEFS. This redundancy factor is similar to the replication factor in the database keystones, but is more granular. Set this value to 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, you can use a value that is equivalent to the replication factor.  
  - `source` the source file to be copied.  
  - `destination` the destination file to be created. |
| **df [options]** | List the DSEFS file system status and disk space usage.  
  - `-h` to list the sizes in human-readable format. Sizes are rounded to three significant places and presented using units:  
    - `K` (for a kilobyte = 1024 bytes),  
    - `M` (for a megabyte = 1024K),  
    - `G` (for a gigabyte = 1024M),  
    - `T` (for a terabyte = 1024G)  
    Without this option sizes are printed in bytes. |
| **exit** | Exit the DSEFS shell client. You can also type Ctrl+D to exit the shell. |
| **fsck** | Perform a file system consistency check and repair file system errors. |
| **get source destination** | Get a file from the DSEFS remote file system and copy the file to the local file system.  
  - `source` is the path to the DSEFS remote file to copy.  
  - `destination` is the path to the local file to create. |
<table>
<thead>
<tr>
<th>DSEFS command</th>
<th>Description and command arguments</th>
</tr>
</thead>
</table>
| `ls` [options] [file_system_entry_or_entries] | List the DSEFS file system entries (files or directories) in the current working directory.  
- `-R` to list subdirectories recursively.  
- `-l` to use a long listing format with metadata.  
- `-h` to list the sizes in human-readable format. Sizes are rounded to three significant places and presented using units:  
  - `K` (for a kilobyte = 1024 bytes),  
  - `M` (for a megabyte = 1024K),  
  - `G` (for a gigabyte = 1024M),  
  - `T` (for a terabyte = 1024G)  
Without this option sizes are printed in bytes.  
- `-1` limits the number of printed columns to one, so one file is printed per line. This allows the output to more easily be parsed by external tools.  
- `file_system_entry_or_entries` is the directory or directories to list the contents of. |
| `mkdir` [options] dir_or_dirs | Make a new directory or directories.  
- `-p` to make parent directories as needed.  
- `-b bytes` is the preferred block size for files stored in this directory.  
- `-c, --compression-encoder value` the encoder name to use for compression. DSE ships with the `lz4` compression encoder.  
- `-n, --redundancy-factor num_nodes` is how many replicas of the file data to create in DSEFS. This redundancy factor is similar to the replication factor in the database keyspaces, but is more granular. Set this value to 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, you can use a value that is equivalent to the replication factor.  
- `-m, --permission-mode value` octal representation of permission mode for owner, group and others.  
- `dir_or_dirs` is the directory or directories to create. |
| `mv` source destination | Move or rename a file or directory.  
- `source` is the path to the DSEFS file system entry to be moved.  
- The destination path on DSEFS:  
  - `destination` is the full destination path, including the name of the file or directory being moved.  
  - `destination/` is the full destination path. If the destination ends with a slash (/) the original file or directory name will be retained. |
<table>
<thead>
<tr>
<th>DSEFS command</th>
<th>Description and command arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>put [options] source destination</td>
<td>Copy a local file to the DSEFS.</td>
</tr>
<tr>
<td></td>
<td>• -o, --overwrite to overwrite the destination file if it exists.</td>
</tr>
<tr>
<td></td>
<td>• -b, --block-size bytes is the preferred block size in bytes.</td>
</tr>
<tr>
<td></td>
<td>• -c, --compression-encoder value the encoder name to use for compression. DSE ships with the lz4 compression encoder.</td>
</tr>
<tr>
<td></td>
<td>• -n, --redundancy-factor num_nodes is how many replicas of the file data to create in DSEFS. This redundancy factor is similar to the replication factor in the database keyspaces, but is more granular. Set this value to 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, you can use a value that is equivalent to the replication factor.</td>
</tr>
<tr>
<td></td>
<td>• -f, --compression-frame-size value the preferred frame size in bytes. Frame is a subject of compression. The bigger the frame the bigger the chance for high compression ratio. For most cases the default value of 131072 bytes is sufficient.</td>
</tr>
<tr>
<td></td>
<td>• -m, --permission-mode value octal representation of permission mode for owner, group and others.</td>
</tr>
<tr>
<td></td>
<td>• source is the path to the local source file.</td>
</tr>
<tr>
<td></td>
<td>• destination is the path to the destination file to be created on DSEFS.</td>
</tr>
<tr>
<td>pwd [path]</td>
<td>Print the working directory of the current file system or specified path.</td>
</tr>
<tr>
<td></td>
<td>• path the current working directory of the file system at the root of the path.</td>
</tr>
<tr>
<td>realpath [options]</td>
<td>Print the resolved absolute path for a specified file or directory.</td>
</tr>
<tr>
<td>path</td>
<td>• -e, --canonicalize-existing all components of the path must exist.</td>
</tr>
<tr>
<td></td>
<td>• -m, --canonicalize-missing no path components need to exist or be a directory.</td>
</tr>
<tr>
<td></td>
<td>• path the path to resolve.</td>
</tr>
<tr>
<td>rename path name</td>
<td>Rename a file or directory in the current location.</td>
</tr>
<tr>
<td></td>
<td>• path is the path to the file system entry to be renamed.</td>
</tr>
<tr>
<td></td>
<td>• name is the new name of the file system entry.</td>
</tr>
<tr>
<td>rm [-r, -R] path</td>
<td>Remove files or directories.</td>
</tr>
<tr>
<td></td>
<td>• -r, -R specifies to recursively remove the files or directories.</td>
</tr>
<tr>
<td></td>
<td>• -v explain what is being done.</td>
</tr>
<tr>
<td></td>
<td>• path is the path to the file system entry to be removed.</td>
</tr>
<tr>
<td>rmdir path</td>
<td>Remove an empty directory or directories.</td>
</tr>
<tr>
<td></td>
<td>• path is the path to the directory to be removed.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>DSEFS command</th>
<th>Description and command arguments</th>
</tr>
</thead>
</table>
| stat file_or_dir [-v] | Display the file system entry status.  
  - file_or_dir is the file system.  
  - -v to print verbose detailed information about the file status. |
| truncate file      | Truncate a file to 0 bytes. Useful for retaining the metadata for the file.  
  - file is the file to truncate. |
| umount [-f] locations | Unmount file system storage locations.  
  - -f to force unmounting, even if the location is unavailable.  
  - locations is the UUID (Universal Unique Identifier) of UUIDs of the locations to unmount. Get the UUID from the df command. |

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

| Location                        | Status | DC | Rack | Host             | Port | Directory          | Used  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Free Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144e587c-11b1-4d74-80f7-dc5e0c744aca</td>
<td>up</td>
<td>GraphAnalytics</td>
<td>rack1</td>
<td>node1.example.com</td>
<td>10.200.179.38</td>
<td>5598</td>
<td>/var/lib/dsefs/data</td>
</tr>
<tr>
<td>29289783296</td>
<td></td>
<td>5368709120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98ca0435-fb36-4344-b5b1-8d776d35c7d6</td>
<td>up</td>
<td>GraphAnalytics</td>
<td>rack1</td>
<td>node2.example.com</td>
<td>10.200.179.39</td>
<td>5598</td>
<td>/var/lib/dsefs/data</td>
</tr>
<tr>
<td>29302099968</td>
<td></td>
<td>5368709120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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`:

```bash
dsefs / >  ls -l

dsefs / >  put file:/bluefile greenfile
```

To view the new file in the DSEFS directory:

```bash
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><code>greenfile</code></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.

```bash
$ dse fs "mkdir /test2" &&
   dse fs "put README.md /test2/README.md"
```

To view the new directory listing:

```bash
$ 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><code>README.md</code></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:

```bash
$ dse fs "mkdir / test2" "put README.md /test/README.md"
```

**DSEFS compression**

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 483) command. The parameter specifies the compression encoder to use for the file that is about to get uploaded.

```
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` (page 486) 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` (page 483) 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
```
The physical size is the actual size of a data stored on the storage device. The physical size is shown by the `df (page 483)` command and by the `stat -v` 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 authentication with the DSE Unified Authentication supports authentication using any combination of DSE Unified Authentication and LDAP pass-through authentication. DSEFS doesn't support Kerberos directly, but will allow users to authenticate with a delegation token using the Digest MD5 authentication protocol. A delegation token is always generated for Spark applications when they are used with Kerberos, and Spark applications are configured to use that token to authenticate the DSEFS client with DSE.

DSEFS authentication applies only to communication between the DSEFS client and the DSEFS server.

Spark applications

For Spark applications, provide authentication credentials in one of these ways:

- **Set with the `dse spark-submit` command:**

  ```
  dse -u username -p password spark-submit
  ```

  Or preferably use the equivalent environment variables for the username and password.

- **Programmatically set the user credentials in the Spark configuration object before the SparkContext is created:**

  ```
  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")
  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 Hadoop XML configuration file. For example, in `/usr/local/lib/dse/resources/hadoop2-client/conf/core-default.xml`:

  ```xml
  <property>
    <name>com.datastax.bdp.fs.client.authentication</name>
    <value>basic</value>
  </property>
  <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 dse-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 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 315).

On this page:
- Owners, groups, and permissions (page 495)
- DSEFS superusers (page 495)
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.

```bash
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.

```bash
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 483) requires its 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 35: 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>X</td>
<td></td>
</tr>
<tr>
<td>chmod a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>chown a/b/c</td>
<td>a/b/c</td>
<td>X</td>
<td>The user must be the owner.</td>
</tr>
<tr>
<td>cp</td>
<td>same as in get and than put</td>
<td>X</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>
</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>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>
<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 315) 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.

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")
```

Copying data from CFS to DSEFS

Use the `dse hadoop fs -cp` command to copy data from CFS to DSEFS. This command works recursively.

1. If necessary, create the new location for the data in DSEFS.

   ```
   $ dse hadoop fs -mkdir dsefs:///test
   ```

2. Copy the data from CFS to DSEFS.

   ```
   $ dse hadoop fs -cp cfs://hostname/test/* dsefs:///test
   ```

3. Verify that the data exists in the new DSEFS location using `dse hadoop fs -ls`. 
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 and Installer-Services: /usr/share/dse/demos
- Tarball installations and Installer-No Services: 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>
<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,</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>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>Method</td>
<td>Status</td>
<td>Comment</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------</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>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>Method</td>
<td>Status</td>
<td>Comment</td>
</tr>
<tr>
<td>---------------------------------------------</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>
<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>Method</td>
<td>Status</td>
<td>Comment</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------</td>
<td>----------------------------------------------</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>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>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Method</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<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>
<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.

dse.yaml
The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/dse/conf/dse.yaml</td>
</tr>
</tbody>
</table>

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**
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).

### About the Cassandra File System (CFS) - deprecated

Analytics jobs require a distributed file system. DataStax Enterprise provides a replacement for the Hadoop Distributed File System (HDFS) called the Cassandra File System (CFS). See also the DataStax Enterprise file system ([DSEFS](page 475)). DSEFS is the default distributed file system on DSE Analytics nodes.

When an analytics node starts up with CFS enabled, DataStax Enterprise creates a CFS rooted at `cfs:/` and an archive file system named `cfs-archive`, which is rooted at `cfs-archive:/`. CFS is available only on analytics nodes. DataStax Enterprise creates a keyspace for the `cfs-archive` file system, and every other CFS file system. The keyspace name is similar to the file system name except the hyphen in the name is replaced by an underscore. For example, the `cfs-archive` file system keyspace is `cfs_archive`.

CFS locations must be specified using the `cfs:/` prefix and the hostname of an analytics node. For example, `cfs://node2/tmp`.

**Increasing the replication factor of default CFS keyspaces**

You must increase the replication factor of default CFS keyspaces to prevent problems when running analytics jobs.

**Encrypting CFS keyspace data**

Spark accesses the Cassandra File System (CFS) as part of the Hadoop File System (HDFS) using the configured authentication. If you encrypt the CFS keyspace sblocks and inode tables, all CFS data is encrypted.

**Configuring a CFS superuser**

A CFS superuser is the DataStax Enterprise daemon user, the user who starts DataStax Enterprise. A cassandra superuser, set up using the CQL `CREATE ROLE` command, is also a CFS superuser.

A CFS superuser can modify files in the CFS without any restrictions. Files that a superuser adds to the CFS are password-protected.

**Deleting files from the CFS**

DSE does not immediately remove deleted data from disk when you use the `dse hadoop fs -rm file` command. Instead, DSE treats the deleted data like any data that is deleted from the database. A tombstone is written to indicate the new data status. Data that is marked with a tombstone exists for a configured time period (defined by the `gc_grace_seconds` value that is set on the table). When the grace period expires, the compaction process permanently deletes the data. You do not have to manually remove expired data.

**Checkpointing with the CFS**

DataStax Enterprise does not support checkpointing to CFS.
Managing the CFS consistency level

The default read and write consistency level for CFS is **LOCAL_QUORUM** or **QUORUM**, depending on the keyspace replication strategy, **SimpleStrategy** or **NetworkTopologyStrategy**, respectively. You can change the consistency level by specifying a value for `dse.consistencylevel.read` and `dse.consistencylevel.write` properties in the `core-site.xml` file.

Setting CFS as the default distributed file system in DataStax Enterprise

DSEFS is the default distributed file system.

To make CFS the default file system, add the following properties to the `core-site.xml` Hadoop configuration file:

```xml
<configuration>
  ...
  <property>
    <name>fs.default.name</name>
    <value>cfs://127.0.0.1/</value>
  </property>
  <property>
    <name>fs.defaultFS</name>
    <value>cfs://127.0.0.1/</value>
  </property>
</configuration>
```

Replace 127.0.0.1 with the value of `broadcast_rpc_address` set in `cassandra.yaml`.

Using multiple Cassandra File Systems

You can use more than one CFS. Typical reasons for using an additional CFS are:

- To isolate analytics jobs
- To configure keyspace replication by job
- To segregate file systems in different physical datacenters
- To separate analytics data in some other way

To create an additional CFS:

1. Open the `core-site.xml` file for editing.

2. Add one or more property elements to `core-site.xml` using this format:

```xml
<property>
  <name>fs.cfs-file_system_name.impl</name>
  <value>com.datastax.bdp.hadoop.cfs.CassandraFileSystem</value>
</property>
```

With multiple CFS, you must override the default file system name for the newly created CFS to avoid conflicts with existing CFS on other datacenters. Each datacenter requires a unique default file system. For example, instead of the default value...
3. Save the file and restart DSE.

DataStax Enterprise creates the new CFS.

4. To access the new CFS, construct a URL using the following format:

```
cfs-file_system_name:path
```

For example, assuming the new file system name is `NewCassandraFS` use the `dse` commands to put data on the new CFS.

```
dse hadoop fs -put /tmp/giant_log.gz cfs-NewCassandraFS://hostname/tmp &
dse hadoop fs distcp hdfs:/// cfs-NewCassandraFS://hostname/ 
```

---

**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, other workloads such as [DSE Graph](#), [DSE Analytics and Search integration](#), and [DSE Analytics](#) 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.
Because DataStax Enterprise is a cohesive data management platform, other workloads such as DSE Graph (page 651), DSE Analytics and Search integration (page 372), and DSE Analytics (page 370) can take full advantage of the indexing and query capabilities of DSE Search.

DSE Search integrates Apache Solr™ 6.0.1 to manage 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.
- CQL index management commands (page 536) simplify search index management.
- Local node (optional) management of search indexing resources with dsetool (page 521) 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 (page 582), and advanced search node resiliency.
- Virtual nodes (vnodes) (page 367) support.
- Set the location of the search index.
- 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).
- Supports all Solr tools and APIs, with several specific unsupported features (page 510).

Solr resources

Resources for more information on using Open Source Solr (OSS):

- Apache Solr documentation
- Solr Tutorial on Apache Lucene site
- Comma-Separated-Values (CSV) file importer
- JSON importer
- Solr cell project, including a tool for importing data from PDFs

## 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 imitations apply to DSE Search:

- Column aliases are not supported in solr_query queries.
- Continuous paging.
- Static columns
- Compound primary keys for COMPACT STORAGE tables
- 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 imitations 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 (page 572) for inserting/updating data.
• Block joins based on the Lucene BlockJoinQuery in search indexes and CQL tables.
• Schemaless 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.
• UnInvertedFieldRealtime field value cache.
• GetReplicationHandler: Store & Restore.
• useDocValuesAsStored in schema fields and as a query request parameter.
• Graph queriesSolr SQLStreaming aggregations.
• Data import handler.
• Tuple/UDT subfield sorting and faceting.
• The dataDir parameter in solrconfig.xml.

Deprecated Solr features

The Tika functionality that is bundled with Apache Solr is deprecated. Instead, use the stand-alone Apache Tika project.

Other unsupported features

• 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.
Using DataStax Enterprise advanced functionality

- Tuple and UDT *(page 557)* 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 521)*` 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:
   ```cql
   CREATE SEARCH INDEX ON demo.health_data;
   ```

2. Alter the search index. For example:
   ```cql
   ALTER SEARCH INDEX CONFIG ON demo.health_data SET autoCommitTime = 30000;
   ```

3. Optionally view the XML of the pending search index. For example:
   ```cql
   DESCRIBE PENDING SEARCH INDEX CONFIG on demo.health_data;
   ```

4. Make the pending changes active. For example:
   ```cql
   RELOAD SEARCH INDEX ON demo.health_data;
   ```

### Sample search index config

```
<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>
```
Using DataStax Enterprise advanced functionality

```xml
<indexConfig>
  <rtOffheapPostings>true</rtOffheapPostings>
  <useCompoundFile>false</useCompoundFile>
  <ramBufferSizeMB>512</ramBufferSizeMB>
  <mergeFactor>10</mergeFactor>
  <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>
  <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="com.datastax.bdp.search.solr.handler.component.CqlSearchHandler" name="/analysis/field" startup="lazy"/>

<requestHandler class="com.datastax.bdp.search.solr.handler.component.CqlSearchHandler" name="/analysis/document" startup="lazy"/>

<requestHandler class="solr.admin.AdminHandlers" name="/admin/"/>

<requestHandler class="solr.PingRequestHandler" name="/admin/ping"/>

<requestHandler class="solr.UpdateRequestHandler" name="/update" startup="lazy"/>

<requestHandler class="solr.UpdateRequestHandler" name="/update/csv" startup="lazy"/>

<requestHandler class="solr.UpdateRequestHandler" name="/update/json" startup="lazy"/>

<requestHandler class="solr.FieldAnalysisRequestHandler" name="/analysis" startup="lazy"/>

<requestHandler class="solr.DocumentAnalysisRequestHandler" name="/analysis" startup="lazy"/>
</requestConfig>
```
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:

   ```
   ALTER SEARCH INDEX CONFIG ON wiki.solr SET autoCommitTime = 30000;
   ```

2. You can view the pending search config:

   ```
   DESCRIBE PENDING SEARCH INDEX CONFIG on wiki.solr;
   ```

   The resulting XML shows the maximum time between updates is 30000 milliseconds:

   ```
   <updateHandler class="solr.DirectUpdateHandler2">
     <autoSoftCommit>
       <maxTime>30000</maxTime>
     </autoSoftCommit>
   </updateHandler>
   ```

3. To make the pending changes active, reload the search index:
Using DataStax Enterprise advanced functionality

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 (page 539).

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.
Using DataStax Enterprise advanced functionality

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:
mergeMaxThreadCount
Must configure with mergeMaxMergeCount. The number of concurrent merges that Lucene can perform for the Solr core. The default mergeScheduler settings are set automatically. Do not adjust this setting.

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.

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. Default is 512.

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;
```

time
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 if you are looking for a little extra performance you can try seek ceiling.

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
Using DataStax Enterprise advanced functionality

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.

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:

   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:

   ALTER SEARCH INDEX CONFIG ON wiki.solr SET
   indexConfig.mergePolicyFactory[@class='org.apache.solr.index.AutoExpungeDeletes TieredMergePolicyFactory'].int[@name='maxMergedSegmentMB'] = 1005;

3. Set the percentage threshold for deleting from the large segments:

   ALTER SEARCH INDEX CONFIG ON wiki.solr SET
   indexConfig.mergePolicyFactory[@class='org.apache.solr.index.AutoExpungeDeletes TieredMergePolicyFactory'].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. The default value is the number of available processors.

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. To prevent writes from overwhelming reads, reduce this value and adjust `max_solr_concurrency_per_core (page 328)` in `dse.yaml`.

**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 521)` 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 (page 539) and Quick Start for CQL index management (page 616) 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>
  <fields>
    <field indexed="true" multiValued="false" name="grade_completed" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_thyroid_disease" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="pets" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="secondary_smoke" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_lupus" stored="true" type="TextField"/>
  </fields>
</schema>
```
Using DataStax Enterprise advanced functionality

```xml
<field indexed="true" multiValued="false" name="gender" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="birthplace" stored="true" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false" name="income_group" stored="true" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="marital_status" stored="true" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false" name="age_months" stored="true" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="bird" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="hay_fever" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_hay_fever" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="routine_medical_coverage" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="annual_income_20000" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="exam_status" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="other_pet" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_stroke" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="employer_paid_plan" stored="true" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false" name="family_sequence" stored="true" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="diagnosed_cataracts" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="major_medical_coverage" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_gout" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="age_unit" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="goiter" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="chronic_bronchitis" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="county" stored="true" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false" name="num_smokers" stored="true" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="screening_month" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_emphysema" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_other_cancer" stored="true" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false" name="id" stored="true" type="TrieIntField"/>
```
dsetool search index commands

dsetool commands for DSE Search

The dsetool commands for DSE Search provide search index management.

- dsetool create_core (page 1029)
- dsetool core_indexing_status (page 1027)
Using DataStax Enterprise advanced functionality

- dsetool get_core_config (page 1034)
- dsetool get_core_schema (page 1036)
- dsetool index_checks (experimental) (page 1040)
- dsetool infer_solr_schema (page 1042)
- dsetool list_index_files (page 1045)
- dsetool read_resource (page 1061)
- dsetool rebuild_indexes (page 1062)
- dsetool reload_core (page 1065)
- dsetool stop_core_reindex (page 1072)
- dsetool unload_core (page 1077)
- dsetool upgrade_index_files (page 1078)
- dsetool write_resource (page 1080)

DataStax recommends using CQL commands to manage search indexes.

**Configuration properties**

Reference information for DSE Search configuration properties.

- Data location in cassandra.yaml (page 522)
- Scheduler settings in dse.yaml (page 522)
- Indexing resources in dse.yaml (page 523)
- Indexing settings in dse.yaml (page 523)
- Safety thresholds in cassandra.yaml (page 524)
- Inter-node communication in dse.yaml (page 525)
- Query options in dse.yaml (page 525)
- Client connections in dse.yaml (page 526)
- Performance in cassandra.yaml (page 526)
- Performance in dse.yaml (page 526)

Data location in cassandra.yaml

See Set the location of search indexes.

**data_file_directories**

The directory location where table data is stored (in SSTables). The database distributes data evenly across the location, subject to the granularity of the configured compaction strategy. Default locations: /var/lib/cassandra/data.

For production, DataStax recommends RAID 0 and SSDs.

Scheduler settings in dse.yaml

Configuration options to control the scheduling and execution of indexing checks.

**ttl_index_rebuild_options**

To ensure that records with TTLs are purged from search indexes when they expire, the search indexes are periodically checked for expired documents.
The `ttl_index_rebuild_options` settings control the schedulers in charge of querying for and removing expired records, and the execution of the checks.

- **fixed_rate_period**
  Schedules how often to check for expired data in seconds. Default: 300

- **initial_delay**
  Speeds startup time by delaying the first TTL checks in seconds. Default: 20

- **max_docs_per_batch**
  Sets the maximum number of documents to check and delete per batch by the TTL rebuild thread. Default: 4096

- **thread_pool_size**
  To manage system resource consumption and prevent many search cores from executing simultaneous TTL deletes, defines the maximum number of cores that can execute TTL cleanup concurrently. Default: 1

Indexing resources in `dse.yaml`

- **solr_resource_upload_limit_mb**
  Default: 10. You can configure the maximum resource file size or disable resource upload. Sets the maximum DSE Search resource upload size limit in megabytes (MB). Set to 0 to disable resource uploading.

Indexing settings in `dse.yaml`

- **max_solr_concurrency_per_core**
  Configures the maximum number of concurrent asynchronous indexing threads per DSE Search index. Default: `number_of_available_CPU_cores`.

  If set to 1, DSE Search reverts to using synchronous indexing behavior, where data is synchronously written to the database in a single thread and indexed for DSE Search.

  To achieve optimal performance, assign this value to number of available CPU cores divided by the number of search cores. For example, with 16 CPU cores and 4 search cores, the suggested value is 4. Also see Configuring and tuning indexing performance.

  To prevent writes from overwhelming reads, reduce this value and adjust `parallelDeleteTasks` (page 519) in the search index config.

  **Note:** Dynamic switching to search concurrency level at 1 is disallowed.

- **enable_back_pressure_adaptive_nrt_commit**
  Allows back pressure system to adapt max auto soft commit time (defined per search index config) to the actual load. Setting is respected only for NRT (near real time) cores. When DSE search cores have real-time (RT) live indexing, adaptive commits are disabled regardless of this property value. See live indexing with RT.

  Default: true

- **back_pressure_threshold_per_core**
  The total number of queued asynchronous indexing requests per search core. When this number is exceeded, back pressure prevents excessive
resource consumption by throttling new incoming requests. DataStax recommends using a back_pressure_threshold_per_core value of 1000 * max_solr_concurrency_per_core (page 328).

Default: 2000

**flush_max_time_per_core**

The maximum time, in minutes, to wait for the flushing of asynchronous index updates, which 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 to ensure 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: 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, expressed. This advanced option should be changed only if exceptions happen during core loading.

Default: 5 (if not specified)

**enable_index_disk_failure_policy**

DSE Search activates the configured disk failure policy if IOExceptions occur during index update operations.

Default: 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. See Managing the location of DSE Search data.

**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 schema.xml file. Then reload the core and reindex. The default value is false. To override false, set useFieldCache=true in the request.

**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**
- Default: 5000. The number of milliseconds that 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**
- Specify settings to tune encryption of search indexes.

**decryption_cache_offheap_allocation**
- Specify whether to allocate shared DSE Search decryption cache off JVM heap.
  - Default: true

**decryption_cache_size_in_mb**
- Sets the maximum size of shared DSE Search decryption cache, in megabytes (MB).
  - Default: 256

**http_principal**
- The http_principal is used by the Tomcat application container to run DSE Search. The Tomcat web server uses GSS-API 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**
- For inter-node communication between DSE Search nodes.

**netty_client_request_timeout**
- Default: 60000. The client request timeout is the maximum cumulative time (in milliseconds) that a distributed search request will wait idly for shard responses.
  - Defines timeout behavior during distributed queries.

**Query options in dse.yaml**
- Options for CQL Solr queries.

**cql_solr_query_paging**
- Options to specify the paging behavior.
  - **off** - Default. Paging is off. Ignore driver paging settings for CQL Solr 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 (page 579).

  - **driver** - Respects driver paging settings. Specifies to use Solr pagination (page 582) (cursors) only when the driver uses pagination. Enabled automatically for DSE SearchAnalytics workloads.

**cql_solr_query_row_timeout**
Using DataStax Enterprise advanced functionality

The maximum time in milliseconds to wait for each row to be read from the database during CQL Solr queries. Default: 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.

**rpc_address**

Default: localhost. The listen address for client connections (Thrift RPC service and native transport). Valid values:

- **unset:**
  
  Resolves the address using the configured hostname configuration of the node. If left unset, the hostname resolves to the IP address of this node using /etc/hostname, /etc/hosts, or DNS.

- **0.0.0.0:**
  
  Listens on all configured interfaces. You must set the broadcast_rpc_address (page 299) to a value other than 0.0.0.0.

- **IP address**

- **hostname**

Related information: Network

**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.

**concurrent_writes**

Default: 32. *note (page 279)* Writes in DataStax Enterprise are rarely I/O bound, so the ideal number of concurrent writes depends on the number of CPU cores on the node. The recommended value is $8 \times \text{number\_of\_cpu\_cores}$.

**memtable_heap_space_in_mb**

Default: 1/4 of heap size. *note (page 279)* 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. For details, see memtable_cleanup_threshold (page 290).

Related information: Tuning the Java heap

**Performance in dse.yaml**

Node routing options.

**node_health_options**

Node health options are always enabled for all nodes. Node health is a score-based representation of how fit a node is to handle search queries.

**refresh_rate_ms**

Default: 60000

**uptime_ramp_up_period_seconds**
Default: 10800 (3 hours). 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.

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

**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 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>
    <mergeFactor>10</mergeFactor>
    <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/>
</config>
```
<updateHandler class="solr.DirectUpdateHandler2">
  <autoSoftCommit>
    <maxTime>10000</maxTime>
  </autoSoftCommit>
</updateHandler>

<query>
  <maxBooleanClauses>1024</maxBooleanClauses>
  <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>
</requestHandler>
Using DataStax Enterprise advanced functionality

You can also view 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" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_thyroid_disease" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="pets" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="secondary_smoke" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_lupus" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="gender" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="birthplace" stored="true" type="TextField"/>
    <field indexed="false" multiValued="true" name="income_group" stored="true" type="TrieIntField"/>
    <field indexed="true" multiValued="false" name="marital_status" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="age_months" stored="true" type="TrieIntField"/>
    <field indexed="true" multiValued="false" name="bird" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="hay_fever" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="diagnosed_hay_fever" stored="true" type="TextField"/>
  </fields>
</schema>
```
Using DataStax Enterprise advanced functionality

```xml
<field indexed="true" multiValued="false"
name="routine_medical_coverage" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="annual_income_20000" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="exam_status" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="other_pet" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_stroke" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="employer_paid_plan" stored="true" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="family_sequence" stored="true" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="diagnosed_cataracts" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="major_medical_coverage" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_gout" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="age_unit" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="goiter" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="chronic_bronchitis" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="county" stored="true" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="num_smokers" stored="true" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="screening_month" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_emphysema" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_other_cancer" stored="true" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false" name="id" stored="true" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="dental_coverage" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="health_status" stored="true" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="monthly_income_total" stored="true" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="fish" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="dog" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="asthma" stored="true" type="TextField"/>
<field indexed="true" multiValued="false" name="ethnicity" stored="true" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="age" stored="true" type="TrieIntField"/>
```
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 1034)
  # dsetool get_core_schema (page 1036)

- **Solr Admin**
  
  View only the last uploaded (pending) resource.

**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 (page 548).
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 `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, regardless of the value of the `stored` attribute value, with the exception of copy fields. Copy field destinations are not stored in the database.

- To store a field with `indexed="false"` and enable the field to be returned on search queries, set `stored="true"`.
- To ignore the field, set both `indexed="false"` and `stored="false"`.
- To enable search but not return the value (for example, to find a user by passport number and return the user but not the passport number), set `indexed="true"` and `stored="false"`.
- To enable search and return the value, set both `indexed="true"` and `stored="true"`.

Sample schema

The following example from *Querying CQL collections (page 600)* 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" stored="true"/>
    <field name="quotes" type="textcollection" indexed="true" stored="true"/>
    <field name="name" type="text" indexed="true" stored="true"/>
    <field name="title" type="text" indexed="true" stored="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 composite 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:** The maximum auto soft commit time in milliseconds.
- **default_query_field:** The query field to use when no field is specified in queries.
- **distributed:** 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_string_copy_fields:** Whether to generate non-stored string copy fields for non-key text fields, so that you can have text both tokenized or non tokenized.
  Default: false

- **exclude_columns:** A comma-separated (CSV) list of columns to exclude.

- **generate_DocValues_for_fields:** The fields to automatically configure DocValues in the generated search index schema. Specify '*' to add all possible fields:
  
  ```yaml
  generate_DocValues_for_fields: '*'
  ```

  or specify a comma-separated list of fields, for example:
Using DataStax Enterprise advanced functionality

```java
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.

```java
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. If resources exist,
- false - Default. Do not automatically generate search index resources.

```java
include_columns=col1, col2, col3, ...
```

A comma-separated (CSV) list of columns to include. Empty = includes all columns.

```java
index_merge_factor:segments
```

How many segments of equal size to build before merging them into a single segment.

```java
index_ram_buffer_size=MB
```

The index ram buffer size in megabytes (MB).

```java
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

```java
resource_generation_profiles
```

To minimize index size, specify a CSV list of profiles to apply while generating resources.

**Table 37: Resource generation profiles**

<table>
<thead>
<tr>
<th>Profile name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceSavingAll</td>
<td>Applies all options: spaceSavingNoTextfield, spaceSavingNoJoin, and spaceSavingSlowTriePrecision.</td>
</tr>
<tr>
<td>spaceSavingNoTextfield</td>
<td>No TextFields. Use StrField instead.</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:

```java
resource_generation_profiles: spaceSavingNoTextfield, 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:

```
CREATE SEARCH INDEX CONFIG ON wiki.solr SET
defaultQueryField='last_name';
```

See About search index management (page 536).

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:

```
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:
Using DataStax Enterprise advanced functionality

```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:

```xml
<searchComponent class="solr.SpellCheckComponent"
  name="suggest_jaspell">
  <lst name="spellchecker">
    <str name="classname">org.apache.solr.spelling.suggest.Suggester</str>
    <str name="lookupImpl">org.apache.solr.spelling.suggest.jaspell.JaspellLookup</str>
    <str name="field">suggest</str>
    <str name="storeDir">suggest</str>
    <bool name="buildOnCommit">true</bool>
    <float name="threshold">0.0</float>
  </lst>
</searchComponent>
```

Configure the parameters for the request handler:

```xml
<requestHandler class="org.apache.solr.handler.component.SearchHandler"
  name="/suggest">
  <lst name="defaults">
    <str name="spellcheck">true</str>
    <str name="spellcheck.dictionary">suggest</str>
    <str name="spellcheck.collate">true</str>
    <str name="spellcheck.extendedResults">true</str>
  </lst>
  <arr name="last-components">
    <str>suggest_jaspell</str>
  </arr>
</requestHandler>
```

**Managing search indexes**

In DSE Search, a search index is an Apache Solr™ core. Each DSE Search index uses an internally stored index configuration pair (schema.xml and solrconfig.xml) that is automatically generated when the index is created.

**About search index management**

Use the following DSE Search CQL commands to manage search indexes:
• **CREATE SEARCH INDEX** Generates a new search index on an existing table with default schema and config.

• **DESCRIBE SEARCH INDEX** Displays the active or pending schema or config in XML format.

• **ALTER SEARCH INDEX CONFIG** Modifies the search index config. After modifying, use reload to push the changes live.

• **ALTER SEARCH INDEX SCHEMA** Modifies the search index schema. After modifying, use reload to push the changes live.

• **RELOAD SEARCH INDEX** Loads pending changes to the index schema and config. Some changes such as adding or removing indexed fields require a rebuild.

• **REBUILD SEARCH INDEX** Reconstructs the search index using the active schema and config.

• **COMMIT SEARCH INDEX** Forces a reload of data into the index after data is added, modified, or removed from the corresponding CQL table.

• **DROP SEARCH INDEX** Removes the search index and corresponding files.

**Tip:** The index configuration pair (schema and config) is stored and persisted in the DSE database table `solr_admin.solr_resources`.

Local node (optional) management of search indexing resources with dsetool (page 521) commands.

**Remember:** In DSE authorization enabled environments, you must grant permission to run search index commands; see Managing search index permissions.

### Adjusting timeout for index management

When running search index management commands on large datasets using cqlsh or dsetool, the process might take longer than the default timeout period (10 minutes).

Temporarily increase the timeout period for index management commands by setting an environment variable:

• **cqlsh:** Before starting a cqlsh session, set the `CQLSH_SEARCH_MANAGEMENT_TIMEOUT_SECONDS` environment variable:

```
export CQLSH_SEARCH_MANAGEMENT_TIMEOUT_SECONDS=900;
```

Overrides the `cqlsh --request-timeout` setting.

• **dsetool:** Before running an index management command, set the `dse.search.client.timeout.secs`:

```
export JVM_OPTS="-Ddse.search.client.timeout.secs=900"
```
Using DataStax Enterprise advanced functionality

Overrides the default timeout.

About search indexes

Use the CQL command **CREATE SEARCH INDEX** to generate a search index for an existing table. Indexes created with CQL commands are automatically distributed to all search nodes in the datacenter.

**Restriction:**

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.

Starting cqlsh on a search node

Connect to a search node to use CQL search management commands.

1. Determine which nodes in the cluster are running search:

   ```
   $ dsetool status
   ```

   **Tip:** DSE Search operations are available only on search-enabled nodes. DataStax recommends single workload datacenters.

   The following example shows a development environment where all nodes in the cluster are in the same physical location, on the same rack, and the nodes have been separated into datacenters based on their workloads.

<table>
<thead>
<tr>
<th>DC: Main</th>
<th>Workload: Cassandra</th>
<th>Graph: no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status=Up/Down</td>
<td>State=Normal/Leaving/Joining/Moving</td>
<td></td>
</tr>
<tr>
<td>Rack</td>
<td>Address</td>
<td>Load</td>
</tr>
<tr>
<td>UN</td>
<td>10.10.10.111</td>
<td>15.51 MiB</td>
</tr>
<tr>
<td>rack1</td>
<td>0.90</td>
<td></td>
</tr>
<tr>
<td>UN</td>
<td>10.10.10.113</td>
<td>19.51 MiB</td>
</tr>
<tr>
<td>rack1</td>
<td>0.90</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DC: Search</th>
<th>Workload: Search</th>
<th>Graph: no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status=Up/Down</td>
<td>State=Normal/Leaving/Joining/Moving</td>
<td></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>State=Normal/Leaving/Joining/Moving</th>
<th>Address</th>
<th>Load</th>
<th>Owns</th>
<th>VNodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>Rack</td>
<td>Health [0,1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN</td>
<td>10.10.10.108</td>
<td>18.13 MiB</td>
<td>?</td>
<td>32</td>
</tr>
<tr>
<td>rack1</td>
<td>0.90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN</td>
<td>10.10.10.110</td>
<td>17.4 MiB</td>
<td>?</td>
<td>32</td>
</tr>
<tr>
<td>rack1</td>
<td>0.90</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. For large datasets, increase the cqlsh timeout:

```bash
export CQLSH_SEARCH_MANAGEMENT_TIMEOUT_SECONDS=900;
```

3. Launch a cqlsh session on a search node:

```bash
$cqlsh hostname
```

A CQL sessions starts on the remote host.

Connected to cluster1 at 10.10.10.108:9042.
[cqlsh 5.0.1 | Cassandra 3.11.0.1805 | DSE 5.1.3 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>

Creating a search index with default values

Use the DataStax Enterprise `CREATE SEARCH INDEX` to generate a search index for an existing table that is automatically distributed to all search nodes.

The search index (schema and config) is generated using default values. The schema and config are stored internally in the `solr_admin.resources` table and displayed in XML format.

Create a search index on an existing table.

```sql
CREATE SEARCH INDEX ON keyspace_name.table_name;
```

All columns are indexed using the default settings.

Setting up default query field

Set up a catch-all field for searches when no field is specified by the query.

**Note:** Adding the leading element `fields.` in `ADD fields.field fieldname` is optional and provides only cosmetic structure.

1. Create a new index-only field:

```sql
ALTER SEARCH INDEX SCHEMA ON wiki.solr
ADD fields.field[ @name='catch_all',
    @type='TextField',
```
Using DataStax Enterprise advanced functionality

Note: Since this new field contains values from two fields, set multiValued to true.

Show the pending schema changes:

```
DESCRIBE PENDING SEARCH INDEX SCHEMA ON wiki.solr;
```

The new field is listed in bold:

```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.TrieDateField" name="TrieDateField"/>
    <fieldType class="org.apache.solr.schema.StrField" name="StrField"/>
  </types>
  <fields>
    <field indexed="true" multiValued="false" name="body" stored="true" type="TextField"/>
    <field docValues="true" indexed="true" multiValued="false" name="real_date" stored="true" type="TrieDateField"/>
    <field indexed="true" multiValued="false" name="title" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="id" stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="date" stored="true" type="TextField"/>
    <field indexed="true" multiValued="true" name="catch_all" type="TextField"/>
  </fields>
  <uniqueKey>id</uniqueKey>
</schema>
```

2. Set up a copy field directive to collect the data from all CQL columns:

```
ALTER SEARCH INDEX SCHEMA ON wiki.solr ADD
    copyField[@source='title', @dest='catch_all'];
ALTER SEARCH INDEX SCHEMA ON wiki.solr ADD
    copyField[@source='body', @dest='catch_all'];
```

Show the pending schema changes:
Using DataStax Enterprise advanced functionality

DESCRIBE PENDING SEARCH INDEX SCHEMA ON wiki.solr;

The new copy field directives are listed in bold below:

```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.TrieDateField"
      name="TrieDateField"/>
    <fieldType class="org.apache.solr.schema.StrField"
      name="StrField"/>
  </types>
  <fields>
    <field indexed="true" multiValued="false" name="body"
      stored="true" type="TextField"/>
    <field docValues="true" indexed="true" multiValued="false"
      name="real_date" stored="true" type="TrieDateField"/>
    <field indexed="true" multiValued="false" name="title"
      stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="id"
      stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="date"
      stored="true" type="TextField"/>
    <field indexed="true" multiValued="true" name="catch_all"
      type="TextField"/>
  </fields>
  <uniqueKey>id</uniqueKey>
  <copyField dest="catch_all" source="body"/>
  <copyField dest="catch_all" source="title"/>
</schema>
```

3. Define the default field in the search index config:

```
ALTER SEARCH INDEX CONFIG ON wiki.solr SET defaultQueryField = 'catch_all' ;
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<config>
  <luceneMatchVersion>LUCENE_6_0_1</luceneMatchVersion>
  <dseTypeMappingVersion>2</dseTypeMappingVersion>
  <directoryFactory class="solr.StandardDirectoryFactory" name="DirectoryFactory"/>
  <indexConfig>
    <ramBufferSizeMB>512</ramBufferSizeMB>
    <rt>false</rt>
```
Using DataStax Enterprise advanced functionality

```xml
</indexConfig>
<jmx/>
<updateHandler>
  <autoSoftCommit>
    <maxTime>10000</maxTime>
  </autoSoftCommit>
</updateHandler>
<query>
  <filterCache class="solr.SolrFilterCache" highWaterMarkMB="2048" lowWaterMarkMB="1024"/>
  <enableLazyFieldLoading>true</enableLazyFieldLoading>
  <useColdSearcher>true</useColdSearcher>
  <maxWarmingSearchers>16</maxWarmingSearchers>
</query>
$requestDispatcher>
  <requestParsers enableRemoteStreaming="true" multipartUploadLimitInKB="2048000"/>
  <httpCaching never304="true"/>
</requestDispatcher>
$requestHandler class="solr.SearchHandler" default="true" name="search">
  <lst name="defaults">
    <str name="df">catch_all</str>
  </lst>
</requestHandler>
$requestHandler class="com.datastax.bdp.search.solr.handler.component.CqlSearchHandler" name="solr_query">
  <lst name="defaults">
    <str name="df">catch_all</str>
  </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>
```
Using DataStax Enterprise advanced functionality

4. Reload the schema and config to make the pending search index schema and config active:

   reload search index on wiki.solr ;

5. Rebuild the index to update the search index for the existing data:

   rebuild search index on wiki.solr ;

Generating an index with joins disabled

By default, the partition key fields are combined into a single field, _partitionKey, and stored as a string field to support joins between indexes. When join is not required, create an index with join disabled.

Note: To disable joins after an index has been created, see Configuring search index joins (page 566).

1. Create a search index with join disabled:

   The PROFILES spaceSavingNoJoin option disables joins when creating a search index. For example:

   create search index on demo.health_data
   with profiles spaceSavingNoJoin;

2. Verify that joins are disabled:

   describe active search index schema on demo.health_data ;

   ...
   <field docValues="false" indexed="false" multiValued="false"
   name="_partitionKey" omitNorms="true" stored="false"
   type="StrField"/>
Managing search index fields

Add, remove, and change indexing definitions for table columns in the search index schema.

Syntax for changing schema settings

Use the `ALTER SEARCH INDEX` to add, set, or drop settings of an existing search index schema.

The search index schema is in XML format and supports most Solr schema.xml elements:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
  <types>
    <fieldType class="class_name" name="type_name"/>
    <analyzer>
      <tokenizer class="class_name"/>
      <filter class="class_name"/>
    </analyzer>
  </fieldType>
</types>
<fields>
  <field indexed="boolean" multiValued="boolean" name="unique_name" stored="boolean" type="type_name"/>
  <dynamicField indexed="boolean" multiValued="boolean" name="fieldname_wildcard_match" stored="boolean" type="type_name"/>
</fields>
<uniqueKey>partition_key</uniqueKey>
<copyField source='field_name' dest='field_name' />
</schema>
```

CQL ALTER SEARCH INDEX SCHEMA basic syntax:

```
ALTER SEARCH INDEX SCHEMA ON keyspace_name.table_name
ADD ([shortcut] | element_path) [ element_definition | WITH $jsonsnippet$$];
```

Using shortcut keywords

Use shortcuts `field`, `fieldType`, and `copyField` to:

- Add or drop table columns from the index, for example:

  ```
  ALTER SEARCH INDEX SCHEMA ON demo.health_data ADD field gender;
  ```

  If the field name matches a column name the field definition is automatically added to the pending schema.
Using DataStax Enterprise advanced functionality

- Identify the element (field, fieldType, and copyField) and then change the setting using an element path or JSON definition:

  ```
  ALTER SEARCH INDEX SCHEMA ON wiki.solr ADD 
  copyField[@source='title', @dest='catch_all'];
  ```

### Using element paths

The element path uniquely describes the setting in the schema XML. Enclose attributes after an element in brackets; to define multiple attributes use a comma-separated list. When adding an element, include all of the attributes.

```
top_level_element_name.child_element_name[@attribute_name='value', ...]
```

For example to add a the Text field type definition:

```
types.fieldType[ @name='TextField_intl' , 
              @class='org.apache.solr.schema.TextField' ]
```

The element path can also be used to describe a sub-element in the schema.

You can use `element_path` to change a sub-element in the schema. For example, to change the ASCIIFoldingFilterFactory in the search analyzer to a ClassicFilterFactory:

```
ALTER SEARCH INDEX SCHEMA ON demo.users
SET types.fieldType[@name='TextField_intl']
   .analyzer[@type='search']
   .filter[@class='solr.ASCIIFoldingFilterFactory']@class=
   'solr.ClassicFilterFactory';
```

Changes the fieldType to:

```
<fieldType class="org.apache.solr.schema.TextField"
  name="TextField_intl">
  ...
  <analyzer type="search">
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.ClassicFilterFactory"/>
    <tokenizer class="solr.StandardTokenizerFactory"/>
  </analyzer>
</fieldType>
```

### Defining complex elements with JSON

This JSON snippet is translated into XML elements and attributes:

- JSON pair translates to XML attribute.
- JSON object translates to XML element.

The JSON is translated into these XML attributes:

```json
$$
  "analyzer": [ 
```

---

DSE 5.1 Developer Guide (Previous version)
The JSON is translated into these XML elements:

```
"analyzer": [ 
  { "type": "index", <analyzer type="index">
    "tokenizer": [ "class": "solr.Standard..." }, <tokenizer class="solr.Standard..."/>
    "filter": [ 
      { "class": "solr.LowerCase..." }, <filter class="solr.LowerCase..."/>
      { "class": "solr.ASCII..." } <filter class="solr.ASCII..."/>
  ]
]

Removing elements or attributes

The CQL command syntax to remove the second filter on the search phase analyzer:

```
ALTER SEARCH INDEX SCHEMA ON demo.users
  DROP types.fieldType[@name='TextField_intl'].analyzer[@type='search'].filter[@class='solr.ClassicFilterFactory'];
```

Changes the fieldType to:

```
<fieldType class="org.apache.solr.schema.TextField"
  name="TextField_intl">
  ...
  <analyzer type="search">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```
Schema

Describes the CQL columns to index, sets the Solr data type, defines how to index and search each field type, and defines the primary key.

The schema displays in XML format. Use element paths to define and identify elements and attributes:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
  <types>
    <fieldType class="class_path" name="fieldtype_name">
      <analyzer>
        <tokenizer class="class_path"/>
        <filter class="class_path"/>
      </analyzer>
    </fieldType>
  </types>
  <fields>
    <field attribute_name="value" docValues="true|false" indexed="true|false" multiValued="true|false" name="column_name" stored="true|false" type="fieldtype_name"/>
    <copyField source="field_name" dest="field_name" />
  </fields>
  <uniqueKey>pk_column_list</uniqueKey>
</schema>
```

**ADD FIELD column_name**

Adds a column from the CQL table to the pending search index schema using the default mapping.

```
ALTER SEARCH INDEX SCHEMA ON demo.health_data ADD field fips;
```

**ADD fields.field[@attribute_name='value', ...]**

Adds a new field to the pending schema and manually set the attributes. For example, to add a column from the table to the index and set the field type to string.

```
ALTER SEARCH INDEX SCHEMA ON demo.health_data
ADD fields.field[@name='fish', @type='StrField', @indexed='true'];
```

**DROP field field_name**

Removes a field from the pending search index schema.
Using DataStax Enterprise advanced functionality

```
ALTER SEARCH INDEX SCHEMA ON demo.health_data
DROP field fips;
```

**SET fields.field[@name='field_name']@attribute_name='value'**

Changes the field identified by the attribute in brackets by adding or replacing the `attribute_to_change`.

Field attributes:
- **name**: Matches a CQL table column name or the name of a copyField destination.
- **type**: Name of a defined fieldType.
- **indexed**: True indicates that the field is indexed. By default, only the fields that are included in the index on creation are displayed.
  
  **Note**: Primary key columns must be indexed (`indexed="true"`).
- **docValues**: Creates a forward index on the field values.
- **multiValued**: Contains more than one value, such as a set, map, list column, or the destination of multiple copyField definitions.

**Restriction**: The stored field attribute is not supported.

```
ALTER SEARCH INDEX SCHEMA ON demo.health_data
SET fields.field[@name='gender_s']@multiValued='true';
```

**ADD types.fieldType[@attribute_name='value', …] WITH $$ { json_map } $$**

Adds a field type definition to the schema for analyzing, tokenizing, and filtering fields in the index.

```
ADD types.fieldType[@name='TrieIntField',
@class='solr.TrieIntField']
```

**Note**: Optionally add the leading element `fields.` in `SET fields.field field_name` to follow a naming convention and provide structure.

### Defining index field types

**Default field type definitions**

A field type definition is required for parsing CQL columns into the corresponding Solr field type. Add processing instructions to the analyzer section of the fieldType definition.

**TrieField types**

Used with a type attribute and value: integer, long, float, double, date.

**TrieDoubleField**

```
<fieldType class="org.apache.solr.schema.TrieDoubleField"
name="TrieDoubleField"/>
```
TrieDateField
Date field for Lucene TrieRange processing, supports indexing negative date. For example: -28011-12-02T00:00:00.002Z. To insert negative dates for the CQL timestamp, insert an epoch time in milliseconds. The TimestampType does not accept a textual representation of negative dates.

```
<fieldType class="org.apache.solr.schema.TrieDateField" name="TrieDateField"/>
```

TrieFloatField

```
<fieldType class="org.apache.solr.schema.TrieFloatField" name="TrieFloatField"/>
```

StringField types

VarIntStrField
Define with the DataStax class to convert a CQL varint.

```
<fieldType class="com.datastax.bdp.search.solr.core.types.VarIntStrField" name="VarIntStrField"/>
```

AsciiStrField
Converts a CQL ascii into a standard Solr StrField.

```
<fieldType class="com.datastax.bdp.search.solr.core.types.AsciiStrField" name="AsciiStrField"/>
```

SimpleDateField
Define with the DataStax class to convert a CQL date field into a compatible Solr date field.

```
<fieldType class="com.datastax.bdp.search.solr.core.types.SimpleDateField" name="SimpleDateField"/>
```

BoolField
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.

```
<fieldType class="org.apache.solr.schema.BoolField" name="BoolField"/>
```

BinaryField

```
<fieldType class="org.apache.solr.schema.BinaryField" name="BinaryField"/>
```

UUIDField
A value of this type is a Type 1 UUID that includes the time of its generation. Values are sorted by conflict-free timestamps. For example, use the TimeUUID type to identify a column, such as a blog entry, by its timestamp and allow multiple clients to write to the same partition key simultaneously. To find data mapped from a TimeUUID to a UUIDField, search for the entire UUID value, not just its time component.

```<fieldType class="org.apache.solr.schema.UUIDField" name="UUIDField"/>
```

Default index field definitions for CQL column types

**Restriction:** Decimal and varint are indexed as strings. Apache Lucene® does not support the precision required by these numeric types. Range and sorting queries do not work as expected if a table uses these types.

### Table 38: Default column definitions

<table>
<thead>
<tr>
<th>CQL data type</th>
<th>Field type name</th>
<th>docValues</th>
<th>multiValued</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii</td>
<td>AsciiStrField</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>bigint</td>
<td>TrieLongField</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>blob</td>
<td>BinaryField</td>
<td>not supported</td>
<td>false</td>
</tr>
<tr>
<td>boolean</td>
<td>BoolField</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>date</td>
<td>SimpleDateField</td>
<td>not supported</td>
<td>false</td>
</tr>
<tr>
<td>decimal</td>
<td>DecimalStrField</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>double</td>
<td>TrieDoubleField</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>float</td>
<td>TrieFloatField</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>inet</td>
<td>InetField</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>int</td>
<td>TrieIntField</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>smallint</td>
<td>TrieIntField</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>text</td>
<td>TextField</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>time</td>
<td>TimeField</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>timestamp</td>
<td>TrieDateField</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>timeuuid</td>
<td>TimeUUIDField</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>tinyint</td>
<td>TrieIntField</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>uuid</td>
<td>UUIDField</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>varchar</td>
<td>TextField</td>
<td>not supported</td>
<td>false</td>
</tr>
<tr>
<td>CQL data type</td>
<td>Field type name</td>
<td>docValues</td>
<td>multiValued</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>varint</td>
<td>VarIntStrField</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>list</td>
<td></td>
<td></td>
<td>true</td>
</tr>
<tr>
<td>map</td>
<td></td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>set</td>
<td></td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>tuple/user defined type (UDT)</td>
<td>TupleField</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

CQL data type compatibility with field type classes

**Table 39: Compatibility matrix**

<table>
<thead>
<tr>
<th>CQL</th>
<th>Field name</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii</td>
<td>AsciiStrField</td>
<td>AsciiType</td>
<td>Indexed as a standard Solr StrField.</td>
</tr>
<tr>
<td>blob</td>
<td>BinaryField</td>
<td>BytesType</td>
<td>Binary data.</td>
</tr>
<tr>
<td>boolean</td>
<td>BoolField</td>
<td>BooleanType</td>
<td>True (1, t, or T) or False (not 1, t, or T)</td>
</tr>
<tr>
<td>DateRangeType</td>
<td>DateRangeField</td>
<td>DateRangeType</td>
<td>Point-in-time with millisecond precision with support for date ranges. See Using date ranges in solr_query (page 602).</td>
</tr>
<tr>
<td>decimal</td>
<td>DecimalStrField</td>
<td>DecimalType</td>
<td>Indexed as a standard Solr StrField.</td>
</tr>
<tr>
<td>text, varchar</td>
<td>EnumField</td>
<td>UTF8Type</td>
<td>A closed set with a pre-determined sort order.</td>
</tr>
<tr>
<td>text, varchar</td>
<td>ExternalFileField</td>
<td>UTF8Type</td>
<td>Values from disk file.</td>
</tr>
<tr>
<td>text, varchar</td>
<td>GeoHashField</td>
<td>UTF8Type</td>
<td>Hash of coordinate pair (latitude,longitude) stored as a string.</td>
</tr>
<tr>
<td>inet</td>
<td>InetField</td>
<td>InetAddressType</td>
<td>InetField is implemented and indexed as a standard Solr StrField.</td>
</tr>
<tr>
<td>text, varchar</td>
<td>LatLonType</td>
<td>UTF8Type</td>
<td>Latitude/Longitude 2-D point, latitude first.</td>
</tr>
<tr>
<td>text, varchar</td>
<td>PointType</td>
<td>UTF8Type</td>
<td>Arbitrary n-dimensional point for spatial search.</td>
</tr>
<tr>
<td>CQL</td>
<td>Field name</td>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------</td>
<td>------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>text, varchar</td>
<td>RandomSortField</td>
<td>UTF8Type</td>
<td>Dynamic field in random order.</td>
</tr>
<tr>
<td>date</td>
<td>SimpleDateField</td>
<td>SimpleDateType</td>
<td>TrieDateField holding a CQL date.</td>
</tr>
<tr>
<td>PointType</td>
<td>SpatialRecursivePrefixTreeFieldType</td>
<td>SpatialRecursivePrefixTreeFieldType</td>
<td>Spatial field type for a point geospatial (page 589) context.</td>
</tr>
<tr>
<td>text, varchar</td>
<td>SpatialRecursivePrefixTreeFieldType</td>
<td>UTF8Type</td>
<td>Spatial field type for a geospatial context.</td>
</tr>
<tr>
<td>text, varchar</td>
<td>StrField</td>
<td>UTF8Type</td>
<td>String (UTF-8 encoded string or Unicode).</td>
</tr>
<tr>
<td>text, varchar</td>
<td>TextField</td>
<td>UTF8Type</td>
<td>Text, usually multiple words or tokens.</td>
</tr>
<tr>
<td>time</td>
<td>TimeField</td>
<td>TimeType</td>
<td>A TrieLongField holding a CQL time.</td>
</tr>
<tr>
<td>timeuuid</td>
<td>TimeUUIDField</td>
<td>TimeUUIDType</td>
<td>Type 1 Universally Unique Identifier (UUID).</td>
</tr>
<tr>
<td>timestamp</td>
<td>TrieDateField</td>
<td>DateType</td>
<td>Date field for Lucene TrieRange processing; supports indexing negative dates.</td>
</tr>
<tr>
<td>double</td>
<td>TrieDoubleField</td>
<td>DoubleType</td>
<td>Double field for Lucene TrieRange processing.</td>
</tr>
<tr>
<td>N/A</td>
<td>TrieField</td>
<td>N/A</td>
<td>Same as any Trie field type.</td>
</tr>
<tr>
<td>float</td>
<td>TrieFloatField</td>
<td>FloatType</td>
<td>Floating point field for Lucene TrieRange processing.</td>
</tr>
<tr>
<td>int, smallint</td>
<td>TrieIntField</td>
<td>Int32Type, ShortType</td>
<td>32-bit signed integer field for Lucene TrieRange processing.</td>
</tr>
<tr>
<td>tinyint</td>
<td>TrieIntField</td>
<td>BYTEType</td>
<td>32-bit signed integer field for Lucene TrieRange processing.</td>
</tr>
<tr>
<td>bigint</td>
<td>TrieLongField</td>
<td>LongType</td>
<td>Long field for Lucene TrieRange processing.</td>
</tr>
<tr>
<td>uuid, timeuuid</td>
<td>UUIDField</td>
<td>UUIDType</td>
<td>Universally Unique Identifier (UUID).</td>
</tr>
<tr>
<td>varint</td>
<td>VarIntStrField</td>
<td>IntegerType</td>
<td>Indexed as a standard Solr StrField.</td>
</tr>
</tbody>
</table>
**Adding a new field type**

Add the Solr field type definitions to the search index schema, and then use the new type.

1. Add the field type definition if it does not exist:

   ```cql
   ALTER SEARCH INDEX SCHEMA ON [keyspace_name.]table_name
   ADD types.fieldtype[@class='field_class', @name='type_name'];
   ```

2. Change the type of the field:

   ```cql
   ALTER SEARCH INDEX SCHEMA ON [keyspace_name.]table_name
   SET field[@name='column_name']@type='fieldtype_name';
   ```

   **Note:** If a field name in the schema matches a table column, the column is indexed.

3. Verify the pending changes:

   ```cql
   DESCRIBE PENDING SEARCH INDEX SCHEMA ON [keyspace_name.]table_name;
   ```

4. Activate the changes:

   ```cql
   RELOAD SEARCH INDEX ON [keyspace_name.]table_name;
   ```

   Copies the pending schema over the active schema. New transactions, such as data inserted into the table, are processed using the active schema. The existing data is not effected by a schema change.

5. Rebuild the index:

   ```cql
   REBUILD SEARCH INDEX ON [keyspace_name.]table_name;
   ```

   The REBUILD SEARCH INDEX regenerates the index using existing data. Rebuilding is required when changing the way that data is indexed, such as changing the type of field or if a field is added to the index.

To run a faceted queries using the gender field, change the type to StrField.

1. Add the Solr string field type to the health_data table:

   ```cql
   ALTER SEARCH INDEX SCHEMA ON demo.health_data
   ```
Using DataStax Enterprise advanced functionality

2. Change the gender field type:

```sql
ALTER SEARCH INDEX SCHEME ON demo.health_data
SET field[@name='gender']@type='StrField';
```

See Adding a column to the index (page 554).

### Adding a column to the index

Add a table column to the index. Field types are inferred when fields are added. The field types are added if they do not exist in the schema. Field type names are generated using the field type name as the simple name of the field type.

1. Add a table column to the index:

   - **Add a regular column**

     For example, to add a field to the wiki demo index:

     ```sql
     ALTER TABLE wiki.solr ADD intfield int;
     ALTER SEARCH INDEX SCHEMA ON wiki.solr ADD field intfield;
     ```

     Adds the following field:

     ```xml
     <field indexed="true" multiValued="false" name="intfield"
             stored="true" type="TrieIntField" />
     ```

     And the following field type:

     ```xml
     <fieldType name="TrieIntField"
                class="org.apache.solr.schema.TrieIntField"/>
     ```

   - **Add a table column that is a Tuple or UDT**

     Tuple columns are added as multiple fields:

     ```sql
     ALTER TABLE solr.wiki ADD fieldname tuple<text,int>;
     ALTER SEARCH INDEX SCHEMA ON solr.wiki ADD
     fields.field fieldname;
     ```

     Adds the following to the schema:

     ```xml
     <field indexed="true" multiValued="false" name="fieldname"
             stored="true" type="TupleField" />
     <field indexed="true" multiValued="false"
            name="fieldname.field1" stored="true" type="TextField" />
     ```
Using DataStax Enterprise advanced functionality

Note: Adding the leading element fields in ADD fields.field
fieldname is optional and provides only cosmetic structure.

2. Verify the pending changes:

```
DESCRIBE PENDING SEARCH INDEX SCHEMA ON [keyspace_name.]table_name;
```

3. Activate the changes:

```
RELOAD SEARCH INDEX ON [keyspace_name.]table_name;
```

Copies the pending schema over the active schema. New transactions, such as data inserted into the table, are processed using the active schema. The existing data is not effected by a schema change.

4. Rebuild the index:

```
REBUILD SEARCH INDEX ON [keyspace_name.]table_name;
```

The REBUILD SEARCH INDEX regenerates the index using existing data. Rebuilding is required when changing the way that data is indexed, such as changing the type of field or if a field is added to the index.

Indexing tuples and UDTs fields

Guidelines

Guidelines for advanced data types, including tuples and user-defined types (UDT):

- The tuple data type holds fixed-length sets of typed positional fields. Use a tuple as an alternative to a UDT.
- A UDT facilitates handling multiple fields of related information in a table. UDTs are a specialization of tuples. All examples and documentation references to tuples apply to both tuples and UDTs.

Simplify applications that require multiple tables by using UDTs to represent the related fields of information, instead of storing the information in a separate table.

DSE Search does not support:

- Tuples and UDTs that are used inside primary key declarations.
- Tuples and UDTs that are used as CQL map values. Instead, use a workaround to simulate a map-like data model (page 558).
- Dynamic fields as tuples or UDTs.
Using DataStax Enterprise advanced functionality

- Tuple/UDT subfield sorting and faceting.

Performance and memory

Tuples and UDTs are read and written as a single unit of information. Consider performance and memory impact when working with tuples and UDTs. Subfields are managed as the full tuple or UDT, and are not handled individually.

Highlights

Add CQL tuple and user-defined type (UDT) columns to an existing search index.

- Define a field for the table column using the DataStax Tuple class 
  (com.datastax.bdp.search.solr.core.types.TupleField).
- Define a field for each value in the CQL tuple or UDT column using the corresponding Solr field type.

  Note: The schema field name is column_name.fieldN where the 
  column_name matches the CQL column and N is the field position starting at 1.

Tuple configuration example

- Tuples

  Tuple columns are added as multiple fields:

  ALTER TABLE solr.wiki ADD fieldname tuple<text,int>;
  ALTER SEARCH INDEX SCHEMA ON solr.wiki ADD fields.field fieldname;

  Adds the following to the schema:

  <field indexed="true" multiValued="false" name="fieldname"
  stored="true" type="TupleField" />
  <field indexed="true" multiValued="false"
  name="fieldname.field1" stored="true" type="TextField" />
  <field indexed="true" multiValued="false"
  name="fieldname.field2" stored="true" type="TrieIntField" />

  Note: Adding the leading element fields. in ADD fields.field 
  fieldname is optional and provides only cosmetic structure.

  Drops the TupleField and all the child fields when dropping the base field name:

  ALTER SEARCH INDEX SCHEMA ON solr.wiki DROP field fieldname;

  To drop individual child fields:

  ALTER SEARCH INDEX SCHEMA ON solr.wiki DROP field 
  "fieldname.field1";

- Tuples
Tuple columns are added as multiple fields:

```
ALTER TABLE solr.wiki ADD fieldname tuple<text,int>;
ALTER SEARCH INDEX SCHEMA ON solr.wiki ADD fields.field fieldname;
```

Adds the following to the schema:

```
<field indexed="true" multiValued="false" name="fieldname" stored="true" type="TupleField" />
  <field indexed="true" multiValued="false" name="fieldname.field1" stored="true" type="TextField" />
  <field indexed="true" multiValued="false" name="fieldname.field2" stored="true" type="TrieIntField" />
```

**Note:** Adding the leading element `fields.` in `ADD fields.field fieldname` is optional and provides only cosmetic structure.

Drops the TupleField and all the child fields when dropping the base field name:

```
ALTER SEARCH INDEX SCHEMA ON solr.wiki DROP field fieldname;
```

To drop individual child fields:

```
ALTER SEARCH INDEX SCHEMA ON solr.wiki DROP field "fieldname.field1";
```

**UDT configuration example**

Example steps to configure a UDT for DSE Search.

In the search schema, declare the UDTField class

```
<fieldType class="com.datastax.bdp.search.solr.core.types.TupleField" name="UDTField"/>
```

**Note:** Use CQL commands to manage search indexes.

Create a type with the UDT

You must create a type for UDTs.

```
CREATE TYPE Address (street text, city text)
```

Create a table with the tuple

```
CREATE TABLE Location ( id text primary key, address frozen<Address> );
```

Configure the UDTField in the search schema

```
<field name="address" type="UDTField" indexed="true" stored="true"/>
```
Using DataStax Enterprise advanced functionality

Nesting tuples and UDTs

DSE Search supports queries for nested tuples and UDTs. For example, you can nest and declare tuples and UDTs inside CQL lists and sets. You cannot nest tuples and UDTs inside maps or keys.

Create a type with the Address tuple

```sql
CREATE TYPE Address (street text, city text, residents set<tuple<text, text>>)
```

Create a table with the Address tuple

```sql
CREATE TABLE Location (id text, address Address)
```

In the search schema, declare the TupleField and the nested TupleField

```sql
<field name="address" type="TupleField" indexed="true" stored="true"/>
<field name="address.street" type="text" indexed="true" stored="true"/>
<field name="address.city" type="text" indexed="true" stored="true"/>
<field name="address.residents" type="TupleField" indexed="true" stored="true" multiValued="true"/>
<field name="address.residents.field1" type="text" indexed="true" stored="true"/>
<field name="address.residents.field2" type="text" indexed="true" stored="true"/>
```

The `residents` nested tuple is `TupleField`. Each nested field is concatenated with each parent tuple or UDT by using periods.

See

Tuples and UDTs as CQL map values

DSE Search does not support using tuples and UDTs as CQL map values. Use this workaround to simulate a map-like data model.

1. Declare a collection of tuples or UDTs that have a type field that represents what would have been the map key:

   Create the tuple type. The tuple type applies to tuples and UDTs.

   ```sql
   CREATE TYPE Address (type text, street text, city text)
   ```

   Create table for UDT:
CREATE TABLE Person (name text primary key, addresses set<frozen<address>>)

Or create a table for a tuple:

CREATE TABLE Person (name text primary key, addresses set<frozen<tuple<text, text, text>>>)

2. Using this collection of tuples or UDTs as a map-like data model, it is possible to query for person addresses of a given type (key).

For example, to query for persons whose home address is in London:

{!tuple}addresses.type:Home AND addresses.city:London

Indexing map columns

DataStax Enterprise Search indexes a CQL map column using a Solr dynamic field (page 591). Dynamic fields apply the field definition using a wildcard match on the name. In the search index schema, DSE sets the dynamic field name to the CQL column name with an asterisk appended. DSE parses the data from a map using the key name and Solr will index only the keys that have the column name as the prefix. Keys that do not have the column name as a prefix are ignored.

For example, when creating a search index with the default settings on the cycling birthday_list table, the blist_map column definition is:

<dynamicField indexed="true" multiValued="false" name="blist_*" type="StrField"/>

When DSE builds the index from the CQL rows, the key name is used (not the column name). Therefore, all keys that have the blist_ as the prefix in the example are indexed and the rest are ignored. Only blist_age and blist_nation are indexed when the following data is inserted:

INSERT INTO cycling.birthday_list (cyclist_name, blist_) VALUES ('Allan DAVIS',
{ 'blist_age':'35',
'bday':'27/07/1980',
'blist_nation':'AUSTRALIA'}});

All key-value pairs in CQL maps have the same data type, the map in the example above sets all values to text (blist_map<text,text>). Because DSE Search loads the data by mapping the key name to the Solr dynamic field name, you can customize field type for each key.

Prerequisites:
Using DataStax Enterprise advanced functionality

This section walks you through the process of customizing the search index for data that has the same three map keys in every record, blist_age, bday (birth date), and blist_nation where only blist_age and blist_nation are indexed. Set up the following keyspace and table to use this example:

- Create the **cycling** keyspace
- Add the **birthday_list** table and data

1. Create an index that excludes the **blist_map** column:

   ```
   CREATE SEARCH INDEX ON cycling.birthday_list
   WITH COLUMNS blist_ {excluded:true};
   ```

2. View the active schema:

   ```
   DESC ACTIVE SEARCH INDEX SCHEMA ON cycling.birthday_list;
   ```

   DSE sets CQL text to Solr StrField type.

   ```
   <?xml version="1.0" encoding="UTF-8" standalone="no"?>
   <schema name="autoSolrSchema" version="1.5">
     <types>
       <fieldType class="org.apache.solr.schema.StrField" name="StrField"/>
     </types>
     <fields>
       <field indexed="true" multiValued="false" name="cyclist_name" type="StrField"/>
     </fields>
     <uniqueKey>cyclist_name</uniqueKey>
   </schema>
   ```

   In order to set **blist_age** to an integer, the type definition is also required.

3. Define the **blist_age** type and configure a field definition:

   ```
   ALTER SEARCH INDEX SCHEMA ON cycling.birthday_list
   ADD types.fieldType[@class='org.apache.solr.schema.TrieIntField',
   @name='TrieIntField'];
   ```

   ```
   ALTER SEARCH INDEX SCHEMA ON cycling.birthday_list
   ADD fields.field[@indexed='true', @multiValued='false',
   @name='blist_age', @type='TrieIntField'];
   ```

4. Define the **blist_nation** field as a string type, which already has a corresponding type definition.

   ```
   ALTER SEARCH INDEX SCHEMA ON cycling.birthday_list
   ```
5. View the pending changes to the schema to ensure that the syntax is correct.

```
DESC PENDING SEARCH INDEX SCHEMA ON cycling.birthday_list;
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
  <types>
    <fieldType class="org.apache.solr.schema.StrField" name="StrField"/>
    <fieldType class="org.apache.solr.schema.TrieIntField" name="TrieIntField"/>
  </types>
  <fields>
    <field indexed="true" multiValued="false" name="cyclist_name" type="StrField"/>
    <field indexed="true" multiValued="false" name="blist_age" type="TrieIntField"/>
    <field indexed="true" multiValued="false" name="blist_nation" type="StrField"/>
  </fields>
  <uniqueKey>cyclist_name</uniqueKey>
</schema>
```

6. Reload the index configuration and schema to push the changes live:

```
RELOAD SEARCH INDEX ON cycling.birthday_list;
```

7. Rebuild the index whenever fields are added.

```
REBUILD SEARCH INDEX ON cycling.birthday_list;
```

8. Use the map fields to filter queries.
   • Limit by age 23:

```
SELECT * FROM cycling.birthday_list WHERE solr_query = 'blist_age:23';
```

```
cyclist_name | blist_ | solr_query
--------------|--------|-----------------------------
--------------|--------|-----------------------------
Claudio HEINEN | {'bday': '27/07/1992', 'blist_age': '23', 'blist_nation': 'GERMANY'} | null
Laurence BOURQUE | {'bday': '27/07/1992', 'blist_age': '23', 'nation': 'CANADA'} | null
```
Using DataStax Enterprise advanced functionality

- Limit by nation GERMANY (which is case sensitive because the type is string):

```sql
SELECT * FROM cycling.birthday_list WHERE solr_query = 'blist_nation:GERMANY';
```

<table>
<thead>
<tr>
<th>cyclist_name</th>
<th>blist_</th>
<th>solr_query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Claudio HEINEN</td>
<td>{'bday': '27/07/1992', 'blist_age': '23', 'blist_nation': 'GERMANY'}</td>
<td>null</td>
</tr>
</tbody>
</table>

(1 rows)

### Dropping columns from the index

Remove a CQL column using the `ALTER SEARCH INDEX SCHEMA` field shortcut. The field is removed based on the name that is defined in the schema.

1. Remove a CQL column from the index:
   - Remove a regular column:
     ```sql
     ALTER SEARCH INDEX SCHEMA ON wiki.solr DROP field intfield;
     ALTER TABLE wiki.solr DROP intfield int;
     ```
   - Remove a tuple column from the index:
     ```sql
     ALTER SEARCH INDEX SCHEMA ON solr.wiki DROP field fieldname;
     ```
     When dropping the base field name, drop the TupleField and all the child fields:
     ```sql
     ALTER SEARCH INDEX SCHEMA ON solr.wiki DROP field fieldname;
     ```
     To drop individual child fields:
     ```sql
     ALTER SEARCH INDEX SCHEMA ON solr.wiki DROP field "fieldname.field1";
     ```

2. Verify the pending changes:
   ```sql
   DESCRIBE PENDING SEARCH INDEX SCHEMA ON [keyspace_name.]table_name;
   ```

3. Activate the changes:
   ```sql
   RELOAD SEARCH INDEX ON [keyspace_name.]table_name;
   ```
Copies the pending schema over the active schema. New transactions, such as data inserted into the table, are processed using the active schema. The existing data is not effected by a schema change.

4. Rebuild the index:

```
REBUILD SEARCH INDEX ON [keyspace_name.]table_name;
```

The REBUILD SEARCH INDEX regenerates the index using existing data. Rebuilding is required when changing the way that data is indexed, such as changing the type of field or if a field is added to the index.

**Indexing a column for different analysis**

DSE Search supports indexing a CQL table column for different types of analysis using the Solr `copyField` directive.

**Tip:** For a complete explanation, see the Solr Reference Guide Copying fields.

When specified during search index creation, DSE automatically defines a new index string field and sets up the data copy. The new field is not stored in the database or returned in query results.

**Restriction:** Copying from/to the same dynamic field and setting the maximum number of characters (maxChars) in the copyField definition are unsupported.

The following example uses copy fields to copy various CQL columns, such as a twitter name and email, to a multiValued field. You can then query the multiValued field using a term to search for all columns in a single query.

1. Create a keyspace using the replication strategy and replication factor that makes sense for your environment. The following example is for a single node test cluster:

```
CREATE KEYSPACE user_info
  WITH REPLICATION = { 'class' : 'SimpleStrategy',
    'replication_factor' : 1 };
```

2. Create a table:

```
CREATE TABLE user_info.users ( id text PRIMARY KEY, name text, email text, skype text, irc text, twitter text ) ;
```
Using DataStax Enterprise advanced functionality

3. Insert some data:

```sql
INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
('user1', 'john smith', 'jsmith@abc.com', 'johnsmith', 'smitty', '@johnsmith');

INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
('user2', 'elizabeth doe', 'lizzy@swbell.net', 'roadwarriorliz', 'elizdoe', '@edoe576');

INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
('user3', 'dan graham', 'etnaboy1@aol.com', 'danielgra', 'dgraham', '@dannyboy');

INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
('user4', 'john smith', 'jonsmit@fyc.com', 'johnsmith', 'jsmith345', '@johnrsmith');

INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
('user5', 'john smith', 'jds@adeck.net', 'jdsmith', 'jdansmith', '@smithjd999');

INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
('user6', 'dan graham', 'hacker@legalb.com', 'dangrah', 'dgraham', '@graham222');
```

4. Create a search index on the table:

```sql
CREATE SEARCH INDEX ON user_info.users;
```

5. Create a field that is only in the index that will contain all the data:

```sql
ALTER SEARCH INDEX SCHEMA ON user_info.users
ADD fields.field[@name='all',
    @type='StrField',
    @multiValued='true'];
```

6. Use copyField to copy the data from all the CQL columns into the new all field of the index:

```sql
ALTER SEARCH INDEX SCHEMA ON user_info.users
ADD copyField[@source='id', @dest='all'];
ALTER SEARCH INDEX SCHEMA ON user_info.users
```
Using DataStax Enterprise advanced functionality

ADD copyField[source='name', dest='all'];
ALTER SEARCH INDEX SCHEMA ON user_info.users
ADD copyField[source='email', dest='all'];
ALTER SEARCH INDEX SCHEMA ON user_info.users
ADD copyField[source='skype', dest='all'];
ALTER SEARCH INDEX SCHEMA ON user_info.users
ADD copyField[source='irc', dest='all'];
ALTER SEARCH INDEX SCHEMA ON user_info.users
ADD copyField[source='twitter', dest='all'];

7. To allow faceting on the name column, set `docValues` to `true`:

ALTER SEARCH INDEX SCHEMA ON user_info.users
SET fields.field[name='name']docValues='true';

8. Reload the schema to make the pending changes active:

RELOAD SEARCH INDEX ON user_info.users;

9. Rebuild the index to apply the new schema to the existing data:

REBUILD SEARCH INDEX ON user_info.users;

10. Filter the query using the index to return all records that contain `smitty` in any of the columns.

SELECT * FROM user_info.users WHERE solr_query = 'all:smitty';

The output is:

<table>
<thead>
<tr>
<th>id</th>
<th>email</th>
<th>irc</th>
<th>name</th>
<th>skype</th>
<th>solr_query</th>
<th>twitter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><a href="mailto:jsmith@abc.com">jsmith@abc.com</a></td>
<td>smitty</td>
<td>john smith</td>
<td>johnsmith</td>
<td>null</td>
<td>@johnsmith</td>
</tr>
</tbody>
</table>

(1 rows)

11. Get a count of unique names (skip nulls):

SELECT name FROM user_info.users
WHERE solr_query= '{"q":"","facet":
{"field":"name","mincount":"1"}}';

At the bottom of the output, the facet results appear: 3 instances of john smith, 2 instances of dan graham, and 1 instance of elizabeth doe:
Using DataStax Enterprise advanced functionality

Configuring search index joins

DataStax Enterprise supports solr_query joins on the partition key field (_partitionKey). By default, the solr_query join functionality is enabled and DSE indexes the partitioning columns in this additional field. This field, _partitionKey, increases search index size. Disabling joins can decrease the amount of disk space the search indexes uses.

Join settings in the schema

DESCRIBE ACTIVE SEARCH INDEX SCHEMA displays the schema settings of a search index. DSE hides the definition of the _partitionKey when joins are enabled.

If the schema contains a field named _partitionKey, support for joins is:

- **Enabled**: attributes docValues and indexed are set to true. For example:

  `<field name="_partitionKey" docValues="true" indexed="true" stored="false" type="StrField"/>

- **Disabled**: attributes docValues and indexed are set to false. For example:

  `<field docValues="false" indexed="false" multiValued="false" name="_partitionKey" omitNorms="true" stored="false" type="StrField"/>

Note: If the schema contains no field definition for _partitionKey, then joins are enabled.

Prerequisite

This section uses the Term and phrase searches using the wikipedia demo (page 618).

Disable joins

Disable join on a search index by setting the _partitionKey field attributes indexed and docValues to false in the schema.

1. Verify if schema has the field _partitionKey and fieldType StrField definitions.

   DESCRIBE ACTIVE SEARCH INDEX SCHEMA ON wiki.solr;

   The example search index has joins enabled with no _partitionKey definition:

   ```xml
   <?xml version="1.0" encoding="UTF-8" standalone="no"?>
   ```
Using DataStax Enterprise advanced functionality

2. If required, add the string type definition:

   ALTER SEARCH INDEX SCHEMA ON wiki.solr
   ADD types.fieldType[@class='org.apache.solr.schema.StrField',
   @name='StrField'];

   The definition is added to the pending schema and is not immediately applied.

3. Define the partition key field:

   • If the search index already has the partition key field, change the indexed and
docValues to false:

   ALTER SEARCH INDEX SCHEMA ON wiki.solr
   SET field[@name='_partitionKey']@docValues='false';
   ALTER SEARCH INDEX SCHEMA ON wiki.solr
   SET field[@name='_partitionKey']@indexed='false';

   • If the schema does not have a _partitionKey definition, add one to override the
default settings:

   ALTER SEARCH INDEX SCHEMA ON wiki.solr
Using DataStax Enterprise advanced functionality

ADD fields.field[@name='_partitionKey', @type='StrField', @docValues='false', @indexed='false'];

**Note:** The type definition StrField is also required.

4. Verify that the schema definition was correctly modified:

```
DESCRIBE PENDING SEARCH INDEX SCHEMA ON wiki.solr;
```

For example, a simple table with three fields and a single partition key:

```
<?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.TrieDateField" name="TrieDateField"/>
    <fieldType class="org.apache.solr.schema.StrField" name="StrField"/>
  </types>
  <fields>
    <field indexed="true" multiValued="false" name="body" stored="true" type="TextField"/>
    <field docValues="true" indexed="true" multiValued="false" name="real_date" stored="true" type="TrieDateField"/>
    <field indexed="true" multiValued="false" name="title" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="id" stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="date" stored="true" type="TextField"/>
    <field docValues="false" indexed="false" name="_partitionKey" type="StrField"/>
  </fields>
  <uniqueKey>id</uniqueKey>
</schema>
```

5. Reload the schema to make it active:

```
RELOAD SEARCH INDEX ON wiki.solr;
```

6. Optional, rebuild the search index:

```
REBUILD SEARCH INDEX ON wiki.solr;
```
Note: Rebuilding from CQL regenerates the index from the existing data on all search nodes, which use significant resources and is not required when disabling joins. When no rebuild command is executed after a schema change, new data in the field is not be duplicated and indexed. Use dsetool rebuild_indexes (page 1062) to regenerate the index on a node-by-node basis.

Enable joins

To enable join on a search index that previously had join disabled, set the _partitionKey, docValues, and indexed attributes to true, reload the schema, and rebuild the index.

Note: Rebuilding the search index on a large dataset might take longer than the default timeout for cqlsh. Before launching cqlsh, you can override the timeout. See Adjusting timeout for index management (page 537).

1. Start cqlsh on a node that is running DSE Search.

2. Set the docValues and indexed attributes to true:

   ```
   ALTER SEARCH INDEX SCHEMA ON wiki.solr
   SET field[@name='_partitionKey']@docValues='true';
   ALTER SEARCH INDEX SCHEMA ON wiki.solr
   SET field[@name='_partitionKey']@indexed='true';
   ```

3. Verify that the schema definition was correctly modified:

   ```
   DESCRIBE PENDING SEARCH INDEX SCHEMA ON wiki.solr;
   ```

   For example, a simple table with three fields and a single partition key:

   ```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.TrieDateField"
      name="TrieDateField="/>
       <fieldType class="org.apache.solr.schema.StrField"
      name="StrField="/>
     </types>
     <fields>
     <field indexed="true" multiValued="false" name="body"
      stored="true" type="TextField"/>
   ```
4. Reload the schema to make it active:

   RELOAD SEARCH INDEX ON wiki.solr;

5. Rebuild the search index:

   REBUILD SEARCH INDEX ON wiki.solr;

**Reloading the search index**

After you modify the search index schema (page 519), config (page 512), or upload custom resource files (like a synonym file), reload the search index to make the pending search index active.

**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:
The CQL command `RELOAD SEARCH INDEX` replaces the active search index with the pending version.

For operations, you can optionally reload a search index (also called a search core) on a single node using `dsetool reload_core` (page 1065).

**Note:** If one or more nodes fail to reload the core in distributed operations, an error message indicates a list of the failing node or nodes. Issue the reload again only on those failing nodes using distributed=false.

Reindexing in place

Setting `reindex=true` and `deleteAll=false` reindexes data and keeps the existing index. During the uploading process, user searches yield inaccurate results. To perform an in-place reindex, use this syntax:

```bash
$ dsetool reload_core keyspace_name.table_name reindex=true deleteAll=false
```

Reindexing in full

Setting `reindex=true` and `deleteAll=true` deletes the index and reindexes the dataset. User searches initially return no or partial documents as the search cores reload and data is reindexed.

```bash
$ dsetool reload_core keyspace_name.table_name reindex=true deleteAll=true
```

During reindexing, a series of criteria routes sub-queries to the nodes most capable of handling them. See Shard routing for distributed queries.

**Removing a search index**

Drop a search index from a table and delete all related data using the DROP SEARCH INDEX command.

The CQL syntax:

```cql
DROP SEARCH INDEX on [keyspace_name.]table_name;
```

Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**Updating the index after data expires (TTL)**

Time-To-Live (TTL) set on a CQL field also applies to the indexed values. The DSE Search engine purges expired and deleted data by rebuilding the index as defined by the
Using DataStax Enterprise advanced functionality

**ttl_index_rebuild_options** *(page 325)* in the *dse.yaml* *(page 312)* file. The default rebuild interval is 300 seconds (5 minutes).

Setting data expiration in CQL

1. Using CQL **INSERT** or **UPDATE**, set the TTL property.

   For example, insert a row with a life of 60 seconds into the *health_data demo* *(page 608)*:

   ```cql
   INSERT INTO demo.health_data (id, age, gender)
   VALUES (9999,88,'female') USING TTL 60;
   ```

2. Force the index to update with the new data:

   ```cql
   COMMIT SEARCH INDEX ON demo.health_data;
   ```

3. After 60 seconds, the row is removed from the CQL table and the search index.

**Inserting, updating, and deleting data**

For DSE Search, inserting and updating data uses the same CQL statements like any update to the database.

Updates to a CQL-based search index replace the entire row. You cannot replace only a field in a CQL table.

To update a CQL-based search index:

- Building on the **Querying CQL collections** *(page 600)* example, insert data into *mykeyspace.mytable* and the search index.

  ```cql
  INSERT INTO mykeyspace.mysolr ('id', 'quotes', 'name', 'title')
  VALUES ('130', 'Life is a beach', 'unknown', 'Life');
  ```

  When you use CQL to update a field, DSE Search implicitly updates individual fields in the Solr document. The reindexing of data occurs automatically.

**Filtering CQL queries with a search index**

DataStax Enterprise supports production-grade implementation of CQL Solr queries in DSE Search.

**Search index filter syntax**

DataStax Enterprise supports production-grade implementation of CQL Solr queries in DSE Search. You can develop CQL-centric applications supporting full-text search without having to work with Apache Solr™-specific APIs. Only full text search queries are supported.
Restriction:

- CQL Solr queries are defaulted to an equivalent LIMIT 10.
- Pagination is off by default. In dse.yaml, the `cql_solr_query_paging` option specifies when to use pagination (also called cursors).
- Solr restrictions apply to pagination. Queries with smaller result sets will see increased performance with paging off.
- Limitations and known Apache Solr issues apply to DSE Search queries. For example: incorrect `SORT` results for tokenized text fields.
- Column aliases are not supported in solr_query queries.
- All of the fields that are queried on DSE SearchAnalytics clusters must be defined in the search index schema definition. Fields that are not defined in the search index schema columns not defined are excluded in the results returned from Spark queries.

Search index query syntax

Execute queries against indexed columns using the `solr_query` option of the `SELECT` statement WHERE clause.

Synopsis

```
SELECT selectors FROM table
WHERE solr_query = 'search_expression' [LIMIT n]
```

There are two types of search index expressions:

- Basic search index queries using only a `q` parameter expression, see Writing a basic index query
- Advanced search index queries using a full JSON expression, see Writing a basic index query

**Note:** Use the `solr_query` option to filter on the search index fields. For example:

```
SELECT * from users WHERE solr_query = 'irc:jdoe';
```

The search indexes cannot be directly queried. For example, this syntax fails:

```
SELECT * FROM users WHERE irc = 'jdoe';
```

Writing a basic index query

The CQL query expression uses the syntax supported by the Solr `q` parameter. In CQL, to use a single quotation mark in a string literal, you must escape it using a single quotation mark (so you'll need to double the single quotation marks). See CQL escaping characters. For example:
Using DataStax Enterprise advanced functionality

SELECT * FROM keyspace.table WHERE solr_query='name: cat name: dog - name: fish'

When you name specific columns, DSE Search retrieves only the specified columns and returns the columns as part of the resulting rows. DSE Search supports projections (SELECT a, b, c...) only, not functions, for the select expression. The following example retrieves only the name column:

SELECT name FROM keyspace.table WHERE solr_query='name: cat name: dog - name: fish'

Use the LIMIT clause to specify how many rows to return. The following example retrieves only 1 row:

SELECT * FROM keyspace.table WHERE solr_query='name: cat name: dog - name: fish' LIMIT 1

Use the count() function in CQL Solr queries to return the number of rows that satisfy the Solr query:

SELECT count(*) FROM table WHERE solr_query = '...';

Using count() in combination with LIMIT or facets results in an error.

All response queries of the drivers have a custom payload where the total number of documents found is returned. This number is keyed as DSESearch.numFound.

**Writing advanced solr_query expressions**

DSE Search supports filtering CQL queries using more advanced Solr searches with JSON-based expressions.

On this page:

- JSON query syntax *(page 575)*
- JSON queries with literal characters that are Apache Solr/Apache Lucene special characters *(page 576)*
- Escaping single quotation marks *(page 580)*
- Field, query, and range faceting with a JSON query *(page 576)*
- Tracing distributed queries *(page 578)*
- JSON single-pass distributed query *(page 578)*
- JSON query name option *(page 578)*
- JSON query commit option *(page 579)*
- Queries to dynamically enable paging *(page 579)*

See also Overriding the default TimeZone (UTC) in search queries *(page 602).*
JSON query syntax

The JSON query expression syntax is a JSON string. The JSON-based query expression supports **local parameters** in addition to the following parameters:

```
{
  "q": query_expression (string),
  "fq": filter_query_expression(s) (string_or_array_of_strings, ...),
  "facet": facet_query_expression (object)
  "sort": sort_expression (string),
  "start": start_index(number),
  timeAllowed (page 579): search_time_limit_ms,
  "TZ" (page 602): zoneID), // Any valid zone ID in java TimeZone
  "paging": "driver" (string),
  "distrib.singlePass": true|false (boolean),
  "shards.failover": true|false (boolean), // Default: true
  "shards.tolerant": true|false (boolean), // Default: false
  "commit": true|false (boolean),
  "route.partition": partition_routing_expression (array_of_strings),
  "route.range": range_routing_expression (array_of_strings),
  "query.name": query_name (string),
}
```

For example:

```
SELECT id FROM nhanes_ks.nhanes WHERE
  solr_query='{"q":"ethnicity:Asian"}';

SELECT id FROM nhanes_ks.nhanes WHERE
  solr_query='{"q":"ethnicity:Mexi*", "sort":"id asc"} LIMIT 3;

SELECT * FROM mykeyspace.mytable WHERE solr_query='{"q" : "{! edismax}quotes:yearning or kills"}';
```

**Note:** To use Apache Solr™ Extended DisMax Query Parser (eDisMax) with `solr_query`, you must include `defaultSearchField` in your schema.

Making distributed queries tolerant of shard failures

Since distributed queries contact many shards, making queries more tolerant of shard failures ensures more successful completions. Use `shards.failover` or `shards.tolerant` parameters to define query failover and tolerance of shard failures during JSON queries:

<table>
<thead>
<tr>
<th>Valid configurations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;shards.failover&quot;: true,</td>
<td>This default configuration enables query failover and disables fault tolerance. Attempt to retry the failed shard requests when errors indicate that there is a reasonable chance of recovery. If any of the nodes (shards) that we scatter to fail before the query is complete, retry the shard query against a replica.</td>
</tr>
<tr>
<td>&quot;shards.tolerant&quot;: false,</td>
<td></td>
</tr>
</tbody>
</table>
Valid configurations

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;shards.failover&quot;: false, &quot;shards.tolerant&quot;: true,</td>
<td>Disable query failover. Enable fault tolerance. Make the query succeed, even if the query only partially succeeded, and did not succeed for all nodes.</td>
</tr>
<tr>
<td>&quot;shards.failover&quot;: false, &quot;shards.tolerant&quot;: false,</td>
<td>Disable query failover. Disable fault tolerance.</td>
</tr>
</tbody>
</table>

**Note:** Failover and tolerance of partial results cannot coexist in the same query. Queries support enabling tolerance for only one parameter.

Other fault tolerance configuration options include: `netty_client_request_timeout` (*page 327*) in `dse.yaml` and `read_request_timeout_in_ms` (*page 297*) in `cassandra.yaml`.

**JSON queries with literal characters that are Apache Solr™/Apache Lucene® special characters**

Lucene supports escaping special characters that are part of the query syntax. Special characters are: +, -, &&, ||, !, (, ), ", ~, *, ?, and :. Using JSON with `solr_query` requires additional syntax for literal characters that are Lucene special characters.

**Syntax for a simple search string:**

| Simple search string | mytestuser1? |
| Solr query | name:mytestuser1? |
| CQL Solr query | solr_query='{"q":"name:mytestuser1\?"}' |

**Syntax for a complex search string:**

| Complex search string | (1+1):2 |
| Solr query | e:\(1\+1\):2 |
| CQL Solr query | solr_query='{"q":"e:\(1\+1\):2"}' |

**Field, query, and range faceting with a JSON query**

Specify the facet parameters inside a facet JSON object to perform field, query, and range faceting inside Solr queries. Distributed pivot faceting is supported. The query syntax is less verbose to specify facets by:

- Specifying each facet parameter without the facet prefix that is required by HTTP APIs.
- Expressing multiple facet fields and queries inside a JSON array.

**Faceted search example**
Using DataStax Enterprise advanced functionality

```sql
SELECT * FROM solr WHERE solr_query='{"q":"id:*","facet":
{"field":"type"})'};
```

**Query facet example**

```sql
SELECT * FROM solr WHERE solr_query='{"q":"id:*","facet":
{"query":"type:0"})'};
```

**Multiple queries example**

```sql
SELECT * FROM solr WHERE solr_query='{"q":"id:*","facet":
{"query":["type:0","type:1"]})'};
```

**Distributed pivot faceting example**

```sql
SELECT id FROM table WHERE solr_query='{"q":"id:*","facet":
{"pivot":"type,value","limit":"-1"})'};
```

**Range facet example**

```sql
SELECT * FROM solr WHERE solr_query='{"q":"id:*","facet":
{"range":"type", "f.type.range.start":-10,
"f.type.range.end":10, "range.gap":1})'};
```

The returned result is formatted as a single row with each column corresponding to the output of a facet (either field, query, or range). The value is represented as a JSON blob because facet results can be complex and nested. For example:

<table>
<thead>
<tr>
<th>facet_fields</th>
<th>facet_queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>{&quot;type&quot;:{&quot;0&quot;:2,&quot;1&quot;:1}}</td>
<td>{&quot;type:0&quot;:2,&quot;type:1&quot;:1}</td>
</tr>
</tbody>
</table>

**Range by date facet example**

```sql
SELECT * FROM solr WHERE
solr_query='{"q":"business_date:*","facet":
{"range":"business_date",
"f.business_date.range.start":"2015-01-01T00:00:00Z",
"f.business_date.range.end":"2015-08-01T00:00:00Z",
"f.business_date.range.gap":"+1MONTH"})';
```

**Warning**: Solr range facets before, after, and between might return incorrect and inconsistent results on multi-node clusters. See SOLR-6187 and SOLR-6375.

**Interval facet example**

```sql
SELECT * FROM solr WHERE solr_query='{"q":"id:*","facet":
{"interval":"id", "interval.set":"[*,500]"});
```
Tracing distributed queries

During a distributed query, every node is responsible for a set of token ranges. A shard is the node/ranges combination. The shard token range is reported:

- In the shards.info response for HTTP queries.
- In the system_traces.events table for HTTP queries that provide cassandra.trace=true and CQL Solr queries that enable tracing at the driver level.

JSON single-pass distributed query

Single-pass distributed queries are supported in CQL Solr queries.

To use a single pass distributed query instead of the standard two-pass query, specify the `distrib.singlePass` Boolean parameter in the JSON query expression:

```json
SELECT * FROM ks.cf WHERE solr_query = '{"q" : "*:*,
"distrib.singlePass" : true}'
```

Using a single-pass distributed query has an operational cost that includes potentially more disk and network overhead. With single-pass queries, each node reads all rows that satisfy the query and returns them to the coordinator node. An advanced feature, a single-pass distributed query saves one network round trip transfer during the retrieval of queried rows. A regular distributed query performs two network round trips, the first one to retrieve IDs from DSE Search that satisfy the query and another trip to retrieve only the rows that satisfy the query from the database, based on IDs from the first step. Single-pass distributed queries are most efficient when most of the documents found are returned in the search results, and they are not efficient when most of the documents found will not be returned to the coordinator node.

For example, a distributed query that only fans out to a single node from the coordinator node will likely be most efficient as a single-pass query.

Single pass distributed queries for CQL are supported when the additional `distrib.singlePass` boolean parameter is included in the JSON query.

With single-pass queries, there is a limitation that only document fields that are defined in the search schema are returned as query results. This limitation also applies to map entries that do not conform to the dynamic field mapping (page 591).

JSON query name option

Using the following syntax to name your queries to support metrics and monitoring for performance objects. Naming queries can be useful for tagging and JMX operations, for example.

```sql
SELECT id FROM nhanes_ks.nhanes WHERE solr_query=' {"query.name":"Asian subjects","q":"ethnicity:Asia*"}' LIMIT 50;
```
JSON query commit option

If you are executing custom queries after bulk document loading, and the auto soft commit is disabled or the configured value is extremely infrequent, and you want the latest data to be visible to your query, use the JSON query commit option to ensure that all pending updates are soft-committed before the query runs. By default, the commit option is set to false.

For example:

```
SELECT id FROM nhanes_ks.nhanes WHERE
solr_query='{"q":"ethnicity:Asia*", "commit":true}' LIMIT 50;
```

**Warning:** Do not use the JSON commit option for live operations against a production cluster. DataStax recommends using the JSON commit option only when you would otherwise be forced to issue a commit though the Solr HTTP interface. The commit option is not a replacement for the normal auto soft commit process.

Queries to dynamically enable paging

To dynamically enable pagination when `cql_solr_query_paging: off` (page 326) in `dse.yaml`, use the "paging":"driver" parameter:

```
select id from wiki.solr where solr_query='{"q":"*", "sort":"id asc", "paging":"driver"}';
```

Limit queries by time

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.

Example with a 30 second timeout:

```
SELECT * FROM users where solr_query = '{ "q": "+:*", "timeAllowed":30000}';
```

Escaping characters in a solr_query

Solr queries require escaping special characters that are part of the query syntax. Special characters are: +, -, &, |, !, ( ), *, ~, ?, and :. To escape these
Using DataStax Enterprise advanced functionality

characters, use a slash (\) before the character to escape. For example, to search for a literal double quotation mark ("), character, escape the " for Solr with ".

When using `solr_query` you can escape special characters in two forms:

**CQL Solr**

...WHERE solr_query='field:value'

**JSON**

WHERE solr_query='{ "q": "field:value"}''

JSON-encoded queries require that values must also be JSON-escaped for special characters.

For queries that contain double quotation marks, use triple slashes `\\\`:

- For query syntax: One slash \ to escape the "
- For the JSON string syntax: Two slashes `\` to escape the \\.

Triple slashes `\\\` escape both characters in \" to produce `\\\` (an escaped escape) and `\"` (an escaped double quote).

**Escaping single quotation marks**

- Double the single quotation mark (‘)

**CQL**

...WHERE solr_query='name:Walter's'

**JSON**

...WHERE solr_query='{ "q": "Walter's"}'

- Use dollar-quotes for the string constant

**CQL**

...WHERE solr_query=$$name:Walter's$$

**JSON**

...WHERE solr_query=$${ "q": "Walter's"}$$

**Query examples for escaping double quotation marks**

**CQL**

Double the single quotation mark (‘) and add the backslash (\) for Solr escaping

...WHERE solr_query='name:Walter\'s'

**JSON**

Escape " to `\\"` to escape both special characters for JSON

...WHERE solr_query='{ "q": "Walter\\"s"}'

**Exact and fuzzy query examples**

**Exact phrase query**

For a row that looks like this, with an email address that includes a double quotation mark `greenr"q@example.com`:
Using DataStax Enterprise advanced functionality

```
INSERT INTO users(id, email) VALUES(1, 'greenr"q@example.com')"
```

Perform a phrase query to search for the email address that is enclosed in double quotation marks:

```
SELECT * FROM users where solr_query = ' {
  "q": ":*:",
  "fq": "email:\"greenr\\\"q@example.com\\\""
}'
```

**Fuzzy query**

For a row that looks like this, with the same email address that includes a double quotation mark:

```
select * from test.users where solr_query='{"q":"email:r\\n\"q@example"}' ;
```

<table>
<thead>
<tr>
<th>id</th>
<th>email</th>
<th>solr_query</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>greenr&quot;<a href="mailto:q@example.com">q@example.com</a></td>
<td>null</td>
</tr>
</tbody>
</table>

(1 rows)

For a term query (fuzzy search) for all email addresses that include r"q@example, remove the double quotation marks but retain triple quotation marks for the escaped double quotation character that is part of the email address:

```
SELECT * FROM users where solr_query = ' {
  "q": ":*:",
  "fq": "email:r\\\"q@example"
}'
```

Using JSON with `solr_query` requires additional syntax for literal characters that are Lucene special characters. See [JSON queries with literal characters that are Solr special characters](page 576).

**Search index filtering best practices**

DataStax recommends following these best practices for running queries in DSE Search:

- Use CQL *(page 572)* to run search queries.
  
  Perform all data manipulation with CQL, except for deleting by query *(page 650)*.

- Use the simplest and best fit Solr types to fulfill the required type for your query. See [Defining index field types](page 548).

- Use profiles when creating *(page 539)* a search index.

- Avoid querying nodes that are indexing.

  For responding to queries, DSE Search ranks the nodes that are not performing search indexing higher than indexing ones. If nodes that are indexing are the only nodes that can satisfy the query, the query does not fail but can return only partial results.
• When vnodes are not used, distributed queries in DSE Search are most efficient when the number of nodes in the queried data center (DC) is a multiple of the replication factor (RF) in that DC.

• Avoid using too many terms in the query, like:

  ```
  SELECT request_id, store_id FROM store_search.transaction_search
  WHERE
  solr_query='{"q":"*:*","shards.failover":true,"shards.tolerant":false,"fq":"store_id:store1a
  store_id:store2b store_id:store2c ... store_id:store19987d"}
  ```

Instead, use a terms filter query.

• When writing collections with few collection updates, DataStax recommends frozen collections over non-frozen collections to address query latency.

  For example, a simple frozen set of text elements:

  ```
  CREATE TABLE foo (id text, values frozen<set<text>>, PRIMARY KEY (id))
  CREATE TYPE name (first text, last text)
  ```

A frozen list of UDTs:

  ```
  CREATE TABLE tableWithList (id text, names frozen<list<frozen<name>>>, PRIMARY KEY (id))
  ```

**Limiting results and paging**

DSE Search integrates native driver paging with Apache Solr™ cursor-based paging. Pagination, also called cursors, supports using a cursor to scan results. Solr pagination restrictions apply.

**Note:** When using CQL Solr queries with pagination enabled, you might experience a performance slowdown because Solr is not able to use its query result cache when pagination is configured. If you do not want to paginate through large result sets, disable pagination when running CQL Solr queries. See the driver documentation.

**Using pagination (cursors) with CQL Solr queries**

In dse.yaml, the `cql_solr_query_paging` (page 326) option specifies when to use pagination (also called cursors):

• When a driver connects to the database and executes a CQL SELECT statement using a search index (`solr_query` option), you can specify to use the driver pagination settings by default by changing the `cql_solr_query_paging` (page 326) to `driver`.

• To enable pagination persistently with CQL Solr queries, set `cql_solr_query_paging: on` (page 326) in dse.yaml and restart the node.

To dynamically enable pagination when `cql_solr_query_paging: off` (page 326) in dse.yaml, use the "paging":"driver" parameter:
Using DataStax Enterprise advanced functionality

```cql
select id from wiki.solr where solr_query='{"q":"*", "sort":"id asc", "paging":"driver"}';
```

**Note:** SearchAnalytics nodes always use driver paging settings. See DSE Analytics and Search integration (page 372).

See the documentation for the CQL shell PAGING command and the driver.

It is not mandatory to use a sort clause. However, if a sort clause is not provided, sorting is undefined.

**Examples**

The word Journal is contained in ~159 entries in the body or title. Use count to determine how many rows match:

```cql
SELECT count(*) FROM wiki.solr WHERE solr_query = 'Journal';
```

Count returns only a single row; it is not effected by the 10 row limit.

<table>
<thead>
<tr>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>159</td>
</tr>
</tbody>
</table>

Run the same query without count (and `cql_solr_query_paging: off` (page 326)):

```cql
SELECT id FROM wiki.solr WHERE solr_query = 'Journal';
```

Only 10 rows are returned.

<table>
<thead>
<tr>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>23759487</td>
</tr>
<tr>
<td>23732986</td>
</tr>
<tr>
<td>23759527</td>
</tr>
<tr>
<td>23759551</td>
</tr>
<tr>
<td>23759455</td>
</tr>
<tr>
<td>23760810</td>
</tr>
<tr>
<td>23731949</td>
</tr>
<tr>
<td>23760697</td>
</tr>
<tr>
<td>23760871</td>
</tr>
<tr>
<td>23738270</td>
</tr>
</tbody>
</table>

(10 rows)

To return all matching IDs, override the `cql_solr_query_paging` setting:

```cql
SELECT id FROM wiki.solr
WHERE solr_query='{"q":"Journal", "paging":"driver"}';
```
Note: If cqlsh PAGING is enabled.

dse.yaml
The location of the dse.yaml file depends on the type of installation:

| Package installations                  | /etc/dse/dse.yaml |
| Installer-Services installations       |                   |
| Tarball installations                  | installation_location/ |
| Installer-No Services installations    | resources/dse/conf/dse.yaml |

Identifying the partition key

Solr CQL queries support restriction to a single partition key. Partition key restrictions work only when _partitionKey is explicitly indexed or the schema explicitly includes all of the components of the database partition key. In your schema, you can override _partitionKey when not using joins.

Example:

```
SELECT id, date, value FROM keyspace.table WHERE id = 'series1' AND solr_query='value:bar*';
```

CQL partition key restrictions work only with fully specified partition keys. For example, with this table:

```
CREATE TABLE vtbl (k1 text, k2 text, valuetext, PRIMARYKEY ((k1, k2)))
```

Avoid using a query like this:

```
SELECT * FROM vtbl WHERE k1 = '50' AND solr_query='value:*'
```

Use a filter query against the partially specified composite partition key:

```
SELECT * FROM valuetable WHERE solr_query='{"q":"value:*", "fq":"k1:50"}'
```

Using the Solr token function

Solr CQL queries support limited use of the token function. The token function enables targeted search that restricts the nodes queried to reduce latency.

Note: Using the Solr token function is for advanced users only and is supported only in specific use cases.

Example:

```
SELECT id, value FROM keyspace.table WHERE token(id) >= -307457345618258601 AND token(id) <= 307457345618258603 AND solr_query='id:*'
```

Example with an open range:
Using DataStax Enterprise advanced functionality

```sql
SELECT id, value FROM keyspace.table WHERE token(id) >= 3074457345618258604 AND solr_query='id:*'
```

Constraints apply to using the token function with Solr CQL queries:

- `token()` cannot be used with `route.range` or `route.partition`
- Wrapping `token()` ranges are not supported
- A specified `token()` range must be owned by a single node; ranges cannot span multiple nodes
- Because DSE uses the Solr single-pass (page 574) queries, only the fields that are declared in the search schema are returned in the query results. If you have columns that do not need to be indexed, but still need to be returned by using a token-restricted query, you can declare the columns as stored non-indexed fields in your `schema.xml` file.

1. Filtering on terms

Filter rows returned by a CQL SELECT statement on terms using the Solr Standard Parser syntax.

The basic syntax to limit queries has the following syntax:

```sql
SELECT column_list FROM table_name
WHERE solr_query = 'standard_term_expression ...';
```

The Solr Standard Parser is a case-sensitive term search that supports boolean expressions with wildcards.

**Tip:** CQL for DSE Search also supports more complex searches using JSON-formatted query strings.

This section uses the Wikipedia Demo included in DataStax Enterprise. Replace `standard_term_expression` with the `solr_query` value from corresponding tables below to return the results:

```sql
SELECT count(*) FROM solr
WHERE solr_query = 'q_search_expression';
```

**Attention:** CQL Solr queries do not support native functions or column aliases as selectors. Only `count(*)` is supported with search index queries. Results use the Solr count process. Results might vary from the native CQL count function.

2. Filtering on words, phrases, or substrings

Find rows that contain words, phrases, or substrings in indexed fields. (Similar to `LIKE` in SQL.)
• **Term**: A word that contains no spaces or punctuation and is separated from other content by a beginning or end of line, space, or punctuation mark.

• **Substring**: Match character patterns in a term. Use asterisk (*) for zero or more characters. Use question mark (?) for zero or one character in a term search.

• **Phrase**: Exact string that contains spaces and/or punctuation. Wrap phrases in double-quotes to search for the complete string when separated from other content by a line beginning or end, space, or punctuation mark.

**Prerequisites**: To run the examples in this section, set up Term and phrase searches using the wikipedia demo (page 618). Use cqlsh on a search node and replace the `search_expression` in the following statement with the example string.

```cql
SELECT count(*) FROM wiki.solr
WHERE solr_query = search_expression;
```

• Search for a single term on any indexed column or a specific column:

<table>
<thead>
<tr>
<th>Table 40: Word examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Search in</strong></td>
</tr>
<tr>
<td>Any column</td>
</tr>
<tr>
<td>Specific column</td>
</tr>
</tbody>
</table>

• Search for substrings:

  Asterisk indicates zero or more characters.

  Question mark indicates zero or one character.

<table>
<thead>
<tr>
<th>Table 41: Substring examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>Beginning of term</td>
</tr>
</tbody>
</table>
### Table 42: Phrase examples

<table>
<thead>
<tr>
<th>Search in</th>
<th>Syntax</th>
<th>Example</th>
<th>Results</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any column</td>
<td>&quot;phrase&quot;</td>
<td>&quot;African Journal&quot;</td>
<td>count ---- 8</td>
<td>Count of rows that contain the complete phrase in the title or body.</td>
</tr>
<tr>
<td>Specific column</td>
<td>'column_name:&quot;phrase&quot;'</td>
<td>'title:&quot;African Journal&quot;'</td>
<td>count ---- 8</td>
<td>Count of rows that contain the complete phrase only in the title.</td>
</tr>
</tbody>
</table>

### Search for a phrase in any column or a specific column:

- Searching for a phrase in any column or a specific column:

### Table 43: Multiple terms or phrases using operators

<table>
<thead>
<tr>
<th>Location</th>
<th>Example</th>
<th>Results</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require multiple terms</td>
<td>'title:(+Journal, +Science)&quot; 'title:Journal AND Science'</td>
<td>count ---- 4</td>
<td>Count of rows that contain both Journal and Science in the title. To use a list of terms, enclose a comma or space separated list of terms and specify a boolean operator. In this case + requires the term; therefore both terms must be in the title.</td>
</tr>
<tr>
<td>Either term</td>
<td>'title:(Journal</td>
<td></td>
<td>Science)'</td>
</tr>
<tr>
<td>Substring beginning of term, including</td>
<td>'title:Africa?'</td>
<td>count ---- 16</td>
<td>Count of rows that have a term that begins with Africa in the title but can only have one additional character.</td>
</tr>
</tbody>
</table>
### Advanced term and phrase searches

- **Find terms with like spelling:**

  The Solr fuzzy search syntax uses the Damerau-Levenshtein Distance algorithm to determine similarity of spelling based on distance. The default distance is 2. Change the distance by specifying 0-2 after the tilde (Kenya~1).

<table>
<thead>
<tr>
<th>Type of query</th>
<th>solr_query value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Like spelling</td>
<td>‘Kenya~’</td>
<td>Searches for strings similar to the word Kenya</td>
</tr>
<tr>
<td>Proximity search</td>
<td>“football Bolivia”~1</td>
<td>Searches for football and Bolivia within 10 words of each other.</td>
</tr>
<tr>
<td>Range searches</td>
<td>‘title:{football TO soccer}’</td>
<td>Supports both inclusive and exclusive bounds using square brackets and curly braces, respectively.</td>
</tr>
<tr>
<td>Term boosting</td>
<td>“football”^4 “soccer”</td>
<td>By default, the boost factor is 1. Must be a positive number.</td>
</tr>
</tbody>
</table>

The following example shows a search for science using the default distance 2:

```sql
SELECT title FROM solr WHERE solr_query = 'title:Science~';
```

The first row is unexpected, the letters are close together but the meaning is dissimilar.
The following example shows a search for science using a smaller distance:

```
SELECT title FROM solr WHERE solr_query = 'title:Science~1';
```

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>African Journal of Marine Science</td>
</tr>
<tr>
<td>African Journal of Range and Forage Science</td>
</tr>
<tr>
<td>African Journal of Science and Technology</td>
</tr>
<tr>
<td>Maarten Schenck</td>
</tr>
<tr>
<td>African Journal of Neurological Sciences</td>
</tr>
<tr>
<td>National Natural Science</td>
</tr>
<tr>
<td>China Association for Science and Technology</td>
</tr>
<tr>
<td>Philippine Science High School Cordillera Administrative Region Campus</td>
</tr>
<tr>
<td>African Journal of Library, Archives and Information Science</td>
</tr>
</tbody>
</table>

Geospatial queries for Point and LineString

Performing geospatial queries for Point and LineString.
Using DataStax Enterprise advanced functionality

Defining schemas for geospatial Point and LineString types

Define the schema for geospatial fields types. For example:

```xml
<?xml version="1.0" ?>
<schema name="spatial-no-jts" version="1.5">
  <types>
    <fieldType name="string" class="solr.StrField" />
    <fieldType name="boolean" class="solr.BoolField"/>
    <!-- When geo="false", indicate worldBounds using ENVELOPE(minX, maxX, maxY, minY) notation -->
    <fieldType name="rpt"
        class="solr.SpatialRecursivePrefixTreeFieldType"
        geo="false"
        worldBounds="ENVELOPE(-1000, 1000, 1000, -1000)"
        maxDistErr="0.001"
        distanceUnits="degrees" />
  </types>
  <fields>
    <field name="id" type="string" indexed="true" stored="true" />
    <field name="point" type="rpt" indexed="true" stored="true" />
    <field name="linestring" type="rpt" indexed="true" stored="true" />
  </fields>
  <uniqueKey>id</uniqueKey>
</schema>
```

Apache Solr™ geospatial field types

For Solr geospatial field types, declare each geospatial field type in the table schema. For example:

```sql
CREATE TABLE test (
  id text PRIMARY KEY,
  point 'PointType', linestring 'LineStringType');
```

Inserting or updating geospatial data

To insert or update data in the database, specify geotypes in the INSERT or UPDATE command. For example:

```sql
INSERT INTO test (id, point, linestring) VALUES ('1', 'POINT(5 50)', 'LINESTRING (30 10, 10 30, 40 40)');
INSERT INTO test (id, point, linestring) VALUES ('2', 'POINT(100 100)', 'LINESTRING (50 20, 20 40, 50 50)');
```

Querying geospatial data

Find points within a 10 unit radius from point (4, 49):
Using DataStax Enterprise advanced functionality

```sql
SELECT * FROM test WHERE solr_query='{"q": "*:*", "fq": "point: IsWithin(BUFFER(POINT(4.0 49.0), 10.0))"}';
```

Find linestring that contains the point (10, 30):

```sql
SELECT * FROM test WHERE solr_query='linestring:"Intersects(POINT(10 30))"';
```

See this tutorial for details on how to index and query geospatial Polygons and MultiPolygons.

**Using dynamic fields**

Using dynamic fields, you can index content in fields that are not explicitly defined by the schema. A common use case for dynamic fields is to identify fields that should not be indexed or to implement a schema-less index.

Search schema fields that are dynamic and multiValued are not supported in CQL-based search indexes.

**Spatial subfields prefix naming conventions**

Dynamic fields for spatial subfields use prefix naming conventions to enable using map types to store geospatial data:

```xml
<types>
  <fieldType class="solr.LatLonType" multiValued="false" name="LatLonType" subFieldPrefix="llt_"/>
  <fieldType name="tdouble" class="solr.TrieDoubleField" precisionStep="8" positionIncrementGap="0"/>
</types>

<fields>
  <dynamicField indexed="true" name="latmap*" stored="true" type="LatLonType"/>
  <dynamicField name="llt_*" type="tdouble" indexed="true" stored="true"/>
</fields>
```

**Best practices**

- Avoid or limit the use of dynamic fields.

  Apache Lucene® allocates memory for each unique field (column) name. For example, for a row with columns A, B, C, and another row with B, D, E, Lucene allocates 5 chunks of memory. For millions of rows, the heap is unwieldy.

- Instead of using dynamic fields, use a default query field (page 539), and then perform queries against the combined field.

  Use the FieldInputTransformer (page 643) (FIT) API.

**To use a dynamic field**

- Include an Apache Solr™ dynamic field in the search index schema.
Using DataStax Enterprise advanced functionality

Name the field using a wildcard at the beginning or end of the field. For example, an asterisk prefix or suffix in the field name in the schema designates a dynamic field.

```#
# dyna_*
# *_s
```

- To define the map collection column in CQL, use the same base name (no asterisk) that you used for the field in search index schema.

For example, use `dyna_*` in the search index schema and `dyna_` for the name of the CQL map collection.

- Use type text for the map key:

```java
CREATE TABLE my_dynamic_table ( 
   ...
   dyna_ map<text, int>,
   ...
);
```

- Using CQL, insert data into the map using the base name as a prefix or suffix in the first component of each map pair:

```java
{ prefix_literal : literal, prefix_literal : literal, ...
```

The CQL map looks like:

```java
'dyna_': {dyna_1 : 1, dyna_2 : 2, dyn_3 : 3}
```

DSE Search maps the dynamic field to a map collection column.

### Joining cores

DSE Search supports the OS Solr query time join through a custom implementation. You can join search documents, including those having different search indexes under these conditions:

- Search indexes must have the same keyspace and same database partition key.
- Both tables that support the search indexes to be joined must be CQL-compatible.
- The type of the unique key (database key validator of the partition key) are the same in both search documents.
- The order of table partition keys and schema unique keys are the same in both search documents.

Using the simplified syntax automatically takes advantage of joins.

### Simplified syntax

DataStax recommends this simplified syntax to join search indexes:

```java
q={!join fromIndex=test/from)field:value
```
The custom DSE Search implementation does not use the to/from parameters that are required by OS Apache Solr™. Based on the key structure, DSE Search determines the parameters. For backward compatibility with applications, the verbose legacy syntax (page 596) is also supported.

Example of using a query time join

This example creates two tables:

- The songs table uses a simple primary key: the UUID of a song.
- The primary key of the songs table is its partition key.
- The lyrics table uses a compound primary: id and song, both of type UUID.
- Both tables use the same partition key.

After joining search indexes, you can construct a single query to retrieve information about songs having lyrics that include "love".

To join the search indexes:

1. **Download and unzip the file.**
   
   This action creates /songs and /lyrics directories, schemas, and config files for indexing data in the songs and lyrics tables.

2. **Start cqlsh, and then create and use a keyspace named internet.**
   
   You can copy from the downloaded commands.txt file.

3. Create two tables, song and lyrics, that share the internet keyspace and use the same partition key.

   ```
   cqlsh> CREATE TABLE songs (song uuid PRIMARY KEY, title text, artist text);
   cqlsh> CREATE TABLE lyrics (song uuid, id uuid, words text, PRIMARY KEY (song, id));
   ```

   Both tables share the song partition key, a uuid. The second table also contains the id clustering column.

4. Insert the data from the downloaded file into the songs table.

5. Insert data into the lyrics table.
   
   The lyrics of songs by Big Data and John Cedrick mention love.

6. Navigate to the songs directory that you created in step 1, and take a look at the schema.xml. Navigate to the lyrics directory and take a look at the schema. Notice that the order of the unique key in the schema and the partition key of the lyrics table are the same: (song, id). Using (id, song) does not work.
7. In the songs directory, create the search index config and schema for the internet.songs table.

8. In the lyrics directory, create the search index config and schema for the internet.lyrics core, and create the search core for internet.lyrics.

9. Search for songs that have lyrics about love.

```
http://localhost:8983/solr/internet.songs/select/?q={!join +fromIndex=internet.lyrics}words:love&indent=true&wt=json
```

The output includes two songs having the word "love" in the lyrics, one by Big Data and the other by John Cedrick:

```
"response":{"numFound":2,"start":0,"docs":[
```
Recursive join support

You can nest a join query to use the result of one join as an input for another join, and another, recursively. All joined data must reside on the same partition. To embed one query in the query string of another, use the magic field name _query_.

Use this syntax to construct a query that recursively joins search indexes:

```cql
F1:V1 AND _query_:"{!join fromIndex=keyspace.table}(F2:V2 AND _query_: 
"{!join fromIndex=keyspace.table}(F3:V3)\")"
```

Where the top level from query includes a nested join query. The nested join in this example is:

```cql
_query_:"{!join fromIndex=keyspace.table}(F3:V3)\"
```

Like an SQL SELECT IN ... (SELECT IN ...) query, the nested join queries run first, enabling multiple nested join queries if required.

**Note:** A join query is not a relational join where the values from the nested join queries are returned in the results.

Example of a recursive join query

This example builds on the solr query time join example. Embed in the query to join songs and lyrics having words:"love" a second query to join award-winning videos using AND _query_:"award:true".

You can copy CQL commands, Solr HTTP requests, and the query from the downloaded commands.txt file.

1. In cqlsh, create a videos table that shares the internet keyspace and uses the same partition key as the songs and lyrics tables.

   ```cql
   cqlsh> CREATE TABLE videos (song uuid, award boolean, title text, PRIMARY KEY (song));
   ```

   All three tables use the song partition key, a uuid.
2. Insert the data from the downloaded file into the videos table. The video data sets the award field to true for the videos featuring songs by Big Data and Brad Paisley.

3. Navigate to the videos directory that was created when you unzipped the downloaded file.

4. In the videos directory, post solrconfig.xml and schema.xml, and create the Search core for internet.videos.

5. Use a nested join query to recursively join the songs and lyrics documents with the videos document, and to select the song that mentions love and also won a video award.

```
http://localhost:8983/solr/internet.songs/select/?q=
  {!join+fromIndex=internet.lyrics}words:love AND _query_: 
  {!join +fromIndex=internet.videos}award:true&indent=true&wt=json
```

Output is:

```
"response":{"numFound":1,"start":0,"docs":[
  {
    "song":"a3e64f8f-bd44-4f28-b8d9-6938726e34d4",
    "title":"Dangerous",
    "artist":"Big Data"}]
}
```

Support for the legacy join query

DataStax Enterprise supports using the legacy syntax that includes to/from fields in the query. The requirements for using the legacy syntax are:

- Tables do not use composite partition key.
- The query includes the force=true local parser parameter, as shown in this example that joins mytable1 and mytable2 in mykeyspace.

**Legacy syntax example**

```
curl 'http://localhost:8983/solr/mykeyspace.mytable1/select/?q=
  {!join+from=id+to=id+fromIndex=mykeyspace.mytable2+force=true}'
```

**Spatial queries with polygons require JTS**

JTS (Java Topology Suite) is required to index polygon/multipolygon and perform queries that include polygon shapes. Dynamic fields for spatial subfields use prefix naming conventions (page 591) to enable map types to store geospatial data. DSE Search includes the Apache Solr™ Spatial4j library that adds advanced spatial types like polygons to search indexes.
Spatial field type with JTS enabled

For optimal indexing of multipolygon shapes, you must set useJtsMulti="false". For example:

```xml
<fieldType autoIndex="true" useJtsMulti="false"
class="solr.SpatialRecursivePrefixTreeFieldType" distErrPct="0.0125"
distanceUnits="kilometers" geo="true" name="WktField"
spatialContextFactory="org.locationtech.spatial4j.context.jts.JtsSpatialContextFactory"/>
```

Advanced spatial queries

Performing spatial queries that include polygon shapes requires installing the JTS (Java Topology Suite) library into the DataStax Enterprise Solr library directory. Download version 1.13 of the jts.jar file from http://central.maven.org/maven2/com/vividsolutions/jts/1.13/ and install in the Solr library path:

The default Solr library path location depends on the type of installation:

- Package installations and Installer-Services: /usr/share/dse/solr/lib
- Tarball installations and Installer-No Services: installation_location/resources/solr/lib

Spatial predicates

DSE Search supports these spatial predicates:

- Intersects
- IsWithin
- IsDisjointTo
- Contains

Examples

Intersects

`fq=geo:"Intersects(-74.093 41.042 -69.347 44.558)"`

IsWithin

`fq=geo:"IsWithin(POLYGON((-10 30, -40 40, -10 -20, 40 20, 0 0, -10 30))) distErrPct=0"`

IsDisjointTo

`fq=geo:"IsDisjointTo(POLYGON((-10 30, -40 40, -10 -20, 40 20, 0 0, -10 30))) distErrPct=0"`

Contains
Using DataStax Enterprise advanced functionality

```sql
fq=geo:"Contains(POLYGON((-10 30, -40 40, -10 -20, 40 20, 0 0, -10 30)))
distErrPct=0"
```

**Limiting queries by time**

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.

Example with a 30 second timeout:

```sql
SELECT * FROM users where solr_query = '{ "q": ":*:*",  "timeAllowed":30000}';
```

**UDT query examples**

You can query nested tuples and UDTs (page 558) inside CQL lists and sets. A UDT facilitates handling multiple fields of related information in a table. UDTs are a specialization of tuples. In these examples, `{!tuple}` applies to both UDTs and tuples.

**Note:** Selecting an entire UDT column in the CQL SELECT clause is supported. Selecting individual fields of a UDT is supported for unfrozen tuples.

**Querying fields**

```cql
{!tuple}address.street:sesame
```

**Querying dynamic fields**

```cql
<dynamicField name="user.position_1" type="text" indexed="true" stored=true/>
{!tuple}user.position_day1:second
{!tuple}user.position_day2:first
```

**Querying collections**

```cql
{!tuple}user.hobbies:swim
```

**Querying across different UDT/tuple fields**

```cql
+{!tuple v='father.name.firstname:Sam'} +{!tuple
v='mother.name.firstname:Anne'}
```
In CQL, to use a single quotation mark in a string literal, you must escape it using a single quotation mark (so you'll need to double the single quotation marks). See CQL escaping characters.

You can also use the discouraged syntax:

```cql
{{tuple}father.name.firstname:Sam AND {{tuple}mother.name.firstname:Anne}
```

**Querying UDT/tuple fields with several conditions**

You can find a tuple that satisfies several conditions. Notice how all the conditions are on the same tuple all the time. For example:

```cql
{{tuple v='address.residents.field1:Alice AND address.residents.field2:Smith'}
```

You can also use the discouraged syntax:

```cql
{{tuple}address.residents.field1:Alice AND address.residents.field2:Smith
```

The difference in syntax specifies to search across tuples or within a tuple.

- **Across tuples:**
  
  `+{{tuple v='condition1'}} +{{tuple v='condition2'}} +{{tuple v='conditionN'}}` searches for documents that satisfy all conditions, but are not necessarily satisfied by the same single tuple/UDT.

- **Within a tuple:**
  
  `{{tuple v='condition1 AND condition2 AND conditionN'}}` searches for documents that satisfy all conditions within a single tuple/UDT.

**Querying nested tuples and UDTs**

To query nested tuples and UDTs, use the same dot notation and the tuple query parser. Because UDTs are a specialization of tuples, use the tuple query parser for tuples and UDTs. In this example, the dot notation identifies `address.resident` as a UDT.

Query for locations that have a resident with the first name *Alice* using the nested `address.residents` tuple:

```cql
{{tuple}address.residents.field1:Alice
```

Query for locations with a resident that has the first name *Alice* and second name *Smith*:

```cql
+{{tuple v='address.residents.field1:Alice AND address.residents.field2:Smith'}}
```
Using DataStax Enterprise advanced functionality

**Note:** Tuples and UDTs are modelled internally as nested documents. The Apache Solr™ block join is used internally to query them. Parents are identified with the `_parent_=true` field. Children are identified with `_parent_=false`. For certain types of queries, including negative queries and empty field queries, you might need to use the `_parent_` field.

**Querying for empty firstnames**

The negation (-) and inclusion (+) operators must precede the `{!tuple}` directive:

```
-({!tuple}_parent_:false AND user.name.firstname:[* TO *])
```

**Negative queries**

Negative queries use this syntax:

```
select * from demo where solr_query='-{!tuple}name.firstname:*
```

Negative queries with more than one condition must follow the Solr rules. Use this syntax:

```
{!tuple v='address.street:* NOT (address.street:sesame AND address.number:32)'}
```

or

```
-{!tuple v='address.street:sesame AND address.number:32'}
```

or

```
{!tuple}address.street:* NOT (address.street:sesame AND address.number:32)
```

**Querying CQL collections**

DSE Search supports CQL collections. In this example, you create a table containing a CQL set collection of famous quotations.

1. Start DataStax Enterprise *(page 1090)* as a DSE Search node.

2. Start `cqlsh`.

3. Create a keyspace and a table for a collection column and other columns, and then insert data.

```
CREATE KEYSPACE mykeyspace
  WITH REPLICAATION = {'class':'NetworkTopologyStrategy', 'Solr':1};

USE mykeyspace;

CREATE TABLE mysolr (
  id text PRIMARY KEY,
  name text,
...
4. **Download** the quotations.zip file.

5. Extract the quotations.zip file, copy the insert commands, and paste each command on the cqlsh command line.

6. Run the following command, which is located in the bin directory of tarball installations. For example, from a tarball installation:

   ```bash
   $ installation_location/bin/dsetool create_core mykeyspace.mysolr
genenerateResources=true reindex=true
   
   If you are recreating the mykeyspace.mysolr core, use the `reload_core` command instead of the `create_core` command.
   There is no output from this command. You can search data after indexing finishes.
   ```

7. In cqlsh, search the indexed data to find quotes like succ*.

   ```sql
   SELECT * FROM mykeyspace.mysolr WHERE solr_query='quotes:succ*';
   ```

   Because you created the core using automatically generated resources, the search index config defines the request handler for using CQL for search queries.

8. Using a browser, search-indexed data using the Solr HTTP API to find titles like succ*.

   ```plaintext
   http://localhost:8983/solr/mykeyspace.mysolr/
   select?q=quotes%3Asucc*&wt=json&indent=on&omitHeader=on
   ```

   ```json
   {  
   "response":{"numFound":2,"start":0,"docs":[
   {  
   "id":"126","title":"Success","quotes":["If A is success in life, then A equals x plus y plus z. Work is x; y is play; and z is keeping your mouth shut."]},  
   "name":"Albert Einstein"},  
   {  
   "id":"125","title":"Success","quotes":["Always bear in mind that your own resolution to succeed is more important than any one thing.",  
   "Better to remain silent and be thought a fool than to speak out and remove all doubt."],  
   "name":"Abraham Lincoln"}
   ]
   }
   ```
**Using date ranges in solr_query**

The Solr `DateRangeField` is supported in DSE Search with mapping of Solr DateRangeField to the CQL type DateRangeType.

The CQL type DateRangeType is supported for use with the latest Java driver, the DSE Python driver, and cqlsh commands.

**Overriding the default TimeZone (UTC) in search queries**

Specify the TZ parameter to overwrite the default TimeZone (UTC) that is used for adding and rounding in date math. The local rules for the specified time zone, including the start and end of daylight saving time (DST) if any, determine when each arbitrary day starts. The time zone rules impact the rounding and adding of DAYs, but also cascades to rounding of HOUR, MIN, MONTH, and YEAR. For example, specifying a different time zone changes the result:

<table>
<thead>
<tr>
<th>Date math</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-03-10T12:34:56Z/YEAR</td>
<td>Default TZ</td>
</tr>
<tr>
<td></td>
<td>2016-01-01T00:00:00Z</td>
</tr>
<tr>
<td></td>
<td>TZ=America/Los_Angeles</td>
</tr>
<tr>
<td></td>
<td>2016-01-01T08:00:00Z</td>
</tr>
<tr>
<td>2016-03-10T08:00:002+1DAY</td>
<td>Default TZ</td>
</tr>
<tr>
<td></td>
<td>2016-03-11T08:00:002</td>
</tr>
<tr>
<td></td>
<td>TZ=America/Los_Angeles</td>
</tr>
<tr>
<td></td>
<td>2016-03-11T07:00:00Z</td>
</tr>
</tbody>
</table>

The value of the TZ parameter can be any zone ID that is supported by the java TimeZone class.

**Primary key or ordinary column**

DateRangeType can be used as a primary key or ordinary column:

```cql
CREATE TABLE taxi_trips(id int PRIMARY KEY, pickup_dropoff_range 'DateRangeType');
```

```cql
CREATE TABLE weather_sensors(weatherstation_id text, event_time 'DateRangeType', temperature text, PRIMARY KEY (weatherstation_id, event_time));
```

**CQL representation**

The CQL representation uses the same syntax as Solr DateRangeField:
Using DataStax Enterprise advanced functionality

```sql
INSERT INTO taxi_trips(id, pickup_dropoff_range) VALUES (1, '[2017-02-02T14:57:00 TO 2017-02-02T15:10:17]');
INSERT INTO taxi_trips(id, pickup_dropoff_range) VALUES (2, '[2017-02-01T09:00:03 TO 2017-02-01T09:32:00.001]');
INSERT INTO taxi_trips(id, pickup_dropoff_range) VALUES (3, '[2017-02-03T12:10:01.358 TO 2017-02-03T12:19:57]');
```

**dateTime precision**

The dateTime precision is preserved from user input. Milliseconds are displayed only when millisecond precision is provided on input.

```sql
SELECT * FROM taxi_trips;
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>pickup_dropoff_range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[2017-02-02T14:57:00Z TO 2017-02-02T15:10:17Z]</td>
</tr>
<tr>
<td>2</td>
<td>[2017-02-01T09:00:03Z TO 2017-02-01T09:32:00.001Z]</td>
</tr>
<tr>
<td>3</td>
<td>[2017-02-03T12:10:01.358Z TO 2017-02-03T12:19:57Z]</td>
</tr>
</tbody>
</table>
```

Create search index:

```sql
CREATE SEARCH INDEX ON taxi_trips;
```

Select all trips from February 2017:

```sql
SELECT * FROM taxi_trips WHERE solr_query = 'pickup_dropoff_range:2017-02';
```

Select all trips started after 2017-02-01 12:00 PM (inclusive) and ended before 2017-02-02 (inclusive):

```sql
SELECT * FROM taxi_trips WHERE solr_query = 'pickup_dropoff_range:[2017-02-01T12 TO 2017-02-02]';
```

Select all trips started after 2017-02-01 12:00 PM (inclusive) and ended before 2017-02-01:23:59:59.999 (inclusive):

```sql
SELECT * FROM taxi_trips WHERE solr_query = 'pickup_dropoff_range:[2017-02-01T12 TO 2017-02-01]';
```

**Single point in time**

**DateRangeField** can represent a single point in time:

```sql
INSERT INTO weather_sensors (weatherstation_id, event_time, temperature)
VALUES ('A1', '2017-10-02T00:00:05', '12C');
INSERT INTO weather_sensors (weatherstation_id, event_time, temperature)
VALUES ('A1', '2017-10-02T00:00:10', '12C');
```
Using DataStax Enterprise advanced functionality

INSERT INTO weather_sensors (weatherstation_id, event_time, temperature) VALUES ('A1', '2017-10-02T00:00:15', '13C');

INSERT INTO weather_sensors (weatherstation_id, event_time, temperature) VALUES ('A1', '2017-10-02T00:00:20', '13C');

INSERT INTO weather_sensors (weatherstation_id, event_time, temperature) VALUES ('A1', '2017-10-02T00:00:25', '12C');

Select all from weather_sensors:

```
SELECT * FROM weather_sensors;
```

```
weatherstation_id | event_time           | temperature
-------------------+----------------------+------------
A1 | 2017-10-02T00:00:05Z |         12C
A1 | 2017-10-02T00:00:10Z |         12C
A1 | 2017-10-02T00:00:15Z |         13C
A1 | 2017-10-02T00:00:20Z |         13C
A1 | 2017-10-02T00:00:25Z |         12C
```

Create a search index on weather_sensors:

```
CREATE SEARCH INDEX ON weather_sensors;
```

Select a specific point in time:

```
SELECT * FROM weather_sensors WHERE solr_query = 'event_time: [2017-10-02T00:00:10 TO 2017-10-02T00:00:20]';
```

```
weatherstation_id | event_time           | solr_query | temperature
-------------------+----------------------+------------+-------------
A1 | 2017-10-02T00:00:10Z |       null |         12C
A1 | 2017-10-02T00:00:15Z |       null |         13C
A1 | 2017-10-02T00:00:20Z |       null |         13C
```

Open bounds

DateRangeField can have open bounds.

Select from a point in time to an open bound:

```
SELECT * FROM weather_sensors WHERE solr_query = 'event_time: [2017-10-02T00:00:10 TO *]';
```

```
weatherstation_id | event_time           | solr_query | temperature
-------------------+----------------------+------------+-------------
A1 | 2017-10-02T00:00:25Z |       null |         12C
A1 | 2017-10-02T00:00:10Z |       null |         12C
A1 | 2017-10-02T00:00:15Z |       null |         13C
A1 | 2017-10-02T00:00:20Z |       null |         13C
```

Select from an open bound up to a point in time:
Using DataStax Enterprise advanced functionality

```sql
SELECT * FROM weather_sensors WHERE solr_query = 'event_time:[* TO 2017-10-02T00:00:20]';
```

| weatherstation_id | event_time           | solr_query | temperature |
|-------------------+----------------------+------------+-------------|
| A1                | 2017-10-02T00:00:10Z | null       | 12C         |
| A1                | 2017-10-02T00:00:15Z | null       | 13C         |
| A1                | 2017-10-02T00:00:20Z | null       | 13C         |
| A1                | 2017-10-02T00:00:05Z | null       | 12C         |

Select from all points in time:

```sql
SELECT * FROM weather_sensors WHERE solr_query = 'event_time:[* TO *]';
```

| weatherstation_id | event_time           | solr_query | temperature |
|-------------------+----------------------+------------+-------------|
| A1                | 2017-10-02T00:00:25Z | null       | 12C         |
| A1                | 2017-10-02T00:00:10Z | null       | 12C         |
| A1                | 2017-10-02T00:00:15Z | null       | 13C         |
| A1                | 2017-10-02T00:00:20Z | null       | 13C         |
| A1                | 2017-10-02T00:00:05Z | null       | 12C         |

Insert an open-bounded range into a table:

```sql
INSERT INTO weather_sensors (weatherstation_id, event_time, temperature)
VALUES ('A1', '[2017-10-02T00:00:25 TO *]', '12C');
SELECT * FROM weather_sensors WHERE solr_query = 'event_time:[* TO *]';
```

| weatherstation_id | event_time           | solr_query | temperature |
|-------------------+----------------------+------------+-------------|
| A1                | 2017-10-02T00:00:25Z | null       | 12C         |
| A1                | 2017-10-02T00:00:10Z | null       | 12C         |
| A1                | 2017-10-02T00:00:15Z | null       | 13C         |
| A1                | 2017-10-02T00:00:20Z | null       | 13C         |
| A1                | 2017-10-02T00:00:05Z | null       | 12C         |
| A1                | [2017-10-02T00:00:25Z TO *] | null   | 12C         |

**Restricted query routing**

This feature is for experts only and should be used with care.

DSE Search restricted query routing is designed for applications that have a data model that supports restricting common queries to a single partition.
**Important:** route.partition and route.range filter only which endpoints to query.

To restrict queries to a token or partition, use the CQL solr_query (page 572) instead.

For example:

```plaintext
SELECT aid, bkt, ts, rid, mt FROM tt.accounttransactions WHERE aid=1096 AND bkt=0 AND solr_query='{"q":"*:*", "sort":"ts asc"}'}
```

to filter:

' ; '

**Partition key routing**

You can restrict routing queries to a limited number of nodes based on a list of partition keys. You can also restrict queries based on a single token range. To specify routing by partition keys, use the route.partition query parameter and set its value to one or more partition keys. DSE Search queries only the nodes that own the given partition keys. The vertical line delimiter separates components of a composite key. The comma delimiter separates different partition keys.

For example:

```plaintext
route.partition=k1c1|k1c2,k2c1|k2c2 . . .
```

If the actual partition key value contains a delimiter character, use a backslash character to escape the delimiter.

**Examples**

You can route Solr HTTP API (page 641) and CQL queries (page 572). This example shows how to use the route queries on a table with a composite partition key, where "nike" and "2" are composite key parts.

```plaintext
http://localhost:8983/solr/test.route/select?
q=*:*&indent=true&shards.info=true&route.partition=nike|2,reebok|2
```

In CQL:

```plaintext
SELECT * FROM test.route WHERE solr_query='{"q" : "*:*",
"route.partition" : ["nike|2","reebok|2"]}'
```

**Token range routing**

Only use token range routing if you thoroughly understand cluster token placement. For simplicity, DataStax recommends routing queries by partition range instead of routing by token range. To specify routing by token range, use the route.range query parameter and set its value to the two token values that represent the range, separated by comma.
For example:

```plaintext
route.range=t1,t2
```

DSE Search queries only the nodes in the given token range.

**Tutorials and demos**

Use these step-by-step tutorials with sample keyspaces, tables, and data that demonstrate DSE Search index functionality.

Useful external resources:

- [Tutorial](#) how to index and query geospatial Polygons and MultiPolygons.
- [Docker container](#) for running Silk on DSE Search.

**Creating a demo keyspace for tutorials**

Step-by-step instructions to create a keyspace for tutorials found in this section.

1. Get a list of datacenter names (DC) in the cluster.

   ```bash
   $ dsetool status
   ```

   The header line contains the datacenter name (DC: *datacenter_name*) and the type of workload.

   ```plaintext
   DC: Cassandra       Workload: Cassandra       Graph: no
   ==---------------------------------------------------------------------==
   ...                     ...                     ...
   DC: Solr            Workload: Search          Graph: no
   ==---------------------------------------------------------------------==
   ...                     ...                     ...
   ```

2. Start a cqlsh session:

   ```bash
   cqlsh
   ```

   **Tip:** To connect cqlsh to a remote node use the host switch with the hostname or IP address.

3. Create a demo keyspace with a replication factor of 1 in each datacenter.

   In multi-datacenter environments use [NetworkTopologyStrategy](#) and set the replication factor for each datacenter to at least one.

   ```cql
   CREATE KEYSSPACE demo
   WITH replication = {
     'class': 'NetworkTopologyStrategy',
   }
   ```
Multi-faceted search using healthcare data

This quick start example provides an overview of creating and altering search indexes using CQL index management commands.

Prerequisites:

1. Create a demo keyspace with a replication factor of at least 1 in the search datacenter, see Creating a demo keyspace for tutorials (page 607).

2. Download the health_data.csv onto a search node.

1. Launch cqlsh on a search node:

   a. Determine which nodes in the cluster are running a search workload:

      

```bash
$ dsetool status
```

   Tip: DSE Search operations are available only on search-enabled nodes. DataStax recommends single workload datacenters.

The following example shows a development environment where all nodes in the cluster are in the same physical location, on the same rack, and the nodes have been separated into datacenters based on their workloads:

<table>
<thead>
<tr>
<th>DC: Main</th>
<th>Workload: Cassandra</th>
<th>Graph: no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status=Up/Down</td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="status.png" alt="Status" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State=Normal/Leaving/Joining/Moving</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-- Address Load Owns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VNodes Rack Health [0,1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN 10.10.10.111 15.51 MiB ? 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rack1 0.90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN 10.10.10.113 19.51 MiB ? 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rack1 0.90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DC: Search</th>
<th>Workload: Search</th>
<th>Graph: no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status=Up/Down</td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="status.png" alt="Status" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State=Normal/Leaving/Joining/Moving</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-- Address Load Owns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VNodes Rack Health [0,1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN 10.10.10.108 18.13 MiB ? 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rack1 0.90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
b. Launch a cqlsh session on a search node from the directory that contains the health_data.csv:

```bash
$ cd ~ && cqlsh -k demo
```

A CQL sessions starts using the demo keyspace (page 607).

Connected to cluster1 at 10.10.10.108:9042.
[cqlsh 5.0.1 | Cassandra 3.11.0.1805 | DSE 5.1.3 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh:demo>

Tip: The active keyspace name appears in the cqlsh prompt.

2. Set up the health_data table with data:

   a. Create the table:

   ```cql
   // START-table
   CREATE TABLE IF NOT EXISTS demo.health_data (  
      "id" INT,
      "num_smokers" INT,
      "age" INT,
      "age_unit" VARCHAR,
      "age_months" INT,
      "major_medical_coverage" VARCHAR,
      "dental_coverage" VARCHAR,
      "routine_medical_coverage" VARCHAR,
      "employer_paid_plan" VARCHAR,
      "secondary_smoke" VARCHAR,
      "county" VARCHAR,
      "screening_month" VARCHAR,
      "pets" VARCHAR,
      "asthma" VARCHAR,
      "bronchitis" VARCHAR,
      "goiter" VARCHAR,
      "hay_fever" VARCHAR,
      "thyroid_disease" VARCHAR,
      "chronic_bronchitis" VARCHAR,
      "diagnosed_asthma" VARCHAR,
      "diagnosed_cataracts" VARCHAR,
      "diagnosed_emphysema" VARCHAR,
      "diagnosed_goiter" VARCHAR,
      "diagnosed_gout" VARCHAR,
      "diagnosed_hay_fever" VARCHAR,
      "diagnosed_lupus" VARCHAR,
   ```
Using DataStax Enterprise advanced functionality

```sql
"diagnosed_other_cancer" VARCHAR,
"diagnosed_skin_cancer" VARCHAR,
"diagnosed_stroke" VARCHAR,
"diagnosed_thyroid_disease" VARCHAR,
"diagnosed_congestive_heart_failure" VARCHAR,
"ethnicity" VARCHAR,
"exam_status" VARCHAR,
"family_sequence" INT,
"family_size" INT,
"fips" VARCHAR,
"grade_completed" VARCHAR,
"household_size" INT,
"health_status" VARCHAR,
"marital_status" VARCHAR,
"bird" VARCHAR,
"cat" VARCHAR,
"dog" VARCHAR,
"fish" VARCHAR,
"other_pet" VARCHAR,
"race" VARCHAR,
"race_ethnicity" VARCHAR,
"gender" VARCHAR,
"birthplace" VARCHAR,
"annual_income_20000" VARCHAR,
"income_group" INT,
"monthly_income_total" INT,
PRIMARY KEY ("id", "age")
WITH gc_grace_seconds = 0;
// END-table

// START-data
copy health_data (  
"id",
"num_smokers",
"age",
"age_unit",
"age_months",
"major_medical_coverage",
"dental_coverage",
"routine_medical_coverage",
"employer_paid_plan",
"secondary_smoke",
"county",
"screening_month",
"pets",
"asthma",
"bronchitis",
"goiter",
"hay_fever",
"thyroid_disease",
"chronic_bronchitis",
"diagnosed_asthma",
"diagnosed_cataracts",
"diagnosed_emphysema",
```
Using DataStax Enterprise advanced functionality

```sql
"diagnosed_goiter",
"diagnosed_gout",
"diagnosed_hay_fever",
"diagnosed_lupus",
"diagnosed_other_cancer",
"diagnosed_skin_cancer",
"diagnosed_stroke",
"diagnosed_thyroid_disease",
"diagnosed_congestive_heart_failure",
"ethnicity",
"exam_status",
"family_sequence",
"family_size",
"fips",
"grade_completed",
"household_size",
"health_status",
"marital_status",
"bird",
"cat",
"dog",
"fish",
"other_pet",
"race",
"race_ethnicity",
"gender",
"birthplace",
"annual_income_20000",
"income_group",
"monthly_income_total")
FROM 'health_data.csv';

// END-data
```

**Tip:** After loading data that contains null values, temporarily set the grace period to zero to clean up tombstones.

b. Load data from the CSV file:

```sql
copy health_data (  
  "id",
  "num_smokers",
  "age",
  "age_unit",
  "age_months",
  "major_medical_coverage",
  "dental_coverage",
  "routine_medical_coverage",
  "employer_paid_plan",
  "secondary_smoke",
  "county",
  "screening_month",
  "pets",
```
"asthma",
"bronchitis",
"goiter",
"hay_fever",
"thyroid_disease",
"chronic_bronchitis",
"diagnosed_asthma",
"diagnosed_cataracts",
"diagnosed_emphysema",
"diagnosed_goiter",
"diagnosed_gout",
"diagnosed_hay_fever",
"diagnosed_lupus",
"diagnosed_other_cancer",
"diagnosed_skin_cancer",
"diagnosed_stroke",
"diagnosed_thyroid_disease",
"diagnosed_congestive_heart_failure",
"ethnicity",
"exam_status",
"family_sequence",
"family_size",
"fips",
"grade_completed",
"household_size",
"health_status",
"marital_status",
"bird",
"cat",
"dog",
"fish",
"other_pet",
"race",
"race_ethnicity",
"gender",
"birthplace",
"annual_income_20000",
"income_group",
"monthly_income_total")
FROM 'health_data.csv';

**Note:** If health_data.csv is not in the directory where you launch cqlsh, specify the full path to the file.

The script loads 20050 with no rows skipped.

Using 1 child processes

Starting copy of demo.health_data with columns
[id, num_smokers, age, age_unit, age_months, major_medical_coverage, dental_coverage, routine_medical_coverage, employer_paid_plan, secondary_smoke, county, screening_month,
c. Verify the number of records:

```
SELECT COUNT(*) FROM demo.health_data;
```

```
count
-------
 20050

(1 rows)
```

Warnings:
Aggregation query used without partition key

3. Create the search index:

```
CREATE SEARCH INDEX ON demo.health_data
WITH COLUMNS * {excluded:false}, age_months, monthly_income_total
{excluded: true}
AND PROFILES spaceSavingNoTextfield;
```

- Only columns identified in the COLUMNS options are included. All columns are included when this option is omitted.

4. Display the schema:

```
DESCRIBE ACTIVE SEARCH INDEX SCHEMA ON demo.health_data;
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
  <types>
    <fieldType class="org.apache.solr.schema.StrField" name="StrField"/>
```
<fieldType class="org.apache.solr.schema.TrieIntField"
name="TrieIntField"/>
</types>

<fields>
    <field indexed="true" multiValued="false"
name="grade_completed" stored="true" type="StrField"/>
    <field indexed="true" multiValued="false"
name="diagnosed_thyroid_disease" stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="pets"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="secondary_smoke"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="diagnosed_lupus"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="gender"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="birthplace"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="income_group"
stored="true" type="TrieIntField"/>
    <field indexed="true" multiValued="false" name="marital_status"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="age_months"
stored="true" type="TrieIntField"/>
    <field indexed="true" multiValued="false" name="bird"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="hay_fever"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="diagnosed_hay_fever"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="routine_medical_coverage"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="annual_income_20000"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="exam_status"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="other_pet"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="diagnosed_stroke"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="employer_paid_plan"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="family_sequence"
stored="true" type="TrieIntField"/>
    <field indexed="true" multiValued="false" name="diagnosed_cataracts"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="major_medical_coverage"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="diagnosed_gout"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="age_unit"
stored="true" type="StrField"/>
    <field indexed="true" multiValued="false" name="goiter"
stored="true" type="StrField"/>
Quick Start for CQL index management

This quick start example provides an overview of creating and altering search indexes using CQL index management commands.

Creating a search index

1. Launch cqlsh and create a tutorial keyspace on a solr node:

```cqlsh
```

2. Set up the table schema and create a default index:

3. Create the keyspace, create the table, and create the search index on the users table from the KillrVideo application:

```cql
CREATE KEYSPACE demo WITH replication = {
    'class': 'SimpleStrategy',
    'replication_factor': 1};
CREATE TABLE demo.users ( userid uuid,
    firstname text,
    lastname text,
    email text,
    created_date timestamp,
    PRIMARY KEY (userid));
CREATE SEARCH INDEX ON demo.users;
```

A new search index is generated on the table. Existing data is reindexed.

4. Use the CQL shell DESCRIBE SEARCH INDEX SCHEMA View the pending search index schema

The generated schema looks like this:

```xml
<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>
    </types>
</schema>
```
5. To increase tolerance of non-ASCII characters in the name field, add a new fieldType to the schema with this ALTER SEARCH INDEX SCHEMA statement:

```sql
ALTER SEARCH INDEX SCHEMA ON demo.users
ADD types.fieldType [
  @name='TextField_intl' ,
  @class='org.apache.solr.schema.TextField' ]
WITH $$
  "analyzer": [ {
    "type": "index",
    "tokenizer": { "class": "solr.StandardTokenizerFactory" },
    "filter": [ { "class": "solr.LowerCaseFilterFactory" },
                { "class": "solr.ASCIIFoldingFilterFactory" } ]
  },
  { "type": "search",
    "tokenizer": { "class": "solr.StandardTokenizerFactory" },
    "filter": [ { "class": "solr.LowerCaseFilterFactory" },
                { "class": "solr.ASCIIFoldingFilterFactory" } ]
  }
]$$;
```

The dollar signs ($$) syntax in the ALTER SEARCH INDEX SCHEMA example are dollar quotes to escape a single quotation mark, see Escaping single quotation marks (page 580). This new fieldType has separate index and search analysis phases:
Using DataStax Enterprise advanced functionality

6. Change the type on the firstname and lastname fields:

   ```
   ALTER SEARCH INDEX SCHEMA ON demo.users
   SET field[@name='firstname']@type = 'TextField_intl';
   ```

   ```
   ALTER SEARCH INDEX SCHEMA ON demo.users
   SET field[@name='lastname']@type = 'TextField_intl';
   ```

7. In contrast to the dsetool search index management commands, all changes made with
   ALTER SEARCH INDEX affect only the pending resources for the search index. To use
   the changes:

   a. Reload the index.

   ```
   RELOAD SEARCH INDEX ON demo.users;
   ```

   b. If there is existing data in the index, rebuild the index.

   ```
   REBUILD SEARCH INDEX ON demo.users;
   ```

Term and phrase searches using the wikipedia demo

The Wikipedia demo scripts automatically download 3,000+ Wikipedia articles, create a
CQL keyspace and table, insert the articles, and create a search index on both the title and
body columns.

Prerequisites: The demo scripts connect to the localhost on the Solr port. Ensure that the
Solr interface and port 127.0.0.1:8983 are accessible.

1. Start DataStax Enterprise (page 1090) as a search node.

2. Go to installation_directory/demos/wikipedia.
3. Run the script to add the wikipedia schema:

```bash
$ ./1-add-schema.sh
```

This script creates the wiki keyspace with a single table solr.

4. To use the demo in a cluster that has more than one node, change the keyspace replication from SimpleStrategy to NetworkTopologyStrategy, and set the factor to 1 in each datacenter:

```bash
$ cqlsh -e 'ALTER KEYSPACE wiki WITH replication = {'class': 'NetworkTopologyStrategy', 'Cassandra' : 1, 'Solr' : 1};
```

In this example, the cluster has two datacenters, Cassandra and Solr. Datacenter names are case sensitive.

5. Load the data and index the table using the second script (2-index.sh).

```bash
$ ./2-index.sh --wikifile wikipedia-sample.bz2
```

3,000 articles are loaded into the solr table and then indexed.

```bash
Start indexing wikipedia...
------------> config properties:
docs.file = wikipedia-sample.bz2
keep.image.only.docs = false
-------------------------------
Indexed 1000
Indexed 2000
Indexed 3000
Finished
Visit http://localhost:8983/demos/wiki/ to see data
```

6. Verify that the data was successfully loaded into the keyspace/table:

```bash
$ cqlsh -e 'DESC KEYSPACE wiki; SELECT count(*) FROM wiki.solr;'
```

The results show the details of the keyspace, table schema, index settings, and number of articles.

```sql
CREATE KEYSPACE wiki WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'} AND durable_writes = true;

CREATE TABLE wiki.solr (id text PRIMARY KEY, body text, date text, solr_query text, title text) WITH bloom_filter_fp_chance = 0.01
```
AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
AND comment = ''
AND compaction = {'class':
    'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
    'max_threshold': '32', 'min_threshold': '4'}
AND compression = {'chunk_length_in_kb': '64', 'class':
    'org.apache.cassandra.io.compress.LZ4Compressor'}
AND crc_check_chance = 1.0
AND dcllocal_read_repair_chance = 0.1
AND default_time_to_live = 0
AND gc_grace_seconds = 864000
AND max_index_interval = 2048
AND memtable_flush_period_in_ms = 0
AND min_index_interval = 128
AND read_repair_chance = 0.0
AND speculative_retry = '99PERCENTILE';
CREATE CUSTOM INDEX wiki_solr_solr_query_index ON wiki.solr
(solr_query) USING
 'com.datastax.bdp.search.solr.Cql3SolrSecondaryIndex';

$ cqlsh -k wiki

CQL shell session starts on the localhost in the wiki keyspace.

  Connected to pw-search at 127.0.0.1:9042.
  [cqlsh 5.0.1 | Cassandra 3.11.0.1805 | DSE 5.1.3 | CQL spec 3.4.4
   | Native protocol v4]
  Use HELP for help.
cqlsh:wiki>

8. Disable paging, for faster query results on small data sets:

PAGING off

**Note:** Paging is turned off only for the session. Paging is enabled after a restart. Use a cqlshrc file to change the default startup parameters for cqlsh.

Disabled Query paging.
9. Display the solr table search index schema:

```
DESCRIBE ACTIVE SEARCH INDEX SCHEMA ON solr;
```

```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.WikipediaTokenizerFactory"/>
      </analyzer>
    </fieldType>
    <fieldType class="org.apache.solr.schema.StrField" name="StrField"/>
  </types>
  <fields>
    <field indexed="true" multiValued="false" name="body" stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="title" stored="true" type="TextField"/>
    <field docValues="true" indexed="true" multiValued="false" name="id" stored="true" type="StrField"/>
    <field docValues="true" indexed="true" multiValued="false" name="date" stored="true" type="StrField"/>
  </fields>
  <uniqueKey>id</uniqueKey>
</schema>
```

10. Execute queries against the table using the index:

- Return the titles of articles that contain the word national:

```
SELECT title FROM solr WHERE solr_query='title:national';
```

Seven records are returned.

```
title
---------------------
team 1999 Bolivia national football
team 2000 Bolivia national football
football team Kenya national under-20
team 2001 Bolivia national football
team 2002 Bolivia national football
hockey team Israel men's national inline
```
Using DataStax Enterprise advanced functionality

List of French born footballers who have played for other national teams

(7 rows)

Using secure cluster

You can run the Term and phrase searches using the wikipedia demo (page 618) on a secure cluster.

Kerberos options

• -a enable Kerberos authentication
• -h hostname server hostname (not required if server hostname resolution is correctly set up)

HTTP basic authentication

Use with DseAuthenticator:

• -u username
• -p password

You can use this option to run the shell scripts:

./1-add-schema.sh -u root -p password && ./2-index.sh -u root -p password

SSL options

• -e cert enable HTTPS for client to node encryption, using cert certificate file
• -k disable strict hostname checking for SSL certificates

Indexing and querying polygons

Using United States of America state data, this tutorial demonstrates how to index and query geospatial shapes, such as Polygons and MultiPolygons.

Geospatial data is stored in the database in WKT (Well Known Text) format. To support Polygons and MultiPolygons fields, set the field type in the table index schema to solr.SpatialRecursivePrefixTreeFieldType.

Tip: SpatialRecursivePrefixTreeFieldType supports multiValued spatial data. Most states are describable as Polygons, but others such as Hawaii and Alaska are MultiPolygons.

Polygonal searches use the following geospatial predicates:

• Intersects: Matches if the indexed value overlaps the search criteria.
• IsWithin: Matches if the indexed value completely encapsulates the search criteria.
IsDisjointTo: Matches if the index value does not overlap or touch the search criteria.


1. **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 [Selecting hardware for DataStax Enterprise implementations](#).

   **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 with the index in place. For example, load data with the driver with the consistency level at LOCAL_ONE (CL.LOCAL_ONE) and a sufficiently high write timeout. Use best practices for data loading.

      **Tip:** Use the DataStax Bulk Loader.
After data loading is completed, there might be lag time because indexing is asynchronous.

5. Verify the indexing QueueSize (page 634) 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 = '*:*';
```

**Note:** The COUNT should stabilize within the configured soft-commit period. For example, if the soft-commit is 30 seconds, then it can be up to 30 seconds before the COUNT is correct.

New data is automatically indexed.

Troubleshooting

If the record count does not stabilize:
- If dropped mutations exist in the nodetool tpstats 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**

To view the indexing status for the local node:

```
dsetool core_indexing_status demo.health_data
```

The local node wiki.solr is currently indexing:

```
[demo.health_data]: INDEXING
```

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 -h 200.192.10.11 core_indexing_status --all
```

To view the indexing status with the progress and estimated time of completion:
Using DataStax Enterprise advanced functionality

dsetool core_indexing_status demo.health_data --progress

The results are displayed:

[demo.health_data]: INDEXING, 38% complete, ETA 452303 milliseconds (7 minutes 32 seconds)

Checking the indexing status using the Core Admin

To check the indexing status, open the Solr Admin and click **Core Admin**.

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.solr

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-
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 623).

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.

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 633) to view the progress of indexing tasks.

The StallMetrics is exposed through the metrics registry com.datastax.bdp.metrics.

The following paths identify the MBeans:

- type=search,index=search_index,name=CommitMetrics (page 626)
- type=search,index=search_index,name=MergeMetrics (page 635)
- type=search,index=search_index,name=QueryMetrics (page 636)
- type=search,index=search_index,name=UpdateMetrics (page 637)
- type=search,index=search_index,name=IndexPool (page 633)
- type=search,index=search_index,name=StallMetrics

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 demo.solr search index under search is expanded.
Using DataStax Enterprise advanced functionality

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:

   ```sh
   $ pgrep -f dse
   368
   668
   45706
   ```

   **Tip:** To verify the PID:

   ```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:
Using DataStax Enterprise advanced functionality

$ 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 *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 and Installer-Services: `/usr/share/dse/demos`
   - Tarball installations and Installer-No Services: `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:

   The script then creates the search index by posting to the following location:

   You can override the script defaults by specifying command line parameters:
Using DataStax Enterprise advanced functionality

5. Execute this script to run the benchmark:

```bash
```

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**
  A comma delimited list of servers to run the benchmark against. For example:
  ```bash
  ```
  Default: http://localhost:8983

The demo creates a Search index named demo.solr and indexes 50,000 documents.

Example CQL commands:

```bash
$ ./run-benchmark.sh --url=http://localhost:8983 --testData=resources/testCqlQuery.txt --solrCore=demo.solr
```
Using DataStax Enterprise advanced functionality

```bash
./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:

![Commit metrics MBean](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 630) of a commit operation on the index.

**Main operational phases**

The main phases of a commit operation on the index are:
Using DataStax Enterprise advanced functionality

**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()`
  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.
Using DataStax Enterprise advanced functionality

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**
  Gets the DSE multi-instance server ID for a remote endpoint. Persists between restarts.

- **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**
Using DataStax Enterprise advanced functionality

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 based on a configurable queue threshold and max pause per task. The implementation is re-entrant if a worker submits a task on the pool, the task is not subjected to any pause.

**Path**

The index pool MBean is identified by the following path:

```
com.datastax.bdp.search.keyspace_name.table_name.IndexPool
```

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 328).

BackPressureThreshold
The back pressure threshold is the target total number of queued asynchronous indexing requests per core; the back pressure mechanism will throttle incoming requests to keep the queue size as close to the threshold as possible. In dse.yaml: back_pressure_threshold_per_core (page 328).

Concurrency
The maximum number of concurrent asynchronous indexing threads. In dse.yaml: max_solr_concurrency_per_core (page 328).

IndexPool MBean view-only attributes
You can get the following MBean operations:

BackPressurePauseNanos
Get the average back pressure pause.

BackPressurePauseOutliers
The number of tasks from the scheduled document expiration job or reindexing that pause longer than the configured maximum pause time.

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.

QueueWeight
The total weight of all in-flight indexing tasks.

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. Should be lower than QueueWeight

dse.yaml
The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
</tbody>
</table>
Tarball installations
Installer-No Services installations

installation_location/
resources/dse/conf/
dse.yaml

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`
- `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 636) 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:

**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.
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.

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.

Update metrics MBean
This MBean records the amount of time spent to execute an index update, split by the following main phases:

WRITE
Comprising the time spent to convert the search document and write it into the database (available only when indexing via the SolrJ HTTP APIs).

QUEUE
Comprising the time spent by the index update task into the index pool.

PREPARE
Using DataStax Enterprise advanced functionality

Comprising the time spent preparing the actual index update.

**EXECUTE**

Comprising 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:

- `setEnable(boolean enabled)`
  Enables/disables metrics recording (enabled by default).

- `isEnabled()`
  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 519](#)) and configuration ([page 512](#)), 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 327](#)) option in dse.yaml.
Note: Using custom resources is not supported by the CQL `CREATE SEARCH INDEX` command.

Resource files 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:

   ```
   $ dsetool write_resource keyspace.table name=schema.xml file=schemaFile.xml
   ```

2. Post the configuration file:

   ```
   $ 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 cores from Solr Admin UI (deprecated)

When DataStax Enterprise authorization is enabled, access to 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><code>solr_admin.solr_resources</code></td>
<td>SELECT</td>
<td>Read a resource</td>
</tr>
<tr>
<td><code>solr_admin.solr_resources</code></td>
<td>MODIFY</td>
<td>Write a resource</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Table</th>
<th>Required permissions</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>all keyspaces</td>
<td>CREATE</td>
<td>Create search index if the keyspace doesn't exist (Thrift only)</td>
</tr>
<tr>
<td>core keyspace</td>
<td>CREATE</td>
<td>Create search index if the table doesn't exist (Thrift only)</td>
</tr>
<tr>
<td>core table</td>
<td>ALTER</td>
<td>Create, reload, and unload search index, and 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>Update core</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:

```sql
GRANT SELECT ON solr_admin.solr_resources 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 and Installer-Services**: `/usr/share/dse/solr/lib`
- **Tarball installations and Installer-No Services**: `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:

```
```
Using DataStax Enterprise advanced functionality

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:

   `<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 (page 570) 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 510) 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 (page 600) 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 name="params">
      <str name="fl">title</str>
      <str name="q">title:Succ*</str>
    </lst>
  </lst>
  <result name="response" numFound="2" start="0">
    <doc>
      <str name="title">Success</str>
    </doc>
    <doc>
      <str name="title">Success</str>
    </doc>
  </result>
</response>
```

**Update request processor (URP) and field transformer (FIT)**

DataStax Enterprise (DSE) recommends 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 discouraged. The DSE custom URP implementation is almost always unnecessary.

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 643) transformer API and the custom URP (page 646).

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>
     ...
   </config>
   
   <fieldOutputTransformer name="dse" class="com.datastax.bdp.cassandra.index.solr.functional.BinaryFieldOutputTransformer">
   </fieldOutputTransformer>
   ...
   ```

2. Write a transformer class something like this reference implementation (page 644) to tweak the data in some way.

3. Export the class to a JAR file. You must place the JAR file in this location:
Using DataStax Enterprise advanced functionality

- **Tarball and Installer-No Services installations**: `install-location/resources/solr/lib`
- **Package and Installer-Services 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](http://www.datastax.com/blog/). 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 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");
        } catch (IOException e) {
            // Handle exception
        }
    }
}
```
```java
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**

```java
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");
    ```
Using DataStax Enterprise advanced functionality

```java
visitor.stringField(binary_name_fi, binary_name);
visitor.stringField(binary_type_fi, binary_type);
visitor.stringField(binary_title_fi, binary_title);
}
}
```

**Custom URP example**

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 almost always unnecessary. Instead, DataStax recommends using the field input/output *(page 643)* (FIT) transformer API.

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.**

   ```xml
   <dseUpdateRequestProcessorChain name="dse">
   <processor
class="com.datastax.bdp.search.solr.functional.DSEUpdateRequestProcessorFactoryExample">
   </processor>
   </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 org.apache.solr.update.AddUpdateCommand;
   import org.apache.solr.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 {
        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 and Installer-No Services installations: install-location/ resources/solr/lib
   - Package and Installer-Services installations: /usr/share/dse/solr/lib

   The JAR is added to the CLASSPATH automatically.

4. Test your implementation. For example:

    package com.datastax.bdp.search.solr.functional;
Using DataStax Enterprise advanced functionality

```java
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`
• **Tarball and Installer-No Services installations:** *installation_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;
    }

    @Override
    public FieldType getStoredFieldType() {
        return new StrField();
    }

    @Override
    public boolean multiValuedFieldCache() {
        return true;
    }

    @Override
    public ListIndexableField createFields(SchemaField sf, Object value) {
        String[] values = ((String) value).split(" ");
        ListIndexableField fields = new ArrayListIndexableField();
        for (String v : values) {
            fields.add(createField(sf, v));
        }
        return fields;
    }
}
```
Deleting by query

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 (page 618)](http://example.com) is built and queried using SolrJ. The query is done using pure Ajax. No DataStax Enterprise API is used for the demo.

You can also use any Thrift API, such as Pycassa or Hector, to access DSE Search. Pycassa supports indexes. You can use indexes in Pycassa just as you use the solr_query expression in DSE Search.

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](http://example.com), built-in [analytics (page 370)](http://example.com) and enterprise [search (page 507)](http://example.com) functionality, and [visual management, monitoring, and development tools (page 1098)](http://example.com).

**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](http://example.com), built-in [analytics (page 370)](http://example.com) and enterprise [search (page 507)](http://example.com) functionality, and [visual management, monitoring, and development tools (page 1098)](http://example.com).
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 699).

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>
<tr>
<td>Support for global graph analytics and batch graph processing (OLAP)</td>
<td>Support for global graph analytics and batch graph processing (OLAP) through the Spark framework.</td>
</tr>
<tr>
<td>Integration with DSE Search</td>
<td>Integrates with DSE Search for efficient indexing.</td>
</tr>
<tr>
<td>Support for geographic, numeric range, and full text search</td>
<td>Support for geographic, numeric range, and full text search for vertices and edges on large graphs.</td>
</tr>
<tr>
<td>Native support for Apache TinkerPop</td>
<td>Native support for the popular property graph data model exposed by Apache TinkerPop.</td>
</tr>
<tr>
<td>Native support for the Gremlin query language</td>
<td>Native support for the graph traversal language Gremlin.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th><strong>Integration of the Gremlin Server</strong></th>
<th>Integration with the Gremlin graph server.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance tuning options</strong></td>
<td>Numerous graph-level configurations provide options for tuning performance.</td>
</tr>
<tr>
<td><strong>Vertex-centric indexes provide optimal querying</strong></td>
<td>Vertex-centric indexes provide vertex-level querying to alleviate issues with the super node problem.</td>
</tr>
<tr>
<td><strong>Optimized disk representation</strong></td>
<td>Provides an optimized disk representation to allow for efficient use of storage and speed of access.</td>
</tr>
</tbody>
</table>

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 773), 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 (page 1098). 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 831) 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 656) and the Gremlin console (page 754).
- GraphSON (page 788) files are JSON files that can exchange graph data and metadata.
- GraphML (page 790) is a standard for exchanging graph data. It can exchange vertex and edge information, but metadata is limited.
- Gryo (page 793) 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 699) provides information and examples.

What tools come with DSE Graph?

DSE Graph comes bundled with a number of tools:

- DataStax Studio (page 1098), a web-based notebook for running Gremlin commands and visualizing graphs
Using DataStax Enterprise advanced functionality

- 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 507) and DSE Analytics (page 370)

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.
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.

Getting started - quick start with DataStax Studio
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. 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.

Figure 3: Recipe Toy Graph

Elements are labeled to distinguish the type of vertices and edges in a graph database using vertex labels and edge labels. A vertex labeled author holds information about an author. An edge between an author and a book is labeled authored. Specifying appropriate labels is an important step in graph data modeling (page 699).

Vertices and edges generally have properties. For instance, an author vertex can have properties name and gender. Edges can also have properties. A created edge can have a year 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.

1. **Install DataStax Enterprise** *(page 222).*

2. **Start DataStax Enterprise with DSE Graph enabled** *(page 1090).*

3. **Installing and running DataStax Studio 2.0** *(page 1099).* Also create a Studio notebook, if needed. This tutorial exists as a Studio notebook, *DSE QuickStart* in Studio 1.0 and *DSE QuickStart v2* in Studio 2.0

4. In DataStax Studio, **create a new connection** *(page 1104).* Choose a graph name; any graph previously unused will work.

   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.

5. In DataStax Studio, **create a new notebook** *(page 1102).* Select the connection created in the last step. Each notebook is connected to a particular graph. Multiple notebooks can be connected to the same graph.

   A blank notebook opens with a single cell. DSE Graph runs a Gremlin Server `tinkerpop.server` on each DataStax Enterprise node. DataStax 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 \( g \) 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.

6. First, set the schema mode to **Development**. Development is a more lenient mode that allows schema to be added at any time during testing. Also allow full scans for testing purposes to inspect the data with broad graph traversals. For production, **Production** schema mode should be set to prevent interactive schema changes that can lead to anomalous behavior, and full scans should be turned off.

   ```java
   schema.config().option('graph.schema_mode').set('Development')
   schema.config().option('graph.allow_scan').set('true')
   ```

7. To check the number of vertices that exist in the graph, use the traversal step \( \text{count()} \). There should currently be none, because we have not added data yet. A graph traversal
Using DataStax Enterprise advanced functionality

$g$ 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()
```

**Caution:** Be aware that queries doing full graph scans with $g.V().count()$ 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.

**Simple example**

Let's start with a simple example from the recipe data model. The data is composed of two vertices, one author (Julia Child) and one book (*The Art of French Cooking, Vol. 1*) with an edge between them to identify that Julia Child authored that book. Without creating any schema, the three elements can be created as shown below. However, DSE Graph makes a best guess at the schema, as we'll talk about below.

8. First, make a vertex for Julia Child. The vertex label is `author` 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. Run the command below and look at the results using the buttons to display the Raw, Table, and Graph views.

```
juliaChild = graph.addVertex(label,'author', 'name','Julia Child', 'gender','F')
```

Each view displays the same information:

- auto-generated id, consisting of a member_id, a community_id and a label
Using DataStax Enterprise advanced functionality

The member_id and the community_id group vertices within the graph storage structure (see Anatomy of a Graph Traversal (page 798))

- vertex label
- properties, name and gender, and their values

**Notice:** Standard auto-generated ids are deprecated with DSE 6.0. Custom ids (page 727) will undergo changes, and specifying vertex ids with partitionKey and clusteringKey will likely become the normal method.

As illustrated in the next command, a property key can be reused for different types of information. Properties are global in the sense that they can be used with multiple vertex labels. However, it is important to understand that you must specify a vertex label in conjunction with a property in a graph traversal.

Run the next command to create a book vertex. Be careful not to run any command twice, or you'll create a duplicate in the graph!

9. Create a book in the graph:

```plaintext
```

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}}.
10. Run the next two commands. The first command creates the edge between the author and book vertices. The second command is a graph traversal that retrieves the two vertices and the edge that connects them. Use **Graph view** to see the relationship. Scroll over elements to display additional information.

```java
juliaChild.addEdge('authored', artOfFrenchCookingVolOne)
g.V()
```
Using DataStax Enterprise advanced functionality

We now have data!

11. Ensure that the data inserted for the author is correct by checking with a `has()` step using the vertex label `author` 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.

```gremlin
g.V().has('author', 'name', 'Julia Child')
```

Use the **Table view** to look at the results, as it is much more readable than the **Raw view**.

The vertex information is displayed for the `author` vertex for Julia Child. A **vertex label** specifies the type of vertex, and the key-value pairs identify the **property key**.
and its value for name and gender. The automatically generated id consists of a vertex label and two components associated with the location of the vertex within the graph. Anatomy of a Graph Traversal (page 798) explains the id components.

12. Another useful traversal is valueMap(), which prints the key-value listing of each property value for specified vertices.

```g.V().hasLabel('author').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').

13. If only the value for a particular property key is desired, use the values() traversal step. This example gets the name of all vertices.

```g.V().values('name')```

Only two vertices exists, so two results are written. If multiple vertices exist, the traversal step returns results for all vertices that have a name.
14. Edge information can also be retrieved. This command filters all edges to find those with an edge label `authored`. The edge information displays details about the incoming and outgoing vertices as well as edge parameters `id`, `label`, and `type`.

```java
g.E().hasLabel('authored')
```

```
{
  "id": "(out_vertex={member_id=0, community_id=1372852736,
  ~label=author},
  local_id=ca2fad30-0e55-11e6-b5e4-0febe4822aa4,
  in_vertex={member_id=0, community_id=14617472,
  ~label=book}, ~type=authored}),
  "label": "authored",
  "type": "edge",
  "inVLabel": "book",
  "outVLabel": "author",
  "inV": "book:14617472:0",
  "outV": "author:1372852736:0"
}
```

15. The traversal step `count()` is useful for counting both the number of vertices and edges. To count edges, substitute `E()` for `V()`. You should have one edge.

```java
g.E().count()
```
16. Re-running the vertex count traversal done at the beginning of this tutorial should now yield two vertices.

```
> g.V().count()
1
1 element returned. Duration: 0.023 s
```

Creating 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 710) and creating indexes (page 734) 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 after data is entered.

17. Create the new schema:

   a. Clear the schema:
Using DataStax Enterprise advanced functionality

```java
schema.clear()
```

b. Create the new property key schema:

```java
// Property Keys
// Check for previous creation of property key with ifNotExists()
ifNotExists()

// single() is optional, as it is the default

schema.propertyKey('name').Text().ifNotExists().create()

// Example of a multiple property that can have several values
// EX: 'livedIn': '1999-2005' 'country': 'Belgium'

schema.propertyKey('livedIn').Text().create()

// A meta-property is a property of a property

schema.propertyKey('livedIn').Text().multiple().create()

// Example of a property that can have several values
// EX: 'nickname': 'John Doe'

schema.propertyKey('nickname').Text().create()
```
Using DataStax Enterprise advanced functionality

Each property must be defined with a data type (page 954). DSE Graph data types are aligned with the DSE database data types. The data types used here are Text, Int, and Timestamp. By default, properties have single cardinality, but can be defined with multiple cardinality (page 723). 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. After property keys are created, vertex labels and edge labels can be defined.

18. Create the schema for vertex labels and edge labels:

```java
// Vertex Labels
schema.vertexLabel('author').ifNotExists().create()  
schema.vertexLabel('recipe').create()  
// Example of creating vertex label with properties  
//
   schema.vertexLabel('recipe').properties('name','instructions').create()  
// Example of adding properties to a previously created vertex label  
//
   schema.vertexLabel('recipe').properties('name','instructions').add()  

schema.vertexLabel('ingredient').create()  
schema.vertexLabel('book').create()  
schema.vertexLabel('meal').create()  
```
Using DataStax Enterprise advanced functionality

```java
// Example of custom vertex id:
// schema.propertyKey('city_id').Int().create()
// schema.propertyKey('sensor_id').Uuid().create()
//
// schema().vertexLabel('FridgeSensor').partitionKey('city_id').clusteringKey('sensor_id').create()
```

```java
// Edge Labels
schema.edgeLabel('authored').ifNotExists().create()
schema.edgeLabel('created').create()
schema.edgeLabel('includes').create()
schema.edgeLabel('includedIn').create()
schema.edgeLabel('rated').connection('reviewer','recipe').create()
```

The schema for vertex labels defines the label type, and optionally defines the properties associated with the vertex label. Two methods exist for defining the association of the properties with vertex labels, either during creation or by adding them after vertex label addition. You can use the `ifNotExists()` for any schema creation.

Vertex ids are automatically generated, but custom vertex ids (page 727) can be created if necessary. The custom vertex id example shown here defines a partition key and a clustering key.

DSE Graph limits the number of vertex labels to 200 per graph.

The schema for edge labels defines the label type, and optionally defines the two vertex labels that are connected by the edge label with `connection()`. The `rated` edge label defines edges between adjacent vertices with the outgoing vertex label `reviewer` and the incoming vertex label `recipe`. By default, edges have multiple
cardinality (page 724), 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.

19. Create the index schema:

```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
//
// schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('category').asString().add()

// Property index using meta-property 'livedIn':
schema.vertexLabel('author').index('byLocation').property('country').by('livedIn').add()

// Edge Index
schema.vertexLabel('reviewer').index('ratedByStars').outE('rated').by('stars').add()
```

Indexing (page 734) 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 indexed.
Using DataStax Enterprise advanced functionality

Edge indexes allow properties on edges to be indexed. Note that indexes are added with `add()` to previously created vertex labels.

20. Examine the schema:

```plaintext
schema.describe()
```

The `schema.describe()` command 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.

Currently, in DSE Graph, schema once created cannot be modified. Additional properties, vertex labels, edge labels, and indexes can be created, but the data type of a property, for instance, cannot be changed. While entering data without schema creation is useful while developing and learning, it is strongly recommended against for actual applications. As a reminder, Production mode disallows schema creation once data is loaded.

21. To find only the schema for a particular type of item in the `describe()` listing, use the following command:

```plaintext
schema.describe().split('\n').grep(~/.*index.*\/)```
Using DataStax Enterprise advanced functionality

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](https://groovy-lang.org/), 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.

**Adding more data**

22. Now that schema is created, add more vertices and edges using the following script to explore more connections in the recipe data model. Enter the following lines in a single DataStax Studio cell and execute. Notice that the first command, `g.V().drop().iterate()` drop all vertex and edge data before reading in new data. Be sure to select the **Graph view** after running the script.

```groovy
// generateRecipe.groovy

// Add all vertices and edges for Recipe
// g.V().drop().iterate()

// author vertices
juliaChild = graph.addVertex(label, 'author', 'name', 'Julia Child', 'gender', 'F')
simoneBeck = graph.addVertex(label, 'author', 'name', 'Simone Beck', 'gender', 'F')
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')

// book vertices

// recipe vertices
beefBourguignon = graph.addVertex(label, 'recipe', '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.')
ratatouille = graph.addVertex(label, 'recipe', 'name', 'Ratatouille', 'instructions', 'Peel and cut the eggplant. Make sure you cut eggplant into lengthwise slices that are about 1-inch wide, 3-inches long, and 3/8-inch thick')
saladeNicoise = graph.addVertex(label, 'recipe', '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.')

wildMushroomStroganoff = graph.addVertex(label, 'recipe', '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 saute pan over medium-high heat.')
spicyMeatloaf = graph.addVertex(label, 'recipe', '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.')
oystersRockefeller = graph.addVertex(label, 'recipe', 'name', 'Oysters Rockefeller', 'instructions', 'Saute the shallots, celery, herbs, and seasonings in 3 tablespoons of the butter for 3 minutes. Add the watercress and let it wilt.')
carrotSoup = graph.addVertex(label, 'recipe', '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.')
Using DataStax Enterprise advanced functionality

roastPorkLoin = graph.addVertex(label, 'recipe', '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."

// ingredients vertices
beef = graph.addVertex(label, 'ingredient', 'name', 'beef')
onion = graph.addVertex(label, 'ingredient', 'name', 'onion')
mashedGarlic = graph.addVertex(label, 'ingredient', 'name', 'mashed
garlic')
butter = graph.addVertex(label, 'ingredient', 'name', 'butter')
tomatoPaste = graph.addVertex(label, 'ingredient', 'name', 'tomato
paste')
eggplant = graph.addVertex(label, 'ingredient', 'name', 'eggplant')
zucchini = graph.addVertex(label, 'ingredient', 'name', 'zucchini')
oliveOil = graph.addVertex(label, 'ingredient', 'name', 'olive oil')
yellowOnion = graph.addVertex(label, 'ingredient', 'name', 'yellow
onion')
greenBean = graph.addVertex(label, 'ingredient', 'name', 'green
beans')
tuna = graph.addVertex(label, 'ingredient', 'name', 'tuna')
tomato = graph.addVertex(label, 'ingredient', 'name', 'tomato')
hardBoiledEgg = graph.addVertex(label, 'ingredient', 'name', 'hard-
boiled egg')
eggNoodles = graph.addVertex(label, 'ingredient', 'name', 'egg
noodles')
mushroom = graph.addVertex(label, 'ingredient', 'name', 'mushrooms')
bacon = graph.addVertex(label, 'ingredient', 'name', 'bacon')
celery = graph.addVertex(label, 'ingredient', 'name', 'celery')
greenBellPepper = graph.addVertex(label, 'ingredient', 'name', 'green
bell pepper')
groundBeef = graph.addVertex(label, 'ingredient', 'name', 'ground
beef')
porkSausage = graph.addVertex(label, 'ingredient', 'name', 'pork
sausage')
shallot = graph.addVertex(label, 'ingredient', 'name', 'shallots')
chervil = graph.addVertex(label, 'ingredient', 'name', 'chervil')
fennel = graph.addVertex(label, 'ingredient', 'name', 'fennel')
parsley = graph.addVertex(label, 'ingredient', 'name', 'parsley')
oyster = graph.addVertex(label, 'ingredient', 'name', 'oyster')
pernod = graph.addVertex(label, 'ingredient', 'name', 'Pernod')
thyme = graph.addVertex(label, 'ingredient', 'name', 'thyme')
carrot = graph.addVertex(label, 'ingredient', 'name', 'carrots')
chickenBroth = graph.addVertex(label, 'ingredient', 'name', 'chicken
broth')
porkLoin = graph.addVertex(label, 'ingredient', 'name', 'pork loin')
redWine = graph.addVertex(label, 'ingredient', 'name', 'red wine')

// meal vertices
// timestamp can also be entered as '2015-01-01' without
Instant.parse()
SaturdayFeast = graph.addVertex(label, 'meal', 'name', 'Saturday
Feast', 'timestamp', '2015-11-30', 'calories', 1000)
Using DataStax Enterprise advanced functionality

EverydayDinner = graph.addVertex(label, 'meal', 'name', 'EverydayDinner', 'timestamp', '2016-01-14', 'calories', 600)
JuliaDinner = graph.addVertex(label, 'meal', 'name', 'JuliaDinner', 'timestamp', '2016-01-14', 'calories', 900)

// 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, 'year', 1961)
juliaChild.addEdge('created', ratatouille, 'year', 1965)
juliaChild.addEdge('created', saladeNicoise, 'year', 1962)
emerilLagasse.addEdge('created', wildMushroomStroganoff, 'year', 2003)
emerilLagasse.addEdge('created', spicyMeatloaf, 'year', 2000)
aliceWaters.addEdge('created', carrotSoup, 'year', 1995)
aliceWaters.addEdge('created', roastPorkLoin, 'year', 1996)
jamesBeard.addEdge('created', oystersRockefeller, 'year', 1970)

// recipe - ingredient edges
beefBourguignon.addEdge('includes', beef, 'amount', '2 lbs')
beefBourguignon.addEdge('includes', onion, 'amount', '1 sliced')
beefBourguignon.addEdge('includes', mashedGarlic, 'amount', '2 cloves')
beefBourguignon.addEdge('includes', butter, 'amount', '3.5 Tbsp')
beefBourguignon.addEdge('includes', tomatoPaste, 'amount', '1 Tbsp')
ratatouille.addEdge('includes', eggplant, 'amount', '1 lb')
ratatouille.addEdge('includes', zucchini, 'amount', '1 lb')
ratatouille.addEdge('includes', mashedGarlic, 'amount', '2 cloves')
ratatouille.addEdge('includes', oliveOil, 'amount', '4-6 Tbsp')
ratatouille.addEdge('includes', yellowOnion, 'amount', '1 1/2 cups or 1/2 lb thinly sliced')
saladeNicoise.addEdge('includes', oliveOil, 'amount', '2-3 Tbsp')
saladeNicoise.addEdge('includes', greenBean, 'amount', '1 1/2 lbs blanched, trimmed')
saladeNicoise.addEdge('includes', tuna, 'amount', '8-10 ozs oil-packed, drained and flaked')
saladeNicoise.addEdge('includes', tomato, 'amount', '3 or 4 red, peeled, quartered, cored, and seasoned')
saladeNicoise.addEdge('includes', hardBoiledEgg, 'amount', '8 halved lengthwise')
wildMushroomStroganoff.addEdge('includes', eggNoodles, 'amount', '16 ozs wmyIde')
wildMushroomStroganoff.addEdge('includes', mushroom, 'amount', '2 lbs wild or exotic, cleaned, stemmed, and sliced')
wildMushroomStroganoff.addEdge('includes', yellowOnion, 'amount', '1 cup thinly sliced')
spicyMeatloaf.addEdge('includes', bacon, 'amount', '3 ozs diced')
spicyMeatloaf.addEdge('includes', onion, 'amount', '2 cups finely chopped')
spicyMeatloaf.addEdge('includes', celery, 'amount', '2 cups finely chopped')
spicyMeatloaf.addEdge('includes', greenBellPepper, 'amount', '1/4 cup finely chopped')
spicyMeatloaf.addEdge('includes', porkSausage, 'amount', '3/4 lbs hot')
spicyMeatloaf.addEdge('includes', groundBeef, 'amount', '1 1/2 lbs chuck')
oystersRockefeller.addEdge('includes', shallot, 'amount', '1/4 cup chopped')
oystersRockefeller.addEdge('includes', celery, 'amount', '1/4 cup chopped')
oystersRockefeller.addEdge('includes', chervil, 'amount', '1 tsp')
oystersRockefeller.addEdge('includes', fennel, 'amount', '1/3 cup chopped')
oystersRockefeller.addEdge('includes', parsley, 'amount', '1/3 cup chopped')
oystersRockefeller.addEdge('includes', oyster, 'amount', '2 dozen on the half shell')
oystersRockefeller.addEdge('includes', pernod, 'amount', '1/3 cup')
carrotSoup.addEdge('includes', butter, 'amount', '4 Tbsp')
carrotSoup.addEdge('includes', onion, 'amount', '2 medium sliced')
carrotSoup.addEdge('includes', thyme, 'amount', '1 sprig')
carrotSoup.addEdge('includes', carrot, 'amount', '2 1/2 lbs, peeled and sliced')
carrotSoup.addEdge('includes', chickenBroth, 'amount', '6 cups')
roastPorkLoin.addEdge('includes', porkLoin, 'amount', '1 bone-in, 4-rib')
roastPorkLoin.addEdge('includes', redWine, 'amount', '1/2 cup')
roastPorkLoin.addEdge('includes', 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', SaturdayFeast)
carrotSoup.addEdge('includedIn', SaturdayFeast)
oystersRockefeller.addEdge('includedIn', SaturdayFeast)
carrotSoup.addEdge('includedIn', EverydayDinner)
roastPorkLoin.addEdge('includedIn', EverydayDinner)
beefBourguignon.addEdge('includedIn', JuliaDinner)
saladeNicoise.addEdge('includedIn', JuliaDinner)

// meal - book edges
EverydayDinner.addEdge('includedIn', artOfSimpleFood)
SaturdayFeast.addEdge('includedIn', simcasCuisine)
JuliaDinner.addEdge('includedIn', artOfFrenchCookingVolOne)
Using DataStax Enterprise advanced functionality

Figure 20: Data for the Recipe Toy Graph

```g.V()
// meal - book edges
EverydayDinner.addEdge('includedIn', artOfSimpleFood)
SaturdayFeast.addEdge('includedIn', sinMasCuisine)
JulieDinner.addEdge('includedIn', artOfFrenchCookingVolOne)
g.V()
```

56 elements returned. Duration: 2.409 s.

The property `timestamp` is a `Timestamp` data type that corresponds to a **valid DSE database timestamp** data type.

The `g.V()` command at the end of the script displays all the vertices created.

23. If a vertex count is run, there is now a higher count of 56 vertices. Run the vertex count again:

```g.V().count()
```

The **DSE Graph Loader** *(page 831)* is available for scripting data loading. It is the recommended method for data loading.
Exploring the graph with graph traversals can lead to interesting conclusions.

24. With several *author* vertices in the graph, to find a particular vertex, provide a specific name. This traversal gets the stored vertex information for the vertex that has the name of Julia Child. Note that the traversal is also constrained by an *author* vertex in the `has` clause.

\[
g.V().has('author', 'name', 'Julia Child')
\]

25. In this next traversal, `has()` gets the vertex information filtered by `name = Julia Child`. The traversal step `outE()` discovers the outgoing edges from that vertex with the `authored` label.

\[
g.V().has('name', 'Julia Child').outE('authored')
\]

In DataStax Studio, either the listing of the **Raw view** edge information:

or the **Graph view** graph visualization where scrolling over a vertex provides additional information.
26. If instead, you want to query for the books that all authors 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`.

```
g.V().outE('authored').inV().values('name')
```

27. 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.

```
g.V().outE('authored').inV().values('name').dedup()
```
28. Refine the traversal by reinserting the `has()` step for a particular author. Find all the books authored by Julia Child.

```gherkin
g.V().has('name','Julia Child').outE('authored').inV().values('name')
```

![Table showing books authored by Julia Child](image)

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Art of French Cooking, Vol. 1</td>
</tr>
<tr>
<td>1</td>
<td>The French Chef Cookbook</td>
</tr>
</tbody>
</table>

29. 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.

```gherkin
g.V().has('name','Julia Child').out('authored').values('name')
```

![Table showing books authored by Julia Child](image)

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.

30. 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.

```gherkin
g.V().has('name','Julia Child').out('authored').has('year','gt(1967)').values('name')
```
31. When developing or testing, often 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.

```
g.V().label().groupCount()
```

32. 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 the entire graph to a file. Other file formats can be written by substituting `gryo()` with `graphml()` or `graphson()`.

```
graph.io(gryo()).writeGraph("/tmp/recipe.gryo")
```

**Note:** `graph.io()` is disabled in sandbox mode.
33. To load a Gryo file, use the graphloader, after creating a mapping script:

```
$ graphloader mappingGRYO.groovy -graph recipe -address localhost
```

Details about loading Gryo data are found in Loading Gryo Data (page 864), in Using DSE Graph Loader (page 831).

34. If you wish to use the gremlin console after working in Studio, two commands are useful: `system.graphs()` (page 757) to get a list of all graphs and `remote config alias g some_graph.g` (page 757) to switch to a different graph.

What's next:

Congratulations! You are well on your way to using DSE Graph for data discovery.

Further adventures in traversing can be found in Creating queries using traversals (page 798). If you want to explore various loading options, check out the DSE Graph Loader (page 831) or Using DSE Graph (page 707).

**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
Using DataStax Enterprise advanced functionality

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:

```
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:

```
gremlin> g.V().in().has('name', 'Julia Child').count().profile()
```

This query completes the following processing:

```
<table>
<thead>
<tr>
<th>Step</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>DsegGraphStep(vertex,[])</td>
<td>61</td>
</tr>
<tr>
<td>query-optimizer</td>
<td>0.563</td>
</tr>
<tr>
<td>_condition=((label = FridgeSensor</td>
<td>label = author</td>
</tr>
<tr>
<td>query-setup</td>
<td>0.048</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>0.979</td>
</tr>
<tr>
<td>_usesCache=false</td>
<td></td>
</tr>
</tbody>
</table>
```

Page 682 DSE 5.1 Developer Guide (Previous version)
Using DataStax Enterprise advanced functionality

\_statement=SELECT "city_id", "sensor_id" FROM
"DSEQuickStart"."FridgeSensor_p" WHERE "~~vertex_exists" = ?
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}
index-query
0.862
\_usesCache=false

\_statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."author_p" WHERE "~~vertex_exists" = ?
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}
index-query
0.679
\_usesCache=false

\_statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."book_p" WHERE "~~vertex_exists" = ?
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}
index-query
1.344
\_usesCache=false

\_statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."ingredient_p" WHERE "~~vertex_exists" = ?
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}
index-query
1.053
\_usesCache=false

\_statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."meal_p" WHERE "~~vertex_exists" = ?
LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 50000
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 4.173
\_usesCache=false
\_statement=SELECT "community_id", "member_id" FROM "DSEQuickStart"."recipe_p" WHERE "~~vertex_exists" = ? 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}

index-query 1.291
\_usesCache=false
\_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{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async=true}

DsegVertexStep(IN,vertex) 78

query-optimizer 0.305
\_condition=((true) & direction = IN)
vertex-query 4.136
\_usesCache=false
\_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{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Optional.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async=true}

\_isPartitioned=false
\_usesIndex=false
vertex-query 0.558
\_usesCache=false
\_statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE "community_id" = ? AND "member_id" = ? LIMIT ?
```

Page 684 DSE 5.1 Developer Guide (Previous version)
ALLOW FILTERING; with params (java.lang.Integer)
1432048000, (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=false}
\_isPartitioned=false
\_usesIndex=false
vertex-query
1.146
\_usesCache=false
\_statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
"community_id" = ? AND "member_id" = ? LIMIT ?
ALLOW FILTERING; with params (java.lang.Integer)
153541376, (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=false}
\_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.Integer) 50000
\_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Optional.empty,
pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async=false}
\_isPartitioned=false
\_usesIndex=false
query-setup
0.015
\_isFitted=false
\_isSorted=true
Using DataStax Enterprise advanced functionality

```plaintext
\_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
>TOTAL
- 154.632 -
```

**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:

```
g.V().in('created').has('name','Julia Child').count()
```

```=>3```

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:

```
gremlin> g.V().in('created').has('name','Julia Child').count().profile()
```

```Traversal Metrics
Step                          Count
Traversers       Time (ms)    % Dur
=============================================================================================================  
DsegGraphStep(vertex,[])                                              61
query-optimizer  
   1.760
   \_condition=((label = FridgeSensor | label = author | label = book |
   label = ingredient | label = meal |
   label = recipe | label = reviewer) & (true))
query-setup       0.071
   \_isFitted=true
   \_isSorted=false
   \_isScan=true
index-query       1.139
   \_usesCache=false
   \_statement=SELECT "city_id", "sensor_id" FROM
   "DSEQuickStart"."FridgeSensor_p" WHERE "~~vertex_exists" =
   ? LIMIT ? ALLOW FILTERING; with params
   (java.lang.Boolean) true, (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}
index-query       2.012
   \_usesCache=false
   \_statement=SELECT "community_id", "member_id" FROM
   "DSEQuickStart"."author_p" WHERE "~~vertex_exists" =
   ? LIMIT ? ALLOW FILTERING; with params
   (java.lang.Boolean) true, (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}
```
Using DataStax Enterprise advanced functionality

```java
\_usesCache=false
\_statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."book_p" WHERE "~~vertex_exists" = ?
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}

\_usesCache=false
\_statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."ingredient_p" WHERE "~~vertex_exists"
= ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 5000
\_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Optional.
empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async=true}

\_usesCache=false
\_statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."meal_p" WHERE "~~vertex_exists" = ?
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}

\_usesCache=false
\_statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."recipe_p" WHERE "~~vertex_exists"
= ? 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}

\_usesCache=false
\_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\{consistency=Optional\[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, async
c=true\}
DsegVertexStep(IN,[created],vertex)                                    8
8         103.458    78.62
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) 1432048000, (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
_\_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) 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 "~~
Using DataStax Enterprise advanced functionality

```java
edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Integer) 588941056, (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, async=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, async=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, async=c=true}
 \_isPartitioned=false
 \_usesIndex=false
```
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?

```java
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:

```java
gremlin> g.V().hasLabel('recipe').in().has('name','Julia Child').count().profile()
```
### Traversal Metrics

<table>
<thead>
<tr>
<th>Step</th>
<th>Count</th>
<th>Time (ms)</th>
<th>% Dur</th>
</tr>
</thead>
<tbody>
<tr>
<td>DsegGraphStep(~label.=(recipe))</td>
<td>8</td>
<td>2.598</td>
<td>9.25</td>
</tr>
<tr>
<td>query-optimizer</td>
<td></td>
<td>0.241</td>
<td></td>
</tr>
<tr>
<td>_condition=((\label = recipe) &amp; (true))</td>
<td></td>
<td>0.187</td>
<td></td>
</tr>
<tr>
<td>index-query</td>
<td></td>
<td>1.225</td>
<td></td>
</tr>
<tr>
<td>DsegVertexStep(IN,vertex)</td>
<td>15</td>
<td>9.668</td>
<td>34.41</td>
</tr>
<tr>
<td>query-optimizer</td>
<td></td>
<td>0.150</td>
<td></td>
</tr>
<tr>
<td>_isFitted=false</td>
<td></td>
<td>0.047</td>
<td></td>
</tr>
<tr>
<td>_isScan=false</td>
<td></td>
<td>0.896</td>
<td></td>
</tr>
<tr>
<td>vertex-query</td>
<td></td>
<td>1.415</td>
<td></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Statement</th>
<th>Query Setup</th>
<th>Is Fitted</th>
<th>Is Sorted</th>
<th>Is Scan</th>
<th>Query Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SELECT * FROM &quot;DSEQuickStart&quot;.&quot;recipe_e&quot; WHERE &quot;community_id&quot; = ? AND &quot;member_id&quot; = ? LIMIT ? ALLOW FILTERING;</code> with params (java.lang.Integer) 96517120, (java.lang.Long) 1, (java.lang.Integer) 50000</td>
<td>0.038</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>0.014</td>
</tr>
<tr>
<td><code>SELECT * FROM &quot;DSEQuickStart&quot;.&quot;recipe_e&quot; WHERE &quot;community_id&quot; = ? AND &quot;member_id&quot; = ? LIMIT ? ALLOW FILTERING;</code> with params (java.lang.Integer) 1598713728, (java.lang.Long) 1, (java.lang.Integer) 50000</td>
<td>0.364</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>0.431</td>
</tr>
<tr>
<td><code>SELECT * FROM &quot;DSEQuickStart&quot;.&quot;recipe_e&quot; WHERE &quot;community_id&quot; = ? AND &quot;member_id&quot; = ? LIMIT ? ALLOW FILTERING;</code> with params (java.lang.Integer) 1146421632, (java.lang.Long) 1, (java.lang.Integer) 50000</td>
<td>0.014</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>0.364</td>
</tr>
</tbody>
</table>
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:
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.

gremlin> g.V().has('author','name','Julia Child').outE('created').count().profile()  
===>Traversal Metrics

<table>
<thead>
<tr>
<th>Step</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traversers</td>
<td></td>
</tr>
<tr>
<td>Time (ms)</td>
<td></td>
</tr>
<tr>
<td>% Dur</td>
<td></td>
</tr>
<tr>
<td>=============================================================================================================</td>
<td></td>
</tr>
<tr>
<td>DsegGraphStep(~label.=(author), name.=(Julia C...)</td>
<td>1</td>
</tr>
<tr>
<td>query-optimizer</td>
<td>7.673</td>
</tr>
<tr>
<td>_condition=((label = author) &amp; (true)) &amp; name = Julia Child) query-setup</td>
<td>0.033</td>
</tr>
<tr>
<td>_isFitted=true _isSorted=false _isScan=false index-query</td>
<td>17.694</td>
</tr>
<tr>
<td>_indexType=Secondary _usesCache=false _statement=SELECT &quot;community_id&quot;, &quot;member_id&quot; FROM &quot;DSEQuickStart&quot;.&quot;author_p&quot; WHERE &quot;name&quot; = ? 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}</td>
<td></td>
</tr>
<tr>
<td>DsegVertexStep(OUT,[created],edge)</td>
<td>3</td>
</tr>
<tr>
<td>query-optimizer</td>
<td>0.200</td>
</tr>
<tr>
<td>_condition=((label = created) &amp; (true)) &amp; direction = OUT) vertex-query</td>
<td>0.586</td>
</tr>
</tbody>
</table>
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 915).

**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 710):

```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()
```
Using DataStax Enterprise advanced functionality

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 737) and retry the queries.

```java
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 \textit{graph\_name\_system}

Each graph created in turn creates three DSE database keyspaces, \textit{graph\_name}, \textit{graph\_name\_system} and \textit{graph\_name\_pvt}. The \textit{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 952) 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.

\textbf{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 \textit{vertex}, and the line adjoining the vertices, or \textit{edge}, identifies the relationship as "created". Vertices and edges have associated \textbf{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 \textbf{property key} and \textbf{property value}. In graph databases, a vertex is \textbf{incident} to an edge, and an edge is \textbf{incident} to a vertex. A vertex is \textbf{adjacent} to another vertex. A generalized view of this data model is shown below:

Figure 37: Generalized data model for author and recipe
Each vertex is assigned a **vertex label** to identify a specific type of vertex. The vertex labels shown here are **author** 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 **year**.

DSE Graph limits the number of vertex labels to 200 per graph.

For more complex graphs, **multiple edges (page 724)** can connect vertices, and multiple properties can be assigned to vertices and edges. Both properties and edges can have **multiple cardinality (page 723)**. Vertex properties can have **meta-properties**, a property on a property (page 725).

An important concept to be aware of is the nature of vertices and edges as addressable elements. **Indexes (page 734)** 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 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 author 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.
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 authors, recipes, and ingredients.

What are the edge labels we can use to identify different types of edges? Previously, you've seen that authors 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.

![Diagram showing the relationship between author, recipe, and ingredient with edge labels and properties]

3. Edges also have attached properties that can be used later in narrowing queries.

What edge property is appropriate for includes?

![Diagram showing the relationship between author, recipe, and ingredient with edge labels and properties]

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 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.
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 710) 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
Using DataStax Enterprise advanced functionality

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 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.
Using DataStax Enterprise advanced functionality

Look at the relationships that result for just one ingredient:

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 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: *vertices*, *edges* and *properties*. Vertices are objects, such as people or artifacts. Edges define the relationships between nodes. Vertices, edges and properties can have properties; for this reason, DSE Graph is classified as a *property graph*. The properties for all elements are an important element of storing and querying information from a property graph.

Property graphs are typically quite large, although the nature of querying the graph will vary 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 an author is labeled `author`. 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 699).

Vertices and edges generally have properties. For instance, an `author` 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 `timestamp` 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 831). CSV, JSON, text parsed with regular expressions, and data selected from a JDBC compliant database can be loaded using a command line tool.

**Gremlin commands**
Data can be added using Gremlin commands (page 754). 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 754) shows some of the common Gremlin commands for creating a graph and running traversals.

**Gryo**
Data can be loaded using Gryo (page 864), 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 866), 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 (page 789).

**GraphML**
Data can be entered using GraphML (page 867), 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.

**Creating a graph in Studio**

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 776). In Development mode, DataStax Studio creates a graph and aliases the graph to a graph traversal automatically for each connection that is created.

1. **Start DSE Graph** (page 1090).

2. **Install and start DataStax Studio** (page 1099). Also create a Studio notebook, if needed.
3. In DataStax Studio, create a new connection (page 1104). Choose a graph name; any graph previously unused will work.

4. In DataStax Studio, create a new notebook (page 1104). Select the connection created in the last step.

A blank notebook will open with a single cell. DSE Graph runs a Gremlin Server `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 `SimpleStrategy`. 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 and return results. A graph traversal is bound to a specific traversal source which is the standard OLTP traversal engine.

### Creating graph schema using Studio

Creating a data model (page 700) 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. Use Gremlin-Groovy to enter scripts into the cells of DataStax Studio.

**Prerequisites:**

Create a graph using Studio (page 709).

1. Optional. If you are reusing a graph that you previously created, drop the graph schema and data (page 784).

2. Optional. If running large scripts, set the `timeout` value to `max` to prevent client-side timeouts. 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 of 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. Copy and paste the Recipe Schema listed in the Example below in a single cell in DataStax Studio. Once the entire script is entered, run the cell. Studio submits the commands to the Gremlin server.

   **NOTE:** Each command submitted is within a single session, so from cell to cell, 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 in cells, 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 954) 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.

   ```
   // 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.
Using DataStax Enterprise advanced functionality

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": "Beef Bourguignon",
        "instructions": "Braise the beef.",
        "createDate": 1967
    }, {
        "type": "recipe",
        "recipeTitle": "Salade Nicoise",
        "instructions": "Break the lettuce into pieces."
    }]
}
```

7. Define the vertex labels. The vertex labels identify the type of vertices that can be created.

```javascript
// Vertex Labels
schema.vertexLabel('author').ifNotExists().create()
schema.vertexLabel('recipe').create()
// Example of creating vertex label with properties
//
schema.vertexLabel('recipe').properties('name', 'instructions').create()
schema.vertexLabel('ingredient').create()
schema.vertexLabel('book').create()
```
Using DataStax Enterprise advanced functionality

```java
schema.vertexLabel('meal').create()
schema.vertexLabel('reviewer').create()
// Example of custom vertex id:
// schema.propertyKey('city_id').Int().create()
// schema.propertyKey('sensor_id').Uuid().create()
//
schema().vertexLabel('FridgeSensor').partitionKey('city_id').clusteringKey('sensor_id').create()
```

Vertex labels can be checked for prior existence using `ifNotExists()`. Vertex labels can be created along with properties. Vertex labels can be created with custom vertex ids (page 727), rather than the standard autogenerated vertex ids (page 798).

**Notice:** Standard auto-generated ids are deprecated with DSE 6.0. Custom ids (page 727) will undergo changes, and specifying vertex ids with `partitionKey` and `clusteringKey` will likely become the normal method.

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 934). Edge labels can identify properties that an edge has using `properties()` (page 945).

9. Define indexes that can speed up the query processing. All types of indexes are presented here. Indexing graph data (page 734) 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
//
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('instructions').asString().add()
```
Using DataStax Enterprise advanced functionality

```java
// 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()
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().create();
// Example meta-property added to property:
// schema.propertyKey('livedIn').Text().create()
//
// schema.propertyKey('country').Text().properties('livedIn').create()

// Vertex Labels
schema.vertexLabel('author').ifNotExists().create()
schema.vertexLabel('recipe').create()
// Example of creating vertex label with properties
//
// schema.vertexLabel('recipe').properties('name','instructions').create()
schema.vertexLabel('ingredient').create()
schema.vertexLabel('book').create()
schema.vertexLabel('meal').create()
schema.vertexLabel('reviewer').create()
// Example of custom vertex id:
// schema.propertyKey('city_id').Int().create()
// schema.propertyKey('sensor_id').Uuid().create()
//
// schema().vertexLabel('FridgeSensor').partitionKey('city_id').clusteringKey('sensor_id').create()

// Edge Labels
schema.edgeLabel('authored').ifNotExists().create()
schema.edgeLabel('created').create()
schema.edgeLabel('includes').create()
schema.edgeLabel('includedIn').create()
schema.edgeLabel('rated').properties('stars').connection('reviewer','recipe').create()

// 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
// schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('category').asString().add()
Using DataStax Enterprise advanced functionality

Advanced schema

There are advanced schema that can be created in DSE Graph.

Date and time schema

Date and time are two data types that are commonly used for both vertex and edge properties.

Create schema and load data

1. Create the following properties in a graph:

   // SCHEMA FOR DATE AND TIME PROPERTIES
   schema.propertyKey('year').Date().ifNotExists().create()
   schema.propertyKey('time').Time().ifNotExists().create()

2. Some additional schema is required for the example queries below:

   // OTHER PROPERTIES
   schema.propertyKey('name').Text().ifNotExists().create()
   schema.propertyKey('gender').Text().ifNotExists().create() // VERTEX LABELS
   schema.vertexLabel('person').properties('name', 'gender').ifNotExists().create()
   // EDGE LABELS
   schema.edgeLabel('born').multiple().connection('person', 'person').ifNotExists().create()
   schema.edgeLabel('born').properties('year', 'time').add()
   // INDEXES
   schema.vertexLabel('person').index('byName').materialized().by('name').add()

   This example uses the date and time properties on the edge label born that identifies the birthdate and birth time for a person.

3. The following data is inserted using DSE Graph Loader (page 835):

   A CSV file of each person:

   name | gender
Using DataStax Enterprise advanced functionality

and a CSV file for the edges:

```
pname1|pname2|year|time
Julia Child|JCMom|1930-01-01|10:00:00.000
Simone Beck|SBMom|1940-01-01|12:00:00.000
Louise Bertholie|LBMom|1950-01-01|13:00:00.000
```

The mapping script loads each file after the schema has been created in a graph:

```
/* SAMPLE INPUT
person: Julia Child|F
personEdges: Julia Child|JCMom|1930-01-01|10:00
*/

// 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)
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/dateTime/'
personInput = File.csv(inputfiledir + "person.csv").delimiter('|')
personEdgeInput = File.csv(inputfiledir +
"personEdges.csv").delimiter('|')

// Specifies what data source to load using which mapper (as defined
inline)
load(personInput).asVertices {
    label "person"
    key "name"
}
load(personEdgeInput).asEdges {
    label "born"
    outV "pname1", {
        label "person"
        key "name"
    }
    inV "pname2", {
        label "person"
        key "name"
    }
```
Using DataStax Enterprise advanced functionality

Querying date and time data

4. Find all born edges that have a birthdate earlier than 1940-01-01:

   // Find all edges that have a birthdate earlier than 1940-01-01
   g.V().hasLabel('person').outE('born').has('year',lt('1940-01-01')).valueMap()

   This query finds all vertices with a label person, then traverses the outgoing edges labelled born, and filters out all edges found to meet the limitation of lt('1940-01-01').

   {year=1930-01-01, time=10:00}

5. List the name of each child and their parent based on having a birthdate earlier than 1940-01-01:

   g.V().hasLabel('person').as('child').outE('born').has('year',lt('1940-01-01')).inV().as('parent').select('child','parent').by('name').by('name')

   This query finds all vertices with a label person and saves it temporarily as child, then traverses the outgoing edges labelled born, and filters out all edges found to meet the limitation of lt('1940-01-01') as the last query did. It continues by finding all the incoming vertices and saves them temporarily as parent, before selecting the two saved items with a select() method, by name in each case.

   {child=Julia Child, parent=JCMom}

Geospatial schema

Three geospatial data types (page 954), point, linestring, and polygon, store data that can be searched with geospatial shapes. After creating schema for these data types, geospatial queries can be constructed (page 815) using them. For most geospatial queries that look for geospatial points, points or linestrings within circles or polygons, DSE Search indexes (page 738) must also be created.

The examples below load geospatial data with graph.addVertex(...) commands, but the DSE Graph Loader can be used to load geospatial data (page 882) starting with DSE 5.0.9 and DSE 5.1.2.

Point schema

• Create schema for a point and add a vertex with a property value for a point:

   schema.propertyKey('name').Text().create()
   schema.propertyKey('point').Point().withGeoBounds().create()
   schema.vertexLabel('location').properties('name','point').create()
Using DataStax Enterprise advanced functionality

A vertex label is created for `location` that has a `point` property.

**Note:** For geospatial linestrings, as with geospatial points, the `withGeoBounds()` method limit 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 the point, using WellKnownText (WKT) format. Note that is specifies longitude first, then latitude.

Check that the point exists:

```javascript
g.V().has('location', 'name', 'Paris').valueMap()
==>{name=[Paris], point=[POINT (2.352222 48.856614)]}
```

**Linestring schema**

- Create schema for a `linestring` and add a vertex with a property value for a linestring:

```javascript
schema.propertyKey('name').Text().create()
schema.propertyKey('line').Linestring().withGeoBounds().create()
schema.vertexLabel('lineLocation').properties('name','line').create()
graph.addVertex(label, 'lineLocation', 'name', 'ParisLondon', 'line', "LINESTRING(2.352222 48.856614, -0.127758 51.507351)"
```

A vertex label is created for `lineLocation` that has a `LineString` property. The same boundary limits (page 719) imposed on points are imposed on linestrings.

Check that the linestring exists:

```javascript
g.V().has('lineLocation','name','ParisLondon').valueMap()
==>{line=[LINESTRING (2.352222 48.856614, -0.127758 51.507351)], name=[ParisLondon]}
```

**Polygon schema**

- Create schema for a `polygon` and add a vertex with a property value for a polygon:

```javascript
schema.propertyKey('name').Text().create()
schema.propertyKey('polygon').Polygon().withGeoBounds().create()
schema.vertexLabel('polyLocation').properties('name','polygon').create()
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 `polyLocation` that has a `Polygon` property. The same boundary limits (page 719) imposed on points are imposed on polygons.

Check that the polygon exists:

```javascript
g.V().has('polyLocation','name','ParisLondonDublin').valueMap()
```
Using DataStax Enterprise advanced functionality

```plaintext
==>{polygon=[POLYGON ((2.352222 48.856614, -0.127758 51.507351, -6.26031 53.349805, 2.352222 48.856614)),
name=[ParisLondonDublin]}
```

**DSE Search indexes**

- 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.

```plaintext
//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:

```plaintext
schema.vertexLabel('location').index('search').search().by('point').withError(0.000009,0.0).add()
```

**What's next:** Geospatial queries (*page 815*) can be created once schema exists.

**Cartesian spatial schema**

Three cartesian spatial data types (*page 954*), `point`, `linestring`, and `polygon` store data that can be searched with spatial shapes. After creating schema for these data types, Cartesian spatial queries can be constructed (*page 822*) using them. For Cartesian queries that look for Cartesian points, points or linestrings within circles or polygons, DSE Search indexes (*page 738*) must be created.

The examples below load Cartesian data with `graph.addVertex(...)` commands, but the DSE Graph Loader can be used to load Cartesian data (*page 882*) starting with DSE 5.0.9 and DSE 5.1.2.

**Point schema**

- Create schema for a `point` and add a vertex with a property value for a point:

```plaintext
schema.propertyKey('name').Text().create()
```
schema.propertyKey('point').Point().withBounds(-3, -3, 3, 3).create()
schema.vertexLabel('location').properties('name','point').create()
graph.addVertex(label, 'location', 'name', 'p0', 'point', Geo.point(0.5, 0.5))

A vertex label is created for location that has a point property. For Cartesian spatial points, the withBounds(x1, y1, x2, y2) method limit searches to a default valid range of values in the x-y grid.

Check that the point exists:
```
g.V().has('location', 'name', 'p0').valueMap() ==>{name=[p0], point=[POINT (0.5 0.5)]}
```

**Linestring schema**

- Create schema for a linestring and add a vertex with a property value for a linestring:

```
schema.propertyKey('name').Text().create()
schema.propertyKey('line').Linestring().withBounds(-3, -3, 3, 3).create()
schema.vertexLabel('lineLocation').properties('name','line').create()
graph.addVertex(label, 'lineLocation', 'name', 'l1', 'line', "LINESTRING(0 0, 1 1)")
```

A vertex label is created for lineLocation that has a LineString property. For Cartesian spatial linestrings, as with Cartesian spatial points, the withBounds(x1, y1, x2, y2) method limit searches to a default valid range of values in the x-y grid.

Check that the linestring exists:
```
g.V().has('lineLocation','name','l1').valueMap() ==>{line=[LINESTRING (0 0, 1 1)], name=[l1]}
```

**Polygon schema**

- Create schema for a polygon and add a vertex with a property value for a polygon:

```
schema.propertyKey('name').Text().create()
schema.propertyKey('polygon').Polygon().withBounds(-3, -3, 3, 3).create()
schema.vertexLabel('polyLocation').properties('name','polygon').create()
graph.addVertex(label, 'polyLocation','name', 'g1', 'polygon', Geo.polygon(0,0,1,1,0,0))
```

A vertex label is created for polyLocation that has a Polygon property. For Cartesian spatial polygons, as with Cartesian spatial points, the withBounds(x1, y1, x2, y2) method limit searches to a default valid range of values in the x-y grid.

Check that the polygon exists:
```
g.V().has('polyLocation','name','g1').valueMap()
```
Using DataStax Enterprise advanced functionality

DSE Search indexes

While DSE Graph natively supports Cartesian 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.

```java
//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()
```

Note: DSE Search does not index polygons.

Caching edges and properties

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 schema API `vertexLabel()` (page 947) method with the `cache()` option. 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.

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.

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` vertices will improve the retrieval of a query asking for all meals with a calorie count less than 850 calories.

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 946), vertex labels (page 947), edge labels (page 941)) that change often to avoid stale data.

Graph cache is useful for rarely changed graph data. The queries that will use graph cache effectively are queries that repeatedly run. If the queries run differ even in the sort order, the graph cache will not be used to reduce the query latency. For instance, caching the `calories` property for `meal` vertices will improve the retrieval of a query asking for all
meals with a calorie count less than 850 calories, if this query is repeated. Note that all properties for all meal vertices will be cached along with calories.

dse.yaml
The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/.resources/dse/conf/dse.yaml</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td></td>
</tr>
</tbody>
</table>

- Cache all properties for author vertices up to an hour (3600 seconds):
  
  ```java
  schema.vertexLabel('author').cache().properties().ttl(3600).add()
  ```

  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()
  ```

**Multiple cardinality property or edge**

Multiple cardinality allows a property to store multiple values within a single key, or multiple edges with the same edge label to connect two vertices.

**Multiple cardinality property**

- Create schema for a multiple cardinality property item_mult and create two vertices, each of which has two items listed for the defined property:
  
  ```java
  schema.propertyKey('item_mult').Text().multiple().ifNotExists().create()
  schema.vertexLabel('fridgeItem_multiple').properties('item_mult').ifNotExists().create()
  graph.addVertex(label,'fridgeItem_multiple','name','item1','item_mult', ['cheese', 'cheddar cheese'])
  graph.addVertex(label, 'fridgeItem_multiple','name', 'item2', 'item_mult', ['yogurt', 'Greek yogurt'], 'item_mult', ['key lime yogurt'])
  g.V().hasLabel('fridgeItem_multiple').valueMap()
  ```

Because item_mult is defined with multiple cardinality, any number of key value insertions can be made, even using two lists to insert the values. Note that all values are stored in a single list.

```java
login: {item_mult=[cheddar cheese, cheese], name=[item1]}
login: {item_mult=[Greek yogurt, key lime yogurt, yogurt], name=[item2]}
```
Using DataStax Enterprise advanced functionality

```sql
g.V().has('fridgeItem_multiple', 'name', 'item2').values('item_mult')
```

```text
==>Greek yogurt
==>key lime yogurt
==>yogurt
```

This output makes it clear that each entry in the list is a separately stored value.

- Check for a single list item, specifying a particular value:

```sql
g.V().hasLabel('fridgeItem_multiple').has('item_mult', 'Greek yogurt').valueMap()
```

```text
==>item_mult=[Greek yogurt, key lime yogurt, yogurt],
    name=[item2])
```

Multiple cardinality edge
- Create schema for a multiple cardinality edge:

```java
// SCHEMA
// PROPERTIES
schema.propertyKey('author').Text().single().create()
schema.propertyKey('city').Text().single().create()
schema.propertyKey('dateStart').Text().single().create()
schema.propertyKey('dateEnd').Text().single().create()
// VERTEX LABELS
schema.vertexLabel('author').properties('author').create()
schema.vertexLabel('city').properties('city').create()
// EDGE LABELS
schema.edgeLabel('livedIn').multiple().connection('author','city').create()
schema.edgeLabel('livedIn').properties('dateStart', 'dateEnd').add()
// INDEXES
schema.vertexLabel('author').index('byAuthor').materialized().by('author').add()
```

Note that the edge label `livedIn` is defined with multi-cardinality. The sample data loaded with graphloader is:

<table>
<thead>
<tr>
<th>author</th>
<th>city</th>
<th>dateStart</th>
<th>dateEnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia Child</td>
<td>Paris</td>
<td>1961-01-01</td>
<td>1967-02-10</td>
</tr>
<tr>
<td>Julia Child</td>
<td>New York</td>
<td>1970-06-06</td>
<td>1971-09-23</td>
</tr>
<tr>
<td>Julia Child</td>
<td>Chicago</td>
<td>1980-04-05</td>
<td>1981-01-01</td>
</tr>
<tr>
<td>Simone Beck</td>
<td>Paris</td>
<td>1960-01-01</td>
<td>1962-09-23</td>
</tr>
</tbody>
</table>

This data includes two different periods of time in which Julia Child lived in Paris (fictiously), and edges will be multiple because of the multi-cardinality.

With this data loaded, the resulting graph shows the multi-cardinal edges exist:
Meta-property of properties

A meta-property allows a property to store another property.

Meta-property

- Create schema for a meta-properties start_date and end_date and create an author vertex label with the property country:

```java
schema.propertyKey("name").Text().single().create()
schema.propertyKey('start_date').Date().create()
schema.propertyKey('end_date').Date().create()
schema.propertyKey('country').Text().multiple().properties('start_date','end_date').create()
schema.vertexLabel("author").properties("name", 'country').create()
```

The meta-properties start_date and end_date are first created as propertyKeys. Then the meta-properties are assigned to the property country using the properties() method.

```java
==>{'item_mult=[cheddar cheese, cheese], name=[item1]}
==>{'item_mult=[Greek yogurt, key lime yogurt, yogurt],
    name=[item2]}
```

- Now create a vertex:

```java
julia=graph.addVertex(label,'author', 'name', 'Julia Child')
props=julia.property(list,'country','France')
props.property('start_date', '1950-01-01')
props.property('end_date', '1960-12-31')
props2 = julia.property(list, 'country', 'USA')
props2.property('start_date', '1961-01-01')
props2.property('end_date', '1984-06-23')
```

This vertex for Julia Child has two values for the property country, each of which has values for the two meta-properties.

An alternative vertex creation statement is:
Using DataStax Enterprise advanced functionality

```java
g.addV('author').
  property('name', 'Emeril Lagasse').
  property('country', 'France', 'start_date', '1973-10-01', 'end_date', '1973-09-09').
  property('country', 'USA', 'start_date', '1973-10-01', 'end_date', '2017-03-10')

Either method is acceptable.

- Explore the inserted properties:

  First, find the listed countries that any author has lived in:

  ```java
g.V().properties()
  ==>vp[name->Julia Child]
  ==>vp[country->France]
  ==>vp[country->USA]
  ==>vp[name->Emeril]
  ==>vp[country->France]
  ==>vp[country->USA]
  ```

  Next, list all the meta-properties of any listed country:

  ```java
g.V().properties().properties()
  ==>p[start_date->1950-01-01]
  ==>p[end_date->1960-12-31]
  ==>p[start_date->1961-01-01]
  ==>p[end_date->1984-06-23]
  ==>p[start_date->1970-04-05]
  ==>p[end_date->1973-09-09]
  ==>p[start_date->1973-02-02]
  ==>p[end_date->2017-03-01]
  ```

  Find specific meta-property values for a given country by adding the `hasValue()` method:

  ```java
g.V().properties('country').hasValue('France').properties()
  ==>p[start_date->1950-01-01]
  ==>p[end_date->1960-12-31]
  ==>p[start_date->1970-04-05]
  ==>p[end_date->1973-09-09]
  ```

  Find just the values of the meta-properties:

  ```java
g.V().properties('country').hasValue('France').properties().value()
  ==>1950-01-01
  ==>1960-12-31
  ==>1970-04-05
  ==>1973-09-09
  ```

  Find the starting date in which an author moved to each country:

  ```java
g.V().as('author').
  properties('country').as('country').
  ```
This query uses the designator *author* to store the author's name, then traverses the country property for each country lived in, designated *country*. From each country, the start date is found by an additional traversal and designated as *start_living_in*. Finally, all three values are selected for each full path traversal and the results are printed, using the designators to map the values. Entering this query into DSE Studio yields the following results:

```javascript
{
  "author": "Julia Child",
  "country": "France",
  "start_living_in": "1950-01-01"
},
{
  "author": "Julia Child",
  "country": "USA",
  "start_living_in": "1961-01-01"
},
{
  "author": "Emeril",
  "country": "France",
  "start_living_in": "1970-04-05"
},
{
  "author": "Emeril",
  "country": "USA",
  "start_living_in": "1973-02-02"
}
```

**Creating a custom vertex id**

A custom vertex id can be created to replace the standard auto-generated vertex id that is normally generated for vertices. The use cases for creating a custom vertex id are:

- When the data stored in DSE Graph is aligned with other data sources, such as other DSE database keyspaces or another database.
- When custom graph partitioning (page 655) is desired.

**Notice:** Standard auto-generated ids are deprecated with DSE 6.0. Custom ids (page 727) will undergo changes, and specifying vertex ids with `partitionKey` and `clusteringKey` will likely become the normal method.

For example, sensor time series data is stored in the DSE database. In addition to the time series queries to the DSE database, relationship information about the sensors is desired, such as how the sensors are networked and where the sensors are located. To load the data into a graph to explore the relationships, but retain the ability to write an application...
Using DataStax Enterprise advanced functionality

that accesses both the time series data and the graph data, custom vertex ids are created that span across both database models.

**Caution:** Keep in mind that if custom vertex ids are used, they must be globally unique within the graph, or duplicate vertices can potentially be loaded into a graph. DSE Graph does not verify the custom vertex ids. Standard auto-generated vertex ids are guaranteed to be unique.

1. Create a vertex label with a custom partitioning key `sensor_id`. The property key `sensor_id` must exist prior to use in creating the vertex label and cannot be a multiple cardinality property.

   ```java
   schema.vertexLabel('FridgeSensor').partitionKey('sensor_id').create()
   ```

2. Add a vertex using the vertex label `FridgeSensor`.

   ```java
   graph.addVertex(label, 'FridgeSensor', 'sensor_id', '60bcae02-f6e5-11e5-9ce9-5e5517507c66')
   ```

3. 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()
   ```

4. Add a vertex using the vertex label `FridgeSensor` with both a partition key and clustering key.

   ```java
   graph.addVertex(label, 'FridgeSensor', 'sensor_id', '60bcae02-f6e5-11e5-9ce9-5e5517507c66', 'city_id', 100)
   ```

5. Create a vertex label with a custom composite partition, using both `city_id` and `sensor_id` as part of the partitioning key.

   ```java
   schema.vertexLabel('FridgeSensor').partitionKey('city_id', 'sensor_id').create()
   ```

**Modifying schema using Studio**

Schema creation is an important part of creating a graph database. It may be necessary to add or modify the schema after initial creation. In Development mode, the schema can be modified after data creation. In Production mode, schema creation and data loading cannot be mixed. Property keys can be added. Adjacencies can be identified.

One important distinction about schema is the difference between `create()` and `add()`. The `create()` command is used to create a property key, vertex label or edge label if it does not already exist. When creating vertex labels and edge labels, `create()` is used along with identification of associated property keys. If a vertex label or edge label is
created without property key identification, then the `add()` command is used to identify associated property keys.

Examine the current schema

- Before making modifications to the schema, examine the current settings.

```java
schema.describe()
```

The schema displayed can be copied and used to reproduce the schema for a graph.

Add property keys to a vertex label

- Add a property key after schema creation. The property key must already exist. In the example, the first command builds the property key for the graph, and the second command adds the property key to the vertex label `author`.

```java
schema.propertyKey('nationality').Text().create()
schema.vertexLabel('author').properties('nationality').add()
```

- Verify that the property key is built for the vertex label `author`. Look for the property key named `nationality`.

```java
schema.vertexLabel('author').describe()
```
The properties name existed prior to the addition of nationality. Any indexes on the vertex label are also displayed.

• Add a value for the newly added property key to a vertex.

   g.V().has('author','name','Julia Child').property('nationality','American')

   ➢ g.V().has('author','name','Julia Child').property('nationality','American')  

   index | id | label  | nationality | name   |
   ---- | ---- | ------ | ----------- | ------ |
   0    | author:772575a1230 | author | American | Julia Child |

Add property keys to an edge label

• Add a property key after schema creation. The property key must already exist. In the example, the first command builds the property key for the graph, and the second command adds the property key to the edge label created.

   schema.edgeLabel('created').properties('timestamp').add()

   GREMLIN

   ➢ schema.edgeLabel('created').properties('timestamp').add()

   null

1 element returned. Duration: 1.261s.

• Verify that the property key is built for the edge label created. Look for the property key named timestamp.

   schema.edgeLabel('created').describe()

   GREMLIN

   schema.edgeLabel('created').describe()

   {} T

   schema.edgeLabel('created').multiple().properties("year","timestamp").create()
   schema.edgeLabel('created').connection("author", "recipe").add()

The properties year existed prior to the addition of timestamp. Any indexes on the vertex label are also displayed.
Creating an edge between two vertices (connection)

- Create a vertex label with properties. All the properties must exist prior to creating the vertex label. Add an edge label that identifies the connection (page 934) outgoing vertex and incoming vertex.

```java
schema.vertexLabel('FridgeItem').properties('name','expiration_date','amount').add()
schema.edgeLabel('isA').connection('ingredient','FridgeItem').create()
```

Dropping data, schema, and graphs

Data, schema, and graphs can be dropped (deleted) in DataStax Studio 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 author vertices, identify the vertices along with a drop traversal step.

```java
g.V().hasLabel('author').drop()
```

**Warning:** Dropping vertices with this command will also drop all edges associated with the vertices. Any vertex at the other end of an edge will remain, but the edges and edge properties will be dropped from the data.

**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('author').limit(100).drop()
```
Using DataStax Enterprise advanced functionality

and repeat until all vertices are dropped.

• To drop a specific value, such as author vertices, identify the vertices along with a drop traversal step.

```
g.V().hasLabel('author').properties('gender').hasValue('M').drop()
```

This query will drop the gender value for all vertices that have a gender value of M.

```
g.V().hasLabel('author').valueMap()
```

returns:

```
==>{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]}
==>{gender=[F], name=[Patricia Simon]}
==>{name=[James Beard]}
==>{name=[Fritz Streiff]}
==>{name=[Emeril Lagasse]}
```

• To drop a property key from an edge, such as rated edges, identify the edges, the property key stars along with a drop traversal step.

```
g.E().hasLabel('rated').properties('stars').drop()
```

This query will drop the property key stars for all edges that have a rated edge label.

```
g.E().hasLabel('rated').properties('stars').valueMap()
```

returns no values.

**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.

**Drop schema**

• To drop the schema and all data without dropping the graph, use a clear() 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.

```
schema.clear()
```

```
GREMLIN |
> schema.clear()

null

1 ELEMENT RETURNED. DURATION: 5.6615.
```
Important: Currently, certain schema elements such as a vertex label cannot be individually modified or removed. If a change to the schema is necessary, drop the whole schema as detailed above and recreate.

Dropping a graph

- Dropping a graph will clear all schema and data as well as deleting the graph. A system command is required to drop a graph. In order to use system commands, the graph traversal alias must be cleared. A configuration reset clears the alias.

```
gremlin> :remote config alias reset
```

Note: System commands are not accessible when a graph is aliased.

```
==>Aliases cleared
```

- Optional: If unsure of the graph name, examine what graphs exist. Note that system commands do not work in Studio, and must be run in Gremlin console.

```
gremlin> system.graphs()
```

```
==>food
```

- Drop the desired graph by running the `drop()` command in Gremlin console.

```
gremlin> system.graph('food').drop()
```

```
==>null
```

Dropping an index

- Dropping indexes (page 752) is described in Indexing.

Adding property data

To change the value of property data for a particular vertex, use the `property()` step. Each property changed must have its own `property()` step.

- If a property value has been dropped (page 731) in order to change the value, a new value can be written. This example illustrates adding the value $M$ to the author vertex with the name $James Beard$.

```
g.V().has('author', 'name', 'James Beard').property('gender', 'M')
```

```
gremlin> g.V().hasLabel('author').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]}
```
Indexing graph data

How to index DSE Graph data.

Indexing graph overview

DSE Graph implements two types of indexes, vertex-centric indexes and global indexes. Vertex-centric indexes are local and specific to a single vertex. Global indexes are specific to a vertex label and property and are graph-wide. All indexes contribute to the performance of graph traversals on large distributed graphs. The type of index lookup will affect performance, and each has pros and cons.

Vertex-centric indexing (VCI) overview

Vertex-centric indexes (VCI) are created locally for a specific vertex, unlike global indexes which are global to the graph and index elements for fast global lookups. VCIs are used once a query has been filtered down to a specific instance of a vertex label, meaning specific vertices. VCIs sort and index the incident edges and adjacent vertices of a vertex according to the incident edge labels or properties. When a vertex is queried, its index is consulted to avoid linear scans of all incident edges. Traversals can be reduced to O(1) or O(log n) from O(n). A typical graph traversal touches numerous vertices, compounding the cost of each incident edge scan if indexes are not consulted.

In DSE Graph, vertex-centric indexing is maintained as materialized views (MVs). 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. Edge indexes (page 943) and property indexes (page 943) are vertex-centric indexes.

Vertex-centric indexing also plays a role in solving the super-node issue. A super-node is a vertex that has an exponentially larger number of incident edges. The example generally given is to compare the number of followers that reader has to those of a celebrity - hundreds or thousands of followers, compared to millions of followers. A graph traversal
Using DataStax Enterprise advanced functionality

Checking the index of a super-node will take an outsized amount of time just to read the index table. Using vertex-centric indexing in conjunction with a partitioned vertex table (PVT), the index can be stored on multiple partitions and distributed across the DSE cluster. For vertices that have in excess of one million edges, graph partitioning is necessary due to the storage limitations of a single DSE database table. Distributing the index tables also enables better response to graph traversals.

**Note:** Partitioned vertex tables are an experimental feature for handling supernodes that are deprecated in DSE 5.1 and will be removed in DSE 6.0. However, data modeling techniques are currently a better avenue for mitigating supernode issues.

Global indexing overview

Indexes can affect traversal query performance. Decreasing the number of starting points for a graph traversal can greatly reduce the latency for a query result. If a traversal must start by checking all the vertices in a graph, time is lost finding the right starting point. If a starting vertex can be identified, that time is not required. Global indexing improves the performance of queries by identifying the starting location of a query using the vertex label and property.

Global indexing in DSE Graph uses DSE secondary indexing or DSE Search indexing (page 507). 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.

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.

Search indexes are used when textual, numeric or geospatial indexing are required and rely on DSE Search (page 507). 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 (page 577). This type of index is best for low selectivity, but lends itself to fuzzy matching. DSE 5.1 adds fuzzy search for both tokenized and non-tokenized indexing.

Composite index keys are not currently supported in DSE Graph.
Indexing best practices

More than one index can be created on the same property, such as creating both a materialized index and a search index on the property `amount`. The DSE Graph query optimizer will automatically use 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. Different index types may be created on different properties as appropriate, based on the selectivity. A special case exists for indexing vertices created with composite keys (page 727); a search index is the only choice for indexing two or more properties, especially for graph loading with the DSE Graph Loader (page 831). Separate materialized view indexes will not be used for the property keys that make up the composite key (custom vertex id) and the DSE Graph Loader will fail to create the vertices.

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 considered.

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 (page 581). 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 do require resources. Each index allocates a minimum of 256MB by default, and each index will require 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.

In DSE 5.1 and later, 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 created. 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.
**Note:** Prior to DSE 5.1, search indexes defaulted to `asText()` for textual property data, if not specified as `asString()`.

It is possible to modify the search index schema ([page 519](#)) 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.

### Creating graph indexes

Creating indexes for a graph can be accomplished with many different characteristics. All indexing identifies a vertex label and a property to index. Edge indexes additionally identify an edge label.

**Note:** A property key can be used in more than one vertex label, as shown with the property key `name` below. Graph traversals will only use indexes if both the vertex label and property key are specified, as shown in Using indexes ([page 741](#)). Indexing that spans all vertex labels in a graph is not supported ([page 741](#)) in DSE Graph if full graph scans are disabled.

#### Secondary index

- Create a secondary index.

```java
schema.vertexLabel('recipe').index('byRecipe').secondary().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 `secondary()` step identifies the index as a secondary index.

#### Materialized index

- Create a materialized view index.

```java
schema.vertexLabel('author').index('byAuthor').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.
Using DataStax Enterprise advanced functionality

Search index

- Create a search index. This search index has one property key indexed. If multiple property keys are indexed, chain additional `by()` steps.

```java
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().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. This index is searched using full text index.

**Note:** Only one search index can be created per vertex label.

- A search index can also specify string indexing option. This example identifies a string index.

```java
schema.vertexLabel('recipe').index('search').search().by('instructions').asString().add()
```

`asString()` is also an available search index option.

- More commonly, a search index will specify multiple columns:

```java
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('name').asString().add()
```

In DSE 5.1 and later, 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 created. 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.

**Note:** Prior to DSE 5.1, search indexes defaulted to `asText()` for textual property data, if not specified as `asString()`.

- Search indexes can also include non-text data types:

```java
schema.vertexLabel('recipe').index('search').search().by('year').by('name').asString().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 (page 548) are used for sorting and range querying.

- 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("coordinates").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 718) for additional information.

- Create a search index for timestamp data:

```java
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()
```

**Edge index**

- Create 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('reviewer').index('ratedByStars').outE(rated).by('stars').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.

- Create an edge index that indexes both incoming and outgoing edges:

```java
schema.vertexLabel('reviewer').index('ratedByStars').bothE(rated).by('stars').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.

**Property index**

- Create a property index. Property indexes are vertex-centric to a particular vertex label.
Using DataStax Enterprise advanced functionality

Identify the vertex label and property keys for the index, in the `vertexLabel()` and `property()` steps, respectively. In the `index()` step, name the index. The `by()` step is used to define the meta-property of the property.

```
schema().vertexLabel('author').index('byLocation').property('country').by('livedIn').add()
```

Verifying and identifying index schema

- Verify the index creation.

```
schema.describe()
```

- Display more information about the indexes by specifying the vertex label and using the `describe()` step. Schema that can be used to create the indexes will be displayed.

```
schema().vertexLabel('author').describe()
```

```
==>schema.vertexLabel("author").properties("name", "gender", "nationality").create()
```

```
schema.vertexLabel("author").index("byName").secondary().by("name").add()
schema.vertexLabel("author").index("byAuthor").materialized().by("name").add()
```

- Display information about all of the indexes using a schema traversal that filters for all `vertexIndex`.

```
schema.traversal().V().hasLabel('vertexIndex').valueMap()
```

```
==>{name=[byName], type=[Secondary]}
==>{unique=[false], name=[byIngredient], type=[Materialized]}
==>{unique=[false], name=[byReviewer], type=[Materialized]}
==>{unique=[false], name=[byRecipe], type=[Materialized]}
==>{unique=[false], name=[byMeal], type=[Materialized]}
```

- Get a count of the number of indexes using a schema traversal that filters for all `vertexIndex`.

```
schema.traversal().V().hasLabel('vertexIndex').valueMap()
```
Using indexes

Indexes will be used in graph traversal queries for the first traversal step reached after the `V()` step, and is used to trim down the number of vertices that are initially fetched. 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.

**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:

```java
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]))
```

- The graph traversal shown uses an index to discover certain author vertices to start the query.

```java
g.V().has('author', 'name', 'Emeril Lagasse').out('created').values('name')
```
Using DataStax Enterprise advanced functionality

This graph traversal uses an index, if the index exists, because the traversal step `has('author', '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 the property key, as two different elements, authors and recipes, use the same property key `name`.

Checking for the use of indexing can be accomplished with the `profile()` method:

```
> g.V().has('author', 'name', 'Emeril Lagasse').out('created').values('name').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([-label.=(author), name.=(Emeril Lagasse)])</td>
<td>1</td>
<td>1</td>
<td>2.196</td>
<td>51.37</td>
</tr>
<tr>
<td>query-optimizer</td>
<td></td>
<td></td>
<td>0.199</td>
<td></td>
</tr>
<tr>
<td>query-setup</td>
<td></td>
<td></td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>index-query</td>
<td></td>
<td></td>
<td>0.946</td>
<td></td>
</tr>
<tr>
<td>DsegVertexStep(OUT,[created],vertex)</td>
<td>2</td>
<td>2</td>
<td>0.935</td>
<td>21.88</td>
</tr>
<tr>
<td>query-optimizer</td>
<td></td>
<td></td>
<td>0.101</td>
<td></td>
</tr>
<tr>
<td>query-setup</td>
<td></td>
<td></td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>vertex-query</td>
<td></td>
<td></td>
<td>0.282</td>
<td></td>
</tr>
<tr>
<td>DsegPropertiesStep([name],value)</td>
<td>2</td>
<td>2</td>
<td>1.030</td>
<td>24.11</td>
</tr>
<tr>
<td>query-optimizer</td>
<td></td>
<td></td>
<td>0.044</td>
<td></td>
</tr>
</tbody>
</table>
Using search indexes

DSE Graph leverages DSE Search indexes (page 735) to efficiently filter vertices by properties, and reducing query latency. DSE Search uses a modified Apache Solr (page 509) 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.

In DSE 5.1 and later, 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 created. 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.

**Note:** Prior to DSE 5.1, search indexes defaulted to `asText()` for textual property data, if not specified as `asString()`.

Property key indexes defined with `asText()` or undefined (since this is the default) can use the following options for search:

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
<th>Count</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>query-setup</td>
<td>0.005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vertex-query</td>
<td>0.347</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vertex-query</td>
<td>0.639</td>
<td></td>
<td></td>
</tr>
<tr>
<td>query-setup</td>
<td>0.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NoOpBarrierStep(2500)</td>
<td></td>
<td>2.64</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>4.276</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Note the *index-query* used in the first step `DsegGraphStep`. If an index was not used, *index-query* would be missing from the profile output.
Using DataStax Enterprise advanced functionality

- token (page 744)
- tokenPrefix (page 745)
- tokenRegex (page 746)

Property key indexes defined with asString() can use the following options for search:

- eq/neq (page 746)
- prefix (page 747)
- regex (page 747)

**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 748)
- fuzzy (page 748)
- tokenFuzzy (page 749)

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 737) 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')
```
Why does this search find these three recipes? Because the instructions for each meet the search requirements:

```
g.V().has('recipe','instructions', Search.token('Sea')).values('name')
```

<table>
<thead>
<tr>
<th>index</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

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().has('recipe','instructions', tokenPrefix('Sea')).values('instructions')
```

<table>
<thead>
<tr>
<th>index</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Roast Pork Loin</td>
</tr>
<tr>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>Oysters Rockefeller</td>
</tr>
<tr>
<td>3</td>
<td>Sauté 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>

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.
Using DataStax Enterprise advanced functionality

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')```

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')```
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.

```gsql
GRELISH

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`.

```gsql
GRELISH

g.V().hasLabel('recipe').has('name', regex('.*ee.*')).values('name')
```
Using DataStax Enterprise advanced functionality

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:

```golang
g.V().hasLabel('recipe').has('name', phrase('Wild Mushroom Stroganoff', 0))
```

The 0 designates that the result must be an exact phrase.

```typescript
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.

Find the exact name of *James Beard* in an author name:

```golang
foundBeard = g.V().hasLabel('author').has('name', phrase('James Beard', 1))
```

```typescript
v[~label=author, community_id=2123369856, member_id=0]
```

The vertex for the correct recipe is returned. A match for

```golang
g.V().hasLabel('recipe').has('name', phrase('Wild Mushroom', 1))
```

will also return the correct vertex, but

```golang
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:
g.V().hasLabel('author').has('name', fuzzy('James Beard', 0)).values('name')

The 0 designates that the result must be an exact match.

James Beard

- Changing the last value in a fuzzy() predicate will find misspellings:

  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.

  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

  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:

  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 DataStax Enterprise advanced functionality

Using two search indexes for a single traversal query

- Create a second search index like an example search index from Creating indexes (page 737) 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'))`.

```java
g.V().has('recipe', 'instructions',
    tokenRegex('Braise.*')).as('r').
V().has('author', 'name',
    prefix('J')).out().where(eq('r')).values('name')
```

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.

```java
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:

```java
schema.vertexLabel('FridgeSensor').index('search').search().
    by('location').ifNotExists().add()
```
Using DataStax Enterprise advanced functionality

```java
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()`.

```java
Distance d = Geo.point(-110,30),20, Geo.Unit.DEGREES)
g.V().hasLabel('FridgeSensor').has('location', Geo.inside(d)).values('name')
```

More information on geospatial queries can be found in Geospatial traversals (page 815).

**Search using numerical values**

- Search indexes can also be used for non-textual values:

```java
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:

```java
graph.addVertex(label, 'person','name','Julia','age',56)
graph.addVertex(label, 'person','name','Emeril','age',48)
graph.addVertex(label, 'person','name','Simone','age',50)
```
Using DataStax Enterprise advanced functionality

```java
graph.addVertex(label, 'person', 'name', 'James', 'age', 52)
```

and the query itself:

```java
g.V().has('person', 'age', gt(50)).values()
```

to find all persons over the age of 50.

```java
=> Julia
=> 56
=> James
=> 52
```

• To sort the previous search, add additional methods:

```java
g.V().hasLabel("person").has("age", gt(50)).order().by("age", incr).values()
```

to get:

```java
=> James
=> 52
=> Julia
=> 56
```

**Dropping indexes**

Dropping indexes from a graph is accomplished with `schema` calls.

**Drop secondary or materialized 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 and find the index name.

```java
schema.vertexLabel('meal').describe()
```

- Using the vertex label and index name, remove the index. Run `describe()` again to verify that the index is removed.

```java
schema.vertexLabel('meal').index('byMeal').remove()
```
Using DataStax Enterprise advanced functionality

```java
schema.vertexLabel('meal').describe()
```

```java
schema.vertexLabel('meal').index('byMeal').remove()
schema.vertexLabel('meal').describe()
```

```java
{}
```

```java
schema.vertexLabel('meal').properties("name", "timestamp", "calories").create()
```

### Drop single property in search index

- To drop a property from a search index in the schema, such as the `nick_name` property, identify the property name. Use `describe()` to examine the search index for the desired vertex label and find the property name.

```java
schema.vertexLabel('author').describe()
```

```java
{}
```

```java
schema.vertexLabel('author').properties("nick_name", "country", "name", "gender").create()
schema.vertexLabel('author').index("byName").secondary().by("name").add()
schema.vertexLabel('author').index("search").search().by("name").asString().by("nick_name").asText().add()
schema.vertexLabel('author').index("bylocation").property("country").by("livedIn").add()
```

- Using the vertex label, property name, and index name, remove the index. Run `describe()` again to verify that the index is removed.

```java
schema.vertexLabel('author').index('search').search().properties('nick_name').remove()
schema.vertexLabel('author').describe()
```
Using DataStax Enterprise advanced functionality

Using the Gremlin console

Inserting data with Gremlin commands in the Gremlin console.

Getting started - quick start with Gremlin console

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 are the perfect representation of the relationships between objects.

Graph databases consist of two 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 you are defining edges.

Both vertices and edges can have properties; for this reason, DSE Graph is classified as a *property graph*. The properties for both vertices and edges are an important element of storing and querying information from a property graph.

Property graphs are typically quite large, although the nature of querying the graph will vary 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 an author is labeled *author*. 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 699).

Vertices and edges generally have properties. For instance, an *author* vertex can have a *name*. Gender and current job are examples of additional properties for an *author* vertex. Edges also have properties. A *created* edge can have a *timestamp* property that identifies when the adjoining *recipe* vertex was created.

Information in a graph database can be retrieved using *graph traversals*. *Graph traversals* “walk” a graph with a single or series of *traversal steps* that can define a starting point for a traversal and filter the results to find the answers to queries about the graph data.
In order to run graph traversals to retrieve information, data must first be inserted. The steps listed in this section will allow you to gain a rudimentary understanding of DSE Graph with a minimum amount of configuration and schema creation.

1. **Install DSE** *(page 222).*

2. **Start DSE Graph** *(page 1090).*

3. **Start the Gremlin Console.**
   ```
   $ bin/dse gremlin-console
   ``

   ```
   \,\,/
   (o o)
   ----o00o--(3)--o00o-----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. A **graph** must be created that is stored as one graph instance per DSE database keyspace.

   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.

4. **Create a graph to hold the data.** The **system** commands are used to run commands that affect graphs in DSE Graph.

   ```
   gremlin> system.graph('test').create()
   ==>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 and return results. A graph traversal is bound to a specific traversal source which is the standard OLTP traversal engine.

5. To list all graphs previously created, use:

```
system.graphs()
```

```
==>test
==>anotherTest
```

6. 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.

```
gremlin> :remote config alias g test.g
```

```
==>g=test.g
```

**Note:** This command is not available if a graph traversal is aliased with the \( \text{:remote config alias g some_graph.g} \) command. In order to access the system command, reset the alias with \( \text{:remote config alias reset} \).

The graph commands usually add vertices or edges to the database, or get other graph information. The \( g \) commands generally do queries to obtain results.

7. First, set the schema mode to **Development**. Development is a more lenient mode that allows schema to be added at any time during testing. Also allow full scans for testing purposes to inspect the data with broad graph traversals. For production, **Production** schema mode should be set to prevent interactive schema changes that can lead to anomalous behavior, and full scans should be turned off.

```
schema.config().option('graph.schema_mode').set('Development')
schema.config().option('graph.allow_scan').set('true')
```

8. Check the number of vertices that exist in the graph using the traversal step \( \text{count()} \). There should currently be none, because we have not added data yet. A graph traversal \( g \) is chained with \( V() \) to get all vertices and \( \text{count()} \) to get the number of vertices.

```
gremlin> g.V().count()
```

```
==>0
```

**Note:**
Using DataStax Enterprise advanced functionality

Note: Be aware that queries doing full graph scans with `g.V().count()` 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.

A simple example is composed of two vertices, one author (Julia Child) and one book (The Art of French Cooking, Vol. 1) with an edge between them to identify that Julia Child authored that book. Without creating any schema, the three elements can be created as shown below. However, DSE Graph makes a best guess at the schema, as we'll talk about below.

9. First, let's make a vertex for Julia Child. The vertex label is `author`, and two property key-value pairs are created for `name` and `gender`. Note the use of label to designate the key for a key-value pair that sets the vertex label. Run the command and look at the results using the buttons to display the Raw, Table, and Graph views.

```gremlin> juliaChild = graph.addVertex(label, 'author', 'name', 'Julia Child', 'gender', 'F')```

```
==>[~label=author, member_id=0, community_id=1080937600]
```

Each view displays the same information:

• an auto-generated id, consisting of a member_id, community_id and label
  
  # The member_id and community_id are used for grouping vertices within the graph (more information (page 798))

Notice: Standard auto-generated ids are deprecated with DSE 6.0. Custom ids (page 727) will undergo changes, and specifying vertex ids with `partitionKey` and `clusteringKey` will likely become the normal method.

As you will see in the next command, a property key can be reused for different types of information. While properties are “global” in the sense that they are 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.

Run the next command to create a book vertex. Don't run any command twice, or you'll create a duplicate in the graph!

10. Create a book in the graph.


```
==>[~label=book, member_id=1, community_id=1080937600]
```
As with the author vertex, you can see the id information about the book vertex created.

Run the next two commands. The first command creates the edge between the author and book vertices. The second command is a graph traversal that retrieves the two vertices using `valueMap()`. Use `valueMap()` to check author vertex property key information. The traversal `g` checks all vertices with the traversal step `V()`, and prints out a key-value listing of the property values for each vertex using the traversal step `valueMap()`.

11. Create an edge and display the vertex data.

```bash
gremlin> juliaChild.addEdge('authored', artOfFrenchCookingVolOne)
gremlin> g.V().valueMap()
```

```bash
===>e[{out_vertex={~label=author, member_id=0,
                  community_id=1080937600},
      local_id=6bd73210-0e70-11e6-b5e4-0febe4822aa4,
      in_vertex={~label=book, member_id=1,
                  community_id=1080937600},
      ~type=authored}]
{~label=author,
  member_id=0, community_id=1080937600}-authored->
{~label=book,
  member_id=1, community_id=1080937600]}
```

```bash
gremlin> g.V().valueMap()
```

```bash
==>{gender=[F], name=[Julia Child]}
==>{name=[The Art of French Cooking, Vol. 1], timestamp=[1961]}
```

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')`.

We now have data! The key-value pairs identify the property key and its value for name and gender for the author vertex created, as well as the name and timestamp for the book vertex created.

12. A graph traversal that is a basic starting point for more complex traversal use the `has()` step along with the vertex label author and the property name = Julia Child to identify a particular vertex. This common graph traversal is used because it narrows the search of the graph with specific information.

```bash
gremlin> g.V().has('author', 'name', 'Julia Child')
```

```bash
===>v[{~label=author, member_id=0, community_id=1080937600}]
```
The *id* is automatically generated and consists of a vertex label and two components associated with the location of the vertex within the graph. The *Anatomy of a Graph Traversal* ([page 798](#)) explains the id components.

13. If only the value for a particular property key is desired, the traversal step `values()` step is used. This example below gets the *name* of all vertices.

```gremlin
gremlin> g.V().values('name')
```

Only two vertices exists, so two results are written. If multiple vertices exist, the traversal step returns results for all vertices with a *name*.

```plaintext
==>Julia Child
==>The Art of French Cooking, Vol. 1
```

14. Edge information can also be retrieved. The next command filters all edges to find those with an edge label *authored*. The edge information displays details about the incoming and outgoing vertices as well as edge parameters *id*, *label*, and *type*.

```gremlin
gremlin> g.E().hasLabel('authored')
```

```plaintext
==>[out_vertex={~label=author, member_id=0, community_id=1080937600},
     local_id=6bd73210-0e70-11e6-b5e4-0febe4822aa4,
     in_vertex={~label=book, member_id=1, community_id=1080937600},
     ~type=authored}
```

15. 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.

```gremlin
gremlin> g.E().count()
```

```plaintext
==>1
```

16. Re-running the vertex count traversal done at the beginning of this tutorial should now yield two vertices.

```gremlin
gremlin> g.V().count()
```

```plaintext
==>2
```

Before adding more data to the graph, let's stop and talk about schema. Schema is used to define the possible properties and their data types that will be used in 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.

More information can be found in the documents about creating schema (page 710) and creating indexes (page 734).

First, let's create schema for the property keys. In the next two cells, the first command clears the schema that was set when we created the first two vertices and edge. After the schema creation is completed, you 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 after data is entered.

17. Clear the previous schema. A return value of null means that the command is successful.

```
17. Clear the previous schema. A return value of null means that the command is successful.

   gremlin> schema.clear()

   =>null

18. Create the property keys.

   // 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().single().create()
   // Example of a multiple property that can have several values
   // schema.propertyKey('nickname').Text().multiple().create() //
   // Next 2 lines define two properties, then create a meta-property
   // 'livedIn' on 'country'
   // A meta-property is a property of a property
   // EX: 'livedIn': '1999-2005' 'country': 'Belgium'
   schema.propertyKey('livedIn').Text().create()
   schema.propertyKey('country').Text().multiple().properties('livedIn').create()
   schema.propertyKey('country').Text().multiple().properties('livedIn').create()

   // A series of null returns will mark the successful completion
   // of all property key creation
```
Each property must be defined with a data type (page 954). DSE Graph data types are aligned with the DSE database data types. The data types used here are Text, Int, and Timestamp. 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()` to prevent overwriting a definition that may already exist. After property keys are created, vertex labels and edge labels can be defined.

19. Create vertex labels and edge labels.

```java
// Vertex Labels
schema.vertexLabel('author').ifNotExists().create()
schema.vertexLabel('recipe').create()
// Example of creating vertex label with properties
//
// schema.vertexLabel('recipe').properties('name','instructions').create()
// Example of adding properties to a previously created vertex label
//
// schema.vertexLabel('recipe').properties('name','instructions').add()

schema.vertexLabel('ingredient').create()
schema.vertexLabel('book').create()
schema.vertexLabel('meal').create()
schema.vertexLabel('reviewer').create()
// Example of custom vertex id:
// schema.propertyKey('city_id').Int().create()
// schema.propertyKey('sensor_id').Uuid().create()
//
// schema().vertexLabel('FridgeSensor').partitionKey('city_id').clusteringKey('sensor_id')

// Edge Labels
schema.edgeLabel('authored').ifNotExists().create()
schema.edgeLabel('created').create()
schema.edgeLabel('includes').create()
schema.edgeLabel('includedIn').create()
schema.edgeLabel('rated').connection('reviewer','recipe').create()
```

A series of null returns will mark the successful completion of all vertex label and edge label creation

```java
=>null
```

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.

DSE Graph limits the number of vertex labels to 200 per graph.

Vertex ids are automatically generated, but custom vertex ids (page 727) can be created if necessary. This custom vertex id example is explained in further detail in the documentation, but note that partition keys and clustering keys can be defined.

The schema for edge labels defines the label type, and optionally defines the two vertex labels that are connected by the edge label with `connection()`. The `rated` edge label defines edges between adjacent vertices with the outgoing vertex label `reviewer` and the incoming vertex label `recipe`. By default, edges have multiple cardinality, 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.

20. Create the indexes.

```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
//
// schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('category').asString().add()
// Property index using meta-property 'livedIn':
schema.vertexLabel('author').index('byLocation').property('country').by('livedIn').add()

// Edge Index
schema.vertexLabel('reviewer').index('ratedByStars').outE('rated').by('stars').add()
```

Indexing (page 734) is a complex and highly important topic. Here, several types of indexes are created. Briefly, secondary and materialized indexes are two types of indexes that use 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 indexed. Edge indexes allow properties on edges to be indexed. Note that indexes are added with `add()` to previously created vertex labels. After running all the cells to create the schema, examine the schema with the following command.

21. Examine the schema.

```bash
gremlin> schema.describe()
```

```bash
==>
  schema.propertyKey("instructions").Text().single().create()
  schema.propertyKey("livedIn").Text().single().create()
  schema.propertyKey("country").Text().multiple().properties("livedIn").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("ISBN").Text().single().create()
  schema.propertyKey("name").Text().single().create()
  schema.propertyKey("comment").Text().single().create()
  schema.propertyKey("category").Text().single().create()
  schema.propertyKey("timestamp").Timestamp().single().create()
  schema.edgeLabel("authored").multiple().create()
  schema.edgeLabel("rated").multiple().properties("stars").create()
  schema.edgeLabel("includedIn").multiple().create()
  schema.edgeLabel("created").multiple().create()
  schema.edgeLabel("includes").multiple().create()
  schema.vertexLabel("meal").properties("name").create()
  schema.vertexLabel("meal").index("byMeal").materialized().by("name").add()
  schema.vertexLabel("ingredient").properties("name").create()
  schema.vertexLabel("ingredient").index("byIngredient").materialized().by("name").add()
  schema.vertexLabel("author").properties("country", "name").create()
  schema.vertexLabel("author").index("byName").secondary().by("name").add()
  schema.vertexLabel("author").index("byLocation").property("country", "livedIn").add()
  schema.vertexLabel("book").create()
  schema.vertexLabel("recipe").properties("name").create()
  schema.vertexLabel("recipe").index("byRecipe").materialized().by("name").add()
  schema.vertexLabel("reviewer").properties("name").create()
  schema.vertexLabel("reviewer").index("byReviewer").materialized().by("name").add()
  schema.vertexLabel("reviewer").index("ratedByStars").outE("rated").by("stars").add()
  schema.edgeLabel("rated").connection("reviewer", "recipe").add()
```

The `schema.describe()` command will display schema that can be used to recreate the schema entered. If you enter data without creating schema, this command verifies the data types set for each property.

Currently, in DSE Graph, schema once created cannot be modified. Additional properties, vertex labels, edge labels, and indexes can be created, but the data type of a property, for instance, cannot be changed. 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.

22. Should you wish to find only the schema for a particular type of item in the `describe()` listing, additional steps can split the output per newline and grep for a string as shown for the `index`. Gremlin as shown in this notebook uses Groovy, so any Groovy commands manipulate graph traversals.

```groovy
gremlin> schema.describe().split("\n").grep(~/*.index*/)  
```

23. 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. Create a script file, `generateRecipe.groovy`, with the information shown below. Note the first command, `g.V().drop().iterate()`; this command drop all vertex and edge data from the graph before reading in new data.

```groovy
// Add all vertices and edges for Recipe
g.V().drop().iterate()

// author vertices
juliaChild = graph.addVertex(label, 'author', 'name', 'Julia Child', 'gender', 'F')
simoneBeck = graph.addVertex(label, 'author', 'name', 'Simone Beck', 'gender', 'F')
louissetteBertholie = graph.addVertex(label, 'author', 'name', 'Louisette 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')
```

// book vertices
Using DataStax Enterprise advanced functionality

```javascript

// recipe vertices
beefBourguignon = graph.addVertex(label, 'recipe', '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.')
ratatouille = graph.addVertex(label, 'recipe', 'name', 'Ratatouille', 'instructions', 'Peel and cut the eggplant. Make sure you cut eggplant into lengthwise slices that are about 1-inch wide, 3-inches long, and 3/8-inch thick')
saladeNicoise = graph.addVertex(label, 'recipe', '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.')
wildMushroomStroganoff = graph.addVertex(label, 'recipe', '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 saute pan over medium-high heat.')
spicyMeatloaf = graph.addVertex(label, 'recipe', '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.')
oystersRockefeller = graph.addVertex(label, 'recipe', 'name', 'Oysters Rockefeller', 'instructions', 'Saute the shallots, celery, herbs, and seasonings in 3 tablespoons of the butter for 3 minutes. Add the watercress and let it wilt.')
carrotSoup = graph.addVertex(label, 'recipe', '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.')
roastPorkLoin = graph.addVertex(label, 'recipe', '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.')

// ingredients vertices
beef = graph.addVertex(label, 'ingredient', 'name', 'beef')
onion = graph.addVertex(label, 'ingredient', 'name', 'onion')
mashedGarlic = graph.addVertex(label, 'ingredient', 'name', 'mashed garlic')
butter = graph.addVertex(label, 'ingredient', 'name', 'butter')
```
Using DataStax Enterprise advanced functionality

tomatoPaste = graph.addVertex(label, 'ingredient', 'name', 'tomato paste')
egggplant = graph.addVertex(label, 'ingredient', 'name', 'eggplant')
zucchini = graph.addVertex(label, 'ingredient', 'name', 'zucchini')
oliveOil = graph.addVertex(label, 'ingredient', 'name', 'olive oil')
yellowOnion = graph.addVertex(label, 'ingredient', 'name', 'yellow onion')
greenBean = graph.addVertex(label, 'ingredient', 'name', 'green beans')
tuna = graph.addVertex(label, 'ingredient', 'name', 'tuna')
tomato = graph.addVertex(label, 'ingredient', 'name', 'tomato')
hardBoiledEgg = graph.addVertex(label, 'ingredient', 'name', 'hard-boiled egg')
eggNoodles = graph.addVertex(label, 'ingredient', 'name', 'egg noodles')
mushroom = graph.addVertex(label, 'ingredient', 'name', 'mushrooms')
bacon = graph.addVertex(label, 'ingredient', 'name', 'bacon')
celery = graph.addVertex(label, 'ingredient', 'name', 'celery')
greenBellPepper = graph.addVertex(label, 'ingredient', 'name', 'green bell pepper')
groundBeef = graph.addVertex(label, 'ingredient', 'name', 'ground beef')
porkSausage = graph.addVertex(label, 'ingredient', 'name', 'pork sausage')
shallot = graph.addVertex(label, 'ingredient', 'name', 'shallots')
chervil = graph.addVertex(label, 'ingredient', 'name', 'chervil')
fennel = graph.addVertex(label, 'ingredient', 'name', 'fennel')
parsley = graph.addVertex(label, 'ingredient', 'name', 'parsley')
oyster = graph.addVertex(label, 'ingredient', 'name', 'oyster')
pernod = graph.addVertex(label, 'ingredient', 'name', 'Pernod')
thyme = graph.addVertex(label, 'ingredient', 'name', 'thyme')
carrot = graph.addVertex(label, 'ingredient', 'name', 'carrots')
chickenBroth = graph.addVertex(label, 'ingredient', 'name', 'chicken broth')
porkLoin = graph.addVertex(label, 'ingredient', 'name', 'pork loin')
redWine = graph.addVertex(label, 'ingredient', 'name', 'red wine')

// meal vertices
SaturdayFeast = graph.addVertex(label, 'meal', 'name', 'Saturday Feast', 'timestamp', '2015-11-30', 'calories', 1000)
EverydayDinner = graph.addVertex(label, 'meal', 'name', 'EverydayDinner', 'timestamp', '2016-01-14', 'calories', 600)
JuliaDinner = graph.addVertex(label, 'meal', 'name', 'JuliaDinner', 'timestamp', '2016-01-14', 'calories', 900)

// author-book edges
juliaChild.addEdge('authored', artOfFrenchCookingVolOne)
simoneBeck.addEdge('authored', artOfFrenchCookingVolOne)
louissetteBertholie.addEdge('authored', artOfFrenchCookingVolOne)
simoneBeck.addEdge('authored', simcasCuisine)
patriciaSimon.addEdge('authored', simcasCuisine)
Using DataStax Enterprise advanced functionality

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', 'year', 1961)
juliaChild.addEdge('created', 'ratatouille', 'year', 1965)
juliaChild.addEdge('created', 'saladeNicoise', 'year', 1962)
emerilLagasse.addEdge('created', 'wildMushroomStroganoff', 'year', 2003)
emerilLagasse.addEdge('created', 'spicyMeatloaf', 'year', 2000)
aliceWaters.addEdge('created', 'carrotSoup', 'year', 1995)
aliceWaters.addEdge('created', 'roastPorkLoin', 'year', 1996)
jamesBeard.addEdge('created', 'oystersRockefeller', 'year', 1970)

// recipe - ingredient edges
beefBourguignon.addEdge('includes', 'beef', 'amount', '2 lbs')
beefBourguignon.addEdge('includes', 'onion', 'amount', '1 sliced')
beefBourguignon.addEdge('includes', 'mashedGarlic', 'amount', '2 cloves')
beefBourguignon.addEdge('includes', 'butter', 'amount', '3.5 Tbsp')
beefBourguignon.addEdge('includes', 'tomatoPaste', 'amount', '1 Tbsp')
ratatouille.addEdge('includes', 'eggplant', 'amount', '1 lb')
ratatouille.addEdge('includes', 'zucchini', 'amount', '1 lb')
ratatouille.addEdge('includes', 'mashedGarlic', 'amount', '2 cloves')
ratatouille.addEdge('includes', 'oliveOil', 'amount', '4-6 Tbsp')
ratatouille.addEdge('includes', 'yellowOnion', 'amount', '1 1/2 cups or 1/2 lb thinly sliced')
saladeNicoise.addEdge('includes', 'oliveOil', 'amount', '2-3 Tbsp')
saladeNicoise.addEdge('includes', 'greenBean', 'amount', '1 1/2 lbs blanched, trimmed')
saladeNicoise.addEdge('includes', 'tuna', 'amount', '8-10 ozs oil-packed, drained and flaked')
saladeNicoise.addEdge('includes', 'tomato', 'amount', '3 or 4 red, peeled, quartered, cored, and seasoned')
saladeNicoise.addEdge('includes', 'hardBoiledEgg', 'amount', '8 halved lengthwise')
wildMushroomStroganoff.addEdge('includes', 'eggNoodles', 'amount', '16 ozs wmyIde')
wildMushroomStroganoff.addEdge('includes', 'mushroom', 'amount', '2 lbs wild or exotic, cleaned, stemmed, and sliced')
wildMushroomStroganoff.addEdge('includes', 'yellowOnion', 'amount', '1 cup thinly sliced')
spicyMeatloaf.addEdge('includes', 'bacon', 'amount', '3 ozs diced')
spicyMeatloaf.addEdge('includes', 'onion', 'amount', '2 cups finely chopped')
spicyMeatloaf.addEdge('includes', 'celery', 'amount', '2 cups finely chopped')
spicyMeatloaf.addEdge('includes', 'greenBellPepper', 'amount', '1/4 cup finely chopped')
spicyMeatloaf.addEdge('includes', porkSausage, 'amount', '3/4 lbs hot')
spicyMeatloaf.addEdge('includes', groundBeef, 'amount', '1 1/2 lbs chuck')
oystersRockefeller.addEdge('includes', shallot, 'amount', '1/4 cup chopped')
oystersRockefeller.addEdge('includes', celery, 'amount', '1/4 cup chopped')
oystersRockefeller.addEdge('includes', chervil, 'amount', '1 tsp')
oystersRockefeller.addEdge('includes', fennel, 'amount', '1/3 cup chopped')
oystersRockefeller.addEdge('includes', parsley, 'amount', '1/3 cup chopped')
oystersRockefeller.addEdge('includes', oyster, 'amount', '2 dozen on the half shell')
carrotpSoup.addEdge('includes', pernod, 'amount', '1/3 cup')
carrotpSoup.addEdge('includes', butter, 'amount', '4 Tbsp')
carrotpSoup.addEdge('includes', onion, 'amount', '2 medium sliced')
carrotpSoup.addEdge('includes', thyme, 'amount', '1 sprig')
carrotpSoup.addEdge('includes', carrot, 'amount', '2 1/2 lbs, peeled and sliced')
carrotpSoup.addEdge('includes', chickenBroth, 'amount', '6 cups')
roastPorkLoin.addEdge('includes', porkLoin, 'amount', '1 bone-in, 4-rib')
roastPorkLoin.addEdge('includes', redWine, 'amount', '1/2 cup')
roastPorkLoin.addEdge('includes', chickenBroth, 'amount', '1 cup')

// book - recipe edges
beefBourguignon.addEdge('includedIn', artOfFrenchCookingVolOne)
saladeNicoise.addEdge('includedIn', artOfFrenchCookingVolOne)
carrotpSoup.addEdge('includedIn', artOfSimpleFood)

// meal - recipe edges
beefBourguignon.addEdge('includedIn', SaturdayFeast)
carrotpSoup.addEdge('includedIn', SaturdayFeast)
oystersRockefeller.addEdge('includedIn', SaturdayFeast)
carrotpSoup.addEdge('includedIn', EverydayDinner)
roastPorkLoin.addEdge('includedIn', EverydayDinner)
beefBourguignon.addEdge('includedIn', JuliaDinner)
saladeNicoise.addEdge('includedIn', JuliaDinner)

// meal - book edges
EverydayDinner.addEdge('includedIn', artOfSimpleFood)
SaturdayFeast.addEdge('includedIn', simcasCuisine)
JuliaDinner.addEdge('includedIn', artOfFrenchCookingVolOne)
g.V()

Run the script by loading it in Gremlin console:

gremlin> :load /tmp/generateRecipe.groovy

replacing "/tmp" with the directory where you write the script.
Using DataStax Enterprise advanced functionality

// A series of returns for vertices and edges will mark the successful completion of the script
// Sample vertex
==>v[{"label=author, member_id=0, community_id=1878171264}]  
// Sample edge
==>e[{out_vertex={"label=meal, member_id=27, community_id=1989847424},
    local_id=545b88b0-0e7b-11e6-b5e4-0febe4822aa4,
    in_vertex={"label=book, member_id=10, community_id=1878171264},
    ~type=includedIn}]
  [{"label=meal, member_id=27, community_id=1989847424}-includedIn->
   {"label=book, member_id=10, community_id=1878171264}]

The property timestamp is a Timestamp data type that corresponds to a valid DSE database timestamp data type.

24. Run the vertex count again.

    gremlin> g.V().count()
    ==>56

A tool, graphloader, is also available for scripting data loading. See the graphloader (page 831) documentation for information.

Exploring the graph with graph traversals can lead to interesting conclusions.

25. With several author 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.

    gremlin> g.V().has('author','name','Julia Child')
    ==>v[{"label=author, member_id=0, community_id=1878171264}]

26. In this next traversal, has() gets the vertex information filtered with name = Julia Child. The traversal step outE() discovers the outgoing edges from that vertex with the authored label.

    gremlin> g.V().has('name','Julia Child').outE('authored')

The edge information is returned:

    ==>e[{out_vertex={"label=author, member_id=0, community_id=1878171264},
          local_id=521f5450-0e7b-11e6-b5e4-0febe4822aa4,
          in_vertex={"label=book, member_id=10, community_id=1878171264},
          ~type=includedIn}]
    [{"label=meal, member_id=27, community_id=1989847424}-includedIn->
     {"label=book, member_id=10, community_id=1878171264}]}
27. If instead, the query is seeking the books that all authors have written, the last example gets 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. Note 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`.

```
gremlin> g.V().outE('authored').inV().values('name')
```

```text
=> The Art of French Cooking, Vol. 1
=> Simca's Cuisine: 100 Classic French Recipes for Every Occasion
=> The Art of French Cooking, Vol. 1
=> The French Chef Cookbook
=> Simca's Cuisine: 100 Classic French Recipes for Every Occasion
=> The Art of French Cooking, Vol. 1
=> The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution
=> The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution
=> The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution
=> The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution
```

28. 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.

```
gremlin> g.V().outE('authored').inV().values('name').dedup()
```

```text
=> The Art of French Cooking, Vol. 1
=> 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
```
29. 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.

```java
gremlin> g.V().outE('authored').inV().values('name').dedup()
```

`==>
Simca's Cuisine: 100 Classic French Recipes for Every Occasion
==>
The Art of French Cooking, Vol. 1
==>
The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution
==>
The French Chef Cookbook`

30. Refine the traversal by reinserting the `has()` step for a particular author. Find all the books authored by Julia Child.

```java
gremlin> g.V().has('name','Julia Child').outE('authored').inV().values('name')
```

`==>
The Art of French Cooking, Vol. 1
==>
The French Chef Cookbook`

31. The last 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 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.

```java
gremlin> g.V().has('name','Julia Child').out('authored').values('name')
```

`==>
The Art of French Cooking, Vol. 1
==>
The French Chef Cookbook`

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.

32. Additional traversal steps continue to fine-tune the results. Adding another chained `has` traversal step finds only books authored by Julia Child that are published after 1967. This example also displays the use of the `gt`, or greater than function.

```java
gremlin> g.V().has('name','Julia Child').out('authored').has('year', gt(1967)).values('name')
```

`==>
The French Chef Cookbook`
33. When developing or testing, oftentimes a check of the number of vertices with each vertex label can confirm that data has been 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.

```gremlin
g.V().label().groupCount()
```

```java
==> {meal=3, ingredient=31, author=10, book=4, recipe=8}
```

34. Write your data to an output file to save or exchange information. A Gryo file is a binary format file that can be used to reload data to DSE Graph. In this next command, graph I/O is used to write the entire graph to a file. Other file formats can be written by substituting `gryo()` with `graphml()` or `graphson()`.

```gremlin
g.graph.io(gryo()).writeGraph("/tmp/recipe.gryo")
```

**Note:** `graph.io()` is disabled in sandbox mode.

```java
==> null
```

35. 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 864)`, in `Using DSE Graph Loader (page 831)`.

**What's next:**

Further adventures in traversing can be found in `Creating queries using traversals (page 798)`. If you want to explore various loading options, check out the `DSE Graph Loader (page 831)` or `Using DSE Graph (page 707)`.

**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`:

```bash
$ bin/dse gremlin-console
```
Using DataStax Enterprise advanced functionality

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

usage: gremlin.sh [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 ARG1 ARG2 ... Execute the specified script
(ARG2 ...) and close the console on completion
-h, --help Display this help message
-i, --interactive=SCRIPT ARG1 ARG2 ... Execute the specified script and leave the console open on completion
-l Set the logging level of components that use standard
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>independent of</th>
<th>logging output</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v, --version</td>
<td>the Console</td>
</tr>
<tr>
<td></td>
<td>Display the version</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:

  ```bash
  $ 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:

  ```bash
  $ 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.

- 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.

  ```groovy
  :help
  ```

  For information about Groovy, visit: [http://groovy-lang.org](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
Using DataStax Enterprise advanced functionality

For help on a specific command type:
:help command

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.

**Creating a graph in the Gremlin console**

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.

1. **Start the Gremlin console** *(page 773).*

2. **Create a simple graph with default settings to hold the data.**

   ```
   gremlin> system.graph('food').create()
   ==null
   ```

   **Note:** This command is not available if a graph traversal is aliased with the :remote config alias g some_graph.g command. In order to access the system command, reset the alias with :remote config alias reset

3. **Create a graph with non-default replication** *(page 952), systemReplication** *(page 953), and configuration settings** *(page 951):*

   ```
   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 database settings must use NetworkTopologyStrategy and a replication factor greater than one. If the graph is created with a replication setting of 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 NetworkTopologyStrategy, whereas for a single node, the replication strategy will default to SimpleStrategy. The number of nodes will determine the default replication factor:

<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>

4. 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.

```bash
> :remote config alias g food.g
==>:g=food.g
```

5. A list of all graphs can be retrieved with the following command:

```bash
> system.graphs()
==>:food
==>:test
```

Creating schema in the Gremlin console

Creating a data model (page 700) 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 packaged with the Apache TinkerPop Gremlin console. Use either Gremlin-Groovy to create a script that contains the Gremlin commands, or enter the commands directly into the Gremlin console.

1. Start the Gremlin console (page 773).

2. Create a new graph (page 776) to store the data and alias a graph traversal to run queries. If you are reusing a graph that you previously created, drop the graph schema and data (page 784).

3. Optional. If running large scripts, set the timeout value to max to prevent client-side timeouts. Use this setting to ensure that script processing will complete. This step cannot be completed in Studio.

```
gremlin> :remote config timeout max
```

4. 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")
```

5. A script that creates schema is shown in the Example at the bottom of this page. The script file is loaded and run on the remote Gremlin Server. This script will not work if you have previously run scripts from the Quick Start (page 754) unless the schema and any data has been cleared (page 784) from the graph.

```
gremlin> :load /tmp/RecipeSchema.groovy
```

6. The following steps show the details of the full script broken down into sections.

7. 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 954) 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 they are deleted and recreated, so use edge properties in situations that warrant their use.

```
// 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()
```
Using DataStax Enterprise advanced functionality

```java
// Example of a multiple property that can have several values
// Next 4 lines define two properties, then create a meta-property
'stars'.Int().create()

// A meta-property is a property of a property
// EX: 'livedIn': '1999-2005' 'country': 'Belgium'

// Example of creating vertex label with properties

// Example of custom vertex id:
`sensor_id`.Uuid().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.

8. Define the vertex labels. The vertex labels identify the type of vertices that can be created.

```java
// Vertex Labels

// Example of custom vertex id:
```

Vertex labels can be checked for prior existence using `ifNotExists()`. Vertex labels can be created along with properties. Vertex labels can be created with custom vertex ids (page 727), rather than the standard vertex ids (page 798).
Using DataStax Enterprise advanced functionality

**Notice:** Standard auto-generated ids are deprecated with DSE 6.0. Custom ids (page 727) will undergo changes, and specifying vertex ids with `partitionKey` and `clusteringKey` will likely become the normal method.

DSE Graph limits the number of vertex labels to 200 per graph.

9. 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('stars').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 934).

10. Define indexes (page 734) that can speed up the query processing. All types of indexes are presented here.

```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
//
// 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();
```

11. After creating the graph schema, examine the schema to verify. This command is included as the last command of the full script.

```java
schema.describe();
```
Using DataStax Enterprise advanced functionality

```java
==>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("ISBN").Text().single().create()
schema.propertyKey("name").Text().single().create()
schema.propertyKey("comment").Text().single().create()
schema.propertyKey("category").Text().single().create()
schema.propertyKey("timestamp").Timestamp().single().create()

schema.edgeLabel("authored").multiple().create()

// 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()
```

// RECIPE SCHEMA

// To run in Studio, copy and paste all lines to a cell and run.

// To run in Gremlin console, use the load command
// :load /tmp/RecipeSchema.groovy
Using DataStax Enterprise advanced functionality

```java
// Example of a multiple property that can have several values
// Next 4 lines define two properties, then create a meta-property
'mealIn' on 'country'
// A meta-property is a property of a property
// EX: 'meatIn': '1999-2005' 'country': 'Belgium'
// schema.propertyKey('nickname').Text().multiple().create()
// schema.propertyKey('country').Text().create()

// schema.propertyKey('meatIn').Text().create()

// Vertex Labels
schema.vertexLabel('author').ifNotExists().create()
schema.vertexLabel('recipe').create()
// Example of creating vertex label with properties
//
// schema.vertexLabel('recipe').properties('name','instructions').create()
// schema.vertexLabel('ingredient').create()
// schema.vertexLabel('book').create()
// schema.vertexLabel('meal').create()
// schema.vertexLabel('reviewer').create()
// Example of custom vertex id:
// schema.propertyKey('city_id').Int().create()
// schema.propertyKey('sensor_id').Uuid().create()
//
// schema().vertexLabel('FridgeSensor').partitionKey('city_id').clusteringKey('sensor_id').create()

// Edge Labels
schema.edgeLabel('authored').ifNotExists().create()
schema.edgeLabel('created').create()
schema.edgeLabel('includes').create()
schema.edgeLabel('includedIn').create()
// schema.edgeLabel('rated').properties('stars').connection('reviewer','recipe').create()

// 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()
```
Using DataStax Enterprise advanced functionality

```java
//
schema.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()

// 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()
```

### Modifying schema

Schema creation is an important part of creating a graph database. It can be necessary to add or modify the schema after initial creation. In **Development mode**, the schema can be modified after data creation. In **Production mode**, schema creation and data loading cannot be mixed. Property keys can be added. Adjacencies can be identified.

#### Add property keys to a vertex label

- Add a property key after schema creation. The property key must already exist. In the example, the first command builds the property key for the graph, and the second command adds the property key to the vertex label `author`.

```java
gremlin> schema.propertyKey('nationality').Text().create()
gremlin>
schema.vertexLabel('author').properties('nationality').add()

==>null
```

- Verify that the property key is built for the vertex label `author`. Look for the property key named `nationality`.

```java
gremlin> schema.vertexLabel('author').describe()

==>schema.vertexLabel("author").properties("nationality",
    "name", "gender").create()
schema.vertexLabel("author").index("byName").secondary().by("name").add()
```

The properties `name` and `gender` existed prior to the addition of `nationality`. Any indexes on the vertex label are also displayed.

- Add a value for the newly added property key to a vertex.
Using DataStax Enterprise advanced functionality

```
gremlin> g.V().has('author','name','Julia Child').property('nationality','American')
gremlin> g.V().hasLabel('author').valueMap()
  ==>[gender:[F], name:[Alice Waters]]
  ==>[gender:[F], name:[Louisette Bertholie]]
  ==>[gender:[M], name:[Fritz Streiff]]
  ==>[gender:[F], nationality:[American], name:[Julia Child]]
```

**Identify the adjacency for two vertices**

- Create a vertex label with properties. All the properties must exist prior to creating the vertex label. Add an edge label that identifies the connection (page 934) outgoing vertex and incoming vertex.

```
gremlin> schema.vertexLabel('FridgeItem').properties('name','expiration_date', 'amount').add()
gremlin> schema.edgeLabel('isA').connection('ingredient','FridgeItem').create()
schema.edgeLabel('isA').describe()
schema.edgeLabel("isA").multiple().create()
schema.edgeLabel("isA").connection("ingredient", "FridgeItem").connection("FridgeItem", "ingredient").add()
```

**Dropping data, schema, and graphs**

Data, schema, and graphs can be dropped in the Gremlin console as follows:

**Drop data**

- To drop all data without dropping a graph and schema, drop all vertices.

```
gremlin> g.V().drop().iterate()
```

- To drop specific data, such as all `author` vertices, identify the vertices along with a `drop` traversal step.

```
gremlin> g.V().hasLabel('author').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:

```
g.V().hasLabel('author').limit(100).drop()
```

and repeat until all vertices are dropped.

- To drop a property key from an edge, such as `rated` edges, identify the edges, the property key `stars` along with a `drop` traversal step.
Using DataStax Enterprise advanced functionality

```java
gremlin> g.E().hasLabel('rated').properties('stars').drop()
```

This query will drop the property key `stars` for all edges that have a `rated` edge label.

```java
gremlin> g.E().hasLabel('rated').properties('stars').valueMap()
```

returns no values.

**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.

### Drop schema
- To drop the schema and all data without dropping the graph, use a `clear()` 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
gremlin> schema.clear()
```  

```java
==>null
```

**Important:** Currently, certain schema elements such as a vertex label cannot be individually modified or removed. If a change to the schema is necessary, drop the whole schema as detailed above and recreate.

### 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.

```java
gremlin> schema.vertexLabel('meal').describe()
```  

```java
==>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').remove()
```  

```java
==>null
```

### Dropping a graph
- Dropping a graph will clear all schema and data as well as deleting the graph. A system command is required to drop a graph. In order to use system commands, the graph traversal alias must be cleared. A configuration reset clears the alias.

```java
gremlin> :remote config alias reset
```
Using DataStax Enterprise advanced functionality

---

**Note:** System commands are not accessible when a graph is aliased.

```---Aliases cleared
```

- Optional: If unsure of the graph name, examine what graphs exist.

```gremlin> system.graphs()
```

```==>food
```

- Drop the desired graph.

```gremlin> system.graph('food').drop()
```

```==>null
```

## Inserting data using Gremlin

The discussion below describes the detailed view of submitting individual commands to create vertices and edges. However, due to the sessionless nature of the interaction between the Gremlin console and the remote Gremlin server, all steps must be submitted in a single script. To load the data shown in the steps below, run the `generateRecipe.groovy` script (page 671).

Two methods are shown below, using graph methods and using graph traversal methods. The graph methods have been benchmarked as several times faster for insertions in informal testing. The graph traversal methods are useful for on-the-fly insertions during development.

Although DSE Graph will insert data without previously creating schema, it is best practice to create schema (page 777) prior to data insertion. If inserted without schema, the data type will often be incorrect. For instance, in the examples below, **ISBN** is a **UUID**. However, if inserted as shown, the data type assigned to **ISBN** will be **Text**.

### Using the graph methods `addVertex()` and `addEdge()`

1. Add an **author** vertex using `addVertex()`.

   ```juliaChild = graph.addVertex(label,'author', 'name','Julia Child', 'gender','F')
   ```

   The vertex is given a descriptive name `juliaChild`, and is given a vertex label of **author**. The **name** and **gender** key-value pairs for the property keys are listed.

2. Add a **book** vertex using `addVertex()`.

A book vertex includes a book name, publishing year, and an ISBN code, if available.

3. Add an edge between an author and a book using addEdge().

juliaChild.addEdge('authored', frenchChefCookbook)

The vertex for Julia Child has an authored edge to the book The French Chef Cookbook.

4. Edges can also have properties, as this example of an edge between a recipe and an ingredient demonstrates.

beefBourguignon.addEdge('includes', beef, 'amount', '2 lbs')

The recipe Beef Bourguignon includes 2 pounds of beef. The amount is stored as an edge property. The recipe and the ingredient vertices must be inserted prior to the edge.

Using the graph traversal methods addV() and add()

5. Add an author vertex using addV(). and property(). Note the use of multiple property() steps, one for each property set.

g.addV('author').property('name', 'Julia Child').property('gender', 'F')

A vertex label of author is defined in the addV() step. The name and gender key-value pairs for the property keys are created using two property() steps.

6. Add a book vertex using addV().


A book vertex includes a book name, publishing year, and an ISBN code, if available.

7. Add an edge between an author and a book using addE().

g.V().has('author', 'name', 'Julia Child').as('a').V().has('book', 'name', 'The French Chef Cookbook').addE('authored').to('a')
The vertex for *Julia Child* has an authored edge to the book *The French Chef Cookbook*. Each vertex is identified by its vertex label and property name. The author vertex is labeled as `a` and used in the `to()` step.

8. Edges can also have properties, as this example of an edge between a *recipe* and an *ingredient* demonstrates.

   ```java
   g.V().has('recipe', 'name', 'Beef Bourguignon').as('a').
   V().has('ingredient', 'name', 'beef').
   addE('includes').from('a').property('amount', '2 lbs')
   ```

   The recipe *Beef Bourguignion* includes 2 pounds of beef. The `amount` is stored as an edge property. The *recipe* and the *ingredient* vertices must be inserted prior to the edge.

**Using GraphSON**

Inserting data with GraphSON, a JSON style file format.

**Inserting data using GraphSON**

GraphSON is JSON data that defines a vertex with its ID and label, outgoing and incoming edges, and properties.

1. Identify the vertex ID and label.

   ```json
   {
     "id": {
       "~label": "author", "member_id": 35, "community_id": 1733329920,
       label": "author",
   }
   ```

   Each vertex must have an identifier. Generally, a label is defined to denote the type of vertex. In this example, the vertex shown is an *author* vertex.

2. Identify the vertex properties.

   ```json
   "properties": {"gender":{,
   "id":{
   "local_id": "00000000-0000-8084-0000-000000000000",
   "~type": "genre",
   "out_vertex":
   {"~label": "author", "member_id": 35, "community_id": 1733329920},
   "value": "F",
   
   "name":{
   "id":{
   "local_id": "00000000-0000-8083-0000-000000000000",
   "~type": "name",
   ```
Using DataStax Enterprise advanced functionality

```
"out_vertex":
{"~label":"author","member_id":35,"community_id":1733329920},
"value":"Simone Beck"}
}
```

Each property is identified with a name, ID, and value. The example shows the author name, which has a value of Simone Beck, and gender valued as F.

3. Add outgoing edges.

```
"outE":{
  "authored":{
    "id":{
      "out_vertex":
      {"~label":"author","member_id":35,"community_id":1733329920},
      "local_id":"7208b3c-0e6c-11e6-b5e4-0febe4822aa4",
      "in_vertex":
      {"~label":"book","member_id":52,"community_id":1733329920},
      "~type":"authored"
    },
    "inV":
    {"~label":"book","member_id":52,"community_id":1733329920}),
  },
  "inV":
  {"~label":"book","member_id":54,"community_id":1733329920}),
}
```

Each edge must have a label and ID. The example shows an authored edge. For an outgoing edge as shown in the example, an incoming vertex, inV, must be identified by vertex identification.

4. Close and name the GraphSON file, using a suffix of ".json".

**Loading and writing data using GraphSON**

The Tinkerpop GraphSON Reader can be used to load and write data.

1. Start the Gremlin console (page 773).

2. Start a graph instance and create the schema (page 777).

3. Use the DSE Graph Loader to load the GraphSON file (page 866).
4. Writing data out of the graph into a GraphSON file while capturing the original data types of the inserted data can be accomplished with a short script:

```java
gremlin> f = new FileOutputStream("/tmp/recipe_lossless.json");
mapper = graph.io(graphson()).mapper().embedTypes(true).create();
graph.io(graphson()).writer().mapper(mapper).create().writeVertex(f,g.V().next(),BOTH)
```==null

A sample of the output shows the class information and data types:

```json
{
   "@class":"java.util.HashMap",
   "id":{
      "@class":"java.util.HashMap",
      "~label":"meal",
      "member_id":["java.lang.Long",25],
      "community_id":1989847424
   },
   "label":"meal",
```

**Using GraphML**

Inserting data with GraphML, an XML file format.

**Identifying graph schema in GraphML**

Defining the vertices and edges along with their properties is a critical first step in creating a graph database in GraphML. GraphML is an XML format that identifies vertices and edges. The first step in defining a graph in GraphML involves defining the vertex and edge properties. Note that a drawback of using GraphML is that indexes cannot be defined.

1. The GraphML header in the file identifies XML information:

```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">
```

2. Create a graph instance to identify schema in a new file.

```xml
<graph id="graph" edgedefault="directed">
```

3. Define the vertex properties.

```xml
<key id="labelV" for="node" attr.name="labelV" attr.type="string"/>
<key id="id" for="node" attr.name="id" attr.type="int"/>
```
The property with id labelV is always defined, as each vertex has a vertex label. Additional vertex properties are defined with an ID, attribute name, and attribute type. Note that the for="node" statement defines each property as a vertex property.

4. Define the edge properties.

The property with id labelE is always defined, as each edge has an edge identifier denoted by the label. Additional edge properties are defined with an ID, attribute name, and attribute type. Note that the for="edge" statement defines each property as an edge property.

**Inserting data using GraphML**

After the vertex and edge properties are defined, the graph data can be defined. Because GraphML is XML data, the node element `<node ...>` is used to define a vertex, and the edge element `<edge ...>` is used to define an edge.

In general, GraphML files have been written from graphs that have previously been created. In DSE Graph, the vertex and edge identification is composed of three parts plus a label value that are created when a vertex is added with another method like a Gremlin command. These instruction dissect a GraphML file to explain the features.
1. Add vertex data by appending to the GraphML file started in Identifying graph schema (page 790), add vertex data. Each vertex is given a vertex label and some properties. Each vertex property is a key-value pair, using the previously created property keys.

```
<node id="{~label=meal, member_id=21, community_id=1579328896}">
  <data key="labelV">meal</data>
  <data key="name">Saturday Feast</data>
  <data key="calories">1000</data>
  <data key="timestamp">2015-11-30T00:00:00Z</data>
</node>
```

Each vertex must have an identifier. Generally, a label is defined to denote the type of vertex. In this example, the vertex shown is a meal vertex. Subsequent properties are defined in this example, such as name and calories.

2. Add an edge.

```
<edge id="{
  out_vertex={~label=meal, member_id=22, community_id=1579328896},
  local_id=62b12e83-0e6a-11e6-b5e4-0febe4822aa4,
  in_vertex={~label=book, member_id=27, community_id=1579328896},
  ~type=includedIn"
  source="{~label=meal, member_id=22, community_id=1579328896}"
  target="{~label=book, member_id=27, community_id=1579328896}"
}"
  <data key="labelE">includedIn</data>
</edge>
```

Each edge must have an identifier. In addition, the source and target vertices must be identified to define the starting vertex and the ending vertex of an edge. As with the vertices, a label is defined to denote the type of edge.

3. A GraphML file has ending tags to close the XML statements.

```
</graph></graphml>
```

4. Close and name the GraphML file, using a graphml file extension.

**Loading data using GraphML**

The Apache TinkerPop GraphML Reader can be used to load data.

1. Start the Gremlin console (page 773).

2. Start a graph instance and create the schema (page 777).
3. Use the DSE Graph Loader to load the GraphML file (page 867).

**Using Gryo**

Inserting data from a Gryo file.

**Loading data using Gryo**

One file format for importing and exporting data to and from DSE Graph is Gryo, a binary file format.

1. Start the Gremlin console (page 773).
2. Start a graph instance and create the schema (page 777).
3. Use the DSE Graph Loader to load the Gryo file (page 864).

**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 `graph.schema()` calls can be used to check how the graph is storing data.

- Use the graph traversal instance `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.

```
g.V().count()
g.V().count()  
```  

```text
=>56
```

- Check the properties of a loaded vertex. Find all the information for the vertex with a name value of Julia Child.

```
g.V().has('name','Julia Child').valueMap()
g.V().has('name', 'Julia Child').valueMap()
```  

```text
=>{gender=[F], name=[Julia Child]}
```

- Check the properties of a loaded edge. Find all the information for the edges with a label of rated.

```
g.E().hasLabel('rated').values()
g.E().hasLabel('rated').values()
```  

```text
=>5  
=>Pretty tasty!  
=>2014-01-01T00:00:00Z
```

- Find the id information for vertices:
• Discover schema information using a `describe()` step. This traversal step provides a
sorted list of the same information as the next alternative below.

```
schema.describe()
```
Using DataStax Enterprise advanced functionality

```java
schema.edgeLabel("includes").connection("ingredient", "recipe").connection("recipe", "ingredient").add()
gremlin> schema.edgeLabel('includes').describe()
===>schema.edgeLabel("includes").multiple().properties("amount").create()
schema.edgeLabel("includes").connection("ingredient", "recipe").connection("recipe", "ingredient").add()
gremlin> schema.vertexLabel('author').describe()
===>schema.vertexLabel("author").properties("name", "gender").create()
```

- An alternative to discover schema information uses a `valueMap()` step on the traversal.

```java
schema.traversal().V().valueMap()
```

```
==>{mode=[Development]}
==>{name=[author]}
==>{name=[recipe]}
==>{name=[ingredient]}
==>{name=[book]}
==>{name=[meal]}
==>{name=[reviewer]}
==>{name=[byName], type=[Secondary]}
==>{name=[includedIn], directionality=[Bidirectional], cardinality=[Multiple]}
==>{name=[fridgeItem_single]}
==>{name=[rated], directionality=[Bidirectional], cardinality=[Multiple]}
==>{name=[fridgeItem_multiple]}
==>{dataType=[Timestamp], name=[timestamp], cardinality=[Single]}
==>{dataType=[Text], name=[ISBN], cardinality=[Single]}
==>{dataType=[Text], name=[category], cardinality=[Single]}
==>{dataType=[Int], name=[year], cardinality=[Single]}
==>{dataType=[Text], name=[gender], cardinality=[Single]}
==>{unique=[false], name=[byIngredient], type=[Materialized]}
==>{dataType=[Text], name=[instructions], cardinality=[Single]}
==>{unique=[false], name=[byReviewer], type=[Materialized]}
==>{unique=[false], name=[byRecipe], type=[Materialized]}
==>{unique=[false], name=[byMeal], type=[Materialized]}
==>{dataType=[Int], name=[stars], cardinality=[Single]}
==>{dataType=[Text], name=[comment], cardinality=[Single]}
==>{dataType=[Int], name=[calories], cardinality=[Single]}
==>{dataType=[Text], name=[blah], cardinality=[Single]}
==>{dataType=[Text], name=[amount], cardinality=[Single]}
==>{name=[created], directionality=[Bidirectional], cardinality=[Multiple]}
==>{name=[includes], directionality=[Bidirectional], cardinality=[Multiple]}
==>{dataType=[Bigint], name=[member_id], cardinality=[Single]}
==>{name=[authored], directionality=[Bidirectional], cardinality=[Multiple]}
```
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.

```java
graph.schema().traversal().V().valueMap(true)
```
Running `traversal()` will supply information about the number of schema element exist for vertices and edges, as well as the `TraversalSource` type.

```
  schema.traversal()
```

```
  >>> graphtraversalsource[tinkergraph[vertices:58 edges:106], standard]
```

A list of all vertex labels using utilities `split()` and `grep()`.

```
  schema.describe().split('\n').grep(~/.*vertexLabel.*\n)
```

```
  gremlin> schema.describe('n').grep(~/.*vertexLabel\n)
  >>> schema.vertexLabel("meal").properties("name", "timestamp", "calories").create()
  >>> schema.vertexLabel("ingredient").properties("name").create()
  >>> schema.vertexLabel("ingredient").index("byIngredient").materialized().by("name").add()
  >>> schema.vertexLabel("test").partitionKey("tester").clusteringKey("foo").create()
  >>> schema.vertexLabel("FridgeSensor").create()
  >>> schema.vertexLabel("FridgeItem").properties("name", "amount").create()
  >>> schema.vertexLabel("recipe").properties("name", "instructions").create()
  >>> schema.vertexLabel("recipe").index("byRecipe").materialized().by("name").add()
  >>> schema.vertexLabel("reviewer").properties("name").create()
  >>> schema.vertexLabel("reviewer").index("byReviewer").materialized().by("name").add()
  >>> schema.vertexLabel("reviewer").index("ratedByStars").outE("rated").by("stars").add()
```
Using DataStax Enterprise advanced functionality

- Get the name of the current graph.
  
  ```java
  graph.name()
  ```
  ```
  ==>quickstart
  ```

## 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 754) traversals that increase in complexity in a stepwise fashion. The examples shown here will continue with the Recipe Toy Graph example (page 708).

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:

```java
  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()
  ==>v[{
  ~label=ingredient, member_id=18, community_id=1989847424}
  ==>v[{
  ~label=ingredient, member_id=19, community_id=1989847424}
  ==>v[{
  ~label=ingredient, member_id=16, community_id=1989847424}
  ==>v[{
  ~label=ingredient, member_id=17, community_id=1989847424}
  ==>v[{
  ~label=ingredient, member_id=22, community_id=1989847424}
  ==>v[{
  ~label=ingredient, member_id=13, community_id=1989847424}
  ==>v[{
  ~label=meal, member_id=25, community_id=1989847424)
  ==>v[{
  ~label=ingredient, member_id=24, community_id=1989847424]
  ==>v[{
  ~label=recipe, member_id=14, community_id=1878171264]
  ==>v[{
  ~label=recipe, member_id=21, community_id=1878171264]
  ==>v[{
  ~label=recipe, member_id=19, community_id=1878171264]
  ==>v[{
  ~label=meal, member_id=27, community_id=1989847424]
  ==>v[{
  ~label=recipe, member_id=20, community_id=1878171264]
  ==>v[{
  ~label=meal, member_id=26, community_id=1989847424]
  ==>v[{
  ~label=book, member_id=13, community_id=1878171264]
  ==>v[{
  ~label=book, member_id=10, community_id=1878171264]
  ==>v[{
  ~label=book, member_id=11, community_id=1878171264]
  ```

  ```
  ```

Page 798 DSE 5.1 Developer Guide (Previous version)
Filter out the vertices labeled as a recipe with `hasLabel('recipe')`
Only the vertices that are recipes will be returned:

```plaintext
gremlin> g.V().hasLabel('recipe')
==>
[~label=recipe, member_id=14, community_id=1878171264]
==>
[~label=recipe, member_id=21, community_id=1878171264]
==>
[~label=recipe, member_id=19, community_id=1878171264]
==>
[~label=recipe, member_id=20, community_id=1878171264]
==>
[~label=recipe, member_id=17, community_id=1878171264]
==>
[~label=recipe, member_id=18, community_id=1878171264]
==>
[~label=recipe, member_id=15, community_id=1878171264]
==>
[~label=recipe, member_id=16, community_id=1878171264]
```

Count the number of vertices with `count()`
The number of vertices returned from the last traversal step is totalled:

```plaintext
gremlin> g.V().hasLabel('recipe').count()
==>
8
```

Standard vertex ids are auto-generated, and are guaranteed to be unique. The standard vertex id consists of three parts:

- **member_id**
  - vertex ID within a group
- **community_id**
  - community ID within a graph
- **label**
  - The specified vertex label

Standard vertex ids are synthetic and have a small footprint. The composition is not tied to a domain and are more flexible. Graph partitioning is an important aspect for retrieving graph objects. DSE Graph uses an optimizing algorithm to set the `member_id` and `community_id` for each vertex. The relationship is:

- A graph is a collection of disjoint communities
- A community is a collection of disjoint member vertices

Disjoint sets have no element in common. Therefore, a vertex is a member of exactly one community. In the example above, all vertices are in a couple of communities. The `member_id` is set to a value within each community.

Custom vertex ids can also be created using natural, or externally generated keys. However, applications using custom vertex ids must be manually partitioned and the guarantee of unique keys are up to the user.

**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`):
Using DataStax Enterprise advanced functionality

// 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', Instant.parse('2014-01-01T00:00:00.00Z'), 'stars', 5, 'comment', 'Pretty tasty!')
johnSmith.addEdge('rated', beefBourguignon, 'timestamp', Instant.parse('2014-01-23T00:00:00.00Z'), 'stars', 4)
janeDoe.addEdge('rated', beefBourguignon, 'timestamp', Instant.parse('2014-02-01T00:00:00.00Z'), 'stars', 5, 'comment', 'Yummy!')
sharonSmith.addEdge('rated', beefBourguignon, 'timestamp', Instant.parse('2015-01-01T00:00:00.00Z'), 'stars', 3, 'comment', 'It was okay.')
johnDoe.addEdge('rated', spicyMeatLoaf, 'timestamp', Instant.parse('2015-12-31T00:00:00.00Z'), 'stars', 4, 'comment', 'Really spicy - be careful!')
sharonSmith.addEdge('rated', spicyMeatLoaf, 'timestamp', Instant.parse('2014-07-23T00:00:00.00Z'), 'stars', 3, 'comment', 'Too spicy for me. Use less garlic.')
janeDoe.addEdge('rated', 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.
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('rated')` would be more precise if the target result are only ratings.

Parsing out the comment property from the rated edges
Using DataStax Enterprise advanced functionality

Here, the `inE()` is specified with the edge label `rated`. The property values are retrieved for the property key `comment`:

```
gremlin> g.V().hasLabel('recipe').has('name', 'Beef Bourguignon').inE('rated').values('comment')
===>Yummy!
===>Pretty tasty!
===>It was okay.
```

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?

```
gremlin>
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 Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution, Recipe:Carrot Soup]
```

One expects that the traversal path from `ingredient` to `recipe` to `book`. To check if this assumption is correct, add `path()` to the end of the traversal.

```
gremlin>
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, member_id=22, community_id=1878171264]},
v[~label=recipe, member_id=14, community_id=1878171264]},
 v[~label=book, member_id=10, community_id=1878171264]},
(Book=The Art of French Cooking, Vol. 1, Recipe=Beef Bourguignon])
==>[v[~label=ingredient, member_id=21, community_id=1989847424]},
v[~label=recipe, member_id=20, community_id=1878171264]},
 v[~label=book, member_id=13, community_id=1878171264]},
(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.

Traversals 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:
Using DataStax Enterprise advanced functionality

gremlin> g.V().hasLabel('recipe').has('name', 'Beef Bourguignon').inE('rated').values('comment').profile()

==>Traversal Metrics

Step                                      Count  Traversers       Time (ms)    % Dur
=============================================================================================================  
DsegGraphStep([~label.eq(recipe), name.eq(Beef ...  1           1           0.979    73.00
  query-optimizer                      0.184
  retrieve-new                        0.115
  iterator-setup                      0.390
DsegVertexStep(IN,[rated],edge)        4           4           0.286    21.37
  query-optimizer                      0.080
  retrieve-new                         0.014
  iterator-setup                       0.062
DsegPropertiesStep([comment],value)    3           3           0.075     5.63
  query-optimizer                      >TOTAL
  retrieve-new                        0.075
  iterator-setup                      1.342

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:

==>Traversal Metrics

Step                                      Count  Traversers       Time (ms)    % Dur
=============================================================================================================  
DsegGraphStep([~label.eq(recipe), name.eq(Beef ...  1           1           0.733    70.62
  query-optimizer                      0.143
  retrieve-new                         0.059
  iterator-setup                       0.289
Using DataStax Enterprise advanced functionality

DsegVertexStep(IN,[rated],edge)
4 4 0.241 23.29
  query-optimizer 0.083
  retrieve-new 0.006
  iterator-setup 0.049
DsegPropertiesStep([comment],value)
3 3 0.063 6.10
>TOTAL
  - - 1.038 -

The change made is subtle. Two traversal steps, `hasLabel('recipe').has('name', 'Beef Bourguignon')` have been replaced by one traversal step, `has('recipe', 'name', 'Beef Bourguignon')`. Although each measurement can vary, generally the second traversal will outperform the first traversal.

In DSE 5.1 and later, DSE Studio 2.0 provides more information on metrics such as `index-query`, showing the type of query used (Search, Materialized, Secondary). The two examples shown here display a materialized view index and a search index in use:

Using a custom vertex id

Vertices can be filtered using a `hasId()` traversal step and providing a particular custom vertex id.

```java
g.V().hasId(['~label':'FridgeSensor','city_id':100,'sensor_id':'60bcae02-f6e5-11e5-9ce9-5e5517507c66']).valueMap()
```

A vertex can be queried directly using a custom vertex id.
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 run the generateRecipe.groovy script (page 671) 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', '2015-12-31T00:00:00.00Z', '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.
Using DataStax Enterprise advanced functionality

```groovy
  gremlin> :load /tmp/generateReviews.groovy
```

The recipes that were previously entered are quired 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
  gremlin> g.V().hasLabel('reviewer').count()
  ==>5
```

List all the reviewer using `values`:

```groovy
  // Get the names of all the reviewers
  gremlin> g.V().hasLabel('reviewer').values('name')
  ==>John Smith
  ==>Sharon Smith
  ==>Betsy Jones
  ==>Jane Doe
  ==>John Doe
```

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.

```javascript
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:

```javascript
=> Pretty tasty!
=> Really spicy - be careful!
```

It might be nice to know which recipes John Doe reviewed, so another traversal can be used.
Using DataStax Enterprise advanced functionality

```
g.V().has('reviewer', 'name', 'John Doe').outE('rated').inV().values('name')
```

resulting in:
```
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:
Using DataStax Enterprise advanced functionality

```sql
gremlin> g.E().hasLabel('rated').has('stars', gt(3)).valueMap()
==>[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')`:

```sql
gremlin> g.E().hasLabel('rated').has('stars',
  gt(3)).inV().values('name')
==>Beef Bourguignon
==>Spicy Meatloaf
==>Beef Bourguignon
==>Carrot Soup
==>Beef Bourguignon
```

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.

```sql
gremlin> g.E().hasLabel('rated').has('stars',
  gte(4)).has('timestamp',
  gte(Instant.parse('2015-01-01T00:00:00.00Z'))).valueMap()
==>[comment:Loved this soup! Yummy vegetarian!,
  timestamp:2015-12-30T00:00:00Z, stars:5]
==>[comment:Really spicy - be careful!, timestamp:2015-12-31T00:00:00Z,
  stars:4]
```

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?

```sql
gremlin> g.E().hasLabel('rated').values('stars').mean()
==>.142857142857143
```

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.
Using DataStax Enterprise advanced functionality

gremlin> g.V().hasLabel('reviewer').map(outE('rated').count()).max() ==>2

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
gremlin> g.V().hasLabel('reviewer').as('reviewer','starCount').select('reviewer','starCount')
```
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:Betsy Jones, starCount:0.0] ==>[reviewer:Jane Doe, starCount:5.0] ==>[reviewer:John Doe, starCount:4.5] ==>[reviewer:John Smith, starCount:4.0] ==>[reviewer:Sharon Smith, starCount:3.0]
```

Note that now Betsy Jones has a starCount of 0.0, the true value.

Find the star rating each reviewer has given to recipes:

```
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.

```plaintext
=> {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.

```plaintext
g.V().hasLabel('recipe').as('recipe','numberOfReviews','meanRating').
    select('recipe','numberOfReviews','meanRating').
    by('name').
    by(inE('rated').count()).
    by(inE('rated').values('stars').mean())
```

```plaintext
=> {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.

```plaintext
g.V().hasLabel('recipe').out().has('name','beef').in().hasLabel('recipe').values('name')
```

```plaintext
=> Beef Bourguignon
```

A modification allows a query that includes either one ingredient or another.

```plaintext
g.V().hasLabel('recipe').out().has('name',within('beef','carrots')).in().hasLabel('recipe')
```

```plaintext
=> Beef Bourguignon
```
Using DataStax Enterprise advanced functionality

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:

- 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:

- `{ISBN=[0-394-40135-2], year=[1968], name=[The French Chef Cookbook]}
- `{year=[1961], name=[The Art of French Cooking, Vol. 1]}`

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:

- 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,

Another example groups all books by year and displays a listing of each year books were published followed by the book titles:

\[g.V().hasLabel('book').group().by('year').by('name')\]

and lists:


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:

\[g.V().hasLabel('author').as('author').out().properties('year').as('year').select('author','year').by('name').by().limit(2)\]

This query results in returning the first two records in the database:

\[==>{author=Julia Child, year=vp[year->1961]}\]
\[==>{author=Julia Child, year=vp[year->1968]}\]

Using local(), change this query to find the first two books that each author in the graph has published:

\[g.V().hasLabel('author').as('author').out().properties('year').as('year').select('author','year').by('name').by().limit(2)\]
Using DataStax Enterprise advanced functionality

```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
```

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.

**Important:** All points must be specified in `(longitude, 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 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. See the note on the difference (page 720) between DSE Search-backed geospatial queries and non-index-backed queries.

**Schema and data**

The examples here use the following schema:
Using DataStax Enterprise advanced functionality

```
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:

```
// 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))
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.polygone(2.352222, 48.856614, -0.127758, 51.507351,
-6.26031, 53.349805))
graph.addVertex(label, 'polyLocation','name', 'LondonDublinAachen',
'polygon',Geo.polygone(-0.127758, 51.507351, -6.26031, 53.349805,
6.083887, 50.775346))
graph.addVertex(label, 'polyLocation','name', 'DublinAachenTokyo',
'polygon',Geo.polygone(-6.26031, 53.349805, 6.083887, 50.775346,
139.691706, 35.689487))
```

The example data has the following approximate distances:

```
// 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):

```java
g.V().
  has('location', 'point', Geo.point(2.352222, 48.856614)).
  valueMap()
```

results in:

```json
{name=[Paris], point=[POINT (2.352222 48.856614)]}
```

Find the stored data that matches a line mapped to the specified points:

```java
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:

```java
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):

```
g.V().
  has('location', 'point', Geo.inside(Geo.point(2.352222, 48.856614),
    4.2, Geo.Unit.DEGREES)).
  values('name')
```

lists:

```text
>>> Paris
>>> London
>>> Aachen
```

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:

```text
>>> 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 if a JAR file from the JTS library is added to DSE Search (page 596). Be sure to add the JAR file before attempting to insert the graph data.

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:

```text
>>> Dublin
```
This result 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:

```java
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, 55.14827321753834))).
values('name')
```

lists:

```java
===>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

```java
g.V()
  .has('lineLocation', 'line', Geo.inside(Geo.polygon(-6.26031, 53.349805, 6.083887, 50.775346, 139.691706, 35.689487))).
values('name')
```

lists:

```java
===>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:

```java
//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()
```
Using DataStax Enterprise advanced functionality

// Search
schema.vertexLabel('author').index('search').search().by('name').asString().ifNotExists().add();

The examples use the following data:

/** VERTICES **/
/** AUTHOR VERTICES **/
juliaChild = graph.addVertex(label, 'author', 'name', 'Julia Child',
  'gender', 'F')
simoneBeck = graph.addVertex(label, 'author', 'name', 'Simone Beck',
  'gender', 'F')
louisetteBertholie = 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, 41.871136));
tokyo = graph.addVertex(label, 'place', 'name', 'Tokyo', 'location',
  Geo.point(139.6917, 35.6895));

/** EDGES **/
juliaChild.addEdge('livesIn', newYork);
simoneBeck.addEdge('livesIn', paris);
louisetteBertholie.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 831)** 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):

```groovy
// 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:

```json
==> {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 805)**.

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:

```groovy
// Order by distance from NYC
```
Using DataStax Enterprise advanced functionality

```java
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')

==>{Location=New York, Author=Patricia Simon}
==>{Location=New York, Author=Julia Child}
==>{Location=New Orleans, Author=Emeril Lagasse}
==>{Location=Los Angeles, Author=Alice Waters}
==>{Location=Paris, Author=Simone Beck}
```

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 720).

**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()```
Using DataStax Enterprise advanced functionality

```java
//SEARCH INDEX – ONLY WORKS FOR POINT AND LINestring

// 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))

// 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:
Using DataStax Enterprise advanced functionality

```
{line=[LINESTRING (0 0, 1 1)], name=[l1]}
```

Find the stored data that matches a polygon mapped to the specified points:

```
g.V().
  has('polyLocation', 'polygon', Geo.polygon(0, 0, 1, 1, 0, 1, 0, 0)).
  valueMap()
results in:
{polygon=[POLYGON ((0 0, 1 1, 0 1, 0 0))], name=[g1]}
```

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):

```
g.V().
  has('location', 'point', Geo.inside(Geo.point(0, 0), 1)).
  values('name')
lists:
==>&gt;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):

```
g.V().has('lineLocation', 'line', Geo.inside(Geo.point(0.0, 0.0), 1.415)).valueMap()
lists:
==>&gt;{line=[LINESTRING (0 0, 1 1)], name=[l1]}
==>&gt;{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 if a JAR file from the JTS library is added to DSE Search (page 596). Be sure to add the JAR file before attempting to insert the graph data.

Find all points within a specified Cartesian spatial polygon:
Using DataStax Enterprise advanced functionality

```
g.V().
has('location', 'point', Geo.inside(Geo.polygon(0, 0, 1, 0, 1, 0, 1, 0, 0))).
values('name')
```

lists:

```=>p0```

find linestrings within a polygon

```
g.V().
has('lineLocation', 'line', Geo.inside(Geo.polygon(0, 0, 1, 0, 1, 0, 1, 0, 0))).
values('name')
```

lists:

```=>l1```

Schema and data

The examples here use the following schema:

```
//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:

```
//VERTICES
// PERSON VERTICES
pam = graph.addVertex(label, 'person', 'name','Pam')
les = graph.addVertex(label, 'person', 'name','Les')
paul = graph.addVertex(label, 'person', 'name','Paul')
```
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):
Using DataStax Enterprise advanced functionality

```java
using DataStax Enterprise advanced functionality
```

```java
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 821).

Now list the stores within a 10 unit radius from Paul's home that have celery:

```java
now list the stores within a 10 unit radius from paul's home that have celery:
```

```java
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 805)** 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
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
finds:

```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 805)**. 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:
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 805).*

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.

```java
import com.esri.core.geometry.ogc.OGCGeometry;

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.

```plaintext
=>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 805).*

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.
Using DataStax Enterprise advanced functionality

g.V().has('name','Julia Child').repeat(out()).times(2).valueMap()

The output for this traversal lists each result:

```sql
==>
{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.

```sql
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:

```sql
==>
v[~label=ingredient, member_id=2, community_id=1442590464]],
v[~label=recipe, member_id=2, community_id=473764096]],
v[~label=book, member_id=0, community_id=568859392]],
(Book=The Art of French Cooking, Vol. 1, Recipe=Beef Bourguignon])
==>
v[~label=ingredient, member_id=1, community_id=684566272]],
v[~label=recipe, member_id=0, community_id=1462084224]],
v[~label=book, member_id=1, community_id=1620680576]],
```
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 Bertholie={}, Salade Nicoise={}

- **The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution**
  - Alice Waters={}, Kelsie Kerr={}, Roast Pork Loin={}, Carrot Soup={}, Fritz Streiff={}, Patricia Curtan={}

- **The French Chef Cookbook**
  - Julia Child={}

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.

```
g.V().hasLabel('recipe').out().tree().by('name').next()
```

The output for this traversal lists each result:

- **Roast Pork Loin**
  - red wine={}, pork loin={}, chicken broth={}, The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution={}

- **Spicy Meatloaf**
  - bacon={}, celery={}, pork sausage={}, onion={}, ground beef={}, green bell pepper={}

- **Beef Bourguignon**
  - mashed garlic={}, butter={}, The Art of French Cooking, Vol. 1={}, onion={}, tomato paste={}, beef={}

- **Carrot Soup**
  - butter={}, onion={}, chicken broth={}, carrots={}, The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution={}, thyme={}

- **Ratatouille**
  - mashed garlic={}, yellow onion={}, olive oil={}, zucchini={}, eggplant={}

- **Salade Nicoise**
  - tuna={}, The Art of French Cooking, Vol. 1={}, hard-boiled egg={}, olive oil={}, tomato={}, green beans={}

- **Wild Mushroom Stroganoff**
  - mushrooms={}, yellow onion={}, egg noodles={}

- **Oysters Rockefeller**
Each recipe lists the ingredients for the recipe and the books that include the recipe.

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 832).

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 832) 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 Gryo 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 888) 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 833) 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
Using DataStax Enterprise advanced functionality

improvement will be seen. Another method of handling mixed new and existing data is the use of `isNew()` (page 907) and `exists()` (page 906). 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 833)* 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 710) before loading data using DSE Graph Loader. Without schema, the correct data types for the data are not enforced. *Creating indexes (page 737)* 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. Use these instructions for installing on Linux-based platforms using the binary tarball.

A sufficiently powerful machine should be used to run DSE Graph Loader. The memory requirements must account for caching the serialized vertices during the loading process, up to ten times (10X) the size of the original data. Ensure that enough shared memory (mmap, or buffer cache) is allocated; properties and edges are bound by the speed of the available I/O. The network connection between the DSE Graph Loader machine and the DSE Graph cluster must have sufficient bandwidth.

**Note:** Do not run DSE Graph Loader on a machine that hosts a DSE Graph node for larger scale datasets.

**Prerequisites:**

- DataStax Academy registration email address and password.
- DataStax Enterprise is installed and configured for DSE Graph.

1. Download the DSE Graph Loader tarball using your DataStax Academy account credentials.

2. Unpack the DataStax Enterprise tarball:

   ```bash
   $ tar -xzvf dse-graph-loader-bin.tar.gz
   
   The files are extracted into the dse-graph-loader directory.
   ```
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 896) including security options (page 902).

- 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.

```java
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_scheme` and `preparation` are both true, then the data is analyzed, compared to the schema, and new schema is created if found missing.

```java
/* CONFIGURATION
/* Configures the data loader to analyze the schema */
cfg preparation: true
*/
```

See the table below (page 834) 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.

```java
/* CONFIGURATION
/* Configures the data loader to create the schema */
cfg create_schema: true
*/
```
Notice: It is strongly recommended that schema is created (page 710) 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.

```plaintext
cfg 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.

```plaintext
cfg load_vertex_threads: 3 load_edge_threads: 0
```

• Multiple configuration settings can be listed together.

```plaintext
cfg load_new: true, dryrun: true, schema_output: '/tmp/loader_output.txt'
```

What's next: Load data. (page 834)

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 (page 835)</td>
</tr>
<tr>
<td>Text</td>
<td>Delimited text data of any format.</td>
<td>Loading TEXT data (page 850)</td>
</tr>
<tr>
<td>Type</td>
<td>Description</td>
<td>Instructions</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</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) (page 854)</td>
</tr>
<tr>
<td>JSON</td>
<td>Data stored in JSON (JavaScript Object Notation) format.</td>
<td>Loading JSON data (page 843)</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. (page 857)</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) (page 859)</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 (page 862)</td>
</tr>
<tr>
<td>Gryo</td>
<td>Data stored in a binary Gryo format.</td>
<td>Loading Gryo data (page 864)</td>
</tr>
<tr>
<td>GraphSON</td>
<td>Data stored in GraphSON format.</td>
<td>Loading GraphSON data (page 866)</td>
</tr>
<tr>
<td>GraphML</td>
<td>Data stored in GraphML format.</td>
<td>Loading GraphML data (page 867)</td>
</tr>
</tbody>
</table>

**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 custom vertex ids (page 727), 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 936) 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 838) and JSON (page 845), 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()`
Using DataStax Enterprise advanced functionality

in the data input line. If more flexibility is desired, such as manipulation of the vertex labels using labelField (page 872), use Loading TEXT data (page 850).

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:

```plaintext
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.

1. If desired, add configuration (page 833) 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.

```plaintext
// 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('|')
```

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:

```plaintext
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 888).

```java
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 869) 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:

```java
/* SAMPLE INPUT
author: Julia Child|F
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

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 833) 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.
Using DataStax Enterprise advanced functionality

// 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:

```
name|gender
```

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"
    }
```
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:

```bash
$ 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 833)* to the mapping script.
- Sample input file:

```groovy
/* 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.

```groovy
// 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+'/*.csv'
```
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 833) 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 833) 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'
Using DataStax Enterprise advanced functionality

```groovy
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:

```bash
$ 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:

```json
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"}
```

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,
Using DataStax Enterprise advanced functionality

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 833) 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.

   ```plaintext
   // 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 888).

   ```plaintext
   authorInput = File.json(inputfiledir + 'author.json')
   ```

4. Create the main body of the mapping script. (page 869) 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:

   ```plaintext
   $ 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:

```plaintext
/* 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 "-filenam" option

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:
Using DataStax Enterprise advanced functionality

// 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.lang.IllegalArgumentException: /tmp/dirSource/data has more than 1 input type.

1. If desired, add configuration (page 833) 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"
   }
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 833) 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
```
Using DataStax Enterprise advanced functionality

// 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 833) 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
Using DataStax Enterprise advanced functionality

// 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 833) 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 833) 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
• 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')
bookInput = File.text(inputfiledir + "book.dat").
   delimiter("|").
   header('name', 'year', 'ISBN')
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 888).

```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 869) 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.
/** SAMPLE INPUT
author: Julia Child|F
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.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.

```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'
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:

```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'
```
Using DataStax Enterprise advanced functionality

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"
    }
}

• 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 833) to the mapping script.
• A sample of the data for load looks like the following:

SAMPLE INPUT
// This file uses tabs between fields
// For the authorREGEX.data file:
name:Julia Child gender:F
// For the bookREGEX.dat file:
name:Simca's Cuisine: 100 Classic French Recipes for Every Occasion
Using DataStax Enterprise advanced functionality

// For the authorBookREGEX.dat file:
bname:Simca's Cuisine: 100 Classic French Recipes for Every Occasion
aname:Simone Beck

• 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}[-]{1}[0-9]{3}[-]{1}[0-9]{1}[0-9]{3}[-]{1}[0-9]{5}[-]{1}[0-9]{1}[0-9]{1}[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 888).

    authorInput = File.text(inputfiledir +
        "authorREGEX.dat").regex("name:(.*)\tgender:([MF])").header('name', 'gender')

• Create the main body of the mapping script. (page 869) 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

author:
name: Julia Child gender: F

book:
name: Simca's Cuisine: 100 Classic French Recipes for Every Occasion

authorBook:
bname: Simca's Cuisine: 100 Classic French Recipes for Every Occasion aname: 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/REGEX/'

authorInput = File.text(inputfiledir + 'authorREGEX.dat').
  regex("name:(.*)\tgender:(\[MF\])").
  header('name', 'gender')

bookInput = File.text(inputfiledir + 'bookREGEX.dat').
  regex("name:(.*)\t\tyear:(\[0-9\]{4})\t\tISBN:(\[0-9\]{1}\[-\]{1}\[0-9\]{3}\[-\]{1}\[0-9\]{1}\[-\]{1}\[0-9\]{5}\[-\]{1}\[0-9\]{0,1})").
  header('name', 'year', 'ISBN')

authorBookInput = File.text(inputfiledir + 'authorBookREGEX.dat').
  regex("bname:(.*)\t\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", {
    label "book"
    key "name"
  }
}
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 833) to the mapping script.
- A sample of the data for load looks like the following:

```plaintext
SAMPLE INPUT
// For the author data:
name:Julia Child gender:F
// For the book data:
name:Simca's Cuisine: 100 Classic French Recipes for Every Occasion
// For the authorBook data:
bname: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 `MySQLDatabase`, 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.

```plaintext
// 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
Using DataStax Enterprise advanced functionality

```java
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 888).

```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 869) 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 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:
 name:Simca's Cuisine: 100 Classic French Recipes for Every Occasion
authorBook:
 bname:Simca's Cuisine: 100 Classic French Recipes for Every Occasion aname:Simone Beck
 */

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true, load_vertex_threads: 3

// DATA INPUT
```
Using DataStax Enterprise advanced functionality

// 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 833) to the mapping script.
- A sample of the CSV data residing on HDFS:

  // 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
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 869) 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 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|F
// For the book.csv file:
// name|year|ISBN
// Simca's Cuisine: 100 Classic French Recipes for Every Occasion|1972|0-394-40152-2
```
Using DataStax Enterprise advanced functionality

// 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"
    }
}
Using DataStax Enterprise advanced functionality

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 833) to the mapping script.
- A sample of the CSV data residing on AWS S3:

```plaintext
// 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:

```plaintext
// 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 869) 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.

```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

// 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"
    key "name"
}
```
Using DataStax Enterprise advanced functionality

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 885) for loading data from each source.

- If desired, add configuration (page 833) 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/Gryo/
recipeInput = Graph.file(inputfiledir + 'recipe.gryo').gryo()
```

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.

```java
// 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().dse()
```
• 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 869](#)) This part of the mapping script is the same regardless of the file format, although Gryo files use a slightly modified version ([page 885](#)).

• To run DSE Graph Loader for Gryo loading as a dry run, use the following command:

```bash
$ 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:

```groovy
/* 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"
    outV "outV", {
        labelField "~label"
    }
```
Using DataStax Enterprise advanced functionality

Loading GraphSON data

The data mapping script for GraphSON data (page 788) 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 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 833) 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()
```

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 869) This part of the mapping script is the same regardless of the file format, although GraphSON files use a slightly modified version (page 887).
• 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 *(page 790)* 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 833) 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 a `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 869) This part of the mapping script is the same regardless of the file format, although GraphML files use a slightly modified version (page 886).
- 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 */

// 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/GraphML/
```
Using DataStax Enterprise advanced functionality

```java
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.

```java
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 349) 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 871). 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 872).

• 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 = {
  label "authored"
  outV "aname", {
    label "author"
    key "name"
  }
  inV "bname", {
    label "book"
    key "name"
  }
```

DSE 5.1 Developer Guide (Previous version)
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)
load(authorInput).asVertices {
    label "book"
    key "bname"
    ignore "name"
    ignore "gender"
}
load(authorInput).asVertices {
```
Using DataStax Enterprise advanced functionality

```java
label "author"
key "name"
outV "book", "authored", {
    label "book"
    key "bname"
}
```

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:

```java
/* 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.

```java
g.V().hasLabel('author').valueMap()
{gender=[F], name=[Julia Child]}
g.V().hasLabel('reviewer').valueMap()
{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:
/* 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"
  }
}
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 727](#)) 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:

```java
/* 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](#) that includes all properties in the composite key. The search index is required to use DSE Graph Loader for inserting composite custom id data.

Check the vertex id results with `id()` to retrieve the full primary key definition:

```java
gremlin> g.V().hasLabel('fridgeSensor').id()
==>{~label=fridgeSensor,
    sensorId=93c4ec9b-68ff-455e-8668-1056ebc3689f, cityId=santaCruz}
==>{~label=fridgeSensor, sensorId=9c23b683-1de2-4c97-a26a-277b373732a, cityId=sacramento}
==>{~label=fridgeSensor, sensorId=eff4a8af-2b0d-4ba9-a063-c170130e2d84, cityId=sacramento}
```

Each vertex stores `fridgeItem` as data:
Using DataStax Enterprise advanced functionality

```java
 gremlin> g.V().valueMap()
==>{fridgeItem=[asparagus]}
==>{fridgeItem=[ham]}
==>{fridgeItem=[eggs]}
```

2. To load edges based on a composite key, a transformation is required:

```java
/* 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"
  }
}
```

The edge file transforms the partition key and clustering key into a map of `cityId` and `sensorId`. This map can then be used to designate the key for a `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:

```java
/* 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:

```
    "inV": 
        "authorCity": [    
            "author|city|dateStart|dateEnd" 
        ], 
    "outV": 
        "authorCity": [    
            "author|city|dateStart|dateEnd" 
        ]
```

### 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-cardinal edges (page 724)** can be accomplished from one file with judicious use of `ignore` while loading vertices:

```java
/* 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) 
// 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`

```
// PROPERTY KEYS
schema.propertyKey('name').Text().single().create()
schema.propertyKey('gender').Text().single().create()
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

```json
reviewer: { "name":"Jon Doe", "gender":"M", "badge" : { "value": "Gold Badge","since" : 2012 } }
*/
```

// CONFIGURATION
// Configures the data loader to create the schema

// 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`.

```plaintext
g.V().valueMap()
{badge=[Gold Badge], gender=[M], name=[Jane Doe]}
g.V().properties('badge').valueMap()
{since=2012}
```

Mapping multiple meta-properties

If the input file includes multiple meta-properties, or properties that have multiple properties, use `vertexProperty`. 
Using DataStax Enterprise advanced functionality

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
reviewer: { "name": "Jane Doe", "gender": "F", "badge": [{ "value": "Gold Badge", "since": 2012 }, { "value": "Silver Badge", "since": 2005 }] }
*/
```

**// 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.json(inputfiledir + "reviewerMultiMeta.json")

// Specifies what data source to load using which mapper (as defined inline)

load(reviewerInput).asVertices{
    label "reviewer"
    key "name"
    vertexProperty "badge", { value "value" }
}
```

Optionally, the data can be loaded from a CSV file if a transform is used before loading:

```java
/* SAMPLE INPUT
```
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>name</th>
<th>gender</th>
<th>value</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane Doe</td>
<td>F</td>
<td>Gold Badge</td>
<td>2011</td>
</tr>
<tr>
<td>Jane Doe</td>
<td>F</td>
<td>Silver Badge</td>
<td>2005</td>
</tr>
<tr>
<td>Jon Doe</td>
<td>M</td>
<td>Gold Badge</td>
<td>2012</td>
</tr>
</tbody>
</table>

`/*

// 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 718) and Cartesian (page 720) 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 738)* 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 833)* 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"
}
import com.datastax.driver.dse.geometry.Point
ptsInput = ptsInput.transform {
    it['point'] = Point.fromWellKnownText(it['point']);
    return it;
}
load(linesInput).asVertices {
    label "lineLocation"
    key "name"
}
import com.datastax.driver.dse.geometry.LineString
linesInput = linesInput.transform {
    it['line'] = LineString.fromWellKnownText(it['line']);
    return it;
}
load(polysInput).asVertices {
    label "polyLocation"
    key "name"
}
import com.datastax.driver.dse.geometry.Polygon
polysInput = polysInput.transform {
    it['polygon'] = Polygon.fromWellKnownText(it['polygon']);
    return it;
}

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

$ graphloader geoMap.groovy -graph testGeo -address localhost

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
//Need to specify a keymap to show how to identify vertices
vertexKeyMap = VertexKeyMap.with('meal','name').with('ingredient','name').with('author','name').with('recipe','name').build();

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 custom 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"
    key "~id", "id"
}
load(recipeInput.edges()).asEdges {
    labelField "~label"
    outV "outV", {
        labelField "~label"
        key "~id", "id"
    }
    inV "inV", {
        labelField "~label"
        key "~id", "id"
    }
}
```

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"
    }
}
```
Using DataStax Enterprise advanced functionality

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 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
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 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_modeset 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:
Using DataStax Enterprise advanced functionality

```java
g.V().hasLabel('author').valueMap()
{gender=[F], name=[Julia Child], id=[0]}
{gender=[F], name=[Simone Beck], id=[3]}
```

**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 888)*
- **flatMap** *(page 892)*
- **map** *(page 894)*

**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
**/
```
Using DataStax Enterprise advanced functionality

```java
inputfiledir = '/tmp/filter_map_flatmap/
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
g.V().hasLabel('chefYoung').valueMap()
```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
**/
```

```java
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"
}
```
Using DataStax Enterprise advanced functionality

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:

```javascript
// List all the living chefs
g.V().hasLabel('chefAlive').valueMap()
==>{gender=F, name=Alice Waters, age=73, status=alive}
==>{gender=F, name=Patricia Curtan, age=66, status=alive}
==>{gender=M, 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=F, name=Amanda Cohen, age=35, status=alive}
==>{gender=M, name=Patrick Connolly, age=31, status=alive}

// List all the deceased chefs
g.V().hasLabel('chefDeceased').valueMap()
==>{gender=F, name=Julia Child, age=500, status=deceased}
==>{gender=F, name=Simone Beck, age=500, status=deceased}
==>{gender=F, name=Louiseette 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>Louiseette 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>500</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:

```javascript
/** SAMPLE INPUT
name|gender|status|age
Jamie Oliver|M|alive|41
```
Using DataStax Enterprise advanced functionality

```java
/**
   // SCHEMA
   // Configures the data loader to create the schema
   config create_schema: false, load_new: true
   
   // DATA INPUT
   // Configures the data loader to create the schema
   // outputfiledir is the directory for the input files that is given in
   // 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"
   }
```

DSE 5.1 Developer Guide (Previous version)
Using DataStax Enterprise advanced functionality

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]}
```
Full flatMap data set

The full sample data set used in this example:

<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()
schema.vertexLabel('recipe').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/'
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"
map

The `map()` (also called `transform()`) applies a function to a field's values before loading the data.

**map converts gender field from to lowercase 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`:

```groovy
g.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=[Patricia Curran], age=[66]}
==>{gender=[f], name=[Kelsie Kerr], age=[57]}
==>{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>
</tbody>
</table>
Patricia Curtan | F | 66
Kelsie Kerr | F | 57
Fritz Streiff | M | 500
Emeril Lagasse | M | 57
James Beard | M | 500
Jamie Oliver | M | 41
Amanda Cohen | F | 35
Patrick Connolly | M | 31

** 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()

// 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 44: 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>

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>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>-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>-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>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>-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 710) 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>-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>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>-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>-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>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>-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>-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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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>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>-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>-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 vertexes 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>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Option</th>
<th>Data type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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:

<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>

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
Using DataStax Enterprise advanced functionality

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.

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 mymapsctipt.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
# 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 mymapsctipt.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_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`:
Using DataStax Enterprise advanced functionality

$ 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:

$ 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`
Using DataStax Enterprise advanced functionality

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**

```plaintext
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:

```plaintext
load(authorBookInput).asEdges {
    label "authored"
    outV "aname", {
        label "author"
        key "name"
        exists()
    }
    inV "bname", {
        label "book"
        key "name"
    }
}
```

**ignore**

**Synopsis**

```plaintext
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 910)* and *key* *(page 909)* 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()
  }
}
```

**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 911) 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`.

```java
//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"}
```
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`

```java
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 910) and `key` (page 909) must be set along with `inV`.

DSE 5.1.3 and later:

```java
//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"}
```

```java
// 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`

```java
inV {
    label "fridgeSensor"
    key cityId: "cityId", sensorId: "sensorId"
    exists()
    ignore "homeId"
    ignore "name"
}
```

**key**

**Synopsis**

```java
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`:

```java
key "name"
```
Using DataStax Enterprise advanced functionality

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:

```
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:

```
key city_id: "city_id", sensor_id: "sensor_id"
```

This definition uses the following pattern:

```
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**

```
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 907), outE (page 911)`).

**Examples**

Set the label in a mapping script to `recipe`.

```
label "recipe"
```

**labelField**

**Synopsis**

```
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 907), outE (page 911)`).
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 910) and `key` (page 909) must be set along with `outE`.

**outV**

Synopsis

```
DSE5.1.2 and earlier:  
outV "field_name", {  
  label "field_name"  
  [ key "key_name" | key key1_name: "key1_name", key2_name: "key2_name" ]  
  [ exists() ]
```

Using DataStax Enterprise advanced functionality

DSE5.1.3 and later:
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 908) 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.

//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 910) and key (page 909) must be set along with outV.

DSE 5.1.3 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
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 906](#)) 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 ([page 723](#)) 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**

```java
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:

```java
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**

```java
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`.

```java
vertexProperty "badge", {
    "value" "since"
}
```

For this mapping, the data could be:
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 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 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. 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()
```
Using DataStax Enterprise advanced functionality

```
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`:

```
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 430](#)) can be set from Gremlin using the `graph.configuration.setProperty` method on the graph.

```
:remote config alias g database.a

g.graph.configuration.setProperty("property name", value)
```

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 403](#)). 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:

```java
: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")
```

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 681) 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.
Using DataStax Enterprise advanced functionality

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 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.

Choosing when to use DseGraphFrame or DSE Graph OLAP queries

**DSE Graph OLAP (page 915)** 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.

```java
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.

```java
// 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.

//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.

```
g.cache()
```

The persist() method requires one of the Spark persist levels as a parameter.

```
g.persist(MEMORY_AND_DISK_SER)
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gf()</td>
<td>Returns a GraphFrame object.</td>
</tr>
<tr>
<td>V()</td>
<td>Returns a DseGraphTraversal[Vertex] object to start a vertex traversal.</td>
</tr>
<tr>
<td>E()</td>
<td>Returns a DseGraphTraversal[Edge] object to start an edge traversal.</td>
</tr>
<tr>
<td>cache()</td>
<td>Cache the graph data with Spark.</td>
</tr>
<tr>
<td>persist(level)</td>
<td>Cache the graph data with one of the Spark persist levels.</td>
</tr>
<tr>
<td>deleteVertices()</td>
<td>Delete vertices.</td>
</tr>
<tr>
<td>deleteVertices(label: String)</td>
<td>Delete all vertices with the specified label.</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 labels 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 labels 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>

**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 46: 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>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Steps</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

Table 47: 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.

```scala
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 the Traversal 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._
```
// convert the iterator to a Scala iterator to get the native Id object
for(i <- g.V().id().asScala) println (i)

{~label=vertex, community_id=748226688, member_id=0}
{~label=custom, name=Name, value=1}

// convert to a Set
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.

g.E().groupCount().by(T.label)
g.V().has("age", P.gt(30)).show

For standalone applications, import theses classes.

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.

g.V().values[Any]("name").next()

Or similarly:

val n: String = g.V().values("name").next()

Explicitly set the type when dropping properties.

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.

g.V().properties("age", "name").drop().show()
Table 48: 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 age properties 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 49: Reserved column names in GraphFrame

<table>
<thead>
<tr>
<th>Reserved column name</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Reserved column name</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
</tr>
<tr>
<td>dst</td>
</tr>
<tr>
<td>new_id</td>
</tr>
<tr>
<td>new_src</td>
</tr>
<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 50: 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><strong>DSE Graph type</strong></td>
<td><strong>Spark SQL type</strong></td>
<td><strong>Conversion rules</strong></td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------</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>
<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.

   ```scala
   scala> g.V.write.json("/tmp/v_json")
   scala> 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:
Using DataStax Enterprise advanced functionality

$ dse hadoop fs -cat /tmp/v_json/* > local_vertices.json
$ dse hadoop fs -cat /tmp/e_json/* > 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 DSE 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"))
   ```

   ```
   val g = spark.dseGraph("graph name")
   g.updateVertices(spark.read.json("path to exported vertices JSON"))
   g.updateEdges(spark.read.json("path to exported edges 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.

   ```
   scala> import org.graphframes._
   scala> val g: GraphFrame = examples.Graphs.friends
   scala> g.vertices.printSchema
   root
   |-- id: string (nullable = true)
   |-- name: string (nullable = true)
   |-- age: integer (nullable = false)
   ```

   ```
   scala> g.edges.printSchema
   root
   |-- src: string (nullable = true)
   |-- dst: string (nullable = true)
   |-- relationship: string (nullable = true)
   ```

   **b.** In the Gremlin console or DSE Studio create the schema.
Using DataStax Enterprise advanced functionality

```java
system.graph('friends').create()
:remote config alias g friends.g
    schema.propertyKey("age").Int().create()
    schema.propertyKey("name").Text().create()
    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
scala> val d = spark.dseGraph("friends")
scala> d.V.printSchema
root
|-- id: string (nullable = false)
|-- ~label: string (nullable = false)
|-- _id: string (nullable = true)
|-- name: string (nullable = true)
|-- age: integer (nullable = true)

scala> d.E.printSchema
root
|-- src: string (nullable = false)
|-- dst: string (nullable = false)
|-- ~label: string (nullable = true)
|-- id: string (nullable = true)
```

d. Convert the edges and vertices to the target format.

```scala
scala> val v = g.vertices.select ($"id" as "_id",
    lit("people") as "~label", $"name", $"age")
scala> 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.
Using DataStax Enterprise advanced functionality

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 1090) in your datacenter.
- Install the DSE Graph Loader (page 832).
- Clone the graph-examples Git repository to the machine on which you are running the Gremlin console.

```bash
$ git clone https://github.com/datastax/graph-examples.git
```

1. Load the Northwind graph and supplemental data using the `graphloader` tool:

```bash
$ 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:

```bash
$ dse gremlin-console
```

3. Alias the traversal to Northwind graph using the default OLTP traversal source:

```bash
gremlin> :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.

```bash
gremlin>
    schema.config().option('graph.schema_mode').set('Development')
```

5. Enable the use of scans and lambdas.

```bash
gremlin> 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:
7. Alias the traversal to the Northwind analytics OLAP traversal source \texttt{a}. Alias \texttt{g} to the OLAP traversal source for one-off analytic queries:

\begin{verbatim}
gremlin> :remote config alias g northwind.a
\end{verbatim}

When you alias \texttt{g} to the OLAP traversal source database name.a, DSE Analytics is the workload back-end.

8. Count the number of vertices using the OLAP traversal source:

\begin{verbatim}
gremlin> g.V().count()
\end{verbatim}

\begin{verbatim}
=> 3294
\end{verbatim}

9. Store subgraphs into snapshots using \texttt{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.

\begin{verbatim}
gremlin> employees = 
    graph.snapshot().vertices('employee').create()

=> graphtraversalsource[hadoopgraph[persistedinputrdd-persistedoutputrdd], sparkgraphcomputer]

gremlin> categories = 
    graph.snapshot().vertices('category').create()

=> graphtraversalsource[hadoopgraph[persistedinputrdd-persistedoutputrdd], sparkgraphcomputer]
\end{verbatim}

The \texttt{snapshot()} method returns an OLAP traversal source using the SparkGraphComputer.

10. Run an operation on the snapshot graphs.

Count the number of employee vertices in the snapshot graph:

\begin{verbatim}
gremlin> employees.V().count()
\end{verbatim}

\begin{verbatim}
=> 9
\end{verbatim}

Count the number of category vertices in the snapshot graph:

\begin{verbatim}
gremlin> categories.V().count()
\end{verbatim}
DSE Graph Tools

In addition to the Gremlin console, other tools are available for working with DSE Graph:

**DataStax Studio (page 1098)**

Web-based notebook-style visualization tool. Currently supports Markdown and Gremlin. Includes a variety of list and graph functions.

**DSE OpsCenter**

Visual management and monitoring tool.

**DSE Lifecycle Manager**

Powerful provisioning and configuration management tool.
**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 941) 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`.

```java
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:

```java
juliaChild.addEdge('created', 'beefBourguignon', T.id, java.util.UUID.fromString('2c85fabd-7c49-4b28-91a7-ca72ae53fd39'), 'createDate', '2017-08-22')
```
Using DataStax Enterprise advanced functionality

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.

**addVertex**

**Synopsis**

```plaintext
addVertex(label, 'label_name', 'key', 'value', 'key', 'value')
```

**Description**

Vertex data is inserted using `addVertex`. A previously created vertex label (page 947) must be specified.

**Examples**

Create a vertex with a vertex label `reviewer` with the properties location and status.

```plaintext
graph.addVertex(label, 'reviewer', 'location', 'Santa Cruz, CA', 'status', 'Rock Star')
```

**io**

**Synopsis**

```plaintext
io([gryo() | graphson() | graphml()]).[readGraph | writeGraph](file_name)
```

**Description**

Graph data is written to a file or read from a file using `io`.

**Examples**

Write the graph data to a file using the Gryo format:

```plaintext
graph.io(gryo()).writeGraph('/tmp/test.gryo')
```

Read the graph data from a file using the Gryo format:

```plaintext
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 831) is a better choice in production. Additionally, a schema setting may need modification for this method to work:
Using DataStax Enterprise advanced functionality

```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`.

```java
jamieOliver = g.V().has('person', 'name', 'Jamie Oliver').next()
jamieOliver.property('gender', 'M').property('nickname', 'jimmy')
```

Update the property `gender` for the vertex juliaChild specifying a property with a property id of 2c85fabd-7c49-4b28-91a7-ca72ae53fd39:

```java
uuid = java.util.UUID.fromString('2c85fabd-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**

```java
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.

```java
graph.tx().config().option("allow_scan", true).open()
```
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.

**clear**

Synopsis

Schema: clear()

Description

Clear schema information for a particular graph using this command. An alias must be created to bind the graph to a graph traversal before running this command.

Examples

Clear the schema and data for a particular graph.

```
gremlin> schema.clear()
```

The result if the clear command is successful:

```
=>null
```

**connection**

Synopsis

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***.

```
schema.edgeLabel('isA').connection('ingredient', 'FridgeItem').create()
```

An adjacency between the vertexLabel ***author*** and ***author*** specifying the edgeLabel ***knows***.
### 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 51: 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>allow_scan</td>
<td>true</td>
<td>Setting to allow costly graph scan queries.</td>
<td>true</td>
</tr>
<tr>
<td>schema_mode</td>
<td>Production</td>
<td>Set mode to Production or Development.</td>
<td>Development</td>
</tr>
<tr>
<td>default_property_key_cardinality</td>
<td>single</td>
<td>Set the cardinality that will be used by default unless otherwise specified.</td>
<td>single</td>
</tr>
<tr>
<td>tx_autostart</td>
<td>true</td>
<td>Set whether transactions are started automatically or must be manually opened.</td>
<td>false</td>
</tr>
</tbody>
</table>

#### Table 52: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`.]

<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>Option argument</td>
<td>Setting Example</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>-------------</td>
<td>---------</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 of 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 53: 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>cache</td>
<td>true</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>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>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 vertex ids (autogenerated vertex ids) actually exist.</td>
<td>true</td>
</tr>
<tr>
<td>external_vertex_verify</td>
<td>false</td>
<td>Set whether a transaction should verify that vertices for externally provided vertex ids (custom 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>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>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>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>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>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Option</th>
<th>Setting Example</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<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 disallow full graph scans in the currently aliased graph.

```java
schema.config().option('graph.allow_scan').set('false')
```

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:

```
REAL_TIME
```

Set the default write consistency for a transaction 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 a transaction 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:

```
true
```
Using DataStax Enterprise advanced functionality

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 verify that external vertex ids exist, 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.

To retrieve a list of configuration options that have been set, use the `describe()` command:

```java
schema.config().describe()
```
resulting in a list of all options that have been manually set:

```java
graph.tx_groups.default.write_consistency: ALL
graph.allow_scan: False
```

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.

To set `restrict_lambda` to `FALSE` in order to test lambda functions (only appropriate for non-production systems):

```java
schema.config().option('graph.traversal_sources.g.restrict_lambda').set('FALSE')
```

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>dse.yaml schema_mode:Production</td>
<td>Development</td>
<td>yes</td>
</tr>
<tr>
<td>graph.schema_mode:Production</td>
<td>Production</td>
<td>no</td>
</tr>
<tr>
<td>graph.schema_mode:Development</td>
<td>Development</td>
<td>no</td>
</tr>
<tr>
<td>graph.allow_scan:true</td>
<td>Production</td>
<td>yes</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

```java
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.

```java
gremlin> schema.describe()
The resulting list:
```

edgeLabel

Synopsis

```java
schema.edgeLabel('edgeLabel').[ single() | multiple() ].
[ connection( outVertex, inVertex ) ].[ ttl ].[ properties(property[, property]) ].[ ifNotExists() ].[ create() | add() | 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 (page 723)`), the connections that are defined between two types of vertices, properties that an edge has defined, and a time-to-live (TTL) to determine the lifecycle of an edge.

Examples

Create an edgeLabel `created`:

```java
schema.edgeLabel('created').create()
```

Create an edgeLabel `includedIn` if the edge label doesn't already exist:

```java
schema.edgeLabel('includedIn').ifNotExists().create()
```

Create an edgeLabel with `multiple cardinality (page 724)`:
Using DataStax Enterprise advanced functionality

```java
schema.edgeLabel('rated').multiple().create()
```

Add properties to an edgeLabel:

```java
schema.edgeLabel('rated').properties('rating', 'last_date').add()
```

Create an edgeLabel with both a property and a connection:

```java
schema.edgeLabel('rated').properties('rating').connection('recipe', 'reviewer').add()
```

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('rated').exists()
```

Get the schema creation command for an edgeLabel using the `describe()` command:

```java
schema.edgeLabel('createDate').describe()
```

### exists

**Synopsis**

```java
schema.<schema_element>('author').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.

```java
gremlin> schema.vertexLabel('author').exists()
```

The resulting list:
index - edge index

Synopsis

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 (\texttt{outE()}) from a vertex label, incoming edges (\texttt{inE()}) from a vertex label, or both outgoing and incoming (\texttt{bothE()}). The last type, \texttt{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 \texttt{index} \texttt{ratedByStars} with an \texttt{outE} edge label using the \texttt{property key} \texttt{stars}. The \texttt{vertex label} is specified as \texttt{reviewer}.

\texttt{schema.vertexLabel('reviewer').index('ratedByStars').outE('rated').by('stars').add()}

Create an \texttt{index} \texttt{ratedByStars2Way} with a \texttt{bothE} edge label using the \texttt{property key} \texttt{year}. The edge index allows queries that find both recipes with a certain year and reviewers who gave a review in a certain year.

\texttt{schema.vertexLabel('recipe').index('byAuthOrRecipe').bothE('created').by('year').ifNotExists().add()}

It can replace two indexes:

\texttt{schema.vertexLabel('recipe').index('toRecipesRated').inE('rated').by('year').add()}
\texttt{schema.vertexLabel('reviewer').index('toReviewersWhoRated').outE('rated').by('year').add()}

index - property index

Synopsis

index('index_name').property('propertykey_name').by('meta-propertykey_name').add()
Using DataStax Enterprise advanced functionality

Description

A property index specifies an index that is built using the meta-property (page 712) 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 of livedIn. The vertex label is specified as author.

```
schema().vertexLabel('author').index('byLocation').property('country').by('livedIn').add()
```

index - vertex index

Synopsis

```
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.

```
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.

```
schema.vertexLabel('meal').index('byMeal').materialized().by('name').add()
```

Create an index as a search index using the property key instructions and specify that the index is a asText(). The vertex label is specified as recipe.

```
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.
Using DataStax Enterprise advanced functionality

```
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('category').add()
```

**partitionKey - clusteringKey**

**Synopsis**

```
partitionKey('id_name').[ clusteringKey('id_name') ]
```

**Description**

`partitionKey` and `clusteringKey` are used to specify a customer vertex id (page 727) 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`.

```
schema.propertyKey('city_id').Int().create()
```

Create a `vertexLabel` using `sensor_id` as a partitioning key.

```
schema().vertexLabel('FridgeSensor').partitionKey('sensor_id').create()
```

Create a vertex label with a custom partitioning key `city_id` and clustering key `sensor_id`.

```
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.

```
graph.addVertex(label, 'FridgeSensor', 'city_id', 100, 'sensor_id', '60bcae02-f6e5-11e5-9ce9-5e5517507c66')
```

Create a `vertexLabel` using `city_id` and `sensor_id` as a composite partitioning key.

```
schema().vertexLabel('FridgeSensor').partitionKey('city_id', 'sensor_id').create()
```

**properties**

**Synopsis**

```
properties('name').add()
```
Using DataStax Enterprise advanced functionality

Description

Properties can be added to vertices and edges. A property key (page 946) 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.

```javascript
schema.vertexLabel('author').properties('nationality').add()
```

Add more than one property to a vertex label.

```javascript
schema.vertexLabel('author').properties('nationality', 'age', 'assocRestaurants').add()
```

propertyKey

Synopsis

```javascript
propertyKey('name').type().[ single() | multiple() ].[ ttl ].
[ properties(metadata_property) ].[ ifNotExists() ].[ create() | add() |
| 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. Allowed characters for the name are alphabetical or underscore. The data type (page 954) must be included. 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.

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.

```javascript
schema.propertyKey('name').Text().create()
```

Create a property key with the name `num_items` of Integer type if the property key doesn't already exist.

```javascript
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 exist.

```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()
```

### vertexLabel

**Synopsis**

```java
schema.vertexLabel('vertexLabel').[ partitionKey(propertyKey, [ partitionKey(propertyKey) ]).[ clusteringKey(propertyKey) ].[ ttl ].[ properties(property, property) ].[ index ].[ partition() ].[ cache() ].[ ifNotExists() ].[ create() | add() | 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 lifecycle of an vertex.

DSE Graph limits the number of vertex labels to 200 per graph.

**Examples**

Create a `vertexLabel` `author`. 
Using DataStax Enterprise advanced functionality

```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 945)`.

Add properties to a vertexLabel.

```java
schema.vertexLabel('author').properties('location','restaurant').add()
```

For indexes, see each index entry (`edge index (page 943)`, `property index (page 943)`, `vertex index (page 944)` 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()
```

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.
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.

```bash
gremlin> system.graph('FridgeItem').create()
```

The resulting list:

```text
===>FridgeItem
```

Create a new graph if it doesn't currently exist by modifying with `ifNotExists()`.

```bash
gremlin> system.graph('FridgeItem').ifNotExists().create()
```

The resulting list:

```text
===>FridgeItem
```

Creating a graph should include setting the replication factor for the `graph` (page 952) and the `graph_system` (page 953). It can also include other options (page 951).

drop

Synopsis

```java
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.

```bash
gremlin> system.graph('FridgeItem').drop()
```

The resulting list:

```bash
==>null
```

Drop an existing graph if it exists.

```bash
gremlin> system.graph('FridgeSensors').ifExists().drop()
```

The resulting list:

```bash
==>null
```

`exists`

Synopsis

```bash
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.

```bash
gremlin> system.graph('FridgeItem').exists()
```

The resulting list:

```bash
==>true
```

`graphs`

Synopsis

```bash
system.graphs()
```

Description

Discover what graphs currently exist using this command.

Examples

Discover all graphs that exist in a DSE cluster.

```bash
gremlin> system.graphs()
```
The resulting list:

```plaintext
==>quickstart
==>test
```

**Note:** This command is not available if a graph traversal is aliased with the `:remote config alias g some_graph.g` command. In order to access the system command, reset the alias with `:remote config alias reset`.

### option

**Synopsis**

```plaintext
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 54: 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>system_replication_config</td>
<td><code>{ 'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }</code></td>
<td>Set replication configuration for a single <code>graph_system (page 956)</code> data.</td>
<td><code>{ 'class' : 'SimpleStrategy', 'replication_factor' : 1 }</code></td>
</tr>
<tr>
<td>default_property_key_cardinality</td>
<td><code>Multiplicity</code></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.
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 935) can be set at graph creation, but must be changed using `schema.config()` if modified later.

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:

```
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**

```
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:

<table>
<thead>
<tr>
<th>number of nodes per datacenter</th>
<th><em>graph_name</em> replication factor</th>
<th><em>graph_name_system</em> 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 953)* 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}"
```

---

DSE 5.1 Developer Guide (Previous version)
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:

<table>
<thead>
<tr>
<th>number of nodes per datacenter</th>
<th><code>graph_name</code> replication factor</th>
<th><code>graph_name_system</code> 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 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 952) must also be set.

**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 (page 548).
<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, &quot;answer&quot;, &quot;blob&quot;, &quot;42&quot;);</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>Note: When dealing with currency, DataStax recommends a currency class that serializes to and from an int or use of the decimal data type.</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></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></td>
</tr>
<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><code>schema.propertyKey('coordinates').Point().withGeoBounds().create()</code></td>
</tr>
<tr>
<td>polygon</td>
<td>Used for geospatial and Cartesian polygons (double .... points)</td>
<td><code>schema.propertyKey('block').Polygon().withGeoBounds().create()</code></td>
</tr>
<tr>
<td>smallint</td>
<td>2 byte integer</td>
<td><code>schema.propertyKey('age').Smallint().create()</code></td>
</tr>
<tr>
<td><strong>DSE Graph Data Type</strong></td>
<td><strong>Description</strong></td>
<td><strong>Schema example</strong></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>text</td>
<td>String or UTF-8 encoded string</td>
<td><code>schema.propertyKey('name').Text().create()</code></td>
</tr>
<tr>
<td>time</td>
<td>Time in the format of ‘10:00:00’ or ‘10:00’.</td>
<td><code>schema.propertyKey('time').Time().create()</code></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 timestamp:</td>
<td><code>schema.propertyKey('mealCreationDate').Timestamp().create()</code></td>
</tr>
<tr>
<td>uuid</td>
<td>A UUID in standard UUID format or timeuuid format</td>
<td><code>schema.propertyKey('authorID').Uuid().create()</code></td>
</tr>
<tr>
<td>varint</td>
<td>Arbitrary-precision integer</td>
<td><code>schema.propertyKey('number').Varint().create()</code></td>
</tr>
</tbody>
</table>

**Graph storage in the DSE database keyspace and tables**

DSE Graph uses the DSE database to store schema and data. Three DSE database keyspaces are created for each graph, `<graphname>`, `<graphname_system>`, and `<graphname_pvt>`. For example, for a graph called *food*, the three keyspaces created will be *food*, *food_system*, and *food_pvt*. 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. The third keyspace *food_pvt* holds information about partitioning for vertices should the graph contain a vertex with a large number of edges that requires the vertex table to be partitioned across the cluster.

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*.

**Apache TinkerPop graph computing framework**

Apache TinkerPop is a graph abstraction layer that works with numerous different graph databases and graph processors. Apache TinkerPop is composed of two elements: a structure API and a process API.

The primary components of the Apache TinkerPop structure API are:

**Graph**
- maintains a set of vertices and edges

**Vertex**
Using DataStax Enterprise advanced functionality

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, 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 Apache TinkerPop process API are:

TraversalSource
a generator of traversals for a particular graph, domain specific language (DSL), and execution engine

Traversal<S,E>
a functional data flow process transforming objects of type S into object of type E

GraphTraversal
a traversal 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 Apache TinkerPop is Gremlin, a graph traversal language and virtual machine. Apache 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.

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.
Using DataStax Enterprise advanced functionality

**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><strong>Smartly replicates data from source clusters to destination clusters</strong></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><strong>Prioritizes data streams</strong></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><strong>Supports ingestion and querying of data at every source</strong></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><strong>Solves problem of periodic downtime</strong></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><strong>Satisfies data sovereignty regulations</strong></td>
<td>Provides configurable streams of selected outbound data, while preventing data changes to inbound data.</td>
</tr>
<tr>
<td><strong>Satisfies data locality regulations</strong></td>
<td>Prevents data from leaving the current geography.</td>
</tr>
</tbody>
</table>

#### 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`:

```bash
$ 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 the `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 964)*
2. Setting up the source cluster *(page 965)*
3. Creating sample keyspace and table *(page 966)*
4. Configuring replication on the source node *(page 966)*
5. Creating the replication channel *(page 972)*
6. Starting replication from source to destination *(page 973)*
7. Inserting data on the source *(page 973)*
8. Testing loss of connectivity *(page 974)*
9. Testing replication start and stop *(page 975)*

**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.

**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**.

### Setting up the destination cluster node

The destination cluster requires DataStax Enterprise 4.8 or later. On the destination node:
1. **Install DataStax Enterprise** *(page 222)* 4.8 or later.

2. Start DataStax Enterprise as a transactional node with the command that is appropriate for the **installation method** *(page 1090)*.

3. Note the public IP address for the destination node.

**Setting up the source cluster**

The source cluster requires DataStax Enterprise 5.1 or later. On the source node:

1. **Install DataStax Enterprise** *(page 222)* 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 278)* 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 1090)*.

7. Once advanced replication is started on a cluster, the source node will create keyspaces and tables that need alteration. See **Keyspaces** *(page 975)* 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 REPLICACTION = {
     '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 1009](#).

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.
- 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:
Using DataStax Enterprise advanced functionality

```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>
<tr>
<td>mydest</td>
<td>driver_ssl_cipher_suites</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
TLS_RSA_WITH_AES_256_CBC_SHA256,
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,
Using DataStax Enterprise advanced functionality

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_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.
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_RSA_WITH_AES_256_GCM_SHA384,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_DSS_WITH_AES_256_GCM_SHA384,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_RSA_WITH_AES_128_GCM_SHA256,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_DHE_DSS_WITH_AES_128_GCM_SHA256,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL_RSA_WITH_3DES_EDE_CBC_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_RC4_128_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL_RSA_WITH_RC4_128_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_RC4_128_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL_RSA_WITH_RC4_128_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_RC4_128_SHA,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL_RSA_WITH_RC4_128_MD5,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

```bash
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

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>
</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 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:

   ```
   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>
<tr>
<td>(0 rows)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. On the source, replication to the destination can be paused or resumed, the latter shown here:

   ```
   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:

   ```
   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:

   ```
   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:

   ```
   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>
<tr>
<td>(0 rows)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

Checking data on the destination

Check data on the destination.

1. On the destination, verify that the data was replicated:

```
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>
</tbody>
</table>

(2 rows)

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.
Using DataStax Enterprise advanced functionality

```
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:

```
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 1009). 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:

```
dse advrep --verbose channel pause --keyspace foo --table bar --collection
```

2. Insert more data.

3. On the source, resume the replication channel:

```
dse advrep --verbose channel resume --keyspace foo --table bar --collection
```

DSE Advanced Replication keyspace overview

Keyspaces 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
Using DataStax Enterprise advanced functionality

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:

```cql
ALTER KEYSPACE dse_advrep
WITH REPLICATION = {
  '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:

```cql
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;<code>data_type</code>&gt;, frozen-set&lt;<code>data_type</code>&gt;, frozen-map&lt;<code>data_type</code>, <code>data_type</code>&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;<code>data_type</code>, <code>data_type</code>, <code>data_type</code>&gt;, frozen-tuple&lt;<code>data_type</code>, <code>data_type</code>, <code>data_type</code>&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: <code>&lt;list&lt;data_type&gt;&gt;&lt;set&lt;data_type&gt;&gt;, &lt;map&lt;data_type, data_type&gt;&gt;</code></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>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 977)*
2. Stopping DSE Advanced Replication *(page 978)*
3. Configuring global configuration settings *(page 979)*
4. Configuring destination settings *(page 980)*
5. Configuring channel settings *(page 993)*
6. Security *(page 994)*
7. Data insert methods *(page 996)*
8. Monitoring operations *(page 996)*

**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.

Starting DSE Advanced Replication

Before you can start and use DSE Advanced Replication, you must create the user keyspace *(page 975)* 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.

   At the end of the 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
   ```
Using DataStax Enterprise advanced functionality

advanced_replication_directory: /var/lib/cassandra/advrep

2. **Enable Capture-Data-Change (CDC) (page 278)** 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 975):**

   ```sql
   DROP TABLE dse_system.advrep_source_config;
   DROP TABLE dse_system.advrep_destination_config;
   DROP TABLE dse_system.advrep_repl_channel_config;
   ```
DROP KEYSSPACE dse_advrep;

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 1009)* with this syntax:

```
dse advrep conf ...
```

To view the source node configuration settings:

```
dse advrep conf list
```

The result is:

```
----------------------------------- |----------------|
| name             | value          |
----------------------------------- |----------------|
| audit_log_file   |/tmp/myaudit.gz|
----------------------------------- |----------------|
| audit_log_enabled| true           |
----------------------------------- |----------------|
```

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 1009)* 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>
</tbody>
</table>
### Configuration key

<table>
<thead>
<tr>
<th>Configuration key</th>
<th>Default value</th>
<th>Description</th>
<th>Restart required</th>
</tr>
</thead>
<tbody>
<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: 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. NONE: Perform no logging.</td>
<td>No</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_adv</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: • Match the audit log file template • And they have not been written to for more than the specified maximum lifespan minutes To disable purging, set to 0.</td>
<td>Yes</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 1009) 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>
</tbody>
</table>

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, |
| | |

DSE 5.1 Developer Guide (Previous version)
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_RSA_WITH_AES_256_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_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_ECDHE_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, | , |
Using DataStax Enterprise advanced functionality

| 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

<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>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>llpdest</th>
<th>driver_ssl_enabled</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>llpdest</td>
<td>addresses</td>
<td>10.200.177.184</td>
</tr>
<tr>
<td>llpdest</td>
<td>driver_read_timeout</td>
<td>15000</td>
</tr>
<tr>
<td>llpdest</td>
<td>driver_connections_max</td>
<td>8</td>
</tr>
<tr>
<td>llpdest</td>
<td>source_id_column</td>
<td>source_id</td>
</tr>
<tr>
<td>llpdest</td>
<td>driver_connect_timeout</td>
<td>15000</td>
</tr>
<tr>
<td>llpdest</td>
<td>driver_ssl_protocol</td>
<td>TLS</td>
</tr>
<tr>
<td>llpdest</td>
<td>driver_consistency_level</td>
<td>ONE</td>
</tr>
<tr>
<td>llpdest</td>
<td>driver_used_hosts_per_remote_dc</td>
<td>0</td>
</tr>
<tr>
<td>llpdest</td>
<td>driver_allow_remote_dcs_for_local_cl</td>
<td>false</td>
</tr>
<tr>
<td>llpdest</td>
<td>driver_compression</td>
<td>lz4</td>
</tr>
<tr>
<td>llpdest</td>
<td>driver_connections</td>
<td>1</td>
</tr>
<tr>
<td>llpdest</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>[TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>,</td>
</tr>
<tr>
<td></td>
<td>[TLS_RSA_WITH_AES_256_CBC_SHA256,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>,</td>
</tr>
<tr>
<td></td>
<td>[TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>,</td>
</tr>
</tbody>
</table>
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_ECDHE_ECDSA_WITH_AES_256_CBC_SHA256, | | |
TLS_ECDHE_RSA_WITH_AES_256_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, | | |
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_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>
</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, |
| SSL_RSA_WITH_3DES_EDE_CBC_SHA, |
Using DataStax Enterprise advanced functionality

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDHE_RSA_WITH_RC4_128_SHA,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL_RSA_WITH_RC4_128_SHA,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDH_ECDSA_WITH_RC4_128_SHA,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_ECDH_RSA_WITH_RC4_128_SHA,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL_RSA_WITH_RC4_128_MD5,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TLS_EMPTY_RENEGOTIATION_INFO_SCSV</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>llpdest</th>
<th>source_id</th>
<th>source1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>llpdest</th>
<th>transmission_enabled</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></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 <strong>driver-consistency-level</strong> 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 <strong>valid DSE consistency level</strong>: 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 command line tool (page 1009) 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:

<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</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>foo</td>
<td>bar</td>
<td>true</td>
<td>true</td>
<td>FIFO</td>
<td>2</td>
<td>foo</td>
<td>bar</td>
<td>source1</td>
<td>source_id</td>
<td>mydest</td>
<td>true</td>
</tr>
</tbody>
</table>

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.
Using DataStax Enterprise advanced functionality

<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 metadata</td>
</tr>
<tr>
<td>source-id-column</td>
<td>Placeholder to override the source-id-column that is defined in advrep_conf metadata</td>
</tr>
<tr>
<td>enabled</td>
<td>If true, replication will start for this table. If false, no more messages 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 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-keyspace</td>
<td>The keyspace on the destination for the replicated table.</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-enabled</td>
<td>Specify if the data collector for the table should be replicated to the 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 Connecting to authentication enabled clusters. 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 980) 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 with the `system_key_directory` property:
     ```yaml
     system_key_directory: /etc/dse/conf
     ```
     The default value is `/etc/dse/conf`.
   - Specify that encryption keys are generated as system keys with the `config_encryption_key_name` property:
     ```yaml
     config_encryption_key_name: 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 1032).

   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 321) 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:

```
dse advrep destination --driver-pwd "Sa9xOVaym7bddjXUT/eeOQ==" --
driver-user "username"
```

5. Start `dse` (page 1090).

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:

- Tables that are defined for compact storage
- `sstableloader` (Cassandra bulk loader)
- OpsCenter restore from backup
- Spark `bulkSaveToCassandra`

`dse.yaml`

The location of the `dse.yaml` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/dse/dse.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer-Services installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
<tr>
<td>Tarball installations</td>
<td><code>installation_location/</code></td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td><code>resources/dse/conf/dse.yaml</code></td>
</tr>
</tbody>
</table>

Monitoring operations

Advanced replication can be monitored with JMX metrics. The outgoing replication queue size is a key factor to watch. See Metrics (page 998) 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**

- 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` 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.
Using DataStax Enterprise advanced functionality

- 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 979) 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 964).

Ensure JMX access

Metrics are stored in the DataStax Enterprise JMX system. JMX access is required.
Using DataStax Enterprise advanced functionality

- For production, DataStax recommends authenticating JMX users, see Setting up JMX users 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 1090) 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:

   ```
   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></td>
<td></td>
<td></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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessageAddErrors</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></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></td>
<td>FifteenMinuteRate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OneMinuteRate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FiveMinuteRate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesAdded</td>
<td>1002</td>
<td>events/second</td>
<td></td>
</tr>
<tr>
<td>0.012688461014514603</td>
<td>9.862886141388435E-39</td>
<td>2.964393875E-314</td>
<td>2.322135514219019E-114</td>
<td></td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesDeleted</td>
<td>0</td>
<td>events/second</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td>0.0</td>
</tr>
</tbody>
</table>

DSE 5.1 Developer Guide (Previous version)
Using DataStax Enterprise advanced functionality

|ReplicationLog|MessagesAcknowledged|1002|events/second|
|0.012688456391385135|9.86403600116801E-39|2.964393875E-314|2.3230339468969963E-114|

|ReplicationLog|CommitLogMessagesRead|16873|events/second|
|0.21366497971804438|0.20580430240786005|0.39126032533612265|0.2277227124698431|

|Group|Type|Value|
|-------------------------------------|
|Transmission|AvailablePermits|30000|

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:
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 decrementing 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 1002)
- Transmission (page 1003)
- AdvancedReplicationHub-[destinationId]-metrics (page 1003)

Metrics are also available per table:
- Performance metrics per table (page 1005)

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>
<tr>
<td>Metric name</td>
<td>Description</td>
<td>Metric type</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<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.
Incomplete 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>
</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>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>

cassandra-env.sh
The location of the `cassandra-env.sh` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>Installer-Services installations</th>
<th>/etc/dse/cassandra/cassandra-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>Installer-No Services installations</td>
<td>installation_location/resources/cassandra/conf/cassandra-env.sh</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/`
Using DataStax Enterprise advanced functionality

<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:

- 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.
Using DataStax Enterprise advanced functionality

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 979), 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:

   ```
   $ 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:

   ```
   $ 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:

   ```
   $ 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.

   ```
   $ 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.
$ 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 Advanced Replication command line tool**

The command line tool provides commands and options for configuring and using DSE Advanced Replication.

DSE Advanced Replication commands

These DSE Advanced Replication commands are available:

- Command options *(page 1009)* for all commands
- Client to DSE connection commands *(page 1009)*
- Replication channel *(page 1012)* commands
- Replication destination *(page 1013)* commands
- Replication configuration *(page 1016)* commands
- Replication log *(page 1018)* commands
- Metrics *(page 1018)* commands

DSE Advanced Replication command options

Synopsis

```
 dse advrep [--v1] [connection_options] [command] [sub_command]
 [sub_command_options]
```

To show the command line help for `dse advrep`:

```
 $ dse advrep help
```

Authentication credentials can be provided in several ways, see Connecting to authentication enabled clusters.

The optional flag, `--v1`, can be used to access advanced replication commands for DSE 5.0; advanced replication commands changed with DSE 5.1. See DSE 5.0 Advanced Replication documentation for commands compatible with the `--v1` flag. An example of using the `--v1` flag is:

```
 dse advrep --v1 edge conf --edge-id "edge1" --edge-id-col-name "edge_id"
 --hub-ip-addresses "10.200.182.148"
```

Client to DSE connection commands

The default port for DSE Advanced Replication is 9042. Connection options are specified as:
### dse advrep [connection_options]

<table>
<thead>
<tr>
<th>Connection options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>--separator</strong>&lt;br&gt;<code>field_separator</code></td>
<td>Specify the field separator if <code>--no-pretty-print</code> is used for printing. Default: comma</td>
</tr>
<tr>
<td><strong>--no-pretty-print</strong></td>
<td>If specified, data is printed as a comma delimited list. If not specified, data is printed as tabular output.</td>
</tr>
<tr>
<td><strong>-u username</strong></td>
<td>DSE username, or alternatively, DSE_USERNAME environment variable can be used.</td>
</tr>
<tr>
<td><strong>--jmx-user</strong></td>
<td>JMX metrics user</td>
</tr>
<tr>
<td><strong>--jmx-port</strong></td>
<td>JMX port (default: 7199)</td>
</tr>
<tr>
<td><strong>--jmx-pwd</strong></td>
<td>JMX metrics password</td>
</tr>
<tr>
<td><strong>-p password</strong></td>
<td>DSE password, or alternatively, DSE_PASSWORD environment variable can be used.</td>
</tr>
<tr>
<td><strong>--use-server-config</strong></td>
<td>Read connection configuration from server YAML files (dse.yaml and cassandra.yaml) instead of reading them from configuration files. Use only when the DSE installation against which the command run is a running node.</td>
</tr>
<tr>
<td><strong>--cipher-suites</strong></td>
<td>A comma-separated list of SSL cipher suites for connection to DSE when SSL is enabled. For example, <code>--cipher-suites=c1,c2,c3</code>.</td>
</tr>
<tr>
<td><strong>--host address</strong></td>
<td>The DSE host RPC broadcast address. The default value is localhost.</td>
</tr>
<tr>
<td><strong>--port port</strong></td>
<td>The DSE native protocol RPC connection port.</td>
</tr>
<tr>
<td><strong>--kerberos-enabled</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>--keystore-password</strong></td>
<td>Keystore password for connection to DSE when SSL client authentication is enabled.</td>
</tr>
<tr>
<td><strong>--keystore-path</strong></td>
<td>Set path to the keystore for connection to DSE when SSL client authentication is enabled.</td>
</tr>
<tr>
<td><strong>--keystore-type</strong></td>
<td>Set 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.</td>
</tr>
<tr>
<td><strong>--truststore-password</strong></td>
<td>Set the truststore password for connection to DSE when SSL is enabled.</td>
</tr>
<tr>
<td><strong>--truststore-path</strong></td>
<td>Set path to truststore for connection to DSE when SSL is enabled.</td>
</tr>
</tbody>
</table>
### Connection options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--truststore-type</code></td>
<td>Set 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.</td>
</tr>
<tr>
<td><code>ssl_truststore_type</code></td>
<td></td>
</tr>
<tr>
<td><code>--sasl-protocol-name</code></td>
<td>The SASL protocol name must match the username of the Kerberos service principal <a href="#">page 316</a> used by the DSE server.</td>
</tr>
<tr>
<td><code>dse_service_principal</code></td>
<td></td>
</tr>
<tr>
<td><code>--ssl</code></td>
<td>Specify whether SSL is enabled for connection to DSE. <code>--ssl-enabled true</code> is the same as <code>--ssl</code>.</td>
</tr>
<tr>
<td><code>--ssl-protocol</code></td>
<td>SSL protocol for connection to DSE when SSL is enabled.</td>
</tr>
<tr>
<td><code>ssl_protocol</code></td>
<td></td>
</tr>
<tr>
<td><code>-t</code></td>
<td>Delegation token which can be used to login, or alternatively, DSE_TOKEN environment variable can be used.</td>
</tr>
</tbody>
</table>

### Connection options examples:

```bash
$ dse advrep --host ip-10-200-300-138.example.lan --kerberos-enabled=true conf list
```

and

```bash
$ dse advrep --use-server-config conf list
```

### Printing option example:

```bash
dse advrep --no-pretty-print destination list-conf --separator "|"
```

will result in output:

```plaintext
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_ECDHE_ECDSA,TLS_RSA_WITH_ECDHE_RSA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_129_CBC_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA256,TLS_RSA_WITH_3DES_EDE_CBC_SHA,TLS_EMPTY_RENEGOTIATION_INFO_SCSV
mydest|driver-ssl-enabled|false
mydest|driver-ssl-protocol|TLS
mydest|name|mydest
mydest|driver-connect-timeout|15000
mydest|driver-max-requests-per-connection|1024
mydest|driver-connections-max|8
mydest|driver-connections|1
mydest|driver-compression|lz4
mydest|driver-consistency-level|ONE
mydest|driver-allow-remote-dcs-for-local-cl|false
mydest|driver-used-hosts-per-remote-dc|0
mydest|driver-read-timeout|15000
```

If `--no-pretty-print` is not used, the output is in tabular format by default.
Replication channel commands

Synopsis

```
$ dse advrep channel command [sub_command] [sub_command_options]
```

To show the command line help for `dse advrep channel`:

```
$ dse advrep help channel
```

Command and command arguments for:

```
$ dse advrep channel sub_command [sub_command_options]
```

<table>
<thead>
<tr>
<th>Sub-command</th>
<th>Sub-command options</th>
</tr>
</thead>
<tbody>
<tr>
<td>create (page 1019)</td>
<td>Create a replication channel for a keyspace:</td>
</tr>
<tr>
<td></td>
<td>--source-keyspace <code>keyspace_name</code> (required)</td>
</tr>
<tr>
<td></td>
<td>--destination-table <code>destination_table_name</code></td>
</tr>
<tr>
<td></td>
<td>--source-id-column</td>
</tr>
<tr>
<td></td>
<td>--fifo-order Specify if the channel should be replicated in FIFO order (default).</td>
</tr>
<tr>
<td></td>
<td>--lifo-order Specify if the channel should be replicated in LIFO order.</td>
</tr>
<tr>
<td></td>
<td>--destination <code>destination</code> (required) Destination where the replication will be</td>
</tr>
<tr>
<td></td>
<td>sent.</td>
</tr>
<tr>
<td></td>
<td>--source-table <code>source_table_name</code> (required) Source table to replicate.</td>
</tr>
<tr>
<td></td>
<td>--collection-enabled true</td>
</tr>
<tr>
<td></td>
<td>--priority <code>channel_priority</code></td>
</tr>
<tr>
<td></td>
<td>--transmission-enabled true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>update (page 1018)</th>
<th>Update a replication channel for a keyspace:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>--source-keyspace <code>keyspace_name</code></td>
</tr>
<tr>
<td></td>
<td>--destination-table <code>destination_table_name</code></td>
</tr>
<tr>
<td></td>
<td>--source-id-column</td>
</tr>
<tr>
<td></td>
<td>--fifo-order Specify if the channel should be replicated in FIFO order (default).</td>
</tr>
<tr>
<td></td>
<td>--lifo-order Specify if the channel should be replicated in LIFO order.</td>
</tr>
<tr>
<td></td>
<td>--destination <code>destination</code> Destination where the replication will be sent.</td>
</tr>
<tr>
<td></td>
<td>--source-table <code>source_table_name</code> Source table to replicate.</td>
</tr>
<tr>
<td></td>
<td>--collection-enabled true</td>
</tr>
<tr>
<td></td>
<td>--priority <code>channel_priority</code></td>
</tr>
<tr>
<td></td>
<td>--transmission-enabled true</td>
</tr>
<tr>
<td>Sub-command</td>
<td>Sub-command options</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
</tr>
</tbody>
</table>
| delete      | Delete replication channel:  
--source-keyspace *keyspace_name* (required)  
--destination *destination* (required) Destination where the replication will be sent.  
--source-table *source_table_name* (required) Source table to replicate.  
--data-center-id *data-center-id* Datacenter for this channel |
| pause       | Pause replication channel for a keyspace:  
--source-keyspace *keyspace_name*  
--data-center-ids *data-center-id(s)* Comma-separated list of data-center-ids on which to filter  
--destinations *destination(s)* Comma-separated list of destinations  
--transmission If specified, no data for the source table is sent to the specified destination.  
--source-table *source_table_name*  
--collection If specified, no data for the source table is collected. |
| resume      | Resume replication for a paused channel:  
--source-keyspace *keyspace_name*  
--data-center-ids *data-center-id(s)* Comma-separated list of data-center-ids on which to filter  
--destinations *destination(s)* Comma-separated list of destinations  
--transmission If specified, no data for the source table is sent to the specified destination.  
--source-table *source_table_name*  
--collection If specified, no data for the source table is collected. |
| status      | Verify status of replication channel:  
--keyspace *keyspace_name*  
--destination *destination* Destination where the replication will be sent.  
--table *source_table_name* Source table to replicate.  
--data-center-id *data-center-id* Datacenter for this channel |
| truncate    | Truncate a channel, all messages currently in the replication log for that channel will not be replicated:  
--source-keyspace *keyspace_name*  
--data-center-ids *data-center-id(s)* Comma-separated list of data-center-ids on which to filter  
--destinations *destination(s)* Comma-separated list of destinations  
--source-table *source_table_name* |

Destination commands

Synopsis
Using DataStax Enterprise advanced functionality

$ dse advrep destination [sub_command] [sub_command_options]

To show the command line help for `dse advrep destination`:

$ dse advrep help destination

Commands and command arguments for edge configuration and replication:

$ dse advrep destination sub_command [sub_command_options]

<table>
<thead>
<tr>
<th>Command</th>
<th>Command arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>Create a channel from a keyspace source table to a destination cluster:</td>
</tr>
<tr>
<td></td>
<td>--name (required)</td>
</tr>
<tr>
<td></td>
<td>--addresses (required)</td>
</tr>
<tr>
<td></td>
<td>--transmission-enabled true</td>
</tr>
<tr>
<td></td>
<td>--driver-user password</td>
</tr>
<tr>
<td></td>
<td>--driver-pwd username</td>
</tr>
<tr>
<td></td>
<td>--driver-connect-timeout ms _driver_waits_to_connect_server</td>
</tr>
<tr>
<td></td>
<td>--driver-read-timeout ms _driver_waits_to_read_server_responses</td>
</tr>
<tr>
<td></td>
<td>--driver-compression lz4</td>
</tr>
<tr>
<td></td>
<td>--driver-connections num_connections_to_create</td>
</tr>
<tr>
<td></td>
<td>--driver-connections-max max_num_connections_to_create</td>
</tr>
<tr>
<td></td>
<td>--driver-max-requests-per-connections max_num_requests_per_connection</td>
</tr>
<tr>
<td></td>
<td>--driver-local-dc dc_name</td>
</tr>
<tr>
<td></td>
<td>--driver-consistency-level consistency [ANY]</td>
</tr>
<tr>
<td></td>
<td>--driver-used-hosts-per-remote-dc number_of_hosts</td>
</tr>
<tr>
<td></td>
<td>--driver-allow-remote-dcs-for-local-cl true</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-enabled true</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-keystore-path ssl_keystore_path</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-keystore-password ssl_keystore_password</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-keystore-type ssl_keystore_type</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-truststore-path ssl_truststore_path</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-truststore-password ssl_truststore_password</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-truststore-type ssl_keystore_type</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-protocol ssl_protocol</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-cipher-suites ssl_cipher_suitess</td>
</tr>
<tr>
<td>Command</td>
<td>Command arguments</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>update</td>
<td>Update a channel from a keyspace source table to a destination cluster:</td>
</tr>
<tr>
<td></td>
<td>--name (required)</td>
</tr>
<tr>
<td></td>
<td>--addresses</td>
</tr>
<tr>
<td></td>
<td>--transmission-enabled true/false</td>
</tr>
<tr>
<td></td>
<td>--driver-user password</td>
</tr>
<tr>
<td></td>
<td>--driver-pwd username</td>
</tr>
<tr>
<td></td>
<td>--driver-connect-timeout ms_driver_waits_to_connect_server</td>
</tr>
<tr>
<td></td>
<td>--driver-read-timeout ms_driver_waits_to_read_server_responses</td>
</tr>
<tr>
<td></td>
<td>--driver-compression lz4_or_snappy_algorithm</td>
</tr>
<tr>
<td></td>
<td>--driver-connections num_connections_to_create</td>
</tr>
<tr>
<td></td>
<td>--driver-connections-max max_num_connections_to_create</td>
</tr>
<tr>
<td></td>
<td>--driver-max-requests-per-connection max_num_requests_per_connection</td>
</tr>
<tr>
<td></td>
<td>--driver-local-dc dc_name</td>
</tr>
<tr>
<td></td>
<td>--driver-consistency-level consistency [ANY</td>
</tr>
<tr>
<td></td>
<td>--driver-used-hosts-per-remote-dc number_of_hosts</td>
</tr>
<tr>
<td></td>
<td>--driver-allow-remote-dcs-for-local-cl true</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-enabled true</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-keystore-path ssl_keystore_path</td>
</tr>
<tr>
<td></td>
<td>--driver_ssl_keystore_password ssl_keystore_password</td>
</tr>
<tr>
<td></td>
<td>--driver_ssl_keystore_type ssl_keystore_type</td>
</tr>
<tr>
<td></td>
<td>--driver_ssl_truststore_path ssl_truststore_path</td>
</tr>
<tr>
<td></td>
<td>--driver_ssl_truststore-password ssl_truststore_password</td>
</tr>
<tr>
<td></td>
<td>--driver_ssl_truststore-type ssl_keystore_type</td>
</tr>
<tr>
<td></td>
<td>--driver_ssl-protocol ssl_protocol</td>
</tr>
<tr>
<td></td>
<td>--driver_ssl-cipher-suites ssl_cipher_suites</td>
</tr>
<tr>
<td>delete</td>
<td>Delete destination:</td>
</tr>
<tr>
<td></td>
<td>--name (required)</td>
</tr>
<tr>
<td>list-conf</td>
<td>List the destination configuration:</td>
</tr>
<tr>
<td></td>
<td>--name (required)</td>
</tr>
</tbody>
</table>
Using DataStax Enterprise advanced functionality

<table>
<thead>
<tr>
<th>Command</th>
<th>Command arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove-conf</td>
<td>Remove configuration for a channel from a keyspace source table to a destination</td>
</tr>
<tr>
<td></td>
<td>cluster:</td>
</tr>
<tr>
<td></td>
<td>--name (required)</td>
</tr>
<tr>
<td></td>
<td>--transmission-enabled true</td>
</tr>
<tr>
<td></td>
<td>--driver-user password</td>
</tr>
<tr>
<td></td>
<td>--driver-pwd username</td>
</tr>
<tr>
<td></td>
<td>--driver-connect-timeout ms_driver_waits_to_connect_server</td>
</tr>
<tr>
<td></td>
<td>--driver-read-timeout ms_driver_waits_to_read_server_responses</td>
</tr>
<tr>
<td></td>
<td>--driver-compression lz4_or_snappy_algorithm</td>
</tr>
<tr>
<td></td>
<td>--driver-connections num_connections_to_create</td>
</tr>
<tr>
<td></td>
<td>--driver-consistency-level consistency [ANY</td>
</tr>
<tr>
<td></td>
<td>--driver-used-hosts-per-remote-dc number_of_hosts</td>
</tr>
<tr>
<td></td>
<td>--driver-allow-remote-dcs-for-local-cl true</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-enabled true</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl-keystore-path ssl_keystore_path</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl_truststore_path ssl_truststore_path</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl_cipher_suites ssl_cipher_suites</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl_protocol ssl_protocol</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl_truststore_password ssl_truststore_password</td>
</tr>
<tr>
<td></td>
<td>--driver-ssl_truststore-type ssl_keystore_type</td>
</tr>
<tr>
<td></td>
<td>list</td>
</tr>
<tr>
<td></td>
<td>List the destination - no options</td>
</tr>
</tbody>
</table>

**Configuration commands**

**Synopsis**

```
$ dse advrep conf [sub_command] [sub_command_options]
```

To show the command line help for `dse advrep conf`:

```
$ dse advrep help conf
```

Commands and command arguments for configuration:

```
$ dse advrep conf sub_command [sub_command_options]
```
<table>
<thead>
<tr>
<th>Command</th>
<th>Command arguments</th>
</tr>
</thead>
</table>
| update    | update the configuration:  
--collection-max-open-files value Number of option files kept  
--audit-log-file audit_log_name Audit log file name  
--audit-log-compression none | gzip Enable audit log compression.  
--audit-log-enabled true | false Enable audit logging.  
--audit-log-max-life-span-mins minutes Audit log max lifespan in minutes before discarding.  
--audit-log-rotate-mins minutes Number of minutes before audit log rotates files.  
--permits Maximum number of messages that can be replicated in parallel over all destinations (default: 30,000).  
--invalid-message-log Error information for messages that could not be replicated are saved to system_log, channel_log (default), or none.  
--collection-time-slice-count count The number of files which are open in the ingestor simultaneously. Default: 5  
--collection-time-slice-width The time period in seconds for each data block ingested. Smaller time widths equal more files. Larger timer widths equal larger files but more data to resend on CRC mismatches. Default: 60 seconds  
--collection-expire-after-write |
| remove    | Remove configuration:  
--collection-max-open-files value Number of option files kept  
--audit-log-file audit_log_name Audit log file name  
--audit-log-compression Disable audit log compression.  
--audit-log-enabled true | false Enable audit logging.  
--audit-log-max-life-span-mins minutes Audit log max lifespan in minutes before discarding.  
--audit-log-rotate-mins minutes Number of minutes before audit log rotates files.  
--permits Maximum number of messages that can be replicated in parallel over all destinations (default: 30,000).  
--invalid-message-log Error information for messages that could not be replicated are saved to system_log, channel_log (default), or none.  
--collection-time-slice-count count The number of files which are open in the ingestor simultaneously.  
--collection-time-slice-width The time period in seconds for each data block ingested. Smaller time widths equal more files. Larger timer widths equal larger files but more data to resend on CRC mismatches.  
--collection-expire-after-write |
| list (page 1019) | List the configuration - no options |
Using DataStax Enterprise advanced functionality

Metrics

Display metrics from the command line once JMX access (page 998) is configured:

```bash
$dse advrep metrics
```

To show the command line help for `dse advrep metrics`:

```bash
$dse advrep help metrics
```

Command options for:

```bash
$dse advrep metrics list [sub_command_options]
```

<table>
<thead>
<tr>
<th>Command</th>
<th>Command options</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>[ --metric-group filter_by_group ] [ --metric-type filter_by_type ]</td>
</tr>
</tbody>
</table>

Replication log commands

Display replication log information from the command line.

```bash
$dse advrep replog
```

To show the command line help for `dse advrep replog`:

```bash
$dse advrep help replog
```

Command options for:

```bash
$dse advrep replog sub_command [sub_command_options]
```

<table>
<thead>
<tr>
<th>Command</th>
<th>Command options</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>--data-center-id data-center-id</td>
</tr>
<tr>
<td></td>
<td>--destination destination (required) Destination where the replication will be sent.</td>
</tr>
<tr>
<td></td>
<td>--source-keyspace keyspace_name (required)</td>
</tr>
<tr>
<td></td>
<td>--source-table table_name (required)</td>
</tr>
<tr>
<td>analyze-audit-log</td>
<td>--file filename Audit log file.</td>
</tr>
</tbody>
</table>

Replication command examples

The `dse advrep` channel command line tool examples and results:

Creating a destination

```bash
$dse advrep --verbose destination create --name mydest --addresses 10.200.182.148 --transmission-enabled true
```
Using DataStax Enterprise advanced functionality

### Creating 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
Created channel dc=Cassandra keyspace=foo table=bar to mydest
```

### Viewing a replication source channel status

```
$ 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| foo     | bar            | true      | false       | FIFO |
| mydest|true        |                 |
```

### Removing a destination channel

```
$ dse advrep delete --name mydest
```

### Removing a replication source channel

```
$ dse advrep delete --source-keyspace foo --source-table bar --destination mydest
```

### Resuming collection from a source to a destination

```
$ dse advrep channel resume --source-keyspace foo --source-table bar --destinations mydest --collection
Channel dc=Cassandra keyspace=foo table=bar collection to mydest was resumed
```

### Resuming transmission from a source to a destination

```
$ dse advrep channel resume --source-keyspace foo --source-table bar --destinations mydest --transmission
Channel dc=Cassandra keyspace=foo table=bar transmission to mydest was resumed
```

### Pausing transmission from a source to a destination

The replication is stopped for the entire edge cluster. Specify the IP address of any node on the edge.

```
$ dse advrep channel pause --source-keyspace foo --source-table bar --destinations mydest --transmission
Channel dc=Cassandra keyspace=foo table=bar transmission to mydest was paused
```
Using DataStax Enterprise advanced functionality

Verifying the record count held in the replication log

```
$ dse advrep replog count --destination mydest --source-keyspace foo --source-table bar
2
```

Metrics command examples

The `dse advrep metrics` command line tool examples and results.

**Show all metrics**

```
$ dse advrep --host localhost --port 7199 metrics list
```

<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReplicationLog</td>
<td>CommitLogsToConsume</td>
<td>1</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessageAddErrors</td>
<td>0</td>
</tr>
<tr>
<td>Tables</td>
<td>MessagesInReplicationLog</td>
<td>0</td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>CommitLogsDeleted</td>
<td>2</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>FifteenMinuteRate</td>
<td>OneMinuteRate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FiveMinuteRate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesAdded</td>
<td>4</td>
<td>events/second</td>
<td>2.6401646866982826E-4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesDeleted</td>
<td>0</td>
<td>events/second</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesAcknowledged</td>
<td>4</td>
<td>events/second</td>
<td>2.6401646866982826E-4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>CommitLogMessagesRead</td>
<td>3551</td>
<td>events/second</td>
<td>0.23438054858983395</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Filter by group**

```
$ dse advrep --host localhost --port 7199 metrics list --metric-group Tables
```

<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
</table>

Page 1020
Filter by group and type

<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></td>
<td></td>
</tr>
<tr>
<td>FiveMinuteRate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReplicationLog</td>
<td>MessagesAdded</td>
<td>4</td>
<td>events/second</td>
<td>2.6221009715537605E-4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.1552879575422323E-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.932609260931711E-108</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.329051114010331E-24</td>
</tr>
</tbody>
</table>
# DataStax Enterprise tools

## dsetool

### About dsetool

dsetool is a command line interface for DSE operations.

### Synopsis

```
dsetool [connection_options] command command_args
```

### Table 56: 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><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>
</tbody>
</table>
Using dsetool command line help

To view a listing of dsetool commands:

```
dsetool help
```

To view help for a specific command:

```
dsetool command help
```

dsetool commands for DSE Search

Search CQL commands (page 536) 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. Using dsetool with SSL requires some JMX setup. See Setting up SSL for nodetool, dsetool, and dse advrep.

**Synopsis**

```
dsetool [connection_options]
```

**Table 57: 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>Italicics</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>

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:** Authentication credentials can be provided in several ways, see Connecting to authentication enabled clusters.

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 jmx_username, --jmxusername jmx_username**  
  User name for authenticating with secure local JMX.

- **-b jmx_password, --jmxpassword jmx_password**  
  Password for authenticating with secure local JMX. If you do not provide a password, you are prompted to enter one.

- **-c dse_port, --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 filename, --config-file filename**  
  File path to configuration file that stores credentials. The credentials in this configuration file override the ~/.dserc credentials.
-h IP_address, --host IP_address
  Connect to the specified hostname or IP address. Do not specify to connect to the
  local node.

-j jmx_port, --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 username
  Role to authenticate for database access.

-p password, --password password
  Password to authenticate for database access.

-s solr_port, --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 autojt (deprecated)**

This command is deprecated. Job Trackers are managed automatically.

**dsetool checkcfs**

Scans a single Cassandra File System (CFS) file, or the whole CFS, for corrupted files.

**Restriction:** Command is supported only on nodes with analytics workloads.
Synopsis

dsetool checkcfs filepath|cfs:/// 

Table 58: 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>

cfs:///  Checks the default Cassandra File System (CFS).
filepath  Get details about a particular file that has been corrupted.
Examples

Scan the Cassandra File System (CFS) for corrupted files:

```
$ dsetool checkcfs cfs:///
```

Get details about a particular file that has been corrupted:

```
$ dsetool checkcfs /tmp/myhadoop/mapred/system/jobtracker.info
```

**dsetool core_indexing_status**

Retrieves the dynamic indexing status of a search index on a DSE Search node, and optionally displays the percent complete and an estimated completion time in milliseconds.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

**Synopsis**

```
dsetool core_indexing_status [keyspace_name.]table_name|--all [|--progress]
```

**Table 59: 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</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>[ -- ]</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>

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 all search indexes.
- **--progress**
  - Display the percent complete and an estimated completion time in milliseconds.

See [Verifying indexing status (page 624)](#).

### Examples

To view the indexing status for the local node:

```
dsetool core_indexing_status demo.health_data
```

The local node wiki.solr is currently indexing:

```
[demo.health_data]: INDEXING
```

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 -h 200.192.10.11 core_indexing_status --all
```

To view the indexing status with the progress and estimated time of completion:

```
dsetool core_indexing_status demo.health_data --progress
```
The results are displayed:

```
[demo.health_data]: INDEXING, 38% complete, ETA 452303 milliseconds (7 minutes 32 seconds)
```

**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 (page 539) for details on docValues.

**Note:** If one or more nodes fail to create the search index in distributed operations, an error message indicates the failing node or nodes. If it failed to create the search index immediately, issue the create again. If it failed to create on some nodes, issue a reload for those nodes to load the newly created search index.

**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)]
[recovery=(true|false]
[reindex=(true|false]
```

**Table 60: 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>
</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>

### 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.

### 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 533).

### recovery=(true | false)

Whether to delete and recreate the search index if it is not able to load due to corruption.

Valid values:
DataStax Enterprise tools

- true - If search index is unable to load, recover the index by deleting and recreating it.
- false - Default. No recovery.

**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.

Valid values:
- true - Reindexes the data. Accepts reads and keeps the current search index while the new index is building.
- false - Default. 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:

```
dsetool create_core demo.health_data generateResources=true
```

Override the default and reindex existing data, specify the `reindex=true` option:

```
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` are `rt: true`:

```
$ 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`:

```
$ dsetool create_core keyspace_name.table_name generateResources=true coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```
DataStax Enterprise tools

dsetool create_core demo.health_data generateResources=true
coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"

Use options in a YAML-formatted file

dsetool create_core demo.health_data
coreOptions="directory_factory_class:solr.EncryptedFSDirectoryFactory"

**dsetool createsystemkey**

Creates an encryption/decryption key for transparent data encryption (TDE). You can specify a file name to create a local key or KMIP options to create a remote key.

See [Transparent data encryption](#).

**Synopsis**

```sh
$ dsetool createsystemkey
[cipher_algorithm[/mode/padding]
[length] [key_name]
[-d filepath] [-k=kmip_groupname
[-t kmip_template] [-n namespace]]
```

**Table 61: 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 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>

### cipher_algorithm[/mode/padding]

DSE supports the following JCE cipher algorithms and corresponding length:

- 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 321)` 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)

### file_name

Unique file name for the generated system key file. Encryption key files can have any valid Unix name. If no name is specified, the default file name is `system_key`. The default key file name is not configurable. The location of the key is specified with `system_key_directory (page 321)` in `dse.yaml`.

### -k kmip_groupname

The name of the KMIP group that is defined in the `kmip_hosts (page 322)` section of `dse.yaml`.

### -t kmip_template

The key template on the specified KMIP provider.

### -n namespace
DataStax Enterprise tools

Namespace on the specified KMIP provider.

Examples

To create an on-server key file:

```bash
$ dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 system_key2
```

where `system_key2` is the unique file name for the generated on-server key file.

To create an off-server key file:

```bash
$ dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 system_key2 -kmip=group2
```

where `group2` is the key server group defined in the `kmip_hosts (page 322)` section of `dse.yaml`.

To create a local key file in a specific directory:

```bash
$ dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 -d /mydir
```

dse.yaml

The location of the `dse.yaml` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Type of Installation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/dse/conf/dse.yaml</td>
</tr>
</tbody>
</table>

**dsetool encryptconfigvalue**

Encrypts sensitive configuration information. This command takes no arguments and prompts for the value to encrypt.

Example

```bash
$ 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**

```bash
dsetool get_core_config keyspace_name.table_name [current=true|false]
```
Table 62: 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.

**current=true|false**
Optionally specify to view the current (active) configuration.
- true - Returns the active live search index config.
DataStax Enterprise tools

- 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
```

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>
    ...
  </requestHandler>
  <admin>
    <defaultQuery>*:*</defaultQuery>
  </admin>
</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

dsetool get_core_schema keyspace_name.table_name [current=true|false]

<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=&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>

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.
DataStax Enterprise tools

- 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>
<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>

Typing dsetool or dsetool help provides a listing of dsetool commands and parameters.
DataStax Enterprise tools

**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**

```
dsetool index_checks keyspace_name.table_name
[coreOptions=yamlFilepath] |
[coreOptionsInline=key1:value1#key2:value2#...]
--index_checks=true|false
--index_checks_stop=true|false
```

**Table 65: 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>[ -- ]</th>
<th>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.</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>

### 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 533).

### 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 533).

### --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:

```bash
dsetool index_checks demo.health_data
```

The LUKE handler information is displayed:

```bash
LUKE handler info: 
------------------
numDocs:0
maxDoc:0
deletedDocs:0
indexHeapUsageBytes:0
version:2
segmentCount:0
current:true
hasDeletions:false
lockFactory=org.apache.lucene.store.NativeFSLockFactory@5c94e0dd
segmentsFile:segments_1
segmentsFileSizeInBytes:71
userData: {}
```

### 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=key1:value1#key2:value2#...]
```

### Table 66: 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>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>

**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 533).

**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 533).
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 infer_solr_schema demo.health_data_udt
```

**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 67: 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>
<th>Syntax conventions</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>

#### Examples

**To view the status for all tables:**

```
dsetool inmemorystatus
```

The results for all tables are displayed:

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Memory to Lock:</td>
<td>3276MB</td>
</tr>
<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>
</tbody>
</table>

No MemoryOnlyStrategy tables found.

**To view the status for a specific table:**

```
dsetool inmemorystatus demo.health_data
```

### 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 68: 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>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>

**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 demo.health_data

The results show file name, encryption, disk usage, decrypted size, and encryption overhead:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Decrypted size</th>
<th>Encryption</th>
<th>Encryption overhead</th>
<th>Disk usage</th>
<th>Encryption overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>segments_1</td>
<td>N/A</td>
<td>N/A</td>
<td>7124 bytes</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>write.lock</td>
<td>N/A</td>
<td>N/A</td>
<td>3240 bytes</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

To list the index files in a specified directory:

$ dsetool list_index_files demo.health_data /My_data_dir

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 69: 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>

### 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` command.

<table>
<thead>
<tr>
<th>Start Token</th>
<th>End Token</th>
<th>Estimated Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>113427455640312821154458202477256070485</td>
<td>113427455640312821154458202477256070485</td>
<td>0</td>
</tr>
<tr>
<td>1234254427595624521227151664615147681247</td>
<td>11264</td>
<td></td>
</tr>
<tr>
<td>1324254427595624521227151664615147681247</td>
<td>11136</td>
<td></td>
</tr>
</tbody>
</table>
dsetool managekmip list

Verifies communication with the specified Key Management Interoperability Protocol (KMIP) server and lists the encryption/decryption keys on that server.

Synopsis

dsetool managekmip list kmip_group_name [namespace=key_namespace]

Table 70: 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>

**kmip_groupname**

The user-defined name of the KMIP group that is configured in the `kmip_hosts` (page 322) 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:

```bash
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>----</td>
</tr>
<tr>
<td>02-449</td>
</tr>
<tr>
<td>02-540</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 71: 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>
</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>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 322) 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:

```
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:

```
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**

```
dsetool managekmip revoke kmip_group_name kmip_key_id
```

<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>
</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>

### 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.

### Examples

To revoke a key to prevent decryption:

```
dsetool managekmip revoke kmipgrouptwo 02-540
```

### 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 73: 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>Syntax conventions</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</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 322) section of dse.yaml.

**kmip_key_id**

The key id on the KMIP provider.

**Examples**

To revoke a key to prevent decryption:

```
dsetool managekmip revoke kmipgrouptwo 02-540
```

After you revoke a key, you can destroy it:

```
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

dsetool node_health [--all]

Table 74: 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>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>
Run the operation on all nodes.

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 75: 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>
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 329) in dse.yaml.

See DSE Performance Service diagnostic table reference and Collecting histogram diagnostics.

Synopsis

dsetool perf subcommand values
### Table 76: 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>

**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 (page 331)`: false setting in `dse.yaml`.

**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
```

dse.yaml

The location of the dse.yaml file depends on the type of installation:
Package installations  
Installer-Services installations  
/etc/dse/dse.yaml  

Tarball installations  
Installer-No Services installations  
installation_location/resources/dse/conf/dse.yaml  

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 77: 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 (<strong>--</strong>). 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 (<strong>'</strong>) 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.

**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
```

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 78: 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 (<strong>[ ]</strong>) surround optional command arguments. Do not type the square brackets.</td>
</tr>
<tr>
<td>( )</td>
<td>Group. Parentheses (<strong>( )</strong>) 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>

**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 repaircfs**

Repairs the CFS file system from orphan blocks.
Synopsis

dsetool repaircfs [file_system]

**Restriction:** Command is supported only on nodes with analytics workloads.

### Table 79: 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>

**file_system**

A CFS file system other than the default CFS file system.
Examples

To repair the default CFS file system:

```
dsetool repaircfs
```

To repair a specific file system other than CFS:

```
dsetool repaircfs othercfs:/
```

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](page 570) 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 80: 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><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>' &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>

### 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.

Default: true

**Warning:** Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

### 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.

Valid values:

- true - Reindexes the data. Accepts reads and keeps the current search index while the new index is building.
- false - Default. Does not reindex the data. You can check and customize search index resources before indexing.

deleteAll

- 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 81: 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</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><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>

This command requires no input.

### Examples

```bash
dsetool ring
```

**dsetool sparkmaster cleanup**

Drops and recreates the Spark Master recovery table.

### Synopsis

```bash
dsetool sparkmaster cleanup [datacenter]
```
### Table 82: 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>

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 sparkmaster cleanup dc1

dsetool sparkworker restart

Manually restarts the Spark Worker on the selected node, without restarting the node.

Synopsis

```
dsetool sparkworker restart
```

Table 83: 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 accept no parameters.

Examples

```
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**

```
dsetool status
```

**Table 84: 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>
<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>
</tbody>
</table>
DataStax Enterprise tools

<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

dsetool status

dsetool stop_core_reindex

**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**

dsetool stop_core_reindex keyspace_name.table_name [timeout_min]

**Table 85: 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>'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 86: 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>

**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:

```bash
dsetool tieredtablestats
```

Output of command:

```bash
ks.tbl
```
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 87: 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>

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 SSL (internode) communications.

**dsetool unload_core**

Removes a search index. Supports DSE authentication with [-l *username* -p *password*].

To drop a search index from a table and delete all related data for the entire cluster, see 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 88: 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</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>

### 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 resources associated with the search index. For example, `solrconfig.xml` and `schema.xml`.

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 89: 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>

keyspace_name.table_name
DataStax Enterprise tools

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.

-\textbf{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.

-\textbf{c port}
  The DSE port on the remote node that contains the encryption configuration.

-\textbf{--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.

-\textbf{--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.

-\textbf{--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.

\textbf{dsetool write_resource}

Uploads the specified search index config or schema.

\textbf{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 \texttt{resource\_upload\_limit} (page 327) option in \texttt{dse.yaml}.

Supports DSE authentication with [-l username -p password].

\textbf{Synopsis}

```
dsetool write_resource keyspace_name.table_name name=res_filename file=path_to_file_to_upload
```
### Table 90: 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; &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 upload.

**file**

The file path of the file to upload.
Examples

To write the resource:

```
dsetool write_resource demo.health_data stopwords.xml
```

To specify the uploaded resource file and the path to the resource file:

```
dsetool write_resource demo.health_data name=ResourceFile.xml file=/myPath1/myPath2/schemaFile.xml
```

dse client-tool

The dse client-tool application connects an external client to a DataStax Enterprise (DSE) node and performs common utility tasks.

Synopsis

```
dse client-tool [connection_options]  
cassandra subcommand |  
configuration byos-export options file |  
configuration import file connection_options |  
spark subcommand
```

Table 91: 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>

Subcommands:

- Connection options (page 1083)
- Operation subcommands (page 1084)
- Configuration subcommands (page 1085)
  - # Exporting DSE cluster configuration (page 1085)
  - # Importing DSE cluster configuration (page 1086)
• **Spark subcommands** *(page 1087)*

To show the command line help for dse client-tool:

```bash
$ dse client-tool --help
```

To show the command line help for a dse client-tool subcommand:

```bash
$ dse client-tool help subcommand
```

**Connection options for dse client-tool**

Different sources of configuration properties are used to connect external clients to a DSE node: DSE configuration in `dse.yaml` and `cassandra.yaml`. Authentication credentials can be provided in several ways, see [Connecting to authentication enabled clusters](#). The dse client-tool subcommands use DSE Unified Authentication, like the Java and other language drivers, not JMX authentication like dsetool.

**Kerberos authentication** with dse client-tool is supported.

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 for CQL execution](#).

Override connection settings with these optional command arguments for:

```bash
$ dse client-tool connection_options
```

**Table 92: External client connection options**

<table>
<thead>
<tr>
<th>Connection options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--cipher-suites</code></td>
<td>A comma-separated list of SSL cipher suites for connection to DSE when SSL is enabled.</td>
</tr>
<tr>
<td><code>ssl_cipher_suites</code></td>
<td></td>
</tr>
<tr>
<td><code>--host address</code></td>
<td>The DSE host RPC broadcast address.</td>
</tr>
<tr>
<td>`--kerberos-enabled true</td>
<td>false`</td>
</tr>
<tr>
<td><code>--keystore-password</code></td>
<td>The keystore password for connection to DSE when SSL client authentication is enabled.</td>
</tr>
<tr>
<td><code>ssl_keystore_password</code></td>
<td></td>
</tr>
<tr>
<td><code>--keystore-path</code></td>
<td>The path to the keystore for connection to DSE when SSL client authentication is enabled.</td>
</tr>
<tr>
<td><code>ssl_keystore_path</code></td>
<td></td>
</tr>
<tr>
<td><code>--keystore-type</code></td>
<td>The keystore type for connection to DSE when SSL client authentication is enabled.</td>
</tr>
<tr>
<td><code>ssl_keystore_type</code></td>
<td></td>
</tr>
<tr>
<td><code>-p password</code></td>
<td>Password for your DSE account. Can use the DSE_PASSWORD environment variable.</td>
</tr>
<tr>
<td><code>--port port</code></td>
<td>The native protocol RPC connection port (Thrift).</td>
</tr>
<tr>
<td>Connection options</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--sasl-protocol-name</td>
<td>The SASL protocol name is the DSE service principal name.</td>
</tr>
<tr>
<td>sasl_protocol_name</td>
<td></td>
</tr>
<tr>
<td>--ssl-enabled</td>
<td>If true, enable SSL for connection to DSE.</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>--ssl-protocol</td>
<td>The SSL protocol for connection to DSE when SSL is enabled. For example, --ssl-protocol=ssl4.</td>
</tr>
<tr>
<td>ssl_protocol</td>
<td></td>
</tr>
<tr>
<td>-t delegation_token</td>
<td>When Kerberos is used, the delegation token to use for login. Can use the DSE_TOKEN environment variable.</td>
</tr>
<tr>
<td>--truststore-password</td>
<td>The truststore password for connection to DSE when SSL is enabled.</td>
</tr>
<tr>
<td>ssl_truststore_password</td>
<td></td>
</tr>
<tr>
<td>--truststore-path</td>
<td>The path to the truststore for connection to DSE when SSL is enabled.</td>
</tr>
<tr>
<td>ssl_truststore_path</td>
<td></td>
</tr>
<tr>
<td>--truststore-type</td>
<td>The truststore type for connection to DSE when SSL is enabled.</td>
</tr>
<tr>
<td>ssl_truststore_type</td>
<td></td>
</tr>
<tr>
<td>-u username</td>
<td>User name of a DSE authentication account. Can use the DSE_USERNAME environment variable.</td>
</tr>
<tr>
<td>--use-server-config</td>
<td>Read connection configuration from server YAML files (dse.yaml and cassandra.yaml) instead of reading them from other configuration files. Use only when the DSE installation against which the command is run is a running node.</td>
</tr>
</tbody>
</table>

Operation subcommands

Optional subcommands for general operations:

$ dse client-tool cassandra subcommand

<table>
<thead>
<tr>
<th>Subcommands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate-token</td>
<td>Generate delegation token (page 413) with the current user as the token renewer. The renewer is not specified and only DSE processes can renew a token. Use to access Kerberos DSE from non-Kerberos clusters. Requires Kerberos authentication mode.</td>
</tr>
<tr>
<td>generate-token --token-renewer username</td>
<td>Generate delegation token (page 413) with the specified user as the token renewer. The specified user can renew and cancel the token. Requires Kerberos authentication mode.</td>
</tr>
<tr>
<td>partitioner</td>
<td>Return the partitioner that is being used by the node.</td>
</tr>
<tr>
<td>renew-token token</td>
<td>Renew the specified token.</td>
</tr>
<tr>
<td>Subcommands</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cancel-token token</td>
<td>Cancel the specified digest authentication token. Requires Kerberos authentication mode.</td>
</tr>
</tbody>
</table>

### Configuration subcommands

The configuration subcommands manage external configurations on DSE client nodes.

```
dse client-tool configuration subcommand [options]
```

<table>
<thead>
<tr>
<th>Subcommands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>export file</td>
<td>On the remote node, export the node configuration to a JAR file. The configuration settings are retrieved from the running DSE node. See Running Spark commands against a remote cluster (page 384).</td>
</tr>
<tr>
<td>byos-export [options] file</td>
<td>Retrieve DSE cluster configuration and export it to a Spark properties file. See Exporting DSE cluster configuration (page 1085).</td>
</tr>
<tr>
<td>import file [options]</td>
<td>Set up DSE installation for integrated client applications to connect to an external running DSE cluster. See Importing DSE cluster configuration (page 1086).</td>
</tr>
</tbody>
</table>

### Exporting DSE cluster configuration

```
dse client-tool configuration byos-export options file
```

The configuration byos-export subcommand retrieves DSE cluster configuration and exports it to a Spark properties file: See the Spark documentation for file distribution options, including the --files parameter.

Store the exported configuration with these `options`:

<table>
<thead>
<tr>
<th>Subcommands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--default-properties</td>
<td>Merge the default Spark properties with the DataStax Enterprise Spark properties. For example:</td>
</tr>
<tr>
<td>path_to_existing_properties</td>
<td>$ dse client-tool configuration byos-export --default-properties /usr/lib/spark/conf/spark-defaults.conf /home/user1/.dse/byos.conf</td>
</tr>
<tr>
<td>--cfs-as-default</td>
<td>Set CFS (page 505) as the default file system. This option was introduced starting in DSE 5.1.6.</td>
</tr>
<tr>
<td>--export-credentials</td>
<td>Store current DataStax Enterprise user and password in the config file.</td>
</tr>
</tbody>
</table>
DataStax Enterprise tools

<table>
<thead>
<tr>
<th>Subcommands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--generate-token</td>
<td>Generate delegation token (page 413) with the current user as the token renewer. The renewer is not specified and only DSE processes can renew a token. Use to access Kerberos DSE from non-Kerberos clusters. Requires Kerberos authentication mode.</td>
</tr>
<tr>
<td>--set-keystore-path path</td>
<td>The path to the SSL keystore when SSL client authentication is enabled. All nodes must store the keystore in the same location.</td>
</tr>
<tr>
<td>--set-keystore-type type</td>
<td>The keystore type when SSL client authentication is enabled. If not specified, the default is JKS.</td>
</tr>
<tr>
<td>--set-keystore-password password</td>
<td>The keystore password for connection to the database when SSL client authentication is enabled.</td>
</tr>
<tr>
<td>--set-truststore-path path</td>
<td>The path to the truststore.</td>
</tr>
<tr>
<td>--set-truststore-type type</td>
<td>The truststore type. If not specified, the default is JKS.</td>
</tr>
<tr>
<td>--set-truststore-password password</td>
<td>The truststore password.</td>
</tr>
<tr>
<td>--token-renewer username</td>
<td>Generate delegation token (page 413) with the specified user as the token renewer. The specified user can renew and cancel the token. Requires Kerberos authentication mode.</td>
</tr>
</tbody>
</table>

Importing DSE cluster configuration

```
$ dse client-tool configuration import file connection_options
```

The configuration settings that were exported can be imported on an external, unconfigured node to allow DSE client applications to remotely access the running DSE cluster.

On the client node, run this command to generate configuration files and a cqlshrc file with the settings from the imported file. The configuration import command sets up DSE installation for integrated client applications to connect to an external running DSE cluster.

Import the node configuration from the specified JAR file with these connection_options:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--set-truststore-type type</td>
<td>The truststore type. If not specified, the default is JKS.</td>
</tr>
<tr>
<td>--set-truststore-path path</td>
<td>The path to the truststore.</td>
</tr>
<tr>
<td>--set-truststore-password password</td>
<td>The truststore password.</td>
</tr>
<tr>
<td>--set-keystore-type type</td>
<td>The keystore type when SSL client authentication is enabled. If not specified, the default is JKS.</td>
</tr>
<tr>
<td>Options</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>--set-keystore-path path</td>
<td>The path to the SSL keystore when SSL client authentication is enabled. All nodes must store the keystore in the same location.</td>
</tr>
<tr>
<td>--set-keystore-password password</td>
<td>The keystore password for connection to the database when SSL client authentication is enabled.</td>
</tr>
<tr>
<td>--cqlshrc file</td>
<td>Generate a cqlshrc file for the DSE client node. When a file is not specified, the default file is the ~/.cassandra/cqlshrc file.</td>
</tr>
<tr>
<td>--force</td>
<td>Force an overwrite of existing configuration files. By default, the import command fails if the configuration files already exist.</td>
</tr>
</tbody>
</table>

Spark subcommands

dse client-tool spark subcommand

Subcommands for integrated Spark operations:

<table>
<thead>
<tr>
<th>Subcommands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>master-address</td>
<td>Returns the current address of the Spark Master in the datacenter that you are connecting to as determined by the host address. The Spark Master address is returned as URI: dse://ip:port?connection.local_dc=dc_name;connection.host=cs_list_contactpoints. The connection.host=cs_list_contactpoints 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.</td>
</tr>
<tr>
<td>leader-address</td>
<td>Returns the IP address of the currently selected leader for the datacenter that the client tool connects to.</td>
</tr>
<tr>
<td>version</td>
<td>Returns the version of Spark that is bundled with DataStax Enterprise.</td>
</tr>
</tbody>
</table>
| sql-schema option | Exports the SQL table creation query with the following options:. With --exclude option, excluded tables
With --keyspace option, keyspaces
With --table option, tables
With --decimal option, hive 0.13+ decimal type parameters in form precision, scale
With --all option, all keyspaces |

DataStax Enterprise stress tools

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

<table>
<thead>
<tr>
<th>type</th>
<th>total ops,</th>
<th>op/s,</th>
<th>pk/s,</th>
<th>row/s,</th>
<th>mean,</th>
<th>med,</th>
<th>.95,</th>
<th>.99,</th>
<th>.999,</th>
<th>max,</th>
<th>time,</th>
<th>stderr,</th>
<th>errors,</th>
<th>gc: #,</th>
<th>max ms,</th>
<th>sum ms,</th>
<th>sdv ms,</th>
<th>mb</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>43148,</td>
<td>42991,</td>
<td>42991,</td>
<td>42991,</td>
<td>4.6,</td>
<td>1.5,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>106.1,</td>
<td>239.3,</td>
<td>255.4,</td>
<td>1.0,</td>
<td>0.00000,</td>
<td>0,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>49,</td>
<td>49,</td>
<td>612</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>98715,</td>
<td>43857,</td>
<td>43857,</td>
<td>43857,</td>
<td>4.6,</td>
<td>1.7,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>98.6,</td>
<td>204.6,</td>
<td>264.5,</td>
<td>2.3,</td>
<td>0.00705,</td>
<td>0,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>45,</td>
<td>45,</td>
<td>619</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>157777,</td>
<td>47283,</td>
<td>47283,</td>
<td>47283,</td>
<td>4.1,</td>
<td>1.4,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>70.6,</td>
<td>251.7,</td>
<td>286.3,</td>
<td>3.5,</td>
<td>0.02393,</td>
<td>0,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>59,</td>
<td>59,</td>
<td>611</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Results:
op rate                   : 46751 [WRITE:46751]
partition rate            : 46751 [WRITE:46751]
row rate                  : 46751 [WRITE:46751]
latency mean              : 4.3 [WRITE:4.3]
latenity median            : 1.3 [WRITE:1.3]
latenity 95th percentile   : 7.2 [WRITE:7.2]
latenity 99th percentile   : 60.5 [WRITE:60.5]
latenity 99.9th percentile : 223.2 [WRITE:223.2]
latenity max               : 503.1 [WRITE:503.1]
Total partitions          : 1000000 [WRITE:1000000]
Total errors              : 0 [WRITE:0]
total gc count            : 18
total gc mb               : 10742
total gc time (s)         : 1
avg gc time (ms)          : 73
stdev gc time (ms)        : 16
Total operation time      : 00:00:21

END

Table 93: Output of cassandra-stress

<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>Data</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</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>
<tr>
<td>mb</td>
<td>Size of the garbage collection in megabytes.</td>
</tr>
</tbody>
</table>
Operations

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.

Starting DataStax Enterprise as a service

Steps for starting the DataStax Enterprise (DSE) service when DataStax Enterprise was installed from RHEL or Debian packages and from the DataStax Installer with the Services option.

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 372) 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.

Note: DSE Graph can be enabled on any node in any datacenter. It no longer needs to be enabled on every node within a 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 371) 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>
<tr>
<td>Transactional-only, BYOH, or BYOS nodes</td>
<td>NODE_TYPES=0 or not present.</td>
</tr>
</tbody>
</table>

Table 94: Examples

<table>
<thead>
<tr>
<th>Node type</th>
<th>Settings</th>
</tr>
</thead>
</table>
| Spark Analytics node                          | SPARK_ENABLED=1  
SOLR_ENABLED=0  
GRAPH_ENABLED=0  
 or  
SPARK_ENABLED=1  
 Note: No entry is the same as disabling it. |
| Spark Analytics, DSE Graph, and DSE Search node | SPARK_ENABLED=1  
GRAPH_ENABLED=1  
SOLR_ENABLED=1 |
| BYOS (Bring Your Own Spark)                   | Set BYOS nodes as transactional nodes:  
All_NODE_TYPES=0 or not present. |
| DSE Graph and BYOS                           | GRAPH_ENABLED=1                                 |
Operations

<table>
<thead>
<tr>
<th>Node type</th>
<th>Settings</th>
</tr>
</thead>
</table>
| SearchAnalytics (page 372) nodes | SPARK_ENABLED=1
| An integrated DSE SearchAnalytics cluster allows analytics jobs to be performed using Search index filter syntax (page 572). | SOLR_ENABLED=1

Prerequisites: Be sure to read the Considerations for starting a cluster (page 1090).

You can also use OpsCenter to start and stop nodes.

1. If DataStax Enterprise is running, stop the node (page 1096).

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 DSE, see Troubleshooting starting and installing DataStax Enterprise.

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 ?
40725dc8-7843-43ae-9c98-7c532b1f517e rack1
```
For example, a running node in a normal state (UN) with DSE Analytics without
vnodes enabled shows:

<table>
<thead>
<tr>
<th>Datacenter: Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status=Up/Down</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>-- Address         Load   Owns    Host ID</td>
</tr>
<tr>
<td>Token                 Rack</td>
</tr>
<tr>
<td>UN 172.16.222.136 103.24 KB ? 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 or from the DataStax Installer with the No Services option.

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 372) 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.

Note: DSE Graph can be enabled on any node in any datacenter. It no longer needs to be enabled on every node within a 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 371) 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 95: Starting examples

<table>
<thead>
<tr>
<th>Node type</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>From the installation_location:</td>
<td></td>
</tr>
<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></td>
</tr>
<tr>
<td>Spark nodes run in separate Spark cluster from a vendor other than DataStax.</td>
<td>$ bin/dse cassandra</td>
</tr>
<tr>
<td>DSE Graph and BYOS</td>
<td>$ bin/dse cassandra -g</td>
</tr>
<tr>
<td>SearchAnalytics (page 372) nodes</td>
<td></td>
</tr>
<tr>
<td>An integrated DSE SearchAnalytics datacenter allows analytics jobs to be performed using search queries (page 572).</td>
<td>$ bin/dse cassandra -k -s</td>
</tr>
</tbody>
</table>

Prerequisites: Be sure to read the Considerations for starting a cluster (page 1093).

You can also use OpsCenter to start and stop nodes.

1. If DataStax Enterprise is running, stop the node (page 1096).
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_location` is either:

- `/usr/share/dse`
- the directory where you installed DataStax Enterprise.

**Note:** If DSE has problems starting DSE, see Troubleshooting starting and installing DataStax Enterprise.

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Load</th>
<th>Tokens</th>
<th>Owns</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>82.43 KB</td>
<td>128</td>
<td>?</td>
<td>40725dc8-7843-43ae-9c98-7c532b1f517e 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

<table>
<thead>
<tr>
<th>Address</th>
<th>Load</th>
<th>Owns</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>172.16.222.136</td>
<td>103.24 KB</td>
<td>?</td>
<td>3c1d0657-0990-4f78-a3c0-3e0c37fc3a06 rack1</td>
</tr>
</tbody>
</table>
```

### Stopping a DataStax Enterprise node

To speed up the restart process, before stopping the dse service, run `nodetool drain`. 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:

$ 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:

$ bin/dse cassandra-stop  ## Use sudo if needed

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.

$ ps auwx | grep dse

$ bin/dse cassandra-stop -p PID  ## Use sudo if needed

**Note:** If you have trouble, see Troubleshooting DataStax Enterprise.

### Clearing the data from DataStax Enterprise

Remove all data from any type of installation.

Package or a DataStax Installer-Services installation

To clear the data from the **default** directories:

1. After **Stop (page 1095)** the service.

2. Run the following command:

   $ sudo rm -rf /var/lib/cassandra/*  ## Remove all data
   $ sudo rm -rf /var/lib/cassandra/data/*  ## Remove only the data directories

Tarball or DataStax Installer-No Services installation

To clear all data from the **default** directories:
1. **Stop (page 1090)** the DataStax Enterprise process.

2. Remove the data:

   ```
   $ cd installation_location
   $ sudo rm -rf data/* commitlog/* saved_caches/* hints/* ## Remove all data
   $ sudo rm -rf data/* ## Remove only the data directories
   ```

   where the `installation_location` is either:
   - `/var/lib/cassandra/data`
   - The directory where you installed DataStax Enterprise.
DataStax Studio

About DataStax Studio

DataStax Studio is an interactive tool for CQL (Cassandra Query Language) and DSE Graph:

- For CQL, it provides the ability to visually create and navigate database objects, create complex queries, and tune CQL queries. Studio includes an intelligent CQL editor that provides syntax highlighting, validation, intelligent code completion, configuration options, and query profiling.
- For DSE Graph, it allows you to explore and view large datasets. It provides an intuitive interface for developers and analysts to collaborate and test theories by mixing code, documentation, and visualizations for query results and profiles into self-documenting notebooks.

Notebooks combine runnable Gremlin code and markdown text in a rich interactive environment. The code is written in the Gremlin graph traversal language. This code is executed by the Gremlin Server that is a part of DataStax Enterprise Graph component. Markdown is a simple language for creating human-readable plain text documents that can be displayed.

Upgrading DataStax Studio

The internal format for Studio settings and notebooks may change when a new version is released. While newly-created notebooks might not work with older versions of Studio, notebooks created in older versions of Studio can be used with new versions. To be on the safe side, you should always back up your older notebooks before opening them in a newer version of Studio (in case you want to downgrade your version of Studio and reopen these notebooks).

DataStax Studio release notes

Studio release notes cover enhancements, known issues, and resolved issues.

For earlier versions, see Studio 1.0.1 and Studio 1.0.2.

**Studio 2.0.0**

Release notes for DataStax Studio

**Important:** DataStax recommends the latest release.

- 2.0.0 Changes and enhancements *(page 1099)*
- 2.0.0 Resolved issues *(page 1099)*
2.0.0 Changes and enhancements

- Code cells in notebooks support CQL code
- DataStax Enterprise Graph improvements
  - Effective schema is scoped per HTTP session
- Changes to how Notebooks are stored. Notebooks are event-sourced aggregates. Notebooks are stored across multiple files located in the $userdata/eventlog and $userdata/snapshots directories (page 1124). (STUDIO-735, STUDIO-1545)
- The Graph field moved from Connection to Notebook. Existing Graph values are moved to each related Notebook. (STUDIO-1053)
- Statement line numbers are reported for failed executions. (STUDIO-1169)
- Editor improvements include electric pairing for matching delimiters: parentheses ( ), square brackets [ ], and single quotation marks ’ ’. (STUDIO-1091)
- Configure edge thickness in graph controls as a static value for all edges in the graph. (STUDIO-959)
- Notebook improvements include support to reset result window and clear previous result. (STUDIO-437)

2.0.0 Resolved issues

- RPC call failures are not adequately managed. (STUDIO-642, STUDIO-829)
- Chart views, including pie, bar, and line, grow in size for each cell rerun. (STUDIO-860)
- Property key is not truncated in the table header. Hover should show full name. (STUDIO-870).
- Edit Connection form: FQDN starting with number does not validate. (STUDIO-934)
- In the Edit Connection form, trailing whitespace is not ignored and causes error message: "Invalid address, address must not be empty". (STUDIO-935)
- When a gremlin traversal steps performs a scan, it should be highlighted and called out. (STUDIO-949)
- Opening schema display does not work with certain schemas. (STUDIO-981)
- Numeric values of text properties like schema.propertyKey('age').Text().create() are not handled correctly. Empty/null values are not ignored. (STUDIO-1057)
- Edit Connection form: graph name is not validated. (STUDIO-1182)
- Profiling query results in a UI freeze. (STUDIO-1192)

Getting started with DataStax Studio

Installing and running DataStax Studio 2.0

Prerequisites:

- DataStax Enterprise 5.1 installed (page 222), configured, and running.
- A supported browser.
- All DataStax Enterprise 5.1 prerequisites (page 222).
DataStax Studio

- Windows platforms: Windows 7 and 10, Java 8.

1. Download DataStax Studio for your platform.

   **Note:** If you installed DSE 5.1 using the DataStax Installer (page 222) and selected Developer Related Tools, Studio has already been installed in the DataStax Enterprise installation directory.

2. Unpack Studio:
   - Linux:
     
     $ tar zvxf datastax-studio-2.0.0.tar.gz
   - Windows:
     a. From File Explorer, right-click datastax-studio-2.0.0.zip.
     b. Enter the destination folder for Studio.
     c. Click Extract.

3. To start Studio, run the Studio Server shell script:
   - Linux:
     
     $ cd DSE_installation_directory/datastax-studio-2.0.0
     $ ./bin/server.sh
     $ ./bin/server.sh

     **Tip:** To start Studio in the background, add an ampersand (&) at the end of the command:

     $ ./bin/server.sh &
   - Windows:
     
     C:/ DSE_installation_directory/datastax-studio-2.0.0/bin \server.bat

     Your result will look similar to:

     Studio is now running at: http://127.0.0.1:9091

4. To open DataStax Studio, by entering the URL in your browser:

   http://URI_running_DSE:9091/
   - For DSE running on localhost, URI_running_DSE is localhost.
For DSE on another machine, `URI_running_DSE` is the hostname or IP address for the remote machine.

What’s next: Use notebooks to get started.

DataStax Studio 2.0 has revisions of these notebooks:

- Working with Graph v2
- DSE Graph QuickStart v2

A new notebook:

- Working with CQL

These notebooks include helpful information on using Studio and examples of using CQL and DSE Graph in Studio.

**Starting and stopping DataStax Studio**

Steps to start and stop Studio.

1. To start Studio, run the Studio Server shell script:

   - Linux:
     
     ```
     $ cd DSE_installation_directory/dostax-studio-2.0.0
     $ ./bin/server.sh
     $ ./bin/server.sh
     ```

DSE 5.1 Developer Guide (Previous version)
Tip: To start Studio in the background, add an ampersand (&) at the end of
the command:

```
$ ./bin/server.sh &
```

- Windows:
  ```
  C:/ DSE_installation_directory\datastax-studio-2.0.0\bin \server.bat
  ```

Your result will look similar to:

```
Studio is now running at: http://127.0.0.1:9091
```

2. To stop Studio:

```
$ pkill -f studio
```

Creating a simple notebook in DataStax Studio

A notebook is a document that contains text and runnable code. A notebook consists of one
or more **cells**.

**Prerequisites:**

- DSE 5.1 installed *(page 222)*, configured, and running.
- If using DSE Graph *(page 651)*, it must be configured, and running.
- DataStax Studio installed *(page 1099)* and running.

To create a new notebook and add some text.

   The DataStax Studio opens in the **Notebooks page**.

2. Select **Add Notebook** (the plus icon (in a square) to the left of the existing notebooks).
   The **Create Notebook** dialog displays.

3. Add information to the fields and **Create**.

   **Name**
   My Notebook

   **Select a connection**
4. **Select Create.**

   The notebook displays with a single empty (default) cell in **CQL** edit mode. This can be changed to **Gremlin** or **Markdown** depending on the desired mode.

The following steps show a Markdown example:

5. In the default cell, change the drop-down to **Markdown**.

6. Add some verbiage to the default cell:

   ```
   My Notebook
   =============
   Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut
   consectetur lectus sit amet erat tempor ornare. Fusce sagittis,
   mauris eu maximus pellentesque, odio purus accumsan lacus, eu
   accumsan ex nisi eu metus.
   ```

7. **Select Run Cell** to render the marked down text.
8. Each notebook has only one connection. Create a connection (page 1104) to a DSE cluster.

Creating a connection in DataStax Studio

Prerequisites:

- DSE 5.1 installed (page 222), configured, and running.
- DataStax Studio installed (page 1099) and running.
- If using DSE Graph (page 651), it must be configured and running.

A connection to a DSE cluster is required.

Each notebook has only one connection. Each connection can have multiple notebooks.

Tip: You can configure more than one host in a Studio connection. Hosts initialize the Cassandra driver connection, so more than one host provides redundancy and failover protection.

A notebook persists its data to a DSE cluster.

**DSE Graph**

The connection identifies a particular graph.

For notebooks using DSE Graph, the connection must be changed to use a different graph.

**CQL**

The connection identifies a database instance.

For notebooks using CQL, the keyspace and table can be changed within the notebook.

To create a connection from a notebook to a DSE cluster:

1. Browse to the URL for the Studio installation.
2. In the menu (#), select the Connections menu item.

   The Connections page in DataStax Studio (page 1129) displays.

3. Create the connection from a notebook to a DSE cluster:
   a. Select Add connection (the plus icon in the top center of the page).
      The Create Connection dialog appears.
   
   b. Enter the connection information:
      
      Name
      Name of the connection.

      Host/IP (comma delimited)
      The host names or IP addresses of the DSE cluster to connect to.
      Default: localhost.

      Username
      Optional. DSE username for logging in.

      Password
      Optional. DSE password for logging in.

      Port
      IP connection port of the DSE cluster. Default: 9042.

      This example connects to a single-node DSE cluster on the local host using the default port.

      Name
      My First Connection

      Host/IP
      127.0.0.1

      Port
      9042

   c. Select Save.

4. Click Test to verify the connection works.

5. To configure a secure encrypted connection, select the Use SSL check box.
   The Truststore and Keystore fields display. See Using SSL connections in DataStax Studio (page 1106).

6. Click Save.
Using SSL connections in DataStax Studio

Prerequisites:

- SSL must be configured and working on your cluster. See Configuring SSL for details on:
  - Client-to-node encryption
  - Node-to-node encryption
  - Preparing server certificates

DataStax recommends using certificates signed by a CA. See Setting up SSL certificates.

- A truststore is required for server verification. The truststore or CA certificate can be shared between all DSE servers and clients. For DataStax Studio, the public key certificate from the CA must be stored in a local truststore file.
- A keystore is required for client verification.
- Java Cryptography Extension (JCE) Unlimited Strength Policy files is required to ensure support for all encryption algorithms.

1. Install the Java Cryptography Extension (JCE) on your client system. See Installing JCE.

2. Download Java 8 for DataStax Enterprise.
   
   Installation directory (jre lib/security):
   - Linux: /usr/lib/jvm/jdk1.major.minor_update/jre/lib/security
   - Mac OS X: /Library/Java/JavaVirtualMachines/jdk1.major.minor_update/Contents/Home/jre/lib/security
   - Windows: C:\Program Files\Java\jre7\lib\security

   Extract the downloaded file and copy the contents of UnlimitedJCEPolicy directory to the jre/lib/security directory.

3. To perform server verification, the client needs to have the public key certificate of each node in the cluster stored in a local truststore file.
   
   a. In the menu (#), select Connections to open the Browse Connections page.

   b. Click + to add a connection or click to edit an existing connection.
      
      The Create Connection dialog displays.

   c. Select the Use SSL check box to show the Truststore and Keystore fields.

   d. For server verification, add the paths to the local truststore file and the truststore password.
The public key certificate from the CA must be stored in a local truststore file.

4. For client verification, add the paths to the keystore and the keystore password. The keystore path is to the Java keystore (JKS). The keystore contains the private key.

5. Select Test to test your connection information. The Connected Successfully message verifies successful connection to DataStax Enterprise nodes.

6. Click Save

7. Restart the Studio server for the JCE policies and other connection changes to take effect.

Creating a graph in a Studio notebook

Creating vertices and edges in DataStax Studio

Prerequisites:
- DSE 5.1 installed (page 222), configured, and running.
- DataStax Studio installed (page 1099) and running.
- A connection (page 1104) from Studio to a DSE cluster.
- An existing notebook (page 1102).

Add Gremlin code to a notebook to create a simple two vertex, one edge graph.

1. If necessary, start DataStax Studio and open the notebook you previously created (page 1102).

When you create a notebook, an empty graph instance is created and named after the value in the graph name field in the notebook's connection. A local variable, g, is defined automatically and bound to that graph.

2. Add a cell to the notebook.

3. Ensure that Gremlin is selected in the menu for the notebook cell editor mode:

4. Add the code to the cell to create some vertices and edges for the graph.

```java
schema.config().option('graph.schema_mode').set('Development')

Vertex firstVertex = graph.addVertex(label, 'user', 'id', 1, 'name', 'Jo Dowe', 'role', 'customer')
```
Vertex secondVertex = graph.addVertex(label,'product', 'id', 2, 'name', 'fountain pen')
firstVertex.addEdge('bought', secondVertex)
g.V()

These lines of code, put DSE Graph in development mode, create two vertices, and connect them with a single edge.

5. Select Run Cell to execute the code.

6. Select Graph in Display toolbar. (By default, the Table view is displayed.)

7. Hover your mouse over a vertex to display its properties.

What’s next: See the DSE Graph documentation (page 651).

Creating a schema in DataStax Studio

Prerequisites:
To create a schema, execute Gremlin code in a Gremlin cell in your notebook. The schema used in this example represents users and products and their relationships to one another.

1. Add a code cell to your notebook.

2. Write code to create a schema:

   a. Drop the schema if it exists.

   ```python
   schema.clear()
   ```

   When creating a notebook, some variables are created by default (if none exist) for you to access in code cells. For example, a graph (graph), a graph traversal (g), and a schema (schema). The graph variable is set to an empty graph with the name that you previously provided in the notebook's connection.

   a. Create the property keys for the schema.

   ```python
   schema.propertyKey('id').Int().create()
   schema.propertyKey('name').Text().create()
   schema.propertyKey('role').Text().create()
   ```

   The id and name property keys are used by both the user and the product vertices. The role property key is only used by the user vertex.

   b. Create the vertex labels for user and product.

   ```python
   schema.vertexLabel('user').create()
   schema.vertexLabel('product').create()
   ```

   c. Create the edge label for use between the user and product vertices.

   ```python
   schema.edgeLabel('bought').create()
   ```

3. Execute the code by selecting Run Cell.

   You have created the schema for the example.

4. View the resulting schema by selecting Schema in the upper-right-hand corner of the notebook.
The schema you created in this task is based on the following data model:

```
user                   bought                product
| id | name  | role      | | id | name         |
|----|-------|-----------| |----|--------------|
```

### Adding data to a graph in DataStax Studio

**Prerequisites:**
- Create the schema *(page 1108).*

To add data to a graph from a cell in your notebook:

1. Add four more vertices and two other edges:

   ```python
   user = graph.addVertex(label, 'user', 'id', 2, 'name', 'Jay Quest', 'role', 'customer')
   product = graph.addVertex(label, 'product', 'id', 3, 'name', 'digital camera')
   user.addEdge('bought', product)
   user = graph.addVertex(label, 'user', 'id', 3, 'name', 'Bartholomew Hicks', 'role', 'employee')
   product = graph.addVertex(label, 'product', 'id', 4, 'name', 'eraser')
   schema.edgeLabel('manufactured').create()
   user.addEdge('manufactured', product)
   ```
2. Create an edge between two existing vertices:

   a. Add this code to a new Gremlin cell:

   ```java
   user = g.V().has('user', 'name', 'Jo Dowe').next()
   product = g.V().has('product', 'name', 'eraser').next()
   user.addEdge('bought', product).next()
   g.V()
   ```

   b. Switch the cell to display a graph.

c. Select the Jo Dowe user vertex.
Interact with data using CQL in DataStax Studio

Creating keyspaces and tables with DataStax Studio

**Prerequisites:**
- DSE 5.1 installed (page 222), configured, and running.
- DataStax Studio installed (page 1099) and running.
- A connection (page 1104) from Studio to a DSE cluster.
- An existing notebook (page 1102).

Add CQL code to a notebook to create a simple keyspace and two tables.

1. Open a notebook.

2. Create a DSE keyspace using the CQL command `CREATE KEYSPACE`:

   ```cql
   CREATE KEYSPACE inventory WITH REPLICAATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 }; 
   ```

   a. Select CQL select as the language.

   b. No keyspace is selected, as none currently exists.

   c. Click the button labelled `CL.ONE` to execute the CQL code.
3. Create two tables using the CQL command **CREATE TABLE**:

```cql
USE inventory;  // or select the default keyspace above, and comment out this line

CREATE TABLE product_info {
    product_id uuid,
    product_name varchar,
    PRIMARY KEY (product_id)
};

CREATE TABLE buyer_info {
    buyer_id uuid,
    buyer_name varchar,
    PRIMARY KEY (buyer_id)
};
```

The keyspace can be configured using the Keyspace pull-down menu in the cell, or with the CQL command **USE**.

4. Use the CQL shell command **DESCRIBE KEYSPACE** to display information about the keyspace that you created:

```cql
DESCRIBE KEYSPACE inventory;
```
Inserting data using CQL in DataStax Studio

Prerequisites:

- Create the schema (page 1108).

To add data to a table from a cell in your notebook:

1. Use the CQL command `INSERT` to add data to the tables:

```
USE inventory;

INSERT INTO product_info (product_id, product_name) VALUES
(99051fe9-6a9c-46c2-b949-38ef78858dd0, 'Coffee mug');
INSERT INTO product_info (product_id, product_name) VALUES
(b3a76c6b-7c7f-4af6-964f-803a9283c401, 'Ethiopian coffee');
INSERT INTO product_info (product_id, product_name) VALUES
(0c3f7e87-f6b6-41d2-9668-2b64d117102c, 'Half and half');
```
2. To display the inserted data, use the CQL command **SELECT**:  

```cql
SELECT * FROM inventory.product_info;
```

### What's next:
The notebook tutorial *Working with CQL* is installed with Studio. For more information about DataStax Enterprise CQL, see [Accessing data using CQL](#).

## Using notebooks in DataStax Studio

A notebook persists its data to a DSE cluster.

### DSE Graph

The connection identifies a particular graph.

For notebooks using DSE Graph, the connection must be changed to use a different graph.

### CQL

The connection identifies a database instance.
For notebooks using CQL, the keyspace and table can be changed within the notebook.

**Listing notebooks using Notebook Manager**

The Notebook Manager lists all notebooks, starred notebooks, and supports filtering by searching.

Connect to the Notebook Manager page in a web browser using the following URL:

To return to the Notebook Manager from a Notebook, select **Notebooks** from the menu (>).

**GUI**

+ To add a notebook, by click the **Add Notebook** button.

**Defining run behavior in DataStax Studio**

With results visualization and profiling capability, Studio serves as debugging tool by executing code in a way that reproduces the settings used in their applications. When using Studio to interact with DSE, you are able to execute code written in CQL or Gremlin. Each language has its own set of configuration options that determine execution behavior.

Execution configurations provide different execution setups by passing a set of options that customize the run execution. Execution configurations are persistent so you can reuse them.

Standard run configurations are provided:
<table>
<thead>
<tr>
<th>Cell type</th>
<th>Execution configuration</th>
<th>Settings</th>
</tr>
</thead>
</table>
| CQL | CL.ONE | • Consistency level: ONE  
• Timeout (MS): 10000  
• Max Results: 5000  
• CQL Tracing: Disabled |
| CQL | CL.Quorum | • Consistency level: QUORUM  
• Timeout (MS): 10000  
• Max Results: 5000  
• CQL Tracing: Disabled |
| CQL | CL.ALL | • Consistency level: ALL  
• Timeout (MS): 10000  
• Max Results: 5000  
• CQL Tracing: Disabled |
| CQL | CL.ONE.TRACE | • Consistency level: ONE  
• Timeout (MS): 10000  
• Max Results: 5000  
• CQL Tracing: Enabled |
| Gremlin | Transactional | Execute statement using real-time OLTP (online transaction processing) engine, traversal source |
| Gremlin | Spark | Execute statement using OLAP (online analytical processing) engine, traversal source |

You can create a custom CQL execution configuration by adjusting these settings for your environment:

- Consistency level: ANY, ONE, TWO, THREE, QUORUM, ALL, LOCAL_QUORUM, EACH QUORUM, SERIAL, LOCAL_SERIAL, LOCAL_ONE
- Timeout (MS): milliseconds
- Max Results: limit_results
- CQL Tracing: Enabled or Disabled

1. In the top right of a code cell, hover over the play button # and then click # to list the existing configurations.

**Tip:**

- In CQL cells, the default execution configuration is CL.ONE so the play button is CL.ONE #.
• In Gremlin cells, the default execution configuration is Executive using real-time (transactional) engine so the play button is **Real-time #**.

2. Select **Manage Configurations**.

3. Select an existing configuration to view settings.

4. **For CQL cells only**: Click **+ Add New Configuration** to create an execution configuration.
   
   a. Enter a self-describing **Name**.
   
   b. Select a **Consistency Level**.
   
   c. Enter a **Timeout** value in milliseconds.
   
   d. Enter **Max Results**.
   
   e. Optionally select the **CQL Tracing** check box.

5. Click **Save**.

   New execution configurations are available to be selected from any CQL notebook cell.

**Notebook cells in DataStax Studio**

Notebook cells contain markdown text, Gremlin code, or CQL.

Creating connections

To create connections to DSE clusters, select the **Connections** menu.

<table>
<thead>
<tr>
<th>Widget</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>📈</td>
<td>View Details.</td>
</tr>
<tr>
<td>⬤</td>
<td>View Schema.</td>
</tr>
<tr>
<td>Widget</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><img src="image" alt="Add Cell" /></td>
<td><strong>Add Cell</strong> adds another cell before or after the current cell.</td>
</tr>
</tbody>
</table>

**Cells**

<table>
<thead>
<tr>
<th>Widget</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Run Cell" /></td>
<td><strong>Run Cell</strong> displays the markdown or runs the Gremlin code and displays the results.</td>
</tr>
<tr>
<td><img src="image" alt="Hide code editor" /></td>
<td><strong>Hide code editor</strong> hides the code editor.</td>
</tr>
<tr>
<td><img src="image" alt="Disable editor validations / Enable editor validations" /></td>
<td><strong>Disable editor validations / Enable editor validations</strong> toggle.</td>
</tr>
<tr>
<td><img src="image" alt="More actions" /></td>
<td><strong>More actions</strong> displays more menu items for the cell.</td>
</tr>
</tbody>
</table>

**Gremlin results display**

The results returned by the last Gremlin statement executed in a cell can be displayed in a number of ways.

<table>
<thead>
<tr>
<th>Widget</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Raw" /></td>
<td><strong>Raw</strong></td>
</tr>
<tr>
<td><img src="image" alt="Table" /></td>
<td><strong>Table</strong></td>
</tr>
<tr>
<td><img src="image" alt="Pie chart" /></td>
<td><strong>Pie chart</strong></td>
</tr>
<tr>
<td><img src="image" alt="Bar" /></td>
<td><strong>Bar</strong></td>
</tr>
<tr>
<td><img src="image" alt="Line" /></td>
<td><strong>Line</strong></td>
</tr>
<tr>
<td><img src="image" alt="Area" /></td>
<td><strong>Area</strong></td>
</tr>
<tr>
<td><img src="image" alt="Scatter" /></td>
<td><strong>Scatter</strong></td>
</tr>
<tr>
<td><img src="image" alt="Graph" /></td>
<td><strong>Graph</strong> displays vertices and edges.</td>
</tr>
<tr>
<td><img src="image" alt="Settings" /></td>
<td><strong>Settings</strong></td>
</tr>
</tbody>
</table>

**Graph** provides a contextual view of a result by automatically populating the entire sub-graph shared between the result's vertices and edges. The Gremlin result is examined:
• All vertices emitted and all vertices touched by emitted edges are aggregated into the set of working vertices.
• All edges that connect this set of vertices are then populated.

Note: This sub-graph result populates the Graph. However, the Gremlin result populates all other views (such as, raw, table, charts).

**Notebook code editor in DataStax Studio**

The Notebook code editor supports a subset of the Apache Groovy™ programming language.

A notebook connects to a DSE cluster based on the connection information. Each notebook has only one connection. One piece of information connection specifies is its Graph Name.

The code editor provides pre-defined alias variables:

- `graph`  The variable that refers to the graph.
- `g`  The traversal source associated with the graph for OLTP graph traversals.
- `a`  The traversal source associated with the graph for OLAP graph traversals.

A drop-down list shows for content-assist *(page 1120)* on a blank line in the editor.

```
graph
```

The code editor supports inline code validation. If some code is invalid or uses Groovy syntax that the editor does not support, the invalid code is displayed with a red underlining.

Content assistance

The code editor provides content assistance *(Ctrl+Space)*.
Content-assistance proposes content based on context:

- Methods
- Variables

Press **Return** to select the highlighted choice from the assistance list.

You can filter proposals by typing enough characters to select a line in the pull-down menu.

Use **Shift-Tab** to return to the beginning of a line.

**Code validation**

The notebook editor performs validation as code is entered into the cell. You can turn off validation on a per-cell basis. To turn off validation in a cell, move the slider to Disable editor validations.

The notebook editor supports these validations:

**Groovy syntax**

The code in a cell is executed within DSE Graph as Groovy. The notebook editor adds enough Groovy syntax support for you to craft your Gremlin statements.

**Type-checking**
The code in a cell is also checked for type. If you try to call a method on an object of the wrong type or pass a parameter of an invalid type, a validation error is displayed.

```groovy
// Example 1

// This method is undefined for the type GraphTraversal<Vertex, Vertex>
g.V().hasPropertyKey()
```

```groovy
// Example 2

// Type mismatch: cannot convert from int to String

g.V().has(123, 'x', 'y')
```

**Domain-specific**

Common errors in code are pointed out when possible.

```groovy
// Example 3

//Traversal has not been iterated; call next(), iterate() or forEachRemaining()
g.V()
```

The code editor cells in a notebook have no Gremlin session scope context and rely on an implicit scope based on their order. Validation occurs from the top cell down in the order in the notebook. If you execute code out of order, validation errors can occur even if the code executes successfully.
Groovy language support

Supported:

- variable declarations in Groovy style:
  
  ```
  # def foo
  # def SomeType foo
  # SomeType foo
  # Shell-style variables
  ```

- method invocations
- optional semicolons to complete statements
- generics
- strings in both forms:
  
  ```
  # '123'
  # "123"
  ```

- for loops: basic and for in syntax
- while loops
- try-catch-finally: a type is required for catching the exception
- if-else statements
- switch statements
- casting in both styles: (SomeType) foo and foo as SomeType
- list literals

Unsupported Groovy language

- closures
- multi-variable assignment and other advanced variable assignments like object deconstruction
- import statements explicitly disabled
- map literals or other complex type literals that are not supported by Java
- try-catch-finally: multi-catch is not supported

Notebook information

Selecting Information displays:

- Connection being used for the notebook.
- Last update to the notebook.
- Version of Studio JAR file being run.
Notebook schema

Selecting Information displays the current graph schema for the notebook.

In addition to visual schema representation, the code-assist feature also provides schema-assist proposals.

That's a lot of proposals. Let's break them down:

- The editor is intelligent enough to know that this is a Vertex-based traversal (because of type inference) and presents only the schema proposals that are relevant for vertex properties and keys.
- Because there are multiple has methods, Studio proposes possibilities for all of these variations.
- The editor concretely makes a proposal for each possible property key.

User data for DataStax Studio notebooks

Studio 2 notebooks are stored across multiple files located in the $userdata/eventlog and $userdata/snapshots directories.
Snapshots are periodically created for notebooks that are active (recently edited or recently executed cells). These snapshots are stored in the $userdata/snapshots/studio/uuid directory which contains snapshot files (snr-n, where the highest snr-n file is the latest snapshot). These snapshots are single file representations of the notebook but might not be fully up to date.

If you have previously stored Studio 1.0.x notebooks in a version control system, the format change described above prevents using the same storage method. You can, however, use one of the snapshots. Snapshots are single file representations of the notebook, but might not be fully up to date.

By default, snapshots are created every five minutes for a notebook that is being actively edited. This time interval can be configured by adding an option in the configuration.yaml file:

```yaml
userData:
  snapshotSaveIntervalInSeconds: 300
```

**Data migration**

Existing notebooks from Studio 1.0.x are migrated to the new format in Studio 2.0.x. Studio creates a backup of 1.0 notebooks in the $userdata/1_x_backup directory during the upgrade migration. If the migration is successful, the $userdata/notebooks/admin directory is deleted.

Studio does not delete your previous notebook files if an error occurs while migrating them to the new format.

**Configuring DataStax Studio**

### Basic configuration options in DataStax Studio

The primary configuration file for Studio is `dse-studio-install-dir/configuration.yaml`.

A sample configuration file in XML format:

```xml
# Maximum number of items returned per cell execution. Additional items will be truncated.
# Unit: count / number of items
<resultSizeLimit>1000</resultSizeLimit>

# Maximum size of a cell result. If a cell result exceeds this size then the cell execution will fail.
# Unit: bytes
<maxResultSizeBytes>524288</maxResultSizeBytes>

# Cell execution timeout. A value of 0 indicates no timeout and will depend on the DSE server timeouts configured in dse.yaml.
# Unit: milliseconds
<cellExecutionTimeout>0</cellExecutionTimeout>
```
executionTimeoutMs: 0

# TraversalSource mapping for real-time and analytic execution engines. These values correspond to the traversal sources
# configured in dse.yaml.
#
traversalSources:
  realTimeTraversalSource: g
  analyticTraversalSource: a

# This refers to the datastax-studio server
server:
  httpPort: 9091
  # WARNING!!! Changing this setting from the default(localhost) could pose a security risk as other users on external machines could then # gain access to notebooks and the clusters those are connected to.
  # Studio is designed to be used as a desktop application, but if deploying centrally you should be aware of the potential security risks.
  # Please visit http://docs.datastax.com/en/latest-studio/studio/reference/configuration.html for information on configuration
  httpBindAddress: localhost

logging:
  fileName: studio.log
  maxLogFileSize: 250 MB
  maxFiles: 10
  directory: ./logs

# User data, where we store data specific to a user
# in the application. For the short-term this is the file system, which # is why we need to explicitly configure a location
userData:
  # defaults to a .datastax_studio folder in your home directory, such as
  # ~/.datastax_studio
  # set to a non-null value to override
  baseDirectory: null

Cell options

resultSizeLimit
Maximum number of items returned per cell execution. Additional items will be truncated. Default: 1000.

maxResultSizeBytes
Maximum size of a cell result. If a cell result exceeds this size then the cell execution will fail. Default: 10485760.

executionTimeoutMs
Cell execution timeout in milliseconds. A value of 0 indicates no timeout override. Uses the DSE server timeouts configured in the dse.yaml file. Default: 0.
traversalSources

Traversal source mapping for real-time and analytic execution engines. These values correspond to the traversal sources configured in the dse.yaml file.

realTimeTraversalSource
Traversal source mapping for real-time execution engine. Default: g

analyticTraversalSource

Server

httpPort
The port on which the Studio server is running. Default: 9091

httpBindAddress
The IP address to which the Studio server is bound. Default: 127.0.0.1.

Note: In Studio version 1.0.0, if you change either of these options, you must also change the corresponding ide options, graphSchemaServiceURL and graphSchemaServicePort.

Logging

Logging-related options:

fileName
Name of the log file. Default: studio.log.

maxLogFileSize
Default: 250 MB.

maxFiles
Maximum number of log files. Default: 10.

directory
Path of the directory in which log files are stored. Default: ./log.

User data

Directories where user data is stored relative to the baseDirectory:

baseDirectory
Default: ~/.datastax_studio

connectionsDirectory
userdata/connections

The location of the dse.yaml file depends on the type of installation:

<table>
<thead>
<tr>
<th>Installations Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/dse/dse.yaml</td>
</tr>
<tr>
<td>Installer-Services installations</td>
<td></td>
</tr>
<tr>
<td>Tarball installations</td>
<td>installation_location/</td>
</tr>
<tr>
<td>Installer-No Services installations</td>
<td>resources/dse/conf/dse.yaml</td>
</tr>
</tbody>
</table>

The location of the cassandra.yaml file depends on the type of installation:
Advanced configuration options in DataStax Studio

There are a number of advanced configuration options that are not explicitly declared and set in the `configuration.yaml` file.

**userData**

- **connectionsDirectory**
  
  The directory where connections are stored. Default: connections.

- **snapshotSaveIntervalInSeconds**
  
  Default: 300.

- **entityCacheIdleTimeoutInSeconds**
  
  Default: 3600.

- **maxKeyspaceSessionsPerConnection**
  
  Maximum number of sessions associated with a specific keyspace that Studio keeps open at any one point in time. Least recently used sessions are closed first. Default: 5.

- **eventReplayTimeoutInSeconds**
  
  Default: 600.

- **eventReplayBatchSize**
  
  Default: 10.

**security**

To make encryption of passwords unique for your installation, you may change the password in this file. But you should ensure it is a reasonably strong generated password, and not just a word or phrase.

- **encryptionPasswordFile**
  
  Default: `conf/security/security.properties`.

**connectionManagement**

- **idleTimeoutInSeconds**
  
  Determines how long before a connection expires and is closed when it is not in use in seconds. Default: 3600 (1 hour).

**Miscellaneous**

- **schemaRefreshIntervalMs**
Schema refresh polling interval in milliseconds. Default: \texttt{3000} (3 seconds).

\section*{DataStax Studio Reference}

\section*{Connections page in DataStax Studio}

Create and modify connections from the \textbf{Connections page}.

Create Connections dialog

Use the \textbf{Create Connection} dialog to create and modify a connection to a DataStax Enterprise cluster:

\begin{itemize}
\item \textbf{Name} \hspace{1cm} Name of the connection.
\item \textbf{Host/IP (comma delimited)} \hspace{1cm} The host names or IP addresses of the DSE cluster to connect to. Default: localhost.
\item \textbf{Username} \hspace{1cm} Optional. DSE username for logging in.
\item \textbf{Password} \hspace{1cm} Optional. DSE password for logging in.
\item \textbf{Port} \hspace{1cm} IP connection port of the DSE cluster. Default: 9042.
\end{itemize}

\section*{Default imports in DataStax Studio}

The following imports are performed by default in a notebook cell.

- Static imports:

\begin{verbatim}
# org.apache.tinkerpop.gremlin.util.TimeUtil.*
\end{verbatim}
# org.apache.tinkerpop.gremlin.structure.Direction.*
# org.apache.tinkerpop.gremlin.process.traversal.Pop.*
# org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversalSource.*
# org.apache.tinkerpop.gremlin.process.traversal.P.*
# org.apache.tinkerpop.gremlin.process.traversal.Order.*
# org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__.*
# org.apache.tinkerpop.gremlin.structure.io.IoCore.*
# org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__.*
# org.apache.tinkerpop.gremlin.process.traversal.Scope.*
# org.apache.tinkerpop.gremlin.structure.Column.*
# org.apache.tinkerpop.gremlin.structure.T.*
# org.apache.tinkerpop.gremlin.process.traversal.Operator.*

**Imports:**

# groovy.grape.Grape
# org.apache.commons.configuration.*
# org.apache.tinkerpop.gremlin.process.traversal.strategyverification.*
# org.apache.tinkerpop.gremlin.process.computer.bulkloading.*
# org.apache.tinkerpop.gremlin.process.computer.traversal.*
# org.apache.tinkerpop.gremlin.util.function.*
# org.apache.tinkerpop.gremlin.structure.*
# org.apache.tinkerpop.gremlin.process.traversal.strategy.decoration.*
# org.apache.tinkerpop.gremlin.process.traversal.engine.*
# org.apache.tinkerpop.gremlin.structure.io.gryo.*
# org.apache.tinkerpop.gremlin.process.traversal.strategy.optimization.*
# org.apache.tinkerpop.gremlin.process.traversal.step.util.event.*
# org.apache.tinkerpop.gremlin.util.*
# org.apache.tinkerpop.gremlin.structure.util.*
# org.apache.tinkerpop.gremlin.structure.io.graphml.*
# org.apache.tinkerpop.gremlin.process.computer.*
# org.apache.tinkerpop.gremlin.process.traversal.strategy.finalization.*
# org.apache.tinkerpop.gremlin.process.computer.clustering.peerpressure.*
# org.apache.tinkerpop.gremlin.structure.util.detached.*
# org.apache.tinkerpop.gremlin.structure.io.graphson.*
# org.apache.tinkerpop.gremlin.process.traversal.*
# org.apache.tinkerpop.gremlin.process.computer.bulkdumping.*
# org.apache.tinkerpop.gremlin.process.traversal.util.*
# org.apache.tinkerpop.gremlin.groovy.function.*

- Extra imports:
  
  # com.datastax.bdp.graph.api.*
  # com.datastax.bdp.graph.api.schema.*
  # com.datastax.bdp.graph.api.id.*
  # com.datastax.bdp.graph.api.config.*
  # org.apache.cassandra.db.marshal.geometry.*
  # com.datastax.bdp.graph.api.system.*
  # java.time.*

**Note:** To get a list from within a Gremlin cell:

```java
DseGraphImports.getInstance().getAllImports()
```

## Keyboard shortcuts in DataStax Studio notebooks

Studio notebooks have keyboard shortcuts to increase your proficiency while writing code and content. Mode depends on cursor location:

- Edit mode when focus is on a cell editor.
- Command mode when focus is on the cell.

### Edit mode keyboard shortcuts

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+?</td>
<td>Display <a href="#">Shortcut help</a></td>
</tr>
<tr>
<td>Esc</td>
<td>Unfocus the editor and focus the cell (switch to <a href="#">Command mode</a>).</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Save code and execute cell.</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Content assist.</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Toggle line numbers.</td>
</tr>
<tr>
<td>Ctrl+/</td>
<td>Toggle comment.</td>
</tr>
<tr>
<td>Ctrl+H</td>
<td>Hide the editor and focus on the cell (switch to <a href="#">Command mode</a>).</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Add a new cell below and switch to its editor.</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+H</td>
<td>Add a new cell above and switch to its editor.</td>
</tr>
<tr>
<td>Ctrl+Shift+S</td>
<td>Toggle Schema view.</td>
</tr>
<tr>
<td>Shortcut</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Ctrl+Shift+Down (Macintosh OS X: #+#+Down)</td>
<td>Focus on the cell below, entering its Command mode.</td>
</tr>
<tr>
<td>Ctrl+Shift+Up (Macintosh OS X: #+#+Up)</td>
<td>Focus on the cell above, entering its Command mode.</td>
</tr>
<tr>
<td>Alt+Backspace (Macintosh OS X: #+Backspace)</td>
<td>Delete from the cursor to the beginning of the line.</td>
</tr>
<tr>
<td>Alt+Delete (Macintosh OS X: #+Delete)</td>
<td>Delete from the cursor to the end of the line.</td>
</tr>
</tbody>
</table>

**Command mode keyboard shortcuts**

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+?</td>
<td>Display Shortcut help.</td>
</tr>
<tr>
<td>Enter</td>
<td>Focus on the cell's code editor (switch to Edit mode).</td>
</tr>
<tr>
<td>Delete (Macintosh OS X: Function +Backspace)</td>
<td>Delete the current cell and focus on next available cell.</td>
</tr>
<tr>
<td>H</td>
<td>Toggle editor visibility.</td>
</tr>
<tr>
<td>N</td>
<td>Add new cell below and switch to its editor.</td>
</tr>
<tr>
<td>Shift+N</td>
<td>Add new cell above and switch to its editor.</td>
</tr>
<tr>
<td>S</td>
<td>Toggle Schema view.</td>
</tr>
<tr>
<td>Up</td>
<td>Switch to the cell above.</td>
</tr>
<tr>
<td>Down</td>
<td>Switch to the cell below.</td>
</tr>
<tr>
<td>Shift+Up</td>
<td>Move current cell up.</td>
</tr>
<tr>
<td>Shift+Down</td>
<td>Move current cell down.</td>
</tr>
</tbody>
</table>

**DataStax Studio FAQ**

**Frequently asked questions**

**Why can I configure more than one host in a Studio connection?**
Hosts initialize the Cassandra driver connection. Configuring more than one host provides redundancy and failover protection.

**How can I view a list of my notebooks?**
Use the Notebook Manager (page 1116) to list and filter notebooks.

**Where have my old notebook files gone?**
When you upgrade Studio to version 2.0, all of your existing notebooks are backed up.
Existing notebooks from Studio 1.0.x are migrated to the new format in Studio 2.0.x. Studio creates a backup of 1.0 notebooks in the $userdata/1_x_backup directory during the upgrade migration. If the migration is successful, the $userdata/notebooks/admin directory is deleted.

**Which web browsers are supported for Studio?**

DataStax Studio is tested on these platforms (all 64-bit) with the latest versions of the specified web browsers.

**Does Studio support connection to SSL-enabled clusters?**

Yes. See Using SSL connections in DataStax Studio (page 1106).

**Troubleshooting DataStax Studio**

Fixing problems in DataStax Studio.

See Troubleshooting DataStax Studio.
CQL

CQL (Cassandra Query Language) is a query language for the DataStax Enterprise database. See CQL for DataStax Enterprise 5.1.