Contents

About DataStax Enterprise .......................... 1
  New Features in DataStax Enterprise 3.0 .................................. 1
  Security for the Community .................................................. 1
  Security for the Enterprise .................................................. 1
  Key Features of DataStax Enterprise ........................................ 1

Installation .................................................. 4
  Installing the DataStax Enterprise package on RHEL-based distributions .................................. 4
  Installing the DataStax Enterprise package on Debian and Ubuntu ............................................. 5
  Installing the DataStax Enterprise tarball on Mac OSX or Any Linux OS ................................... 6
  Installing the DataStax Enterprise Tarball on SUSE Enterprise .................................................. 8
  Installing on cloud providers ................................................ 8

Upgrading to DataStax Enterprise 3.0.x ......................... 15
  Preparing to upgrade ....................................................... 15
  Upgrading DataStax Enterprise real-time Cassandra nodes ......................................................... 17
  Upgrading DSE Search/Solr nodes .......................................... 21
  Upgrading Analytics/Hadoop nodes ............................. 25
  Upgrading DataStax Community--tarball ...................................... 27
  Upgrading DataStax Community--Debian/Ubuntu .................................. 28
  Upgrading DataStax Community--RHEL-based distributions ..................................................... 29
  Upgrading legacy versions .................................................. 30
  Upgrading the DataStax AMI ............................................... 31

Security ................................................ 32
  Security management .................................................... 32
  Authenticating a DataStax Enterprise cluster with Kerberos ...................................................... 34
  Client-to-node encryption .................................................. 36
  Node-to-node encryption ................................................... 37
  Preparing server certificates ............................................. 38
  Installing the cqlsh security packages .................................... 38
  Transparent data encryption .............................................. 40
  Configuring and using data auditing .................................... 44
  Configuring and using internal authentication ................................................. 49
  Managing object permissions using internal authorization ................................................ 51
  Configuring dse_auth keyspace replication ........................................... 52
  Configuring firewall port access ......................................... 52
  Alphabetical CQL command reference .................................... 54
  ALTER USER .................................................... 54
  CREATE USER ..................................................... 54
About DataStax Enterprise

DataStax Enterprise is a big data platform built on Apache Cassandra that manages real-time, analytics, and enterprise search data. DataStax Enterprise leverages Cassandra, Apache Hadoop, and Apache Solr to shift your focus from the data infrastructure to using your data strategically.

New Features in DataStax Enterprise 3.0

DataStax Enterprise (DSE) 3.0 features internal and external authentication, object permission management, transparent data encryption, data auditing, client-to-node encryption, and an enterprise-class administration interface for secure database management.

The new version provides core security capabilities to the entire Cassandra community, as well as the advanced data protection that businesses expect in an enterprise-grade database. DSE 3.0 supplies the type of security framework that allows modern enterprises to confidently adopt NoSQL databases as they safely scale their big data infrastructure.

Security for the Community

With the release of DSE 3.0, DataStax is providing security features both to the open source community and to the enterprise. For the Cassandra NoSQL community, DataStax is making the following security enhancements freely available to everyone who uses Apache Cassandra:

- **Authentication based on internally controlled login accounts/passwords and internal authorization**
- **Object permission management** based on the familiar relational database GRANT/REVOKE paradigm
- **Client-to-node encryption**, which protects data in flight from client machines to a database cluster

Security for the Enterprise

In addition to the general security features being made available to the entire Cassandra NoSQL community, DSE 3.0 supplies the following features to enterprises that require advanced security when handling mission-critical data:

- **External authentication** via Kerberos, which is one of the most trusted external security packages in use today, especially by governments and financial institutions. External authentication allows DSE to provide single sign-on capability to all Cassandra, Hadoop, and Solr nodes in a DSE cluster.
- **Transparent data encryption** protects data at rest from theft, and from unauthorized read access at the file system level.
- **Data auditing** enables administrators to create granular audit trails of some or all activity in a database cluster.

Key Features of DataStax Enterprise

The key features of DataStax Enterprise include:

- **Production Certified Cassandra** – DataStax Enterprise contains a fully tested, benchmarked, and certified version of Apache Cassandra that is suitable for mission-critical production deployments.
- **No Single Point of Failure** - In the Hadoop Distributed File System (HDFS) master/slave architecture, the NameNode entry point into the cluster stores configuration metadata about the cluster. If the NameNode fails, the Hadoop system goes down. DataStax Enterprise improves upon this architecture by making nodes peers. Being peers, any node in the cluster can load data files, and any analytics node can assume the responsibilities of job tracker for MapReduce jobs.
- **Reserve Job Tracker** - DataStax Enterprise keeps a job tracker in reserve to take over in the event of a problem that would affect availability.
About DataStax Enterprise

- **Multiple Job Trackers** - In the Cassandra File System (CassandraFS), you can run one or more job tracker services across multiple data centers and create multiple CassandraFS keyspaces per data center. Using this capability has performance, data replication, and other benefits.

- **Hadoop MapReduce using Multiple Cassandra File Systems** - CassandraFS is an HDFS-compatible storage layer. DataStax replaces HDFS with CassandraFS to run MapReduce jobs on Cassandra's peer-to-peer, fault-tolerant, and scalable architecture. In DataStax Enterprise 2.1 and later, you can create additional CassandraFS's to organize and optimize Hadoop data.

- **Analytics Without ETL** - Using DataStax Enterprise, you run MapReduce jobs directly against your data in Cassandra. You can even perform real-time and analytics workloads at the same time without one workload affecting the performance of the other. Starting some cluster nodes as Hadoop analytics nodes and others as pure Cassandra real-time nodes automatically replicates data between nodes.

- **Streamlined Setup and Operations** - In Hadoop, you have to set up different mode configurations: stand-alone mode or pseudo-distributed mode for a single node setup, or cluster mode for a multi-node configuration. In DataStax Enterprise, you configure only one mode (cluster mode) regardless of the number of nodes.

- **Hive Support** - Hive, a data warehouse system, facilitates data summarization, ad-hoc queries, and the analysis of large data sets stored in Hadoop-compatible file systems. Any JDBC compliant user interface connects to Hive from the server. Using the Cassandra-enabled Hive MapReduce client in DataStax Enterprise, you project a relational structure onto Hadoop data in the Cassandra file systems, and query the data using a SQL-like language. Cassandra nodes share the Hive metastore automatically, eliminating repetitive HIVE configuration steps.

- **Pig Support** - The Cassandra-enabled Pig MapReduce client included with DataStax Enterprise is a high-level platform for creating MapReduce programs used with Hadoop. You can analyze large data sets, running jobs in MapReduce mode and Pig programs directly on data stored in Cassandra.

- **Enterprise Search Capabilities** - DataStax Enterprise Search fully integrates Apache Solr for ad-hoc querying of data, full-text search, hit highlighting, multiple search attributes, geo-spatial search, and for searching rich documents, such as PDF and Microsoft Word, and more.

- **Migration of RDBMS data** - Apache Sqoop in DataStax Enterprise provides easy migration of RDBMS data, such as Oracle, Microsoft SQL Server, MySQL, Sybase, and DB2 RDBMS, and non-relational data sources, such as NoSQL into the DataStax Enterprise server.

- **Runtime Logging** - DataStax Enterprise transfers log-based data directly into the server using log4j. Apache log4j is a Java-based logging framework that provides runtime application feedback and control over the size of log statements. Cassandra Appender can store the log4j messages in the Cassandra table-like structure for in-depth analysis using the Hadoop and Solr capabilities.

- **Support for Mahout** - The Hadoop component, Apache Mahout, incorporated into DataStax Enterprise 2.1 and later offers machine learning libraries. Machine learning improves a system, such as the one that recreates the Google priority inbox, based on past experience or examples.

- **Full Integration with DataStax OpsCenter** - Using DataStax OpsCenter, you can monitor, administer, and configure one or more DataStax Enterprise clusters in an easy-to-use graphical interface. Schedule automatic backups, explore Cassandra data, and see detailed health and status information about clusters, such as the up or down status of nodes, graphs of performance metrics, storage limitations, and progress of Hadoop MapReduce jobs.
Installation

Installing the DataStax Enterprise package on RHEL-based distributions

DataStax provides a packaged release for installing DataStax Enterprise on the following systems:

- CentOS systems
- Oracle Linux
- Red Hat Enterprise Linux (RHEL)

For a complete list of supported platforms, see DataStax Enterprise Supported Platforms.

To install OpsCenter, see Installing OpsCenter from CentOS, OEL, and RHEL packages.

Prerequisites

- Before you can install, you must register with DataStax to get a username and password.
- The Yum Package Management application installed.
- Root or sudo access to the install machine.
- The latest version of Oracle Java SE Runtime Environment (JRE) 6 is installed. Java 7 is not recommended.
- Java Native Access (JNA) is required for production installations. See Installing JNA.
- If you are installing DataStax Enterprise on a 64-bit Oracle Linux, first install 32-bit versions of glibc libraries.

Also, see Recommended Settings for Production Installations.

DataStax Enterprise and OpsCenter installation steps

DataStax provides Yum repositories for CentOS, Oracle Linux, and RHEL systems. Use these instructions to install DataStax Enterprise 3.0.x.

1. Check which version of Java is installed by running the following command in a terminal window:

   `# java -version`

   Use the latest version of Java 6 on all nodes. Java 7 is not recommended. If you need help installing Java, see Installing the JRE on RHEL or CentOS Systems.

2. (CentOS 5.x/RHEL 5.x only) Make sure you have EPEL (Extra Packages for Enterprise Linux) installed. EPEL contains dependent packages required by OpsCenter. To install for both 32- and 64-bit systems:

   `# rpm -Uvh http://dl.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm`

3. Add a yum repository file for DataStax in `/etc/yum.repos.d`. For example:

   `# /etc/yum.repos.d/datastax.repo`
4. In this file add the following lines:

    [datastax]
    name= DataStax Repo for Apache Cassandra
    baseurl=http://<username>:<password>@rpm.datastax.com/enterprise
    enabled=1
    gpgcheck=0

5. Install the DataStax Enterprise packages:

    # yum install dse-full-3.0.6-1

The packaged releases create a user cassandra. When starting DataStax Enterprise as a service, the Cassandra and Hadoop tracker services run as this user. A service initialization script is located in /etc/init.d/dse. Run levels are not set by the package.

You can install any 3.0.x version using the following format:

    # yum install dse-full-<version>-1

**Next steps**

- Set the configuration properties on each node in the cluster for *single* or *multiple* data center deployment.
- Install or configure OpsCenter.
- Configuration file locations.
- Configure the heap dump directory to avoid server crashes.
- Start DataStax Enterprise.

**Installing the DataStax Enterprise package on Debian and Ubuntu**

DataStax provides a packaged release for installing DataStax Enterprise and OpsCenter on Debian and Ubuntu systems. For a complete list of supported platforms, see DataStax Enterprise Supported Platforms.

To install OpsCenter, see Installing OpsCenter from Debian or Ubuntu packages.

**Prerequisites**

- Before you can install, you must register with DataStax to get a username and password.
- The Aptitude Package Manager is installed.
- Root or sudo access to the install machine.
- The latest version of Oracle Java SE Runtime Environment (JRE) 6 is installed. Java 7 is not recommended.
- Java Native Access (JNA) is required for production installations. See Installing JNA.

**Note**

If you are using Ubuntu 10.04 LTS, you need to update to JNA 3.4, as described in Install JNA on Ubuntu 10.04.

- Your DataStax username and password (provided in your DataStax registration confirmation email). If you do not have a DataStax username and password, register before attempting to download the software.

Also see Recommended Settings for Production Installations.
DataStax Enterprise and OpsCenter installation steps

DataStax provides Debian package repositories for Debian and Ubuntu systems. Use these instructions to install DataStax Enterprise 3.0.x.

1. Check which version of Java is installed by running the following command in a terminal window:

   # java -version

   Use the latest version of Java 6 on all nodes. Java 7 is not recommended. If you need help installing Java, see Installing the JRE on Debian or Ubuntu Systems.

2. Add the DataStax repository to the /etc/apt/sources.list.d/datastax.sources.list file. For example:

   deb http://<username>:<password>@debian.datastax.com/enterprise stable main

3. Add the DataStax repository key to your Aptitude trusted keys:

   $ curl -L https://debian.datastax.com/debian/repo_key | sudo apt-key add -

   Note
   If you have trouble adding the key, use http instead of https.

4. Install the DataStax Enterprise packages:

   $ sudo apt-get update
   $ sudo apt-get install dse-full=3.0.4-1 dse=3.0.4-1 dse-hive=3.0.4-1 dse-pig=3.0.4-1 dse-demos=3.0.4-1 ...

   The packaged releases create a user cassandra. When starting DataStax Enterprise as a service, the Cassandra and Hadoop tracker services run as this user. A service initialization script is located in /etc/init.d/dse. Run levels are not set by the package.

   You can install any 3.0.x version using the following format:

   $ sudo apt-get install dse-full=<version>-1 dse=<version>-1 dse-hive=<version>-1 dse-pig=<version>-1 ...

Next steps

- Set the configuration properties on each node in the cluster for single or multiple data center deployment.
- Install or configure OpsCenter.
- Configuration file locations.
- Configure the heap dump directory to avoid server crashes.
- Start DataStax Enterprise.

Installing the DataStax Enterprise tarball on Mac OSX or Any Linux OS

DataStax provides a binary tarball distribution for installing on platforms that do not have package support, such as Mac, or if you do not have or want to do a root installation. For a complete list of supported platforms, see DataStax Enterprise Supported Platforms.

To install the Opscenter, see Installing OpsCenter tarball distributions on Linux and Mac OSX.

Prerequisites
Before you can install, you must register with DataStax to get a username and password.

The latest version of Oracle Java SE Runtime Environment (JRE) 6 is installed. Java 7 is not recommended.

Java Native Access (JNA) is required for production installations. See Installing JNA.

**Note**
If you are using Ubuntu 10.04 LTS, you need to update to JNA 3.4, as described in Install JNA on Ubuntu 10.04.

Also see Recommended production settings.

**DataStax Enterprise and OpsCenter installation steps**

1. Check which version of Java is installed by running the following command in a terminal window:

   ```
   # java -version
   ```

   Use the latest version of Java 6 on all nodes. Java 7 is not recommended. If you need help installing Java, see Installing Oracle JRE.

2. Download the tarball from the Previous Releases section of the Download DataStax Enterprise page.

3. Unpack the distributions:

   ```
   $ tar -xzvf dse.tar.gz
   $ rm *.tar.gz
   ```

4. By default, DataStax Enterprise is configured to use /var/lib/cassandra and /var/log/cassandra directories.

   If you do not have root access to the default directories, ensure you have write access as follows:

   ```
   $ 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
   ```

5. (Optional) If you do not want to use the default data and logging directories, you can define your own directory locations:

   a. Make the directories for data and logging directories. For example:

      ```
      $ mkdir <install_location>/dse-data
      $ cd dse-data
      $ mkdir commitlog
      $ mkdir saved_caches
      ```

   b. Go the directory containing the cassandra.yaml file. For example:

      ```
      $ cd <install_location>/resources/cassandra/conf
      ```

   c. Edit the following lines in the cassandra.yaml file. For example:

      ```
      data_file_directories: <install_location>/dse-data
      commitlog_directory: <install_location>/dse-data/commitlog
      saved_caches_directory: <install_location>/dse-data/saved_caches
      ```

**Next steps**

- Set the configuration properties on each node in the cluster for *single* or *multiple* data center deployment.
Installing the DataStax Enterprise Tarball on SUSE Enterprise

DataStax provides a binary tarball distribution for installing on SUSE Linux. For a complete list of supported platforms, see DataStax Enterprise Supported Platforms.

Prerequisites

- Before you can install, you must register with DataStax to get a username and password.
- The latest version of Oracle Java SE Runtime Environment (JRE) 6 is installed. Java 7 is not recommended. See Installing the JRE on SUSE Systems.
- Java Native Access (JNA) is required for production installations. See Installing JNA. Also see Recommended Settings for Production Installations.

Installing DataStax Enterprise and OpsCenter on SUSE

To install DataStax Enterprise on SUSE:

1. Install DataStax Enterprise using the Binary Tarball Distribution.
2. Set up DataStax Enterprise as described in Deployment.

To install OpsCenter on SUSE:

1. Install OpsCenter using the OpsCenter Tarball Distribution.
2. Deploy OpsCenter agents as described in Manually Deploying Agents - Tarball Installations.

Installing on cloud providers

Initializing a DataStax Enterprise Cluster on Amazon EC2

For instructions on installing the DataStax AMI (Amazon Machine Image), see the latest AMI documentation.

Installing DataStax Enterprise on HP Cloud

This is a step-by-step guide to setting up a DataStax Enterprise (DSE) cluster in the HP Cloud. DataStax supports installation on Ubuntu 11.04 Natty Narwhal and Ubuntu 11.10 Oneiric Ocelot. Installation includes the following steps:

- Creating a key pair
- Creating a security group
- Creating the server
- Connecting to the server
- Install the JRE and JNA

- Install or configure OpsCenter.
- Configuration file locations.
- Configure the heap dump directory to avoid server crashes.
- Start DataStax Enterprise.
Creating a key pair

You need a key pair (.pem file) to login to your DataStax Enterprise nodes.

1. From the HP Cloud Dashboard, click Manage Servers or Activate in one of the Availability Zones.
2. Click Key Pairs.
3. Click Add KeyPair.
   - If you do not have an existing key pair, specify only the Key Name, click Create Key, and then copy the contents into a text file that has .pem extension on your local machine.
   - If you already have an existing key pair, specify both the Key Name and Public Key.

Creating a security group

A security group acts as a firewall that allows you to choose which protocols and ports are open in your cluster. A Cassandra cluster requires that certain ports are open for inter-node, OpsCenter, and SSH communication. You can specify the protocols and ports either by a range of IP addresses or by security group. It is much simpler and requires less maintenance to define port access by security group. Currently the HP Cloud console does not provide the capability to specify ports by security group. However, you can install and use the HP Extended Python Novaclient for this purpose.

The HP Security Groups document provides information on defining rules for security groups.

To create a security group:

1. Using the HP Extended Python Novaclient, create a security group:

   nova secgroup-create DSESecurityGroup "Security group for DataStax Enterprise"
2. Create the rules for the security group. For example, to create a rule that opens port 7000 to other nodes in the security group:

```
nova secgroup-add-group-rule DSESecurityGroup DSESecurityGroup --ip_proto tcp --from_port 7000 --to_port 7000
```

<table>
<thead>
<tr>
<th>IP Protocol</th>
<th>From Port</th>
<th>To Port</th>
<th>IP Range</th>
<th>Source Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp</td>
<td>7000</td>
<td>7000</td>
<td></td>
<td>DSESecurityGroup</td>
</tr>
</tbody>
</table>

A Cassandra/DataStax Enterprise cluster requires the following ports:

<table>
<thead>
<tr>
<th>Port</th>
<th>IP Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>icmp</td>
<td>Use for ping</td>
</tr>
</tbody>
</table>

**Public Facing Ports**

<table>
<thead>
<tr>
<th>Port</th>
<th>IP Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>tcp</td>
<td>Default SSH port</td>
</tr>
</tbody>
</table>

**DataStax Enterprise Specific**

<table>
<thead>
<tr>
<th>Port</th>
<th>IP Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8012</td>
<td>tcp</td>
<td>Hadoop Job Tracker client port</td>
</tr>
<tr>
<td>8983</td>
<td>tcp</td>
<td>Solr port and Demo applications website port (Portfolio, Search, Search log)</td>
</tr>
<tr>
<td>50030</td>
<td>tcp</td>
<td>Hadoop Job Tracker website port</td>
</tr>
<tr>
<td>50060</td>
<td>tcp</td>
<td>Hadoop Task Tracker website port</td>
</tr>
</tbody>
</table>

**OpsCenter Specific**

<table>
<thead>
<tr>
<th>Port</th>
<th>IP Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8888</td>
<td>tcp</td>
<td>OpsCenter website port</td>
</tr>
</tbody>
</table>

**Internode Ports**

**Cassandra Specific**

<table>
<thead>
<tr>
<th>Port</th>
<th>IP Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024+</td>
<td>tcp (use security group)</td>
<td>JMX reconnection/loopback ports</td>
</tr>
<tr>
<td>7000</td>
<td>tcp (use security group)</td>
<td>Cassandra intra-node port</td>
</tr>
<tr>
<td>7199</td>
<td>tcp (use security group)</td>
<td>Cassandra JMX monitoring port</td>
</tr>
<tr>
<td>9160</td>
<td>tcp (use security group)</td>
<td>Cassandra client port</td>
</tr>
</tbody>
</table>

**DataStax Enterprise Specific**

<table>
<thead>
<tr>
<th>Port</th>
<th>IP Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9290</td>
<td>tcp (use security group)</td>
<td>Hadoop Job Tracker Thrift port</td>
</tr>
</tbody>
</table>

**OpsCenter Specific**

<table>
<thead>
<tr>
<th>Port</th>
<th>IP Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>50031</td>
<td>tcp (use security group)</td>
<td>OpsCenter HTTP proxy for Job Tracker</td>
</tr>
<tr>
<td>61620</td>
<td>tcp (use security group)</td>
<td>OpsCenter intra-node monitoring port</td>
</tr>
<tr>
<td>61621</td>
<td>tcp (use security group)</td>
<td>OpsCenter agent ports</td>
</tr>
</tbody>
</table>

**Note**

Generally, when you have firewalls between machines, it is difficult to run JMX across a network and maintain security. This is because JMX connects on port 7199, handshakes, and then uses any port within the 1024+ range. Instead use SSH to execute commands to remotely connect to JMX locally or use the DataStax OpsCenter.
3. After you are done adding the port rules, you can also view them on the HP Cloud console:

**Rules for DSESecurity**

<table>
<thead>
<tr>
<th>IP Protocol</th>
<th>From Port</th>
<th>To Port</th>
<th>Type</th>
<th>CIDR IPS</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>icmp</td>
<td>-1</td>
<td>-1</td>
<td>IPs</td>
<td>0.0.0.0/0</td>
<td></td>
</tr>
<tr>
<td>tcp</td>
<td>22</td>
<td>22</td>
<td>IPs</td>
<td>0.0.0.0/0</td>
<td></td>
</tr>
<tr>
<td>tcp</td>
<td>8888</td>
<td>8888</td>
<td>IPs</td>
<td>0.0.0.0/0</td>
<td></td>
</tr>
<tr>
<td>tcp</td>
<td>1024</td>
<td>65535</td>
<td>group</td>
<td></td>
<td>DSESecurity</td>
</tr>
</tbody>
</table>

*Add Rule* to create new rule

**Warning**
This security configuration shown in the above graphic opens ports 22 and 8888 to incoming traffic from any IP address (0.0.0.0/0). If you desire a more secure configuration, see the HP Security Groups document.

**Creating the server**

1. From the HP Cloud Dashboard, click **Manage Servers** or **Activate** in one of the Availability Zones.
2. Under **Create Servers**, select the following:
   - **Flavor**: standard.large (or greater).
   - **Security Group**: The DSE security group that you created earlier.
   - **Install Image**: Ubuntu Oneric 11.10.
   - **Key Pair**: The key pair that you created earlier.
3. Click **Create**.
4. Click **Create** for each additional instance.

**Connecting to the server**

1. If this is the first time you are connecting, copy your private key file (<keyname>.pem) you downloaded earlier to your home directory, and change the permissions so it is not publicly viewable. For example:
   
   ```bash
   chmod 400 DataStaxKey.pem
   ```

2. After the instance is running, click **Connect**.

3. From the Instance dialog box, copy the example and change the connection user from **root** to **ubuntu**, then paste it into your SSH client.

   ```bash
   ssh -i DataStaxKey.pem root@15.185.171.82
   ```

**Install the JRE and JNA**

Oracle Java SE Runtime Environment (JRE) 6 is required to run DataStax Enterprise. The latest version is recommended.

1. The easiest way to put the Oracle JRE on an HP Cloud instance is to download it to your local machine from [Oracle Java SE Downloads](https://www.oracle.com/java/technologies/javase-jre-downloads.html) and then use the secure copy command to copy it onto the node:

   ```bash
   scp -i DataStaxKey.pem jre-6u43-linux-x64.bin ubuntu@<ip_address>:~/
   ```

2. Install the JRE as described in **Installing the JRE on Debian or Ubuntu Systems**.

3. Install the JNA as described in **Installing JNA**.

**Install DataStax Enterprise**

Install DataStax Enterprise as described in **Installing the DataStax Enterprise package on Debian and Ubuntu**.
Note
You only need to install OpsCenter on one node.

Configure DataStax Enterprise
You can configure DataStax Enterprise as described in Single data center deployment or Single data center deployment using the following guidelines.

- Single availability zone:
  - If necessary, change the default the delegated_snitch to DSESimpleSnitch. It is located in the /etc/dse/dse.yaml configuration file.
    
    delegated_snitch: com.datastax.bdp.snitch.DseSimpleSnitch

  - In the /etc/dse/cassandra/cassandra.yaml configuration file, use the private IP addresses of the nodes, not the public IP addresses:
    
    seed_provider:
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
      - seeds: "<private_ip_of_seed1>,<private_ip_of_seed2>"
      listen_address: <private_ip_of_the_node>

- Multiple availability zones:
  - In the /etc/dse/dse.yaml configuration file, set the delegated_snitch to PropertyFileSnitch:
    
    delegated_snitch: org.apache.cassandra.locator.PropertyFileSnitch

  - In the /etc/dse/cassandra/cassandra.yaml configuration file, use the public IP addresses for the seeds and set the broadcast_address:
    
    seed_provider:
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
      - seeds: "<public_ip_of_seed1>,<public_ip_of_seed2>"
      listen_address: <private_ip_of_the_node>
      broadcast_address: <public_ip_of_the_node>

Configuring OpsCenter and Agents
DataStax Enterprise OpsCenter is installed when you install DataStax Enterprise using the sudo apt-get install dse-full opscenter command. If you have not already installed OpsCenter, install it as described in Installing OpsCenter from Debian or Ubuntu Packages.

Note
If you are installing OpsCenter on Ubuntu 11.10, be sure to install OpenSSL 0.9.8 on the node where OpsCenter is installed:

$ sudo apt-get install libssl0.9.8
1. In the /etc/opscenter/opscenterd.conf configuration file, set the [webserver] interface to the private IP address of the OpsCenter node:

   [webserver]
   port = 8888
   interface = <private_ip_of_the_opscenter_node>

2. Connect to the OpsCenter using the following URL:

   http://<private_ip_of_the_opscenter_node>:8888

3. Install the agents as described in Automatically Deploying Agents - Packaged Installations.

   • In the Welcome to DataStax OpsCenter! dialog box, use the private IP address for each node.
   • In the Node SSH Credentials dialog box, use ubuntu for the user name and the private key from the key pair you use to connect to the HP Cloud.
Upgrading to DataStax Enterprise 3.0.x

Preparing to upgrade
You can upgrade from these releases to DataStax Enterprise 3.0.x:

- A previous release of DataStax Enterprise
- Cassandra 0.7.10, 0.8.10, or 1.0.x - 1.1.x

You can upgrade to DataStax Enterprise 3.0 from a version of DataStax Community or Cassandra that is earlier, but not later, than the one included in DataStax Enterprise 3.0.x. See the release notes for version information.

Upgrade one data center at a time.
To start the upgrade process, perform the pre-upgrade steps, and then follow the link in step 4 for the particular version you are upgrading.

Limitations on use during the upgrade
During an upgrade, limit activity on the node:

- Do not update schemas.
- Do not re-index Solr unless you are following an instruction in these upgrade procedures to re-index.
- Do not run nodetool repair.
- Do not use new features.
- Do not issue these types of queries during a rolling restart: DDL, TRUNCATE, and Solr queries
- Do not issue Solr queries after upgrading from DataStax Enterprise 2.1.x or earlier until all nodes are upgraded and schema disagreements are resolved.
- Do not attempt to set up Kerberos authentication. First upgrade the cluster, and then set up Kerberos.

Pre-upgrade steps

1. Make a backup of the data by taking a snapshot of the node to be upgraded.
2. Run nodetool drain to flush the commit log of the old installation:

   nodetool drain -h <hostname>

   If you are upgrading a DSE Search/Solr node, this step is mandatory. This step is highly recommended for upgrading other nodes because, in the event of a power failure during the upgrade, data loss could occur.
3. On Debian/Ubuntu, save the cassandra.yaml (and cassandra-topology.properties file if you use the PropertyFileSnitch) from the old installation in a safe location.

   On RHEL-based platforms, RPM saves the file automatically during the upgrade process instead of overwriting it. RPM output looks something like this:

   warning: /etc/cassandra/default.conf/cassandra.yaml saved as /etc/cassandra/default.conf/cassandra.yaml.rpmsave

   On tarball platforms, you install the new release in a different location, so the old files are not overwritten.

   Regardless of the platform, if you customized any other files, copy the files from the old installation to a safe location before performing an in-place upgrade that overwrites customized files.
4. Upgrade your installation according to these instructions:
   - Upgrading from DataStax Community 1.0.x to DataStax Enterprise 3.0.x:
     - Tarball
     - Debian/Ubuntu
     - RHEL-based platforms
     - Upgrading DataStax Enterprise real-time Cassandra nodes
     - Upgrading DSE Search/Solr nodes
     - Upgrading Analytics/Hadoop nodes
     - Upgrading legacy versions

**Order of upgrading nodes**

Observe the following order for upgrading nodes in a mixed workload cluster:

1. Analytics: Jobtracker, remaining seeds, remaining task trackers
2. Cassandra: Seeds, then remaining nodes
3. Solr: Seeds, then remaining nodes

DSE Search/Solr nodes are upgraded last because they are more sensitive to schema disagreements.

**Component version changes**

The versions of components that changed for each DataStax release are listed in the Release notes for that version. The cassandra.yaml file included in DSE 3.0.x contains security options that are not included in the Apache Cassandra 1.1.9 file.

For a complete list of Cassandra changes and new features, see:

**Upgrading data on disk**

Upgrading SSTables is highly recommended under any of these conditions:

- If you use counter columns
- If you are upgrading from Cassandra 1.0.x or earlier
- If you are upgrading from a DataStax Enterprise version having Cassandra 1.0.x or earlier

Upgrade SSTables before doing these operations:

- move
- repair
- bootstrap

Because these operations copy SSTables within the cluster and the on-disk format sometimes changes between major versions, upgrade SSTables now to prevent possible SSTable incompatibilities. Follow upgrade procedures in this document to upgrade SSTables at the end of the process.

- Tarball: <install_location>/bin/nodetool -h upgradesstables
- Package or AMI: nodetool -h upgradesstables
About expected schema disagreements

Between the time the first node in a cluster begins the upgrade process until the last node completes the process, a schema disagreement condition exists. Cassandra throws a SchemaDisagreementException when a schema disagreement occurs. This is expected behavior. Upgrade procedures include steps for resolving schema disagreements.

When the schema disagreement exists, client interfaces block the following operations:

- DDL
- TRUNCATE
- Solr queries

DDL, TRUNCATE, and Solr queries are not supported during a rolling restart. For example, during a rolling restart, these are the CQL commands that are and are not supported:

<table>
<thead>
<tr>
<th>OK to Run</th>
<th>Do Not Run</th>
<th>Do Not Run (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>ALTER TABLE</td>
<td>DROP TABLE</td>
</tr>
<tr>
<td>INSERT</td>
<td>CREATE TABLE</td>
<td>DROP INDEX</td>
</tr>
<tr>
<td>SELECT</td>
<td>CREATE INDEX</td>
<td>DROP KEYSPACE</td>
</tr>
<tr>
<td>UPDATE</td>
<td>CREATE KEYSPACE</td>
<td>TRUNCATE</td>
</tr>
</tbody>
</table>

Exception: After upgrading from DataStax Enterprise 2.1.x or earlier, Solr queries are not supported until all nodes are upgraded and schema disagreements are resolved.

Tapping NEWS.txt for upgrading information

NEWS.txt contains late-breaking information about upgrading from previous versions of Cassandra.

A NEWS.txt or a NEWS.txt archive is installed in the following locations:

- Tarball: `<install_location>/resources/cassandra`
- Package: `/usr/share/doc/dse-libcassandra`

NEWS.txt is also posted on the Apache Cassandra project web site.

Unpack NEWS.txt.gz if it is an archive. For example:

```
cd /usr/share/doc/dse-libcassandra-3.0.
sudo gunzip NEWS.txt.gz
```

Take a look at the information that is pertinent to your old version if there is any. For example, if you upgrade from some early versions, it might be necessary to upgrade SSTables.

Upgrading DataStax Enterprise real-time Cassandra nodes

To upgrade DataStax Enterprise 1.x.x - 2.2.x to 3.0.x, perform these upgrade steps on each node in the cluster. If the cluster is a mixed workload cluster, upgrade in the order described in Order of upgrading nodes. Complete all steps on one node before starting to upgrade the next node.

To upgrade a tarball release

Follow the instructions in Upgrading DataStax Community--tarball.

To upgrade a packaged release

1. Run the Yum (CentOS/RHEL/Oracle Linux) or Aptitude (Debian/Ubuntu) update commands.
2. Run the install commands shown in Installing the DataStax Enterprise package on Debian and Ubuntu or Installing the DataStax Enterprise package on RHEL-based distributions.

3. Start the first node.

4. Configure the node: Open the old cassandra.yaml. Open the new cassandra.yaml:
   - **Debian/Ubuntu:** /etc/dse/cassandra
   - **RHEL-based:** /etc/dse/cassandra/cassandra.yaml
     Diff the new and old cassandra.yaml files. Merge the diffs by hand from the old file to the new one except do not merge the snitch setting.

5. **Configure the snitch setting.**

6. Configure and start the node.

### Configuring and starting a node

A few configuration settings need to be made before completing the upgrade of a DataStax Enterprise or a DataStax Community node.

#### To configure and start the node

1. If you customized property files, other than the cassandra-topology.properties, update files by hand. Merge the settings of old property files, other than cassandra-topology.properties, into the new property files instead of overwriting the files. Users who overwrite property files, other than cassandra-topology.properties, have reported problems.

   It is ok to overwrite the old with the new cassandra-topology.properties file as instructed in Configuring the snitch setting.

2. **Start the node.**

3. If necessary, upgrade any CQL drivers and client libraries, such as python-cql, Hector, or Pycassa that are incompatible with the new DSE version. You can download CQL drivers and client libraries from the DataStax web site.

   The CQL utility is included in the DataStax Enterprise installation, so no upgrade of the CQL utility is necessary.

4. Restart client applications.

5. You can use a rolling restart to upgrade a cluster. Monitor the log files for any issues.

6. If you created column families using the default SizeTieredCompaction, continue to the next step. If you created column families having LeveledCompactionStrategy, scrub the SSTables that store those column families.

7. **Check for schema disagreements.**

8. If you meet conditions for upgrading SSTables, upgrade SSTables now.

### Configuring the snitch setting

Perform one of the following tasks, depending on the endpoint_snitch setting in your old cassandra.yaml file:

<table>
<thead>
<tr>
<th>endpoint_snitch URL</th>
<th>Upgrade Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.apache.cassandra.locator.SimpleSnitch</td>
<td>Leave the DseDelegateSnitch in the cassandra.yaml and the default delegated_snitch in the new dse.yaml unchanged.</td>
</tr>
</tbody>
</table>
Copy/paste the cassandra-topology.properties from the old installation to: `<install_location>/resources/cassandra/conf for binary installs, or to /etc/dse/cassandra for packaged installs. This action overwrites the new properties file. Set the delegated_snitch setting in the new dse.yaml file to: org.apache.cassandra.locator.PropertyFileSnitch.

| Any other snitch URL | Change the default delegated_snitch in the new dse.yaml file to your current snitch setting. |

The default delegated_snitch (com.datastax.bdp.snitch.DseSimpleSnitch) is specified in the new dse.yaml file. The dse.yaml is located in <install_location>/resources/dse/conf for binary installs or /etc/dse for packaged installs.

**Using a rolling restart**

Using a rolling restart, you upgrade and start one node at a time, instead of bringing down the entire cluster and starting all nodes at once. The procedures for a rolling restart of DSE Search/Solr and a rolling restart of Cassandra real-time nodes are different.

**Performing a rolling restart of Cassandra real-time nodes**

To perform a rolling restart of Cassandra real-time nodes:

1. Restart the first node.
2. Check for and resolve schema disagreements for the node.
3. Repeat steps 1 and 2 to restart and resolve schema disagreements for another node.

**Performing a rolling restart of Analytics/Hadoop Nodes**

You can restart an Analytics/Hadoop node using the same rolling restart as Cassandra real-time nodes, however, a rolling restart is not fully supported. Exceptions, which you can ignore, flood the log file.

The exceptions are caused by the Hadoop job tracker repeatedly logging exceptions until all analytics nodes are upgraded. If you can tolerate these exceptions being added to the log file, use the rolling restart. The runtime exceptions you might see when starting analytics nodes look something like these snippet.

```java

```

Ignore these exceptions. When the last node upgrades, restarts, and joins the cluster, the exceptions cease. As previously mentioned, upgrade and start the new job tracker node first.

**Post-upgrade problems?**

In the event of a post-upgrade problem, such as a schema disagreement, contact Support before attempting further DDL operations.

**Checking for and resolving schema disagreements**

When the output of DESCRIBE CLUSTER indicates a schema disagreement, or if a node is UNREACHABLE, perform these steps:
1. Using the Command Line Interface (CLI), run the DESCRIBE CLUSTER command. For example,

   $ cassandra-cli -host localhost -port 9160
   [default@unknown] DESCRIBE cluster;

   If any node is UNREACHABLE, you see output something like this:

   [default@unknown] describe cluster;
   Cluster Information:
   Snitch: com.datastax.bdp.snitch.DseDelegateSnitch
   Partitioner: org.apache.cassandra.dht.RandomPartitioner
   Schema versions:
   UNREACHABLE: [10.202.205.203, 10.80.207.102, 10.116.138.23]

2. Restart unreachable nodes.

3. Repeat steps 1 and 2 until the DESCRIBE cluster command shows that all nodes have the same schema version number: only one schema version appears in the output of DESCRIBE cluster.

Scrubbing SSTables

If you created column families having LeveledCompactionStrategy, you need to scrub the SSTables that store those column families.

First, upgrade all nodes to the latest version of DataStax Enterprise, according to the platform-specific instructions presented earlier in this document. Next, complete steps in Configuring and starting a node. At this point, all nodes are upgraded and started.

Finally, follow these steps to install the sstablescrub utility and scrub the affected SSTables:

Tarball Installations

Download the sstablescrub and dse.in.sh utilities.

   1. Place the downloaded sstablescrub script into the $DSE_HOME/bin directory.
   2. Copy dse.in.sh script to the $DSE_HOME/bin directory.

Packaged Installations (deb/rpm)

Download the sstablescrub and dse.in.sh

   1. Place the attached sstablescrub in the /usr/bin directory.
   2. Replace dse.in.sh in the /usr/share/dse directory with the version you downloaded.

   Note
   Do not replace dse-env.sh in the /etc/dse directory.

To scrub SSTables:

   1. Shut down the nodes, one-at-a-time.
   2. On each offline node, run the sstablescrub utility.

      For example, on a tarball installation:

      cd <install directory>/bin
      ./sstablescrub mykeyspace mycolumnfamily

      To get help about sstablescrub:

      usage: sstablescrub -h
3. Restart each node and client applications, one node at-a-time.

   4. **Check for schema disagreements.**

   If you do not scrub the affected SSTables, you might encounter the following error during compactions on column families using LeveledCompactionStrategy:

   ```
   ERROR [CompactionExecutor:150] 2012-07-05 04:26:15,570 AbstractCassandraDaemon.java (line 134)
   Exception in thread Thread[CompactionExecutor:150,1,main]
   java.lang.AssertionError
   at org.apache.cassandra.db.compaction.LeveledManifest.promote
   (LeveledManifest.java:214)
   ```

### Upgrading DSE Search/Solr nodes

To upgrade DataStax Enterprise 1.x.x - 2.2.x to 3.0.x, perform these upgrade steps on each node in the cluster. If the cluster is a mixed workload cluster, upgrade in the order described in *Order of upgrading nodes*. You need to restart the nodes as real-time Cassandra nodes before upgrading as described in this procedure. Restarting the nodes as real-time Cassandra nodes prevents unwanted schema changes from occurring when you start the upgraded node. Complete all steps on one node before starting to upgrade the next node.

#### Tarball release

1. Stop the first node to be upgraded and restart it in real-time Cassandra mode:
   ```
   dse cassandra
   ```

2. Create a directory for the new installation, download the tarball, and move it to that directory.

3. Unpack the DataStax Enterprise 3.0.x tarball in the new install location.
   ```
   tar -xzvf <dse-3.0.x tarball name>
   ```

4. If you customized the location of the data in the old installation, create a symbolic link to the old data directory:
   ```
   cd <new install location>
   ln -s <old data directory> <new install location>/<new data directory>
   ```

### To configure the upgraded node

1. In the new installation, open the cassandra.yaml for writing. The file is located in:
   ```
   <install location>/resources/cassandra/conf
   ```

2. In the old installation of Cassandra, open the cassandra.yaml. The file is located in:
   ```
   <install location>/conf
   ```

3. Diff the new and old cassandra.yaml files.

4. Merge the diffs by hand from the old file into the new one, except do not merge snitch settings.
   ```
   If you are migrating data and set up the symbolic link described in the previous procedure, ensure that you merge the data_file_directories, commitlog_directory, and saved_caches_directory properties correctly.
   ```

5. **Configure the snitch setting.**
Upgrading DSE Search/Solr nodes

6. If you customized property files, other than the cassandra-topology.properties, update files by hand. Merge the settings of old property files, other than cassandra-topology.properties, into the new property files instead of overwriting the files. Users who overwrite property files, other than cassandra-topology.properties, have reported problems.

   It is ok to overwrite the old with the new cassandra-topology.properties file as instructed in Configuring the snitch setting.

7. Start each node in real-time Cassandra mode during the rolling restart.

8. Check for schema disagreements on each node.

9. After all nodes are upgraded and the schemas agree, restart the nodes in Solr mode using rolling restart for Solr nodes that includes recovering the index.

10. Update the old Solr configuration files and indexes.

11. If you are upgrading from a version of DataStax Enterprise 3.0 or earlier, make the default legacy type mapping effective by commenting out the dseTypeMappingVersion element.

   <!-- <dseTypeMappingVersion>1</dseTypeMappingVersion> -->

12. Repeat the previous steps for all nodes. Monitor the log files for any issues.

Packaged release

1. Stop the dse service, and then disable Solr by setting the options in /etc/default/dse: SOLR_ENABLED=0

2. Restart the dse service.

3. Run the Yum (CentOS/RHEL/Oracle Linux) or Aptitude (Debian/Ubuntu) update commands.

4. Run the install commands shown in Installing the DataStax Enterprise package on Debian and Ubuntu or Installing the DataStax Enterprise package on RHEL-based distributions.

5. Start the first node.

6. Configure the node: Open the old cassandra.yaml. Open the new cassandra.yaml:

   Debian/Ubuntu: /etc/dse/cassandra
   RHEL-based: /etc/dse/cassandra/cassandra.yaml

   Diff the new and old cassandra.yaml files. Merge the diffs by hand from the old file to the new one except do not merge the snitch setting.

7. Configure the snitch setting.

8. If you customized property files, other than the cassandra-topology.properties, update files by hand. Merge the settings of old property files, other than cassandra-topology.properties, into the new property files instead of overwriting the files. Users who overwrite property files, other than cassandra-topology.properties, have reported problems.

   It is ok to overwrite the old with the new cassandra-topology.properties file as instructed in Configuring the snitch setting.

9. Start up each node in real-time Cassandra mode during the rolling restart (Solr mode disabled).

10. Check for schema disagreements.

11. After all nodes are upgraded and the schemas agree, reset the mode from real-time Cassandra to Solr.

To reset the mode to Solr

1. Stop the dse service, and then enable Solr by setting this option in /etc/default/dse: SOLR_ENABLED=1

2. Start the dse service using a rolling restart for Solr nodes that includes recovering the index.
3. **Update the old Solr configuration files and indexes.**

4. If you are upgrading from a version of DataStax Enterprise 3.0 or earlier, make the default legacy type mapping effective by commenting out the dseTypeMappingVersion element.

   ```xml
   <!-- <dseTypeMappingVersion>1</dseTypeMappingVersion> -->
   ``

5. Repeat the previous steps for each node. Monitor the **log files** for any issues.

### Performing a rolling restart of DSE Search/Solr nodes

To perform a rolling restart, restart each node in a rolling fashion, one node at a time, instead of bringing down the entire cluster and starting all nodes at once. If you are upgrading from a 2.x release, you likely will see error messages that resemble those shown in the list of Solr errors. If you get error messages, you typically recreate the core using the recovery option of the CREATE command. The recovery option is used when the core refuses to load due problems, such as incompatible changes during an upgrade or serious index errors. The default is false.

This diagram depicts the steps in the following procedure:

![Diagram of rolling restart process](image)

**To restart the cluster using a rolling restart**

1. Restart the first node.

   If you do not see errors, proceed to step 2; otherwise, skip step 2 and do step 3 (run the CREATE OR RELOAD command).

2. Check the Solr admin console to see if your cores are listed. If your cores are up, skip step 3 and do step 4 (check/resolve schema disagreements); otherwise, do step 3.

3. Run the CREATE command to recover and not to distribute the core for each index.

   For example, to run the CREATE command for the demo, wikipedia, and log_search indexes, on the node you've just upgraded:

   ```bash
   curl -v "http://localhost:8983/solr/admin/cores?action=CREATE&name=demo.solr&recovery=true&distributed=false"
   curl -v "http://localhost:8983/solr/admin/cores?action=CREATE&name=wiki.solr&recovery=true&distributed=false"
   curl -v "http://localhost:8983/solr/admin/cores?action=CREATE&name=Logging.log_entries&recovery=true&distributed=false"
   ``

   If you get an error message telling you that the core already exists, run the **RELOAD command** instead of the CREATE command.
4. **Check for and resolve schema disagreements** for the node. Monitor the log files for any issues.

5. If you use counters or Cassandra 1.1 or earlier, **upgrade SSTables**.

6. Repeat the procedure for the next node.

**Expected Solr error messages during the upgrade**

When installing DataStax Enterprise 3.0.x, you may see errors, such as:

```
ERROR 00:57:17,785 Cannot activate core: ks.cf_10000_keys_50_cols
ERROR 00:57:17,786 <indexDefaults> and <mainIndex> configuration sections are discontinued.
   Use <indexConfig> instead.
```

```
ERROR 01:29:55,145 checksum mismatch in segments file (resource:
   ChecksumIndexInput(MMapIndexInput(path="/var/lib/cassandra/data/solr.data/ks.
   cf_10000_keys_50_cols/index/segments_6")))
ERROR 01:29:55,145 Solr index ks.cf_10000_keys_50_cols seems to be corrupted:
   please CREATE the core again with recovery=true to start reindexing data.
ERROR 01:29:55,146 checksum mismatch in segments file (resource: ChecksumIndexInput
   (MMapIndexInput(path="/var/lib/cassandra/data/solr.data/ks.
   cf_10000_keys_50_cols/index/segments_6")))
org.apache.lucene.index.CorruptIndexException: checksum mismatch in segments file
   (resource: ChecksumIndexInput(MMapIndexInput
      ( path="/var/lib/cassandra/data/solr.data/ks.cf_10000_keys_50_cols/index/segments_6")))
```

**Updating Solr configuration and indexes**

After upgrading a DSE Search/Solr node, you might need to update the old solrconfig.xml file and Solr indexes to run on DataStax Enterprise 3.0. If a DSE Search/Solr node hangs and the system.log *shows errors*, you need to modify your custom solrconfig.xml file to make it Solr 4.0.x-compliant.

**To modify your custom Solr configuration file**

1. Open the system.log and look for the message about the Solr error.
   
   The error message briefly describes the changes you need to make.

2. Correct these errors in your solrconfig.xml files, then post the corrected files.
   
   Existing cores cannot be loaded until the solrconfig.xml errors are resolved.

3. HTTP-POST the modified Solr configuration file and recover indexes as described in the next procedure.

**To HTTP-POST modified solrconfig.xml files and recover indexes**

This procedure uses as examples the demo solrconfig.xml files included with DataStax Enterprise 3.0.

1. Go to the directory containing your Solr app. For example, go to the demos directory:

   - Binary installation
     
     `cd <install_location>/demos`

   - Package installation
     
     `cd /usr/share/dse-demos`
2. Run the following commands to HTTP-POST your modified custom solrconfig.xml to DSE-Search. For example, from the demos or dse-demos directory, run the following commands:

   - From the solr_stress directory
     
     ```
     curl -v --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8'
     ```
   
   - From the wikipedia directory
     
     ```
     curl -v --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8'
     ```
   
   - From the log_search directory
     
     ```
     curl -v --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8'
     ```

   After running each curl command, a SUCCESS message appears.

   This step is only required once, when the first node is upgraded.

3. If you use a rolling restart, run the CREATE command with the recovery and distributed (false) options for each index as shown in rolling restart.

   If you do not want to use rolling restart, you can run these commands on one node after the other by upgrading all nodes, bringing down the entire cluster, and restarting all nodes. These commands use a different value--true, which is the default, for the distributed option.

   ```
   curl -v "http://localhost:8983/solr/admin/cores?action=CREATE&name=demo.solr&recovery=true"
   curl -v "http://localhost:8983/solr/admin/cores?action=CREATE&name=wiki.solr&recovery=true"
   curl -v "http://localhost:8983/solr/admin/cores?action=CREATE&name=Logging.log_entries&recovery=true"
   ```

4. Monitor the log files for any issues.

5. Check for schema disagreements.

6. If necessary, upgrade any CQL drivers and client libraries, such as python-cql, Hector, or Pycassa that are incompatible with the new DSE version. You can download CQL drivers and client libraries from the DataStax download page.

   The CQL utility is included in the DataStax Enterprise installation, so no upgrade of the CQL utility is necessary.

7. Restart client applications.

8. If you created column families using the default SizeTieredCompaction, continue to the next step. If you created column families having LeveledCompactionStrategy, scrub the SSTables that store those column families.

9. Check for schema disagreements.

10. If you meet conditions for, upgrading SSTables, upgrade SSTables now.

    By default, DataStax Enterprise 3.0.x enables legacy DSE Search/Solr type mapping. You can configure the solrconfig.xml to use the new type mapping.

### Upgrading Analytics/Hadoop nodes

To upgrade DataStax Enterprise 1.xx - 2.2.x to 3.0.x, perform these upgrade steps on each node in the cluster. If the cluster is a mixed workload cluster, upgrade in the order described in Order of upgrading nodes. You need to restart the nodes as real-time Cassandra nodes before upgrading as described in this procedure. Restarting the nodes as real-time Cassandra nodes prevents unwanted schema changes from occurring when you start the upgraded node. Complete all steps on one node before starting to upgrade the next node.
Tarball release

1. Stop the first node to be upgraded and restart it in real-time Cassandra mode:
   
   dse cassandra

2. Create a directory for the new installation, download the tarball, and move it to that directory.

3. Unpack the DataStax Enterprise 3.0.x tarball in the new install location.
   
   tar -xzvf <dse-3.0.x tarball name>

4. If you customized the location of the data in the old installation, create a symbolic link to the old data directory:
   
   cd <new install location>
   ln -s <old data directory> <new install location>/<new data directory>

To configure the upgraded node

1. In the new installation, open the cassandra.yaml for writing. The file is located in:
   
   <install location>/resources/cassandra/conf

2. In the old installation of Cassandra, open the cassandra.yaml. The file is located in:
   
   <install location>/conf

3. Diff the new and old cassandra.yaml files.

4. Merge the diffs by hand from the old file into the new one, except do not merge snitch settings.

   If you are migrating data and set up the symbolic link described in the previous procedure, ensure that you merge the data_file_directories, commitlog_directory, and saved_caches_directory properties correctly.

5. Configure the snitch setting.

6. If you customized property files, other than the cassandra-topology.properties, update files by hand. Merge the settings of old property files, other than cassandra-topology.properties, into the new property files instead of overwriting the files. Users who overwrite property files, other than cassandra-topology.properties, have reported problems.

   It is ok to overwrite the old with the new cassandra-topology.properties file as instructed in Configuring the snitch setting.

7. Start each node in real-time Cassandra mode during the rolling restart.

8. Check for schema disagreements on each node.

9. After all nodes are upgraded and the schemas agree, restart the nodes in Hadoop mode using a rolling restart.

10. In DataStax 3.0, the ownership of the Hadoop mapred staging directory in the CassandraFS changed. After upgrading, you need to set the owner of /tmp/hadoop-<dseuser>/mapred/staging to the dse user. For example, if you run DataStax Enterprise 3.0.x as root, use the following command on Linux:

    dse hadoop fs -chown root /tmp/hadoop-root/mapred/staging

11. If you created column families using the default SizeTieredCompaction, continue to the next step. If you created column families having LeveledCompactionStrategy, scrub the SSTables that store those column families.

12. Check for schema disagreements again.

13. If you meet conditions for upgrading SSTables, upgrade SSTables now.

Packaged release
1. Stop the dse service, and then disable Hadoop by setting options in /etc/default/dse: HADOOP_ENABLED=0
2. Restart the dse service.
3. Run the Yum (CentOS/RHEL/Oracle Linux) or Aptitude (Debian/Ubuntu) update commands.
4. Run the install commands shown in Installing the DataStax Enterprise package on Debian and Ubuntu or Installing the DataStax Enterprise package on RHEL-based distributions.
5. Start the first node.
6. Configure the node: Open the old cassandra.yaml. Open the new cassandra.yaml:
   - Debian/Ubuntu: /etc/dse/cassandra
   - RHEL-based: /etc/dse/cassandra/cassandra.yaml
   Diff the new and old cassandra.yaml files. Merge the diffs by hand from the old file to the new one except do not merge the snitch setting.
7. Configure the snitch setting.
8. If you customized property files, other than the cassandra-topology.properties, update files by hand. Merge the settings of old property files, other than cassandra-topology.properties, into the new property files instead of overwriting the files. Users who overwrite property files, other than cassandra-topology.properties, have reported problems.
   It is ok to overwrite the old with the new cassandra-topology.properties file as instructed in Configuring the snitch setting.
9. Start up each node in real-time Cassandra mode during the rolling restart (Hadoop mode disabled).
10. Check for schema disagreements.
11. After all nodes are upgraded and the schemas agree, reset the mode from real-time Cassandra to Hadoop.

To restart the mode to Hadoop
1. Stop the dse service, and then enable Hadoop by setting this option in /etc/default/dse: HADOOP_ENABLED=1
2. Start the dse service using a rolling restart.
3. In DataStax 3.0, the ownership of the Hadoop mapred staging directory in the CassandraFS changed. After upgrading, you need to set the owner of /tmp/hadoop-<dseuser>/mapred/staging to the dse user. For example, if you run DataStax Enterprise 3.0.x as root, use the following command on Linux:

   dse hadoop fs -chown root /tmp/hadoop-root/mapred/staging
4. Repeat the previous steps for each node. Monitor the log files for any issues.
5. If you created column families using the default SizeTieredCompaction, continue to the next step. If you created column families having LeveledCompactionStrategy, scrub the SSTables that store those column families.
6. Check for schema disagreements.
7. If you meet conditions for upgrading SSTables, upgrade SSTables now.

Upgrading DataStax Community--tarball

This procedure shows how to upgrade a cluster of nodes from DataStax Community 1.0.x/1.1 to DataStax Enterprise 3.0.x.

Upgrading a node and migrating the data

Perform these upgrade steps on each node in the cluster. If the cluster is a mixed workload cluster, upgrade in the order described in Order of upgrading nodes. Complete all steps on one node before starting to upgrade the next node.
To upgrade a node and migrate the data

1. Create a directory for the new installation, download the tarball, and move it to that directory.

   tar -xzvf <dse-3.0.x tarball name>

2. Unpack the DataStax Enterprise 3.0.x tarball in the new install location.

   3. If you customized the location of the data in the old installation, create a symbolic link to the old data directory:

      cd <new install location>
      ln -s <old data directory> <new install location>/<new data directory>

To configure the upgraded node

1. In the new installation, open the cassandra.yaml for writing. The file is located in:

   <install location>/resources/cassandra/conf

2. In the old installation of Cassandra, open the cassandra.yaml. The file is located in:

   <install location>/conf

3. Diff the new and old cassandra.yaml files.

4. Merge the diffs by hand from the old file into the new one, except do not merge snitch settings.

   If you are migrating data and set up the symbolic link described in the previous procedure, ensure that you merge the data_file_directories, commitlog_directory, and saved_caches_directory properties correctly.

5. Configure the snitch setting.

6. Configure and start the node.

7. If you meet conditions for, upgrading SSTables, upgrade SSTables now.

Upgrading DataStax Community--Debian/Ubuntu

This procedure shows how to upgrade a cluster of Ubuntu 11.10 nodes from DataStax Community 1.0.x/1.1 to DataStax Enterprise 3.0.x. The installer migrates the data from the old to the new version of Cassandra automatically.

Upgrading a node

Perform these upgrade steps on each node in the cluster. If the cluster is a mixed workload cluster, upgrade in the order described in Order of upgrading nodes. Complete all steps on one node before starting to upgrade the next node.

To upgrade a node

1. Remove the old packages and dependencies on them.

   sudo apt-get remove dsc cassandra
   sudo apt-get autoremove

   This action shuts down Cassandra on the node.

2. Add the DataStax repository to the /etc/apt/sources.list.

   deb http://<username>:@<password>@debian.datastax.com/enterprise stable main

   Get the <username> and <password> from your DataStax registration confirmation email. If you don’t have the email, register on the DataStax web site.
3. Upgrade the node.
   ```
sudo apt-get update
sudo apt-get install dse-full
   ```

4. If the prompt appears informing you of the disk space to be used, type Y to continue.

5. Configure the upgraded node as described in the next section.

To configure the upgraded node

1. Open the old cassandra.yaml located in `/etc/cassandra`.
2. Open the new cassandra.yaml located in `/etc/dse/cassandra`.
3. Diff the new and old cassandra.yaml files, merging the diffs by hand from the old file into the new one, except do not merge snitch settings.
4. **Configure the snitch setting.**
5. **Configure and start the node.**
6. If you meet conditions for upgrading SSTables, upgrade SSTables now.

---

### Upgrading DataStax Community--RHEL-based distributions

This procedure shows how to upgrade a cluster of RHEL-based nodes from DataStax Community 1.0.x/1.1 to DataStax Enterprise 3.0.x on the following systems:

- CentOS systems
- Oracle Linux
- Red Hat Enterprise Linux (RHEL)

The installer migrates the data from the old to the new version of Cassandra automatically.

---

### Upgrading a Node

Perform these upgrade steps on each node in the cluster. If the cluster is a mixed workload cluster, upgrade in the order described in *Order of upgrading nodes*. Complete all steps on one node before starting to upgrade the next node.

To upgrade a node

1. Remove dependencies on the old version of the package:
   ```
   rpm -e apache-cassandra1 -noscripts
   ```
   
   The old Cassandra configuration file is renamed to cassandra.yaml.rpmsave as shown in the output of this command.

   ```
   warning: /etc/cassandra/default.conf/cassandra.yaml saved as /etc/cassandra/default.conf/cassandra.yaml.rpmsave
   ```

2. Open the yum repository file for DataStax in `/etc/yum.repos.d` for editing.
   ```
   sudo vi /etc/yum.repos.d/datastax.repo
   ```
3. Replace the contents of the file with the following lines:

   [datastax]
   name= DataStax Repo for Apache Cassandra \
   baseurl=http://<username>:<password>@rpm.datastax.com/enterprise
   enabled=1
   gpgcheck=0

4. Upgrade the node.

   sudo yum clean all
   sudo yum install dse-full

   A prompt informs you of the download size and asks for confirmation to continue. For example:

   Total download size: 252 M
   Is this ok [y/N]: y

5. Type y.

6. Configure the upgraded node as described in the next section.

To configure the upgraded node

1. Open the old, renamed configuration file.
   
   vi /etc/cassandra/default.conf/cassandra.yaml.rpmsave

2. Open the new configuration file for editing.
   
   sudo vi /etc/dse/cassandra/cassandra.yaml

3. Diff the new and old cassandra.yaml files.

4. Merge the diffs by hand from the old file to the new one except do not merge the snitch setting.

5. Configure the snitch setting.

6. Configure and start the node.

7. If you meet conditions for upgrading SSTables, upgrade SSTables now.

Upgrading legacy versions

The procedures listed in these sections are based on upgrading from DataStax Community 1.0.x to DataStax Enterprise 3.0.x:

- Upgrading DataStax Community--tarball
- Upgrading DataStax Community--Debian/Ubuntu
- Upgrading DataStax Community--RHEL-based distributions

Generally, these procedures work for upgrading from other releases of Cassandra or DataStax Community to DataStax Enterprise 3.0.x. Exceptions are:

- Upgrading directly from versions 0.8.8 and 1.0.3-1.0.5 to 3.0.x does not work. These older versions generate cross-dc forwarding and are incompatible with Cassandra 1.1, a component of DataStax Enterprise 3.0.x. First upgrade to 0.8.11 or 1.0.11, then upgrade to 3.0.x, and upgrade SSTables.
- To upgrade from a Brisk release, contact Support.
- To upgrade from Cassandra 0.8 or earlier, refer to NEWS.txt.
Upgrading the DataStax AMI

Before upgrading, be sure to make a backup. After upgrading, read NEWS.txt, to learn about any late-breaking upgrade information.

Note

If you have analytics nodes in the cluster, upgrade and restart the job tracker node first.

1. On each node ensure that the DataStax repository is listed in the /etc/apt/sources.list:
   
   ```
   deb http://<username>:<password>@debian.datastax.com/enterprise stable main
   ```

   where <username> and <password> are the DataStax account credentials from your registration confirmation email.

2. If necessary, add the DataStax repository key to your aptitude trusted keys.

   ```
   $ curl -L http://debian.datastax.com/debian/repo_key | sudo apt-key add -
   ```

3. On each node, run the following command:

   ```
   $ sudo apt-get update
   $ sudo apt-get install dse-full
   ```

4. Compare the new and old version of the cassandra.yaml file and other property files that may have changed in /etc/dse directory.

   After installing the upgrade, a backup of the cassandra.yaml is created in the /etc/dse/cassandra directory. Use this copy to compare the two configurations and make appropriate changes.

   a. Diff the following configuration files:

      • The cassandra.yaml from the old installation
      • The new DSE 3.0 cassandra.yaml

   b. Merge the versions by hand from the old cassandra.yaml into the new DSE 3.0 cassandra.yaml:

      Don’t add snitch settings from the old file to the new file. The new default snitch in the cassandra.yaml is com.datastax.bdp.snitch.DseDelegateSnitch. In previous versions, the default snitch was com.datastax.bdp.snitch.DseSimpleSnitch.

      Don’t copy property files from the prior release and overwrite files in the new release. Users who have attempted this have reported problems.

5. Configure the snitch setting as described in Configuring the snitch setting.

6. If necessary, upgrade any CQL drivers and client libraries, such as python-cql, Hector, or Pycassa that are incompatible with the new DSE version. You can download CQL drivers and client libraries from the DataStax download page.

7. Run nodetool drain to flush the commit log.

8. Restart the node:

   ```
   sudo service dse restart
   ```

9. Restart client applications.
Security

Security management

DataStax Enterprise 3.0 and later includes a number of features for securing data. The security framework provides advanced data protection for enterprise-grade databases. You can secure a DataStax Community or DataStax Enterprise cluster using these features.

- **Internal authentication** using login accounts and passwords
- **Object permission management** based on the GRANT/REVOKE paradigm
- **Client to node encryption** using SSL for data going from the client to the Cassandra cluster

DataStax Enterprise offers additional security, not included in DataStax Community, to enterprises for mission-critical data:

- **Kerberos authentication**: a network authentication protocol that allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner using tickets.
- **Transparent data encryption**: the encoding of data flushed from the memtable in system memory to the SSTables on disk (at rest data) to be unreadable to unauthorized users. Encryption and decryption occurs without user intervention.
- **Data auditing**: the administrator capability to create detailed audit trails of cluster activity.

DataStax Enterprise 3.0 uses the versions of CQL available in Cassandra 1.1.x and backports additional CQL security commands from later versions of Cassandra.

Limitations

Assuming you configure security features, this table describes exactly which data is secured (or not) based on the workload type: real-time Cassandra (DSE/Cassandra), analytics (Hadoop), and DSE/Search (Solr).

<table>
<thead>
<tr>
<th>Feature</th>
<th>DSE/Cassandra</th>
<th>Hadoop</th>
<th>Solr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal authentication</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Kerberos authentication</td>
<td>Yes [5]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data auditing</td>
<td>Yes</td>
<td>Partial [8]</td>
<td>Partial [8]</td>
</tr>
</tbody>
</table>

[1] Permissions to access objects stored in Cassandra are checked. The Solr cache and indexes and the Hadoop cache are not under control of Cassandra, and therefore are not checked. You can, however, set up permission checks to occur on column families that store Hadoop or Solr data.

[2] The inter-node gossip protocol is protected using SSL.

[3] The Thrift interface between Hadoop and the Cassandra File System (CFS) is SSL-protected. Inter-tracker communication is Kerberos authenticated, but not SSL secured. Hadoop access to Cassandra is SSL- and Kerberos-protected.

[4] HTTP access to the DSE Search/Solr data is protected using SSL. Node-to-node encryption using SSL protects internal Solr communication.

[5] The inter-node gossip protocol is not authenticated using Kerberos. Node-to-node encryption using SSL can be used.
Security

[6] Cassandra commit log data is not encrypted, only at rest data is encrypted.

[7] Data in DSE/Search Solr column families is encrypted by Cassandra. Encryption has a slight performance impact, but ensures the encryption of original documents after Cassandra permanently stores the documents on disk. However, Solr cache data and Solr index data (metadata) is not encrypted.

[8] Hadoop and Solr data auditing is done at the Cassandra access level, so requests to access Cassandra data is audited. Node-to-node encryption using SSL protects communication over inter-node gossip protocol.

**Using Kerberos and SSL at the same time**

Both the Kerberos and SSL libraries provide authentication, encryption, and integrity protection:

- **Kerberos**: If you enable Kerberos authentication, integrity protection is also enabled. However, you can enable integrity protection without encryption.

- **SSL**: If you use SSL, authentication, integrity protection, and encryption are all enabled or disabled.

- **Kerberos and SSL**: It is possible to enable both Kerberos authentication and SSL together. However, this causes some overlap because authentication is performed twice by two different schemes: Kerberos authentication and certificates through SSL. DataStax recommends choosing one and using it for both encryption and authentication. These settings are described in the `dse.yaml` configuration file.

**Securing DSE Search services**

The `security table` summarizes the security features of DSE Search/Solr and other integrated components. DSE Search data is completely or partially secured by using these DataStax Enterprise security features:

- **Object permission management**
  Access to Solr documents, excluding cached data, can be limited to users who have been granted access permissions. Permission management also secures tables used to store Solr data.

- **Transparent data encryption**
  Data at rest in Cassandra tables, excluding cached and Solr-indexed data, can be encrypted. Encryption occurs on the Cassandra side and impacts performance slightly.

- **Client-to-node encryption**
  You can encrypt HTTP access to Solr data and internal, node-to-node Solr communication using SSL. Enable SSL node-to-node encryption on the Solr node by setting encryption options in the `dse.yaml` file as described in **Client-to-node encryption**.

- **Kerberos authentication**
  You can authenticate DSE Search users through Kerberos authentication using Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO). To use the SolrJ API against DSE Search clusters with Kerberos authentication, client applications should use the SolrJ-Auth library and the DataStax Enterprise SolrJ component as described in the `solrj-auth-README.md` file.

You can also use **HTTP Basic Authentication**, but this is not recommended.

**Configuring HTTP Basic Authentication for Solr**

When you enable Cassandra's internal authentication by specifying `authenticator: com.datastax.bdp.cassandra.auth.PasswordAuthenticator` in `cassandra.yaml`, clients must use **HTTP Basic Authentication** to provide credentials to Solr services. Due to the stateless nature of HTTP Basic Authentication, this can have a significant performance impact as the authentication process must be executed on each HTTP request. For this reason, DataStax does not recommend using internal authentication on DSE Search clusters in production. To secure DSE Search in production, enable DataStax Enterprise **Kerberos** authentication.

To configure DSE Search to use Cassandra's internal authentication, follow this configuration procedure:
To configure password authentication for accessing DSE Search/Solr data:

1. Uncomment the PasswordAuthenticator in cassandra.yaml to enable HTTP Basic authentication for Solr.
   ```
   #authentication backend, implementing IAuthenticator; used to identify users
   authenticator: org.apache.cassandra.auth.AllowAllAuthenticator
   authenticator: com.datastax.bdp.cassandra.auth.PasswordAuthenticator
   authenticator: com.datastax.bdp.cassandra.auth.KerberosAuthenticator
   ```

2. Configure the replication strategy for the dse_auth keyspace.

3. Start the server.

4. Open a browser, and go to the service web page, for example http://localhost:8983/demos/wikipedia/.
   The browser asks you for a Cassandra username and password.

**Authenticating a DataStax Enterprise cluster with Kerberos**

This section provides information about configuring security for a DataStax Enterprise (DSE) cluster using Kerberos.

Kerberos is a computer network authentication protocol that allows nodes communicating over a non-secure network to prove their identity to one another in a secure manner using tickets. This section does not provide detailed information on installing and setting up Kerberos. For this information, see the [MIT Kerberos Consortium](https://kerberos.org).

**Note**

When using Kerberos security, you need to be aware of the scope of Kerberos tickets. Using the su or sudo command leaves any existing credentials behind and requires you to re-authenticate as that new user. If you encounter authentication issues, please ensure that you have a proper Kerberos ticket.

For information about using Kerberos with SSL, see [Using Kerberos and SSL at the same time](https://www.datastax.com/documentation/datastax-enterprise/5.1/datastax-enterprise/deploy/security/ssl/kerberos).

**Kerberos Recommendations**

The following are general guidelines for setting up Kerberos:

- Before installing DSE, set up your Kerberos servers.
- Set up several machines as authentication servers (Key Distribution Center [KDC]). One will be the primary or administration KDC, the others will be secondary.
- Do not install the KDC servers on DSE nodes.
- Set up firewalls on each KDC server.
- Physically protect the KDC machines.
- Secure the keytab files owned by the user running DSE. The file should be readable and writeable only by the owner without permissions for any other user (chmod 0600).

**AES-256 support**

Currently, DataStax Enterprise does not support AES-256 because of security issues. You must remove it as an allowed cypher for each principal in one of the following ways:

- Before creating the principals, use the `-e` flag to specify encryption:salt type pairs. For example: `-e "arcfour-hmac: normal des3-hmac-sha1: normal"`. This method requires Kerberos 5-1.2 on the KDC.
- After creating the principals, modify the Kerberos principals using the `-e` flag as described above and then recreate the keytab file. This method requires Kerberos 5-1.2 on the KDC.
• Modify the /etc/krb5kdc/kdc.conf file by removing any entries containing aes256 from the supported_enctypes variable for the realm in which the DSE nodes are members.

**Securing DataStax Enterprise nodes**

Do not upgrade DataStax Enterprise and set up Kerberos at the same time; see *Limitations on use during the upgrade.*

To set up Kerberos on your DSE nodes, do the following on every node:

1. **Install the Kerberos client software.**
2. **Use Kerberos to generate one keytab file for each node:**

   ```bash
   kadmin -p <username>/admin
   addprinc -randkey dse/<FQDN>
   addprinc -randkey HTTP/<FQDN>
   ktadd -k dse.keytab dse/<FQDN>
   ktadd -k dse.keytab HTTP/<FQDN>
   quit
   ```

   • `-randkey` creates a random password.
   • `-ktadd -k` creates a keytab for the dse and HTTP principals; `-k` specifies the keytab file name. In this example, the keytab entry is added to the dse.keytab file in the current directory.
3. **In the cassandra.yaml configuration file, set the authenticator:**

   ```yaml
   authenticator: com.datastax.bdp.cassandra.auth.KerberosAuthenticator
   ```
4. **Change the replication strategy and default replication factor for the dse_auth keyspace.** See *Configuring dse_auth keyspace replication.*

   DataStax recommends configuring dse_auth keyspaces for fault tolerance (in case of failure). In a multi-node cluster, if the node storing the user data goes down, using the default replication factor of 1 for the dse_auth keyspace precludes logging into any secured node.
5. Set the DSE service principals, keytab location, and qop (Quality of Protection) in the `dse.yaml` configuration file:

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

   - Set the `service_principal` that the Cassandra and Hadoop processes run under. It must use the form `<dse_user>/_HOST@REALM`, where `<dse_user>` is `cassandra` in package installs (the name of the user running the service) and the name of the UNIX user that starts the service in binary installs. It must be consistent everywhere: in the `dse.yaml`, present in the keytab, and in the `.cqlshrc` file (where it is separated into the `<service>/<hostname>`).
   - Leave `_HOST` as is. The name before the "/" must match the name in the service principal. DSE automatically substitutes the FQDN (Fully Qualified Domain Name) of the host where it runs. There must be credentials for this principal in the keytab file and readable by the user that Cassandra runs as, usually `cassandra`.
   - The `http_principal` is used by the application container, which is tomcat and used to run Solr. The web server uses GSS-API mechanism (SPNEGO) to negotiate the GSSAPI security mechanism (Kerberos). To set up password authentication for a DSE Search/Solr node, follow instructions in Running the demo on a secure cluster.
   - The `qop` is a comma delimited list of Quality of Protection 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 available settings 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.

### Enabling and disabling Kerberos security

After setting up Kerberos as described above, you can turn it on and off by changing the authenticator in the `cassandra.yaml` file:

- On: `com.datastax.bdp.cassandra.auth.KerberosAuthenticator`
- Off: any other authenticator

### Using cqlsh with Kerberos security

To use cqlsh with Kerberos, you must install the `python-kerberos` and `python-pure-sasl` packages, and create a `.cqlshrc` file in your home or client program directory. See Installing the cqlsh security packages.

### Client-to-node encryption

Client-to-node encryption protects data in flight from client machines to a database cluster. It establishes a secure channel between the client and the coordinator node. Unlike Kerberos, SSL is fully distributed and does not require setting up a shared authentication service. For information about generating SSL certificates, see Preparing server certificates.

### SSL settings for DataStax Enterprise client-to-node encryption

To enable client-to-node SSL, you must set the client encryption options in the `dse.yaml` file.
Node-to-node encryption

On each node, under client_encryption_options:

- Enable encryption.
- Set the paths to your .keystore and .truststore files.
- Provide the passwords used when generating the keystore and truststore.

```yaml
client_encryption_options:
  enabled: true
  keystore: resources/dse/conf/.keystore
  keystore_password: <keystore password>
  keystore_type: JKS
  truststore: resources/dse/conf/.truststore
  truststore_password: <truststore password>
```

For information about using Kerberos with SSL, see Using Kerberos and SSL at the same time.

**Initializing Solr to support SSL encryption**

When you enable SSL in the dse.yaml, it automatically enables the authentication/authorization filters in Solr web.xml and configures an SSL connector in Tomcat. This means that you don't have to change your web.xml or server.xml.

**Node-to-node encryption**

Node-to-node encryption protects data transferred between nodes in a cluster using SSL (Secure Sockets Layer). For information about generating SSL certificates, see Preparing server certificates.

**SSL settings for node-to-node encryption**

To enable node-to-node SSL, you must set the encryption options in the cassandra.yaml file.

On each node, under encryption_options:

- Enable the internode_encryption options (described below).
- Set the appropriate paths to your .keystore and .truststore files.
- Provide the required passwords. The passwords must match the passwords used when generating the keystore and truststore.
- To enable peer certificate authentication, set require_client_auth to true.

The available inter-node options are:

- **all**
- **none**
- **dc**: Cassandra encrypts the traffic between the data centers.
- **rack**: Cassandra encrypts the traffic between the racks.

```yaml
encryption_options:
  internode_encryption: <internode_option>
  keystore: resources/dse/conf/.keystore
  keystore_password: <keystore password>
  truststore: resources/dse/conf/.truststore
  truststore_password: <truststore password>
  require_client_auth: <true or false>
```
Preparing server certificates

This topic provides information about generating SSL certificates for client-to-node encryption or node-to-node encryption. If you generate the certificates for one type of encryption, you do not need to generate them again for the other: the same certificates are used for both.

All nodes must have all the relevant SSL certificates on all nodes. A keystore contains private keys. The truststore contains SSL certificates for each node and doesn't require signing by a trusted and recognized public certification authority.

To prepare server certificates:

1. Generate the private and public key pair for the nodes of the cluster.
   A prompt for the new keystore and key password appears.
2. Leave key password the same as the keystore password.
3. Repeat steps 1 and 2 on each node using a different alias for each one.
   
   keytool -genkey -alias <dse_node0> -keystore .keystore

4. Export the public part of the certificate to a separate file and copy these certificates to all other nodes.
   
   keytool -export -alias dse -file dsenode0.cer -keystore .keystore

5. Add the certificate of each node to the truststore of each node, so nodes can verify the identity of other nodes.
   A prompt for setting a password for the newly created truststore appears.
   
   keytool -import -v -trustcacerts -alias <dse_node0> -file <dse_node0>.cer -keystore .truststore
   keytool -import -v -trustcacerts -alias <dse_node1> -file <dse_node1>.cer -keystore .truststore
   . . .
   keytool -import -v -trustcacerts -alias <dse_nodeN> -file <dse_nodeN>.cer -keystore .truststore

6. Distribute the .keystore and .truststore files to all DSE nodes.
7. Make sure .keystore is readable only to the DSE daemon and not by any user of the system.

Installing the cqlsh security packages

To use cqlsh with a Kerberized cluster, you must install the PyKerberos and python-pure-sasl packages. The PyKerberos package is a high-level wrapper for Kerberos (GSSAPI) operations. The python-pure-sasl package is a pure Python client-side SASL (Simple Authentication and Security Layer) implementation.

Adding the required modules

The steps required for adding the necessary modules depends on the type of install:

Debian-based installs:

$ sudo apt-get install python-pure-sasl

RHEL installs:

# yum install python-pure-sasl

Binary installs:

To install from the binary tarball, you first install the PyKerberos module and then the pure-sasl module. DataStax recommends using APT or Yum because installing the dependencies can be difficult/time consuming. To install the modules:
1. Ensure all dependencies are properly installed for:
   - **Debian-based systems:**
     $ apt-cache show python-kerberos
     Look at the Depends field.
   - **RHEL-based systems:**
     $ yum deplist python-kerberos
2. Update your system to meet any dependencies.
3. Download the PyKerberos tarball:
   $ curl -OL <username>:<password>@http://downloads.datastax.com/enterprise/kerberos-1.1.2+DSE1.tar.gz
4. Extract the tarball:
   $ tar -xzf kerberos-1.1.2+DSE1.tar.gz
5. From the directory where you untrusted PyKerberos:
   $ python setup.py build
6. From the install directory:
   $ python setup.py install
7. Download the pure-sasl module tarball:
   $ curl -OL http://pypi.python.org/packages/source/p/pure-sasl/pure-sasl-0.1.3.tar.gz
8. Extract the tarball:
   $ tar -xzf pure-sasl-0.1.3.tar.gz
9. From the install directory:
   $ sudo python setup.py install

**Running cqlsh**

To run cqlsh, you need to create a .cqlshrc file in your home directory. Sample files are available in the following directories:

- **Packaged installs:** /usr/share/doc/dse-libcassandra
- **Binary installs:** <install_location>/resources/cassandra/conf

**Kerberos example**

```
[connection]
hostname = 192.168.1.2
port = 9160
factory = cqlshlib.kerberos.kerberos_transport_factory

[kerberos]
hostname = cassandra01.example.com
service = cassandra
principal = bill/cassandra-admin@example.com ;; Optional.
gops = auth-conf ;; Optional, see the following paragraph.
```
192.168.1.3 = cassandra01.example.com
192.168.1.4 = cassandra02.example.com

If `qops` is not specified the default (auth) is used. On the client side, the `qops` option is a comma-delimited list of the QOP values allowed by the client for the connection. The client (cqlsh) value list must contain at least one of the QOP values specified on the server. To clarify, the client can have multiple QOP values, while the server can have only a single QOP value (specified in the dse.yaml).

The Kerberos hostname and service are mandatory settings and must be provided either in the configuration file or as environment variables. The environment variables (KRB_HOST, KRB_SERVICE, and KRB_PRINCIPAL) override any options set in this file. For more information about these settings, see [Securing DataStax Enterprise nodes](https://datastax.com/doc/). The hostname and service must match the values set in the dse.yaml.

**SSL example**

```
[connection]
hostname = 127.0.0.1
port = 9160
factory = cqlshlib.ssl.ssl_transport_factory

[ssl]
certfile = ~/keys/cassandra.cert
validate = true ;; Optional, true by default.

[certfiles] ;; Optional section, overrides the default certfile in the [ssl] section.
192.168.1.3 = ~/keys/cassandra01.cert
192.168.1.4 = ~/keys/cassandra02.cert
```

When validate is enabled, the host in the certificate is compared to the host of the machine that it is connected to. The SSL certificate must be provided either in the configuration file or as an environment variable. The environment variables (SSL_CERTFILE and SSL_VALIDATE) override any options set in this file.

**Kerberos and SSL**

For information about using Kerberos with SSL, see [Using Kerberos and SSL at the same time](https://datastax.com/doc/).

The settings for using both Kerberos and SSL are a combination of the Kerberos and SSL sections in the above examples, except the factory setting:

```
factory = cqlshlib.kerberos_ssl.kerberos_ssl_transport_factory
```

The supported environmental variables are KRB_HOST, KRB_SERVICE, KRB_PRINCIPAL, SSL_CERTFILE, and SSL_VALIDATE variables.

**Transparent data encryption**

Transparent data encryption (TDE) protects at rest data. At rest data is data that has been flushed from the memtable in system memory to the SSTables on disk.
As shown in the diagram, data stored in the commit log is not encrypted. If you need commit log encryption, store the commit log on an OS-level encrypted file system using Gazzang, for example. Data can be encrypted using different algorithms, or you can choose not to encrypt data at all. SSTable data files are immutable (they are not written to again after they have been flushed to disk). SSTables are encrypted only once when they are written to disk.

The high-level procedure for encrypting data is:

1. Back up SSTables.
2. Set permissions so that only the user/group running DataStax Enterprise can change the keytab file. If JNA is installed, JNA takes care of setting these permissions.
3. Ensure that the user encrypting data has been granted ALTER permission on the table containing the data to be encrypted. You can use LIST PERMISSIONS to view the permissions granted to a user.
4. Specify encryption options when you create a table (column family).
5. Rewrite all SSTables using nodetool scrub, or use nodetool flush to flush to disk all new data using the current settings for encryption.

Requirements

TDE requires a secure local file system to be effective. The encryption certificates are stored locally; therefore, an invasion of the local file system invalidates encryption.

Options

To get the full capabilities of TDE, download and install the Java Cryptography Extension (JCE), unzip the jar files and place them under $JAVA_HOME/jre/lib/security. JCE-based products are restricted for export to certain countries by the U.S. Export Administration Regulations.

Limitations and recommendations

Data is not directly protected by TDE when accessed using the following utilities.

<table>
<thead>
<tr>
<th>Utility</th>
<th>Reason Utility Is Not Encrypted</th>
</tr>
</thead>
<tbody>
<tr>
<td>json2sstable</td>
<td>Operates directly on the sstables.</td>
</tr>
<tr>
<td>nodetool</td>
<td>Uses only JMX, so data is not accessed.</td>
</tr>
<tr>
<td>sstable2json</td>
<td>Operates directly on the sstables.</td>
</tr>
<tr>
<td>sstablekeys</td>
<td>Operates directly on the sstables.</td>
</tr>
<tr>
<td>sstableloader</td>
<td>Operates directly on the sstables.</td>
</tr>
<tr>
<td>sstablescrub</td>
<td>Operates directly on the sstables.</td>
</tr>
</tbody>
</table>
The local file system could be protected through a third party whole-disk encryption solution. You choose ssl, kerberos authentication, encrypted file system, or other ways to secure nodes. DataStax recommends that you do not export local file systems if possible. If you must export a local file system, ensure that mounting the file system on the node is a server-side capability.

Compression and encryption introduce performance overhead.

**Encrypting Data**

You designate encryption on a per table (column family) basis. When using encryption, each node generates a separate key used for only that node’s sstables.

For example, log in as the default superuser:

```
./cqlsh -3 -u cassandra -p cassandra
```

The ALTER TABLE syntax for setting encryption options is the same as the syntax for setting data compression options.

For example, to set compression options in the chores table:

```
ALTER TABLE chores
  WITH compression_parameters:sstable_compression = 'DeflateCompressor'
  AND compression_parameters:chunk_length_kb = 64;
```

To set encryption options in the chores table using CQL 3, for example:

```
ALTER TABLE chores
  WITH compression_parameters:sstable_compression = 'Encryptor'
  AND compression_parameters:cipher_algorithm = 'AES/ECB/PKCS5Padding'
  AND compression_parameters:secret_key_strength = 128;
  AND compression_parameters:chunk_length_kb = 1;
```

Designating data for encryption using ALTER TABLE doesn’t encrypt existing SSTables, just new SSTables that are generated. When setting up data to be encrypted, but not compressed, set the chunk_length_kb option to the lowest possible value, 1, as shown in the previous example. Setting this option to 1 improves read performance by limiting the data that needs to be decrypted for each read operation to 1 KB.

**Setting encryption and compression together**

Encryption and compression occur locally, which is more performant than trying to accomplish these tasks on the Cassandra-side. Encryption can be set together with compression using a single statement. The single statement in CQL 3 is:

```
ALTER TABLE chores
  WITH compression_parameters:sstable_compression = 'EncryptingSnappyCompressor'
  AND compression_parameters:cipher_algorithm = 'AES/ECB/PKCS5Padding'
  AND compression_parameters:secret_key_strength = 128
  AND compression_parameters:chunk_length_kb = 128;
```

**Encryption/compression options and sub-options**

Using encryption, your application can read and write to SSTables that use different encryption algorithms or no encryption at all. Using different encryption algorithms to encrypt SSTable data is similar to using different compression algorithms to compress data. This section lists the options and sub-options.

The high-level container option for encryption and/or compression used in the ALTER TABLE statement are:

- Encryptor
## Transparent data encryption

- EncryptingDeflateCompressor
- EncryptingSnappyCompressor
- DeflateCompressor
- SnappyCompressor (default)

### The cipher_algorithm sub-option

The cipher_algorithm options and acceptable secret_key_strength for the algorithms are:

<table>
<thead>
<tr>
<th>cipher_algorithm</th>
<th>secret_key_strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES/CBC/PKCS5Padding</td>
<td>128, 192, or 256</td>
</tr>
<tr>
<td>AES/ECB/PKCS5Padding</td>
<td>128, 192, or 256</td>
</tr>
<tr>
<td>DES/CBC/PKCS5Padding</td>
<td>56</td>
</tr>
<tr>
<td>DESede/CBC/PKCS5Padding</td>
<td>112 or 168</td>
</tr>
<tr>
<td>Blowfish/CBC/PKCS5Padding</td>
<td>32-448</td>
</tr>
<tr>
<td>RC2/CBC/PKCS5Padding</td>
<td>40-128</td>
</tr>
</tbody>
</table>

You can install custom providers for your JVM. The AES-512 is not supported out-of the box.

### The secret_key_provider_factory_class sub-option

The secret_key_provider_factory_class is:

com.datastax.bdp.cassandra.crypto.LocalFileSystemKeyProviderFactory

### The secret_key_file sub-option

The secret_key_file option is the location of the keyfile. The default location is /etc/dse/conf, but it can reside in any directory.

### The chunk_length_kb sub-option

On disk, SSTables are encrypted and compressed by block (to allow random reads). This subproperty of compression defines the size (in KB) of the block and is a power of 2. Values larger than the default value might improve the compression rate, but increases the minimum size of data to be read from disk when a read occurs. The default value (64) is a good middle-ground for compressing tables.

Using just encryption and no compression, the size of SSTables are dramatically different. For example, during an internal test, starting with a 3.2M .db file and in using these options, resulted in a 236K encrypted .db file:

- sstable_compression = EncryptingDeflateCompressor
- cipher_algorithm = 'AES/CBC/PKCS5Padding',
- secret_key_strength = 256
- secret_key_file = '/home/automaton/newencrypt/keyfile'
- chunk_length_kb = 128

Altering the table to use the EncryptingDeflateCompressor and the same options as before resulted in a file size of 236K, so combining encryption and compression is probably a good idea.

### The iv_length sub-option
Not all algorithms allow you to set this sub-option, and most complain if it is not set to 16 bytes. Either use 16 or accept the default.

The syntax for setting this sub-option is similar to setting a compression algorithm to compress data.

```
ALTER TABLE chores
    WITH compression_parameters:sstable_compression = 'EncryptingSnappyCompressor'
    AND compression_parameters:cipher_algorithm = 'AES/ECB/PKCS5Padding'
    AND compression_parameters:secret_key_strength = 128
    AND compression_parameters:iv_length = 16;
```

### Using nodetool to complete encryption operations

Use the `nodetool scrub` utility to rewrite all the SSTables. Use `nodetool flush` to flush to disk all new data using the current settings for encryption.

### About the keytab file

After designating the data to be encrypted, a keytab file is created in the directory set by the `secret_key_file`. If the directory doesn't exist, it is created. A failure to create the directory probably indicates a permissions problem.

Example values in the keytab file are:

AES/ECB/PKCS5Padding:256:bxegm8vh4wE3S2hO9J36RL2gIdBLx0O46J/QmoC3W3U=
AES/CBC/PKCS5Padding:256:FUhaiy7NGB8oeSfe7cOo3hvojVi2iI/wbBCFH6hsE=
RC2/CBC/PKCS5Padding:128:5lwh3GW3GqE6y/6Bglc3tLw==

Deleting, moving, or changing the data in the keytab file causes errors when the node restarts and you lose all your data. Consider storing the file on a network server or encrypting the entire file system of the nodes using a third-party tool.

### CassandraFS

The CassandraFS (Cassandra file system) is accessed as part of the Hadoop File System (HDFS) using the configured authentication. If you encrypt the CassandraFS keyspace's sblocks and inode tables, all CassandraFS data gets encrypted.

### Using SolrJ-Auth

Follow instructions in the solrj-auth-README.md file to use the SolrJ-Auth libraries to implement encryption. The SolrJ-auth-README.md file is located in the following directory:

- **Debian installations**: `/usr/share/doc/dse-libsolr`
- **RHEL-based installations**: `/usr/share/doc/dse-libsolr`
- **Binary installations**: `resources/solr`

These SolrJ-Auth libraries are included in the DataStax Enterprise distribution:

- **Debian installations**: `/usr/share/dse/clients`
- **Binary installations**: `<install_location>/clients`

### Configuring and using data auditing

Auditing is implemented as a [log4j-based](https://logging.apache.org/log4j) integration. DataStax Enterprise places the audit log in the directory indicated by a `log4j` property. After the file reaches a threshold, it rolls over, and the file name is changed. The file names include a numerical suffix determined by the `maxBackupIndex`. 

44
The audit logger logs information on the node set up for logging. For example, node 0 has audit turned on, node 1 does not. Issuing updates and other commands on node 1 does not generally show up on node 0’s audit log. To get the maximum information from data auditing, turn on data auditing on every node. The log4j supports data stored on the file system or in Cassandra.

Auditing is configured through a text file in the file system, so the file is vulnerable to OS-level security breaches. Store the file on an OS-level encrypted file system using Gazzang, for example, to secure it.

**Configuring data auditing**

You can configure which categories of audit events should be logged and also whether operations against any specific keyspaces should be omitted from audit logging.

**To configure data auditing:**

1. Open the log4j-server.properties file in the following directory.

   **Packaged installs**
   `/etc/dse/cassandra`

   **Binary installs**
   `/resources/cassandra/conf`

2. To configure data auditing, uncomment these properties, and ensure that the default properties are set.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log4j.logger.DataAudit</td>
<td>INFO, A</td>
<td>Produce INFO-level logs.</td>
</tr>
<tr>
<td>log4j.additivity.DataAudit</td>
<td>false</td>
<td>Prevents logging to the root appender.</td>
</tr>
<tr>
<td>log4j.appender.A.File</td>
<td>/var/log/cassandra/audit.log</td>
<td>Sets the file and path of the log file.</td>
</tr>
<tr>
<td>log4j.appender.A.bufferedIO</td>
<td>true</td>
<td>True improves performance but will not be real time; set to false for testing.</td>
</tr>
</tbody>
</table>

To disable data auditing, comment out log4j.logger.DataAudit, log4j.additivity.DataAudit, and log4j.appender.A. This removes almost all auditing overhead. The Log4J audit logger logs at INFO level, so the DataAudit logger must be configured at INFO (or lower) level in log4j-server.properties. Setting the logger to a higher level, such as WARN, prevents any log events from being recorded, but it does not completely disable the data auditing. Some overhead occurs beyond that caused by regular processing.

2. Set other general options to tune the logging, for example uncomment these properties and accept the following defaults:

   - log4j.appender.A.maxFileSize=200MB
   - log4j.appender.A.maxBackupIndex=5
   - log4j.appender.A.layout=org.apache.log4j.PatternLayout
   - log4j.appender.A.layout.ConversionPattern=%m%n
   - log4j.appender.A.filter.1=com.datastax.bdp.cassandra.audit.AuditLogFilter
3. Uncomment and set log4j.appender.A.filter.1.ActiveCategories to ALL or to a combination of these settings:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMIN</td>
<td>Logs describe schema versions, cluster name, version, ring, and other admin events</td>
</tr>
<tr>
<td>ALL</td>
<td>Logs everything: DDL, DML, queries, and errors</td>
</tr>
<tr>
<td>AUTH</td>
<td>Logs login events</td>
</tr>
<tr>
<td>DML</td>
<td>Logs insert, update, delete and other DML events</td>
</tr>
<tr>
<td>DDL</td>
<td>Logs object and user create, alter, drop, and other DDL events</td>
</tr>
<tr>
<td>DCL</td>
<td>Logs grant, revoke, create user, drop user, and list users events</td>
</tr>
<tr>
<td>QUERY</td>
<td>Logs all queries</td>
</tr>
</tbody>
</table>

Set the ActiveCategories property to a comma separated list of the categories to include in the audit log output. By default, this list is empty so unless specified, no events are included in the log. Events are generated even if not included in the log, so set this property.

4. You can disable logging for specific keyspaces. Set this property as follows to prevent logging to specified keyspaces:

```
log4j.appender.A.filter.1.ExemptKeyspaces=do_not_log,also_do_not_log
```

To prevent the audit logger from logging information about itself when using the Cassandra log4j appender, exempt the keyspaces from the appender logs.

The audit log section of the log4j-server.properties file should look something like this:

```
log4j.logger.DataAudit=INFO, A
log4j.additivity.DataAudit=false
log4j.appender.A=org.apache.log4j.RollingFileAppender
log4j.appender.A.File=/var/log/cassandra/audit.log
log4j.appender.A.bufferedIO=true
log4j.appender.A.maxFileSize=200MB
log4j.appender.A.maxBackupIndex=5
log4j.appender.A.layout=org.apache.log4j.PatternLayout
log4j.appender.A.layout.ConversionPattern=%m%n
log4j.appender.A.filter.1=com.datastax.bdp.cassandra.audit.AuditLogFilter
log4j.appender.A.filter.1.ActiveCategories=ALL
log4j.appender.A.filter.1.ExemptKeyspaces=do_not_log,also_do_not_log
```

### Format of logs

The log format is a simple set of pipe-delimited name/value pairs. The pairs themselves are separated by the pipe symbol ("|"), and the name and value portions of each pair are separated by a colon. A name/value pair, or field, is only included in the log line if a value exists for that particular event. Some fields always have a value, and are always present. Others might not be relevant for a given operation. The order in which fields appear (when present) in the log line is predictable to make parsing with automated tools easier. For example, the text of CQL statements is unquoted but if present, is always the last field in the log line.

<table>
<thead>
<tr>
<th>Field Label</th>
<th>Field Value</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>dse node address</td>
<td>no</td>
</tr>
<tr>
<td>source</td>
<td>client address</td>
<td>no</td>
</tr>
<tr>
<td>user</td>
<td>authenticated user</td>
<td>no</td>
</tr>
<tr>
<td>timestamp</td>
<td>system time of log event</td>
<td>no</td>
</tr>
</tbody>
</table>
Configuring and using data auditing

<table>
<thead>
<tr>
<th>category</th>
<th>DML/DDL/QUERY for example</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>API level operation</td>
<td>no</td>
</tr>
<tr>
<td>batch</td>
<td>batch id</td>
<td>yes</td>
</tr>
<tr>
<td>ks</td>
<td>keyspace</td>
<td>yes</td>
</tr>
<tr>
<td>cf</td>
<td>column family</td>
<td>yes</td>
</tr>
<tr>
<td>operation</td>
<td>textual description</td>
<td>yes</td>
</tr>
</tbody>
</table>

The textual description value for the operation field label is currently only present for CQL.

Auditing is completely separate from authorization, although the data points logged include the client address and authenticated user, which may be a generic user if the default authenticator is not overridden. Logging of requests can be activated for any or all of the first list of categories covered by log4j.appender.A.filter.1.ActiveCategories (shown in step 3 in Configuring data auditing).

**CQL Logging examples**

Generally, SELECT queries are placed into the QUERY category. The INSERT, UPDATE, and DELETE statements are categorized as DML. CQL statements that affect schema, such as CREATE KEYSPACE and DROP KEYSPACE are categorized as DDL.

**CQL USE**

```sql
USE dsp904;
```

```sql
host:/192.168.56.1|source:/192.168.56.101|user:#<User allow_all groups=[]>
|timestamp:1351003707937|category:DML|type:SET_KS|ks:dsp904|operation:use dsp904;
```

**CLI USE**

```sql
USE dsp904;
```

```sql
host:/192.168.56.1|source:/192.168.56.101|user:#<User allow_all groups=[]>
|timestamp:1351004648848|category:DML|type:SET_KS|ks:dsp904
```

**CQL query**

```sql
SELECT * FROM t0;
```

```sql
host:/192.168.56.1|source:/192.168.56.101|user:#<User allow_all groups=[]>
|timestamp:1351003741953|category:QUERY|type:CQL_SELECT|ks:dsp904|cf:t0|operation:select * from t0;
```

**CQL BATCH**

```sql
BEGIN BATCH
  INSERT INTO t0(id, field0) VALUES (0, 'foo')
  INSERT INTO t0(id, field0) VALUES (1, 'bar')
  DELETE FROM t1 WHERE id = 2
APPLY BATCH;
```

```sql
host:192.168.56.1|source:/192.168.56.101|user:#<User allow_all groups=[]>
|timestamp:1351005482412|category:DML|type:CQL_UPDATE
|batch:fc386364-245a-44c0-a5ab-12f165374a89|ks:dsp904|cf:t0
|operation:INSERT INTO t0 ( id , field0 ) VALUES ( 0 , 'foo' )
```

```sql
host:192.168.56.1|source:/192.168.56.101|user:#<User allow_all groups=[]>
|timestamp:1351005482413|category:DML|type:CQL_UPDATE
|batch:fc386364-245a-44c0-a5ab-12f165374a89|ks:dsp904|cf:t0
```

47
Configuring and using data auditing

```
operation: INSERT INTO t0 (id, field0) VALUES (1, 'bar')
host:192.168.56.1|source:/192.168.56.101|user:#<User allow_all groups=[]>
timestamp:1351005482413|category:DML|type:CQL_DELETE
batch:fc386364-245a-44c0-a5ab-12f165374a89|ks:dsp904|cf:t1
operation: DELETE FROM t1 WHERE id = 2
```

CQL DROP KEYSPACE

```
DROP KEYSPACE dsp904;
```

CQL prepared statement

```
host:/10.112.75.154|source:/127.0.0.1|user:allow_all
timestamp:1356046999323|category:DML|type:CQL_UPDATE
ks:ks|cf:cf|operation: INSERT INTO cf (id, name) VALUES (?, ?) [id=1,name=vic]
```

Thrift batch_mutate

```
host:/192.168.56.1|source:/192.168.56.101|user:#<User allow_all groups=[]>
timestamp:1351005073561|category:DML|type:INSERT
batch:7d13a423-4c68-4238-af06-a779697088a9|ks:Keyspace1|cf:Standard1
host:/192.168.56.1|source:/192.168.56.101|user:#<User allow_all groups=[]>
timestamp:1351005073562|category:DML|type:INSERT
batch:7d13a423-4c68-4238-af06-a779697088a9|ks:Keyspace1|cf:Standard1
host:/192.168.56.1|source:/192.168.56.101|user:#<User allow_all groups=[]>
timestamp:1351005073562|category:DML|type:INSERT
batch:7d13a423-4c68-4238-af06-a779697088a9|ks: Keyspace1 |cf: Standard1
```

Batch updates

Batch updates, whether received via a Thrift batch_mutate call, or in CQL BEGIN BATCH....APPLY BATCH block, are logged in the following way: A UUID is generated for the batch, then each individual operation is reported separately, with an extra field containing the batch id.

Configuring auditing for a DSE Search/Solr cluster

By default, DSE Search/Solr nodes need no configuration for data auditing except setting up the `log4j-server.properties` file. If the filter-mapping element in the Solr web.xml file is commented out, the auditor cannot log anything from Solr and you need to configure auditing as described in the next section.

If necessary, uncomment the filter-mapping element in the Solr web.xml.

```
<filter-mapping>
  <filter-name>DseAuditLoggingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

The Solr web.xml is located in the following directory:

Packaged installations

```
/usr/share/dse/solr/web/solr/WEB-INF/web.xml
```
Configuring and using internal authentication

Binary installations

/resources/solr/web/solr/WEB-INF/web.xml

Example of a Solr Audit Log

Here is an example of the data audit log of a Solr query:

```
host:/10.245.214.159|source:127.0.0.1|user:jdoe|timestamp:1356045339910|category:QUERY
|type:SOLR_QUERY|ks:wiki|cf:solr|operation:/wiki.solr/select/?q=body:trains
```

Configuring and using internal authentication

Like object permission management that uses internal authorization, internal authentication is based on
Cassandra-controlled login accounts and passwords. Internal authentication works for the following clients when you
provide a user name and password to start up the client:

- Astyanax
- cassandra-cli
- cqlsh
- Hector
- pycassa

Internal authentication stores user names and bcrypt-hashed passwords in the dse_auth.credentials column family.

Limitations

The dsetool and Hadoop utilities are not supported by internal authentication.

Configuring and using authentication

To configure Cassandra to use internal authentication, first you change the superuser password, and then you make a
few changes to the cassandra.yaml as described in this procedure. To use authentication, you start up the client using
the default user name and password (cassandra/cassandra).

1. Change the authenticator option in the cassandra.yaml to PasswordAuthenticator by uncommenting only the
   PasswordAuthenticator:

   ```
   #authentication backend, implementing IAuthenticator; used to identify users
   #authenticator: org.apache.cassandra.auth.AllowAllAuthenticator
   authenticator: com.datastax.bdp.cassandra.auth.PasswordAuthenticator
   #authenticator: com.datastax.bdp.cassandra.auth.KerberosAuthenticator
   ```

2. Configure the replication strategy for the dse_auth keyspace.

3. Restart DataStax Enterprise. The syntax for starting up the Cassandra client for the first time after configuring
   native authentication is:

   ```
   <client startup string> -u cassandra -p cassandra
   ```

4. Start cqlsh using the same superuser name and password (cassandra) that you use to start the supported client.
   For example, to start cqlsh in CQL 3 mode on Linux:

   ```
   ./cqlsh -3 -u cassandra -p cassandra
   ```

You can now change the superuser's user name and password.
**Changing the default superuser**

By default, each installation of Cassandra includes a superuser account named cassandra whose password is also cassandra. A superuser grants initial permissions to access Cassandra data, and subsequently a user may or may not be given the permission to grant/revoke permissions.

To change the superuser account name and password:

1. **Configure internal authentication** if you have not already done so.

2. Create another superuser, not named cassandra.
   - Use the `CREATE USER` command.
   - You can now set up user accounts and authorize users to access the database objects by using CQL to **grant them permissions** on those objects.

3. Log in as that new superuser.

4. Change the cassandra user password to something long and incomprehensible, and then forget about it. It won’t be used again.

5. Take away the cassandra user's superuser status.

CQL 3 supports the following authentication statements, which are described in the CQL alphabetical security command reference:

- `ALTER USER`
- `CREATE USER`
- `DROP USER`
- `LIST USERS`

**Enable internal security in DataStax Enterprise 3.0.3 without downtime**

The TransitionalAuthenticator and TransitionalAuthorizer allow internal authentication and authorization to be enabled without downtime or modification to client code or configuration.

To implement:

1. On each node, in the cassandra.yaml file:
   - Set the `authenticator` to `com.datastax.bdp.cassandra.auth.TransitionalAuthenticator`.
   - Set the `authorizer` to `com.datastax.bdp.cassandra.auth.TransitionalAuthorizer`.

2. Perform a rolling restart.

3. Once the restarts are complete, use cqlsh with the default superuser login to setup the users, credentials, and permissions.

4. Once the setup is complete, edit the cassandra.yaml file again and perform another rolling restart:
   - Change the `authenticator` to `com.datastax.bdp.cassandra.auth.PasswordAuthenticator`.
   - Change the `authorizer` to `com.datastax.bdp.cassandra.auth.CassandraAuthorizer`.

5. After the restarts have completed, remove the default superuser and create at least one new superuser as described above.

**Logging in with cqlsh**

To avoid having to pass credentials for every login using cqlsh, you can create a `.cqlshrc` file your home directory. When present, it passes default login information to cqlsh. For example:
Managing object permissions using internal authorization

[authentication]
username = fred
password = !!bang!!$

Be sure to set the correct permissions and secure this file so that no unauthorized users can gain access to database login information.

**Note**
Sample .cqlshrc files are available in the following directories:

- **Packaged installs**: /usr/share/doc/dse-libcassandra
- **Binary installs**: <install_location>/resources/cassandra/conf

Managing object permissions using internal authorization

You use familiar relational database GRANT/REVOKE paradigm to grant or revoke permissions to access Cassandra data. A superuser grants initial permissions, and subsequently a user may or may not be given the permission to grant/revoke permissions. Object permission management is independent of authentication (works with Kerberos or Cassandra).

**Accessing system resources**

Read access to these system tables is implicitly given to every authenticated user because the tables are used by most Cassandra tools:

- system.schema_keyspace
- system.schema_columns
- system.schema_columnfamilies
- system.local
- system.peers

**Configuration**

To **configure internal authorization for managing object permissions**:

1. Specify the authorizer in the cassandra.yaml by uncommenting this option:
   ```yaml
   authorizer: com.datastax.bdp.cassandra.auth.CassandraAuthorizer
   ```
   You can use any authenticator except AllowAll.

2. **Configure the dse_auth keyspace replication factor.**

3. Fetching permissions can be an expensive operation. If necessary, adjust the validity period for permissions caching by setting the `permissions_validity_in_ms` option in the cassandra.yaml. You can also disable permission caching by setting this option to 0.

CQL 3 supports the following authorization statements, which are described in the CQL alphabetical security command reference:

- **GRANT**
- **LIST PERMISSIONS**
- **REVOKE**
Configuring dse_auth keyspace replication

**Note**
To enable internal authorization on existing clusters, see *Enable internal security in DataStax Enterprise 3.0.3 without downtime*.

**Configuring dse_auth keyspace replication**

You need to configure the default dse_auth keyspace replication factor to prevent a potential problem logging into a secure cluster. Do not use the default replication factor of 1 for the dse_auth keyspace if you use any of these authenticators and/or this authorizer:

- authenticator: com.datastax.bdp.cassandra.auth.KerberosAuthenticator
- authenticator: com.datastax.bdp.cassandra.auth.PasswordAuthenticator
- authorizer: com.datastax.bdp.cassandra.auth.CassandraAuthorizer

In a multi-node cluster, using the default of 1 precludes logging into any node when the node that stores the user data is down.

For all dse_auth-related queries, Cassandra uses a consistency level of QUORUM. For more information, see *About Data Consistency in Cassandra*.

**Setting the replication factor**

1. Open the `cassandra.yaml` configuration file for editing.
2. Change the `auth_replication_options` using the same options that you would use when creating a keyspace.

   **Example for SimpleStrategy:**

   ```yaml
   auth_replication_options:
     replication_factor: 3
   ```

   **Example for NetworkTopologyStrategy:**

   ```yaml
   auth_replication_options:
     DC1: 3
     DC2: 3
   ```

3. If you change the `auth_replication_options` on an existing cluster:
   a. Make sure every node uses the same settings.
   b. Restart every node after updating the `cassandra.yaml` file.
   c. Run a `nodetool repair` on each node.

**About the dse_auth keyspace**

Cassandra uses the dse_auth keyspace for storing security authentication and authorization information:

- **Cassandra**: the internal user list (in dse_auth.users column family).
- **PasswordAuthenticator**: the users’ hashed passwords (in dse_auth.credentials column family)
- **CassandraAuthorizer**: the users’ permissions (in dse_auth.permissions column family)

**Configuring firewall port access**
If you have a firewall running on the nodes in your Cassandra or DataStax Enterprise cluster, you must open up the following ports to allow communication between the nodes, including certain Cassandra ports. If this isn't done, when you start Cassandra (or Hadoop in DataStax Enterprise) on a node, the node will act as a standalone database server rather than joining the database cluster.

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Public Facing Ports</strong></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>SSH (default)</td>
</tr>
<tr>
<td><strong>DataStax Enterprise Specific</strong></td>
<td></td>
</tr>
<tr>
<td>8012</td>
<td>Hadoop Job Tracker client port. The Job Tracker listens on this port for job submissions and communications from task trackers; allows traffic from each Analytics node in a cluster.</td>
</tr>
<tr>
<td>8983</td>
<td>Solr port and Demo applications website port (Portfolio, Search, Search log)</td>
</tr>
<tr>
<td>50030</td>
<td>Hadoop Job Tracker website port. The Job Tracker listens on this port for HTTP requests. If initiated from the OpsCenter UI, these requests are proxied through the opscenterd daemon; otherwise, they come directly from the browser.</td>
</tr>
<tr>
<td>50060</td>
<td>Hadoop Task Tracker website port. Each Task Tracker listens on this port for HTTP requests coming directly from the browser and not proxied by the opscenterd daemon.</td>
</tr>
<tr>
<td><strong>OpsCenter Specific</strong></td>
<td></td>
</tr>
<tr>
<td>8888</td>
<td>OpsCenter website. The opscenterd daemon listens on this port for HTTP requests coming directly from the browser.</td>
</tr>
<tr>
<td><strong>Inter-node Ports</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Cassandra Specific</strong></td>
<td></td>
</tr>
<tr>
<td>1024+</td>
<td>JMX reconnection/loopback ports. See description for port 7199.</td>
</tr>
<tr>
<td>7000</td>
<td>Cassandra inter-node cluster communication.</td>
</tr>
<tr>
<td>7199</td>
<td>Cassandra JMX monitoring port. After the initial handshake, the JMX protocol requires that the client reconnects on a randomly chosen port (1024+).</td>
</tr>
<tr>
<td>9160</td>
<td>Cassandra client port (Thrift). OpsCenter agents makes Thrift requests to their local node on this port. Additionally, the port can be used by the opscenterd daemon to make Thrift requests to each node in the cluster.</td>
</tr>
<tr>
<td><strong>DataStax Enterprise Specific</strong></td>
<td></td>
</tr>
<tr>
<td>9290</td>
<td>Hadoop Job Tracker Thrift port. The Job Tracker listens on this port for Thrift requests coming from the opscenterd daemon.</td>
</tr>
<tr>
<td><strong>OpsCenter Specific</strong></td>
<td></td>
</tr>
<tr>
<td>50031</td>
<td>OpsCenter HTTP proxy for Job Tracker. The opscenterd daemon listens on this port for incoming HTTP requests from the browser when viewing the Hadoop Job Tracker page directly.</td>
</tr>
<tr>
<td>61620</td>
<td>OpsCenter monitoring port. The opscenterd daemon listens on this port for TCP traffic coming from the agent.</td>
</tr>
<tr>
<td>61621</td>
<td>OpsCenter agent port. The agents listen on this port for SSL traffic initiated by OpsCenter.</td>
</tr>
</tbody>
</table>

**Alphabetical CQL command reference**

DataStax Enterprise 3.0 uses the versions of CQL available in Cassandra 1.1.x:

- The pre-release version of CQL 3
- CQL 2
ALTER USER

DataStax Enterprise 3.0 adds these additional CQL security commands:

- ALTER USER
- CREATE USER
- DROP USER
- GRANT
- LIST PERMISSIONS
- LIST USERS
- REVOKE

For information about using other CQL commands, see the Cassandra 1.1 documentation.

ALTER USER

Alter existing user options.

Synopsis

ALTER USER user_name
   WITH PASSWORD 'password' NOSUPERUSER | SUPERUSER

Synopsis legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable

A semicolon that terminates cqlsh statements is not included in the synopsis.

Description

Superusers can change a user’s password or superuser status. To prevent disabling all superusers, superusers cannot change their own superuser status. Ordinary users can change only their own password.

Enclose the user name in single quotation marks if it contains non-alphanumeric characters. Enclose the password in single quotation marks.

Example

ALTER USER moss WITH PASSWORD 'bestReceiver';

CREATE USER

Create a new user.

Synopsis
CREATE USER user_name
   WITH PASSWORD 'password' NOSUPERUSER | SUPERUSER

Synopsis legend

Description
CREATE USER defines a new database user account. By default users accounts do not have superuser status. Only a superuser can issue CREATE USER requests.

User accounts are required for logging in under internal authentication and authorization.

Enclose the user name in single quotation marks if it contains non-alphanumeric characters. You cannot recreate an existing user. To change the superuser status or password, use ALTER USER.

Creating internal user accounts
You need to use the WITH PASSWORD clause when creating a user account for internal authentication. Enclose the password in single quotation marks.

Example
   CREATE USER spillman WITH PASSWORD 'Niner27';
   CREATE USER akers WITH PASSWORD 'Niner2' SUPERUSER;
   CREATE USER boone WITH PASSWORD 'Niner75' NOSUPERUSER;

If internal authentication has not been set up, you do not need the WITH PASSWORD clause:

   CREATE USER test NOSUPERUSER;

DROP USER

Synopsis
   DROP USER user_name

Synopsis legend

Description
DROP USER removes an existing user. You have to be logged in as a superuser to issue a DROP USER statement. A user cannot drop themselves.

Enclose the user name in single quotation marks only if it contains non-alphanumeric characters.

GRANT
Provides users access to database objects.

Synopsis
GRANT permission_name PERMISSION
| GRANT ALL PERMISSIONS
    ON resource TO user

permission_name is one of these:

- ALTER
- AUTHORIZE
- CREATE
- DROP
- MODIFY
- SELECT

resource is one of these:

- ALL KEYSPACES
- KEYSPACE keyspace_name
- TABLE keyspace_name.table_name

Synopsis legend

Description

Permissions to access all keyspaces, a named keyspace, or a table can be granted to a user. Enclose the user name in single quotation marks if it contains non-alphanumeric characters.

This table lists the permissions needed to use CQL statements:

<table>
<thead>
<tr>
<th>Permission</th>
<th>CQL Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>ALTER KEYSPACE, ALTER TABLE, CREATE INDEX, DROP INDEX</td>
</tr>
<tr>
<td>AUTHORIZE</td>
<td>GRANT, REVOKE</td>
</tr>
<tr>
<td>CREATE</td>
<td>CREATE KEYSPACE, CREATE TABLE</td>
</tr>
<tr>
<td>DROP</td>
<td>DROP KEYSPACE, DROP TABLE</td>
</tr>
<tr>
<td>MODIFY</td>
<td>INSERT, DELETE, UPDATE, TRUNCATE</td>
</tr>
<tr>
<td>SELECT</td>
<td>SELECT</td>
</tr>
</tbody>
</table>

To be able to perform SELECT queries on a table, you have to have SELECT permission on the table, on its parent keyspace, or on ALL KEYSPACES. To be able to CREATE TABLE you need CREATE permission on its parent keyspace or ALL KEYSPACES. You need to be a superuser or to have AUTHORIZE permission on a resource (or one of its parents in the hierarchy) plus the permission in question to be able to GRANT or REVOKE that permission to or from a user. GRANT, REVOKE and LIST permissions check for the existence of the table and keyspace before execution. GRANT and REVOKE check that the user exists.

Examples

Give ‘spillman’ permission to perform SELECT queries on all tables in all keyspaces:

```sql
GRANT SELECT ON ALL KEYSPACES TO spillman;
```

Give ‘akers’ permission to perform INSERT, UPDATE, DELETE and TRUNCATE queries on all tables in the 'field' keyspace:

```sql
GRANT MODIFY ON KEYSPACE field TO akers;
```
Give 'boone' permission to perform ALTER KEYSPACE queries on the 'forty9ers' keyspace, and also ALTER TABLE, CREATE INDEX and DROP INDEX queries on all tables in 'forty9ers' keyspace:

```
GRANT ALTER ON KEYSPECE forty9ers TO boone;
```

Give 'boone' permission to run all types of queries on ravens.plays table:

```
GRANT ALL PERMISSIONS ON ravens.plays TO boone;
```

To grant access to a keyspace to just one user, assuming nobody else has ALL KEYSPACES access, you use this statement:

```
GRANT ALL ON KEYSPECE keyspace_name TO user_name
```

**LIST PERMISSIONS**

Lists permissions granted to a user.

**Synopsis**

```
LIST permission_name PERMISSION
```

```
| LIST ALL PERMISSIONS
| ON resource OF user_name
| NORECURSIVE
```

permission_name is one of these:

- ALTER
- AUTHORIZE
- CREATE
- DROP
- MODIFY
- SELECT

resource is one of these:

- ALL KEYSPACES
- KEYSPACE keyspace_name
- TABLE keyspace_name.table_name

**Synopsis legend**

**Description**

Permissions checks are recursive. If you omit the NORECURSIVE specifier, permission on the requests resource and its parents in the hierarchy are shown.

- Omitting the resource name (ALL KEYSPACES, keyspace, or table), lists permissions on all tables and all keyspaces.
- Omitting the user name lists permissions of all users. You need to be a superuser to list permissions of all users. If you are not, you must add of <myusername>.
- Omitting the NORECURSIVE specifier, lists permissions on the resource and its parent resources.
- Enclose the user name in single quotation marks only if it contains non-alphanumeric characters.
After creating users in *Creating internal user accounts* and granting the permissions in the *GRANT* examples, you can list permissions that users have on resources and their parents.

**Example**

Assuming you completed the examples in *Examples*, list all permissions given to akers:

```
LIST ALL PERMISSIONS OF akers;
```

**Output**

```
username | resource           | permission
----------+--------------------+------------
akers    | <keyspace field>   | MODIFY
```

List permissions given to all the users:

```
LIST ALL PERMISSIONS;
```

**Output**

```
username | resource             | permission
----------+----------------------+------------
akers    | <keyspace field>     | MODIFY
boone    | <keyspace forty9ers> | ALTER      
boone    | <table ravens.plays> | CREATE     
boone    | <table ravens.plays> | ALTER      
boone    | <table ravens.plays> | DROP       
boone    | <table ravens.plays> | SELECT     
boone    | <table ravens.plays> | MODIFY     
boone    | <table ravens.plays> | AUTHORIZE  
spillman | <all keyspaces>      | SELECT     
```

List all permissions on the plays table:

```
LIST ALL PERMISSIONS ON ravens.plays;
```

**Output**

```
username | resource             | permission
----------+----------------------+------------
boone    | <table ravens.plays> | CREATE     
boone    | <table ravens.plays> | ALTER      
boone    | <table ravens.plays> | DROP       
boone    | <table ravens.plays> | SELECT     
boone    | <table ravens.plays> | MODIFY     
boone    | <table ravens.plays> | AUTHORIZE  
spillman | <all keyspaces>      | SELECT     
```

List all permissions on the ravens.plays table and its parents:

```
LIST ALL PERMISSIONS ON ravens.plays NORECURSIVE;
```

**Output**

```
username | resource             | permission
----------+----------------------+------------
boone    | <table ravens.plays> | CREATE     
boone    | <table ravens.plays> | ALTER      
boone    | <table ravens.plays> | DROP       
boone    | <table ravens.plays> | SELECT     
boone    | <table ravens.plays> | MODIFY     
boone    | <table ravens.plays> | AUTHORIZE  
```
LIST USERS

Lists existing users and their superuser status.

Synopsis

LIST USERS

Synopsis legend

Description

Assuming you use internal authentication, created the users in Creating internal user accounts, and have not yet changed the default user, the following example shows the output of LIST USERS.

Example

LIST USERS;

Output

<table>
<thead>
<tr>
<th>name</th>
<th>super</th>
</tr>
</thead>
<tbody>
<tr>
<td>cassandra</td>
<td>True</td>
</tr>
<tr>
<td>boone</td>
<td>False</td>
</tr>
<tr>
<td>akers</td>
<td>True</td>
</tr>
<tr>
<td>spillman</td>
<td>False</td>
</tr>
</tbody>
</table>

REVOKE

Synopsis

REVOKE permission_name PERMISSION
| REVOKE ALL PERMISSIONS
ON resource FROM user_name

permission_name is one of these:

- ALTER
- AUTHORIZE
- CREATE
- DROP
- MODIFY
- SELECT

resource is one of these:

- ALL KEYSPACES
- KEYSPACE keyspace_name
- TABLE keyspace_name.table_name

Synopsis legend
**Description**

Permissions to access all keyspaces, a named keyspace, or a table can be revoked from a user. Enclose the user name in single quotation marks if it contains non-alphanumeric characters.

This table lists the permissions needed to use CQL statements:

<table>
<thead>
<tr>
<th>Permission</th>
<th>CQL Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>ALTER KEYSPACE, ALTER TABLE, CREATE INDEX, DROP INDEX</td>
</tr>
<tr>
<td>AUTHORIZE</td>
<td>GRANT, REVOKE</td>
</tr>
<tr>
<td>CREATE</td>
<td>CREATE KEYSPACE, CREATE TABLE</td>
</tr>
<tr>
<td>DROP</td>
<td>DROP KEYSPACE, DROP TABLE</td>
</tr>
<tr>
<td>MODIFY</td>
<td>INSERT, DELETE, UPDATE, TRUNCATE</td>
</tr>
<tr>
<td>SELECT</td>
<td>SELECT</td>
</tr>
</tbody>
</table>

**Example**

REVOKE SELECT ON ravens.plays FROM boone;

The user boone can no longer perform SELECT queries on the ravens.plays table. Exceptions: Because of inheritance, the user can perform SELECT queries on ravens.plays if one of these conditions is met:

- The user is a superuser
- The user has SELECT on ALL KEYSPACES permissions
- The user has SELECT on the ravens keyspace
Deployment

Production deployment planning

This section provides guidelines for determining the size of your production Cassandra cluster based on the data you plan to store.

Planning includes the following activities:

- Prerequisites
- Selecting hardware for enterprise implementations
- Planning an Amazon EC2 cluster
- Calculating usable disk capacity
- Calculating User Data Size

Prerequisites

Before starting to plan a production cluster, you need:

- A good understanding of the size of the raw data you plan to store.
- A good estimate of your typical application workload.
- A plan to model your data in Cassandra (number of column families, rows, columns per row, and so on).

Selecting hardware for enterprise implementations

As with any application, choosing appropriate hardware depends on selecting the right balance of the following resources: memory, CPU, disks, number of nodes, and network.

Note

Hadoop and Solr nodes require their own nodes/disks and have specific hardware requirements. See the Hadoop and Solr documentation for more information when determining your capacity requirements.

Memory

The more memory a Cassandra node has, the better read performance. More RAM allows for a larger file system cache and reduces disk I/O for reads. The ideal amount of RAM depends on the anticipated size of your hot data.

DataStax recommends the following memory requirements:

- For dedicated hardware, a minimum of 8GB of RAM is needed. For most implementations you should use 16GB to 32GB.
- Java heap space should be set according to Cassandra 1.1 guidelines.
- For a virtual environment use a minimum of 4GB, such as Amazon EC2 Large instances. For production clusters with a healthy amount of traffic, 8GB is more common.
- For Solr and Hadoop nodes, use 32GB or more of total RAM.

CPU

Insert-heavy workloads are CPU-bound in Cassandra before becoming memory-bound. Cassandra is highly concurrent and uses as many CPU cores as available.
Deployment

- For dedicated hardware, 8-core processors are the current price-performance sweet spot.
- For virtual environments, consider using a provider that allows CPU bursting, such as Rackspace Cloud Servers.

**Disk**

What you need for your environment depends a lot on the usage, so it's important to understand the mechanism. Cassandra writes data to disk for two purposes:

- All data is appended to the commit log for durability.
- When thresholds are reached, Cassandra periodically flushed in-memory data structures (memtables) to immutable SSTable data files for storage of column family data.

Commit logs receive every write made to a Cassandra node, but are only read during node start up. Commit logs are purged after the corresponding data is flushed. Conversely, SSTable (data file) writes occur asynchronously and are read during client look-ups. Additionally, SSTables are periodically compacted. Compaction improves performance by merging and rewriting data and discarding old data. However, during compaction (or node repair), disk utilization and data directory volume can substantially increase. For this reason, DataStax recommends leaving an adequate amount of free disk space available on a node (50% [worst case] for tiered compaction, 10% for leveled compaction).

**Recommendations:**

- DataStax neither supports nor recommends using Network Attached Storage (NAS) because of performances issues, such as network saturation, I/O overload, pending-task swamp, excessive memory usage, and disk contention.
- When choosing disks, consider both capacity (how much data you plan to store) and I/O (the write/read throughput rate). Most workloads are best served by using less expensive SATA disks and scaling disk capacity and I/O by adding more nodes (with more RAM).
- Solid-state drives (SSDs) are a valid choice for Cassandra. Cassandra's sequential, streaming write patterns minimize the undesirable effects of write amplification associated with SSDs.
- Ideally Cassandra needs at least two disks, one for the commit log and the other for the data directories. At a minimum the commit log should be on its own partition.
- Commit log disk - this disk does not need to be large, but it should be fast enough to receive all of your writes as appends (sequential I/O).
- Data disks - use one or more disks and make sure they are large enough for the data volume and fast enough to both satisfy reads that are not cached in memory and to keep up with compaction.
- RAID - the compaction process can temporarily require up to double the normal data directory volume. This means when approaching 50% of disk capacity, you should use RAID 0 or RAID 10 for your data directory volumes. RAID also helps smooth out I/O hotspots within a single SSTable.
  - Use RAID0 if disk capacity is a bottleneck and rely on Cassandra's replication capabilities for disk failure tolerance. If you lose a disk on a node, you can recover lost data through Cassandra's built-in repair.
  - Use a setra setting of 512, especially on Amazon EC2 RAID0 devices. See Optimum blockdev --setra Settings for RAID.
  - Use RAID10 to avoid large repair operations after a single disk failure, or if you have disk capacity to spare.
  - Because data is stored in the memtable, generally RAID is not needed for the commit log disk, but if you need the extra redundancy, use RAID 1.
- Extended file systems - On ext2 or ext3, the maximum file size is 2TB even using a 64-bit kernel. On ext4 it is 16TB.

Because Cassandra can use almost half your disk space for a single file, use XFS when raiding large disks together, particularly if using a 32-bit kernel. XFS file size limits are 16TB max on a 32-bit kernel, and essentially unlimited on 64-bit.
Number of nodes

The amount of data on each disk in the array isn’t as important as the total size per node. Using a greater number of smaller nodes is better than using fewer larger nodes because of potential bottlenecks on larger nodes during compaction.

Network

Since Cassandra is a distributed data store, it puts load on the network to handle read/write requests and replication of data across nodes. Be sure to choose reliable, redundant network interfaces and make sure that your network can handle traffic between nodes without bottlenecks.

- Recommended bandwidth is 1000 Mbit/s (Gigabit) or greater.
- Bind the Thrift interface (listen_address) to a specific NIC (Network Interface Card).
- Bind the RPC server interface (rpc_address) to another NIC.

Cassandra is efficient at routing requests to replicas that are geographically closest to the coordinator node handling the request. Cassandra will pick a replica in the same rack if possible, and will choose replicas located in the same data center over replicas in a remote data center.

Ports

If using a firewall, make sure that nodes within a cluster can reach each other. See Configuring firewall port access.

Planning an Amazon EC2 cluster

DataStax provides an Amazon Machine Image (AMI) to allow you to quickly deploy a multi-node Cassandra cluster on Amazon EC2. The DataStax AMI initializes all nodes in one availability zone using the SimpleSnitch. If you want an EC2 cluster that spans multiple regions and availability zones, do not use the DataStax AMI. Instead, initialize your EC2 instances for each Cassandra node and then configure the cluster as a multiple data-center cluster.

Use the following guidelines when setting up your cluster:

- For most production clusters, use or Extra Large instances with local storage.

Note

Amazon Web Service recently reduced the number of default ephemeral disks attached to the image from four to two. Performance will be slower for new nodes unless you manually attach the additional two disks; see Amazon EC2 Instance Store.

- For low to medium data throughput production clusters, use Large instances with local storage (which are generally adequate for about a year).
- RAID0 the ephemeral disks, and put both the data directory and the commit log on that volume. This has proved to be better in practice than putting the commit log on the root volume (which is also a shared resource).
- For data redundancy, consider deploying your cluster across multiple availability zones or using EBS volumes to store your backup files.
• EBS volumes are not recommended for Cassandra data volumes. Their network performance and disk I/O are not good fits for Cassandra for the following reasons:
  
  • EBS volumes contend directly for network throughput with standard packets. This means that EBS throughput is likely to fail if you saturate a network link.
  
  • EBS volumes have unreliable performance. I/O performance can be exceptionally slow, causing the system to backload reads and writes until the entire cluster becomes unresponsive.
  
  • Adding capacity by increasing the number of EBS volumes per host does not scale. You can easily surpass the ability of the system to keep effective buffer caches and concurrently serve requests for all of the data it is responsible for managing.
  
  For more information and graphs related to ephemeral versus EBS performance, see the blog article at http://blog.scalyr.com/2012/10/16/a-systematic-look-at-ec2-io/.

Calculating usable disk capacity

To calculate how much data your Cassandra nodes can hold, calculate the usable disk capacity per node and then multiply that by the number of nodes in your cluster. Remember that in a production cluster, you will typically have your commit log and data directories on different disks. This calculation is for estimating the usable capacity of the data volume.

Start with the raw capacity of the physical disks:

\[
\text{raw\_capacity} = \text{disk\_size} \times \text{number\_of\_disks}
\]

Account for file system formatting overhead (roughly 10 percent) and the RAID level you are using. For example, if using RAID-10, the calculation would be:

\[
\left(\text{raw\_capacity} \times 0.9\right) / 2 = \text{formatted\_disk\_space}
\]

During normal operations, Cassandra routinely requires disk capacity for compaction and repair operations. For optimal performance and cluster health, DataStax recommends that you do not fill your disks to capacity, but run at 50-80 percent capacity. With this in mind, calculate the usable disk space as follows (example below uses 50%):

\[
\text{formatted\_disk\_space} \times 0.5 = \text{usable\_disk\_space}
\]

Calculating User Data Size

Typically in data storage systems, the size of your raw data will be larger once it is loaded into the database due to storage overhead. On average, raw data is about 2 times larger on disk after it is loaded into the Cassandra, but could vary in either direction depending on the characteristics of your data and column families. The calculations in this section account for data persisted to disk, not for data stored in memory.

• Column Overhead - Every column in Cassandra incurs 15 bytes of overhead. Since each row in a column family can have different column names as well as differing numbers of columns, metadata is stored for each column. For counter columns and expiring columns, add an additional 8 bytes (23 bytes column overhead). So the total size of a regular column is:

\[
\text{total\_column\_size} = \text{column\_name\_size} + \text{column\_value\_size} + 15
\]

• Row Overhead - Just like columns, every row also incurs some overhead when stored on disk. Every row in Cassandra incurs 23 bytes of overhead.

• Primary Key Index - Every column family also maintains a primary index of its row keys. Primary index overhead becomes more significant when you have lots of skinny rows. Sizing of the primary row key index can be estimated as follows (in bytes):

\[
\text{primary\_key\_index} = \text{number\_of\_rows} \times (32 + \text{average\_key\_size})
\]
Replication Overhead - The replication factor obviously plays a role in how much disk capacity is used. For a replication factor of 1, there is no overhead for replicas (as only one copy of your data is stored in the cluster). If replication factor is greater than 1, then your total data storage requirement will include replication overhead.

\[
\text{replication\_overhead} = \text{total\_data\_size} \times (\text{replication\_factor} - 1)
\]

### Configuring replication

Cassandra performs replication to store multiple copies of data on multiple nodes for reliability and fault tolerance. You need to choose a data partitioner and replica placement strategy to configure replication. Data partitioning determines how to place the data across the nodes in the cluster. Choosing a partitioner determines on which node to place the first copy of data. Choosing the replica placement strategy determines which nodes get additional copies of data.

Nodes communicate with each other about replication and other things using the gossip protocol. This section covers the gossip settings and other node configuration information. It also covers how to change a replication setting.

### Partitioner settings

When you deploy a Cassandra cluster, make sure that each node is responsible for roughly an equal amount of data. To accomplish this load balancing, configure the partitioner for each node, and assign the node an initial-token value.

DataStax strongly recommends using the RandomPartitioner (the default) for all cluster deployments. Assuming use of this partitioner, each node in the cluster is assigned a token that represents a hash value within the range of 0 to \(2^{127} - 1\).

You can calculate tokens for a cluster having nodes in a single data center by dividing the range by the total number of nodes in the cluster. In multiple data center deployments, tokens should be calculated such that each data center is individually load balanced as well. Partition each data center as if it were its own distinct ring. See Generating Tokens for the different approaches to generating tokens for nodes in single and multiple data center clusters.

### Snitch settings

The snitch is responsible for knowing the location of nodes within your network topology. The location of nodes affects where replicas are placed and how requests are routed between replicas. All nodes must have exactly the same snitch configuration.

The endpoint_snitch property configures the snitch for a node. In cassandra.yaml, the snitch is set to the DSE Delegated Snitch (endpoint_snitch: com.datastax.bdp.snitch.DseDelegateSnitch). The following sections describe a few commonly-used snitches. All snitches are described in the Apache Cassandra documentation. The default endpoint_snitch is the DseDelegateSnitch. The default snitch delegated by this snitch is the DseSimpleSnitch (org.apache.cassandra.locator.DseSimpleSnitch). You set the snitch used by the DseDelegateSnitch in the dse.yaml file:

- **Packaged installations:** /etc/dse/dse.yaml
- **Tarball installations:** <install_location>/resources/dse/conf/dse.yaml

#### DseSimpleSnitch

DseSimpleSnitch is used only for DataStax Enterprise (DSE) deployments. To segregate analytics and real-time workloads, this snitch logically configures Hadoop analytics nodes in a separate data center from Cassandra real-time nodes. Use DseSimpleSnitch for mixed-workload DSE clusters located in one physical data center or for multiple data center DSE clusters that have exactly two data centers: one with all Analytics nodes and the other with all Cassandra real-time nodes.

When defining your keyspace strategy_options, use Analytics or Cassandra for your data center names.

#### SimpleSnitch
For a single data center (or single node) cluster, using SimpleSnitch is usually sufficient. However, if you plan to expand your cluster at a later time to multiple racks and data centers, it is easier if you choose a rack and data center aware snitch from the start, such as the RackInferringSnitch. All snitches are compatible with all placement strategies.

**Configuring the PropertyFileSnitch**

The PropertyFileSnitch allows you to define your data center and rack names to be whatever you want. Using this snitch requires that you define network details for each node in the cluster in a cassandra-topology.properties configuration file. This file is located in /etc/dse/cassandra/conf/cassandra.yaml in packaged installations or <install_location>/resources/cassandra/conf/cassandra.yaml in binary installations.

Every node in the cluster should be described in this file, and specified exactly the same on every node in the cluster.

For example, supposing you had non-uniform IPs and two physical data centers with two racks in each, and a third logical data center for replicating analytics data:

```
# Data Center 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

# Data Center Two
110.56.12.120=DC2:RAC1
110.50.13.201=DC2:RAC1
110.54.35.184=DC2:RAC1

50.33.23.120=DC2:RAC2
50.45.14.220=DC2:RAC2
50.17.10.203=DC2:RAC2

# Analytics Replication Group
172.106.12.120=DC3:RAC1
172.106.12.121=DC3:RAC1
172.106.12.122=DC3:RAC1

# default for unknown nodes
default=DC3:RAC1
```

Make sure the data center names defined in the /etc/dse/cassandra/cassandra-topology.properties file correlates to what you name your data centers in your keyspace strategy-options.

**Choosing keyspace replication options**

When you create a keyspace, you must define the replica placement strategy and the number of replicas you want. DataStax recommends choosing NetworkTopologyStrategy for single and multiple data center clusters. It is as easy to use as SimpleStrategy and allows for expansion to multiple data centers in the future, should that become useful. It is much easier to configure the most flexible replication strategy up front, than to reconfigure replication after you have already loaded data into your cluster.

NetworkTopologyStrategy takes as options the number of replicas you want per data center. Even for single data center (or single node) clusters, you can use this replica placement strategy and just define the number of replicas for
one data center. For example (using the Beta CQL 3):

```sql
CREATE KEYSPACE test
    WITH strategy_class = 'NetworkTopologyStrategy'
    AND strategy_options: "us-east" = 6;
```

Or for a multiple data center cluster:

```sql
CREATE KEYSPACE test2 WITH strategy_class = 'NetworkTopologyStrategy'
    AND strategy_options: DC1 = 3 AND strategy_options: DC2 = 3;
```

When declaring the keyspace `strategy_options`, what you name your data centers depends on the `snitch` you have chosen for your cluster. The data center names must correlate to the snitch you are using in order for replicas to be placed in the correct location.

As a general rule, the number of replicas should not exceed the number of nodes in a replication group. However, it is possible to increase the number of replicas, and then add the desired number of nodes afterwards. When the replication factor exceeds the number of nodes, writes will be rejected, but reads will still be served as long as the desired consistency level can be met.

In DataStax Enterprise 3.0.1, the default consistency level has changed from ONE to QUORUM for reads and writes to resolve a problem finding a CassandraFS block when using consistency level ONE on a Hadoop node.

### Changing replication settings

The default replication of 1 for keyspaces is suitable only for development and testing of a single node. For production environments, it is important to change the replication of keyspaces from 1 to a higher number. To avoid operations problems, changing the replication of these system keyspaces is especially important:

- **HiveMetaStore, cfs, and cfs_archive keyspaces**
  
  If the node is an Analytics node that uses Hive, increase the HiveMetaStore and cfs keyspace replication factors to 2 or higher to be resilient to single-node failures. If you use cfs_archive, increase it accordingly.

- **dse_system keyspace**
  
  On an Analytics/Hadoop node, this keyspace contains information about the location of the job tracker. If only a single node contains the job tracker replica, other nodes cannot find the job tracker when that node is unavailable for some reason.

To change the replication these keyspaces

1. Check the name of the data center of the node.

   ```bash
   bin/nodetool -h localhost ring
   ```

   The output tells you the name of the data center for the node, for example, datacenter1.
2. Change the replication of the cfs and cfs_archive keyspaces from 1 to 3, for example:

   Using the Beta CQL 3:
   
   ```
   ALTER KEYSPACE cfs
   WITH strategy_class = NetworkTopologyStrategy
   AND strategy_options: datacenter1=3;
   
   ALTER KEYSPACE cfs_archive
   WITH strategy_class = NetworkTopologyStrategy
   AND strategy_options: datacenter1=3;
   ```

   Using CLI:
   
   ```
   [default@unknown] UPDATE KEYSPACE cfs
   WITH strategy_options = {datacenter1:3};
   
   [default@unknown] UPDATE KEYSPACE cfs_archive
   WITH strategy_options = {datacenter1:3};
   ```

   How high you increase the replication depends on the number of nodes in the cluster, as discussed in the previous section.

3. If you use Hive, update the HiveMetaStore keyspace to increase the replication from 1 to 3, for example.

4. Update the dse_system keyspace to increase the replication from 1 to 3, for example.

5. If the keyspaces you changed contain any data, run `nodetool repair` to avoid having missing data problems or data unavailable exceptions.

**Choosing node configuration options**

A major part of planning your Cassandra cluster deployment is understanding and setting the various node configuration properties. This section explains the various configuration decisions that need to be made before deploying a Cassandra cluster, be it a single-node, multi-node, or multiple data center cluster.

The properties mentioned in this section are set in the `cassandra.yaml` configuration file. Each node should be correctly configured before starting it for the first time.

**Storage settings**

By default, a node is configured to store the data it manages in `/var/lib/cassandra`. In a production cluster deployment, you should change the `commitlog_directory` so it is on a different disk device than the `data_file_directories`.

**Gossip settings**

The gossip settings control a node's participation in a cluster and how the node is known to the cluster.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cluster_name</code></td>
<td>Name of the cluster that this node is joining. Should be the same for every node in the cluster.</td>
</tr>
<tr>
<td><code>listen_address</code></td>
<td>The IP address or hostname that other Cassandra nodes will use to connect to this node. Should be changed from <code>localhost</code> to the public address for the host.</td>
</tr>
</tbody>
</table>
Purging gossip state on a node

Gossip information is also persisted locally by each node to use immediately next restart without having to wait for gossip. To clear gossip history on node restart (for example, if node IP addresses have changed), add the following line to the cassandra-env.sh file. This file is located in /usr/share/cassandra or $CASSANDRA_HOME/conf in Cassandra installations.

-Dcassandra.load_ring_state=false

Single data center deployment

In this scenario, data replication is distributed across a single data center in mixed workload clusters. For example, if the cluster has 3 Hadoop nodes, 3 Cassandra nodes, and 2 Solr nodes, the cluster has 3 data centers: one for each type of node. A multiple data center cluster has more than one data center for each type of node.

Data replicates across the data centers automatically and transparently - no ETL work is necessary to move data between different systems or servers. You can configure the number of copies of the data in each data center and Cassandra handles the rest, replicating the data for you. To configure a multiple data center cluster, see Multiple data center deployment.

Prerequisites

To correctly configure a multi-node cluster, requires the following:

- DataStax Enterprise is installed on each node.
- The total number of nodes in the cluster.
- A name for the cluster.
- The IP addresses of each node in the cluster.
- For a mixed-workload cluster, the purpose of each node.
- Which nodes will serve as the seed nodes. (Cassandra nodes use this host list to find each other and learn the topology of the ring.)
- If the nodes are behind a firewall, make sure you know what ports you need to open. See Configuring firewall port access.
- Other configuration settings you may need are described in Choosing Node Configuration Options and Node and Cluster Configuration.

This information is used to configure Node and Cluster Initialization Properties in the cassandra.yaml configuration file on each node in the cluster. Each node should be correctly configured before starting up the cluster.
**Configuration example**

This example describes installing a six node cluster spanning two racks in a single data center.

**Location of the property file:**

You set properties for each node in the cassandra.yaml file. This file is located in different places depending on the type of installation:

- **Packaged installations:** `/etc/dse/cassandra/cassandra.yaml`
- **Binary installations:** `<install_location>/resources/cassandra/conf/cassandra.yaml`

**Note**

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

**To configure a mixed-workload cluster:**

1. The nodes have the following IPs, and one node per rack will serve as a seed:
   - node0 110.82.155.0 (Cassandra seed)
   - node1 110.82.155.1 (Cassandra)
   - node2 110.82.155.2 (Cassandra)
   - node3 110.82.155.3 (Analytics seed)
   - node4 110.82.155.4 (Analytics)
   - node5 110.82.155.5 (Analytics)
   - node6 110.82.155.6 (Search - seed nodes are not required for Solr.)
   - node7 110.82.155.7 (Search)

2. Calculate the token assignments using the *Token Generating Tool* for a single data center.

<table>
<thead>
<tr>
<th>Node</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>node0</td>
<td>0</td>
</tr>
<tr>
<td>node1</td>
<td>21267647932558653966460912964485513216</td>
</tr>
<tr>
<td>node2</td>
<td>42535295865117307932921825928971026432</td>
</tr>
<tr>
<td>node3</td>
<td>63802943797675961899382738893456539648</td>
</tr>
<tr>
<td>node4</td>
<td>85070591730234615865843651857942052864</td>
</tr>
<tr>
<td>node5</td>
<td>106338239662793269832304564822427566080</td>
</tr>
<tr>
<td>node6</td>
<td>127605887595355192379876547778691307929</td>
</tr>
<tr>
<td>node7</td>
<td>148873535527910577765226390751398592512</td>
</tr>
</tbody>
</table>

3. If you have a firewall running on the nodes in your Cassandra or DataStax Enterprise cluster, you must open certain ports to allow communication between the nodes. See *Configuring firewall port access.*
4. Stop the nodes and clear the data.
   - For packaged installs, run the following commands:
     $ sudo service dse stop (stops the service)
     $ sudo rm -rf /var/lib/cassandra/* (clears the data from the default directories)
   - For binary installs, run the following commands from the install directory:
     $ ps auwx | grep cassandra (finds the Cassandra and DataStax Enterprise Java process ID [PID])
     $ sudo kill <pid> (stops the process)
     $ sudo rm -rf /var/lib/cassandra/* (clears the data from the default directories)

5. Modify the following property settings in the cassandra.yaml file for each node:

   **Note**
   In the - seeds list property, include the internal IP addresses of each seed node.

   **node0**
   
   cluster_name: 'MyDemoCluster'
   initial_token: 0
   seed_provider:
     - class_name: org.apache.cassandra.locator.SimpleSeedProvider
       parameters:
         - seeds: "110.82.155.0,110.82.155.3"
   listen_address: 110.82.155.0
   rpc_address: 0.0.0.0

   **node1 to node7**
   The properties for the rest of the nodes are the same as **Node0** except for the initial_token and listen_address:

<table>
<thead>
<tr>
<th>Node</th>
<th>initial_token</th>
<th>Listen address</th>
</tr>
</thead>
<tbody>
<tr>
<td>node1</td>
<td>21267647932558653966460912964485513216</td>
<td>110.82.155.1</td>
</tr>
<tr>
<td>node2</td>
<td>42535295865117307932921825928971026432</td>
<td>110.82.155.2</td>
</tr>
<tr>
<td>node3</td>
<td>63802943797675961899382738893456539648</td>
<td>110.82.155.3</td>
</tr>
<tr>
<td>node4</td>
<td>85070591730234615865843651857942052864</td>
<td>110.82.155.4</td>
</tr>
<tr>
<td>node5</td>
<td>106338239662793269832304564822427566080</td>
<td>110.82.155.5</td>
</tr>
<tr>
<td>node6</td>
<td>12760588759535192379876547778691307929</td>
<td>110.82.155.6</td>
</tr>
<tr>
<td>node7</td>
<td>14887353527910577765226390751398592512</td>
<td>110.82.155.7</td>
</tr>
</tbody>
</table>

6. After you have installed and configured DataStax Enterprise on all nodes, start the seed nodes one at a time, and then start the rest of the nodes.

   **Note**
   If the node has restarted because of automatic restart, you must stop the node and clear the data directories, as described in above.

   - Packaged installs: See *Starting DataStax Enterprise as a service*
   - Binary installs: See *Starting DataStax Enterprise as a stand-alone process*
7. Check that your ring is up and running:
   - **Packaged installs:** `nodetool ring -h localhost`
   - **Binary installs:**
     
     ```
     $ cd /<install_directory>
     $ bin/nodetool ring -h localhost
     ```

---

### Multiple data center deployment

In this scenario, a mixed workload cluster has more than one data center for each type of node. For example, if the cluster has 4 Hadoop nodes, 4 Cassandra nodes, and 2 Solr nodes, the cluster could have 5 data centers: 2 data centers for Hadoop nodes, 2 data centers for Cassandra nodes, and 1 data center for Solr nodes. A single data center cluster has only 1 data center for each type of node.

Data replication can be distributed across multiple, geographically dispersed data centers, between different physical racks in a data center, or between public cloud providers and on-premise managed data centers. Data replicates across the data centers automatically and transparently - no ETL work is necessary to move data between different systems or servers. You can configure the number of copies of the data in each data center and Cassandra handles the rest, replicating the data for you. To configure a single data center cluster, see [Single data center deployment](#).

### Prerequisites

To correctly configure a multi-node cluster with multiple data centers, requires:

- DataStax Enterprise is installed on each node.
- The total number of nodes in the cluster.
- A name for the cluster.
- The IP addresses of each node in the cluster.
- Which nodes will serve as the seed nodes. (DataStax Enterprise nodes use this host list to find each other and learn the topology of the ring.)
- If the nodes are behind a firewall, make sure you know what ports you need to open. See [Configuring firewall port access](#).
- Other configuration settings you may need are described in [Choosing Node Configuration Options](#) and [Node and Cluster Configuration](#).

In DataStax Enterprise 3.0.1, the default consistency level has changed from ONE to QUORUM for reads and writes to resolve a problem finding a CassandraFS block when using consistency level ONE on a Hadoop node. This information is used to configure the following properties on each node in the cluster:

- The [Node and Cluster Initialization Properties](#) in the `cassandra.yaml` file.
Assigning the data center and rack names to the IP addresses of each node in the `cassandra-topology.properties` file.

**Configuration example**

This example describes installing a six node cluster spanning two data centers. The steps for configuring multiple data centers on binary and packaged installations are the same except the configuration files are located in different directories.

**Location of the property files in packaged installations:**

- `/etc/dse/cassandra/cassandra.yaml`
- `/etc/dse/cassandra/cassandra-topology.properties`
- `/etc/dse/dse.yaml`

**Location of the property files in binary installations:**

- `<install_location>/resources/cassandra/conf/cassandra.yaml`
- `<install_location>/resources/cassandra/conf/cassandra-topology.properties`
- `<install_location>/resources/dse/conf/dse.yaml`

**Note**

After changing properties in these files, you must restart the node for the changes to take effect.

**To configure a cluster with multiple data centers:**

1. Suppose you install DataStax Enterprise on these nodes:

   10.168.66.41
   10.176.43.66
   10.168.247.41
   10.176.170.59
   10.169.61.170
   10.169.30.138

2. Assign tokens so that data is evenly distributed within each data center by calculating the token assignments with the `Token Generating Tool` and offset the token for the second data center:

<table>
<thead>
<tr>
<th>Node</th>
<th>IP Address</th>
<th>Token</th>
<th>Offset</th>
<th>Data Center</th>
</tr>
</thead>
<tbody>
<tr>
<td>node0</td>
<td>10.168.66.41</td>
<td>0</td>
<td>NA</td>
<td>DC1</td>
</tr>
<tr>
<td>node1</td>
<td>10.176.43.66</td>
<td>56713727820156410577229101238628035242</td>
<td>NA</td>
<td>DC1</td>
</tr>
<tr>
<td>node2</td>
<td>10.168.247.41</td>
<td>113427455640312821154458202477256070485</td>
<td>NA</td>
<td>DC1</td>
</tr>
<tr>
<td>node3</td>
<td>10.176.170.59</td>
<td>10</td>
<td>10</td>
<td>DC2</td>
</tr>
<tr>
<td>node4</td>
<td>10.169.61.170</td>
<td>56713727820156410577229101238628035252</td>
<td>10</td>
<td>DC2</td>
</tr>
<tr>
<td>node5</td>
<td>10.169.30.138</td>
<td>113427455640312821154458202477256070495</td>
<td>10</td>
<td>DC2</td>
</tr>
</tbody>
</table>

For more information, see *Calculating tokens for a multiple data center cluster*. 
3. Stop the nodes and clear the data.
   - For packaged installs, run the following commands:
     "$ sudo service dse stop" (stops the service)
     "$ sudo rm -rf /var/lib/cassandra/*" (clears the data from the default directories)
   - For binary installs, run the following commands from the install directory:
     "$ ps auwx | grep dse" (finds the Cassandra and DataStax Enterprise Java process ID [PID])
     "$ sudo kill <pid>" (stops the process)
     "$ sudo rm -rf /var/lib/cassandra/*" (clears the data from the default directories)

4. Modify the following property settings in the `cassandra.yaml` file for each node:
   - `initial_token`: <token from previous step>
   - `seeds`: <internal IP_address of each seed node>
   - `listen_address`: <localhost IP address>

   **node0**:
   ```yaml
   initial_token: 0
   seed_provider:
     - class_name: org.apache.cassandra.locator.SimpleSeedProvider
       parameters:
         - seeds: "10.168.66.41,10.176.170.59"
   listen_address: 10.168.66.41
   ```

   **Note**
   You must include at least one node from each data center. It is a best practice to have at more than one seed node per data center.

   **node1 to node5**
   The properties for the rest of the nodes are the same as **Node0** except for the `initial_token` and `listen_address`:

<table>
<thead>
<tr>
<th>Node</th>
<th>initial_token</th>
<th>listen address</th>
</tr>
</thead>
<tbody>
<tr>
<td>node1</td>
<td>56713727820156410577229101238628035242</td>
<td>10.176.43.66</td>
</tr>
<tr>
<td>node2</td>
<td>11342745540312821154458202477256070485</td>
<td>10.168.247.41</td>
</tr>
<tr>
<td>node3</td>
<td>10</td>
<td>10.176.170.59</td>
</tr>
<tr>
<td>node4</td>
<td>56713727820156410577229101238628035252</td>
<td>10.169.61.170</td>
</tr>
<tr>
<td>node5</td>
<td>11342745540312821154458202477256070495</td>
<td>10.169.30.138</td>
</tr>
</tbody>
</table>

5. For each node, change the `dse.yaml` file to specify the snitch to be delegated by the DseDelegateSnitch. For more information about snitches, see the About Snitches. For example, to specify the `PropertyFileSnitch`, enter:
   ```yaml
degarded_snitch: org.apache.cassandra.locator.PropertyFileSnitch
   ```

6. Determine a naming convention for each data center and rack, for example: DC1, DC2 or 100, 200 and RAC1, RAC2 or R101, R102.
7. In the `cassandra-topology.properties` file, assign data center and rack names to the IP addresses of each node, and assign a default data center name and rack name for unknown nodes. For example:

```bash
# Cassandra Node IP=Data Center:Rack
10.168.66.41=DC1:RAC1
10.176.43.66=DC2:RAC1
10.168.247.41=DC1:RAC1
10.176.170.59=DC2:RAC1
10.169.61.170=DC1:RAC1
10.169.30.138=DC2:RAC1

# default for unknown nodes
default=DC1:RAC1
```

8. After you have installed and configured DataStax Enterprise on all nodes, start the seed nodes one at a time, and then start the rest of the nodes.

**Note**

If the node has restarted because of automatic restart, you must stop the node and clear the data directories, as described above.

- Packaged installs: See [Starting DataStax Enterprise as a service](#)
- Binary installs: See [Starting DataStax Enterprise as a stand-alone process](#)

9. Check that your ring is up and running:

- Packaged installs: `nodetool ring -h localhost`
- Binary installs:
  ```bash
  $ cd /install_directory
  $ bin/nodetool ring -h localhost
  ```

<table>
<thead>
<tr>
<th>Address</th>
<th>DC</th>
<th>Rack</th>
<th>Status</th>
<th>State</th>
<th>Load</th>
<th>Owns</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.168.66.41</td>
<td>DC1</td>
<td>RAC1</td>
<td>Up</td>
<td>Normal</td>
<td>126.79 KB</td>
<td>16.6%</td>
<td>0</td>
</tr>
<tr>
<td>10.176.43.66</td>
<td>DC2</td>
<td>RAC1</td>
<td>Up</td>
<td>Normal</td>
<td>131.18 KB</td>
<td>16.6%</td>
<td>56713727820156410577229101238628035242</td>
</tr>
<tr>
<td>10.168.247.41</td>
<td>DC1</td>
<td>RAC1</td>
<td>Up</td>
<td>Normal</td>
<td>126.79 KB</td>
<td>16.6%</td>
<td>113427455640312821154456202477256070485</td>
</tr>
<tr>
<td>10.176.170.59</td>
<td>DC2</td>
<td>RAC1</td>
<td>Up</td>
<td>Normal</td>
<td>133.62 KB</td>
<td>16.6%</td>
<td>10</td>
</tr>
<tr>
<td>10.169.61.170</td>
<td>DC1</td>
<td>RAC1</td>
<td>Up</td>
<td>Normal</td>
<td>126.79 KB</td>
<td>16.6%</td>
<td>56713727820156410577229101238628035252</td>
</tr>
<tr>
<td>10.169.30.138</td>
<td>DC2</td>
<td>RAC1</td>
<td>Up</td>
<td>Normal</td>
<td>126.79 KB</td>
<td>16.6%</td>
<td>113427455640312821154456202477256070495</td>
</tr>
</tbody>
</table>

**More information about configuring data centers**

Links to more information about configuring a data center:

- Configuring nodes
- Choosing keyspace replication options
- Replication in a physical or virtual data center

**Generating tokens**

Tokens assign a range of data to a particular node within a data center.

When you start a DataStax Enterprise cluster, you must choose how the data (column family rows) is divided across the nodes in the cluster. A partitioner determines what each node stores by row (key). A token is a partitioner-dependent
element of the cluster. Each node in a cluster is assigned a token and that token determines the node’s position in the ring and what data the node is responsible for in the cluster. The tokens assigned to your nodes need to be distributed throughout the entire possible range of tokens. Each node is responsible for the region of the ring between itself (inclusive) and its predecessor (exclusive). As a simple example, if the range of possible tokens was 0 to 100 and you had 4 nodes, you would want the tokens for your nodes to be: 0, 25, 50, 75. This approach ensures that each node is responsible for an equal range of data. Each data center should be partitioned as if it were its own distinct ring.

For more detailed information, see About Data Partitioning in Cassandra.

**Note**
Each node in the cluster must be assigned a token before it is started for the first time. The token is set with the initial_token property in the cassandra.yaml configuration file.

**Token generating tool**
DataStax provides a Python program for generating tokens. Tokens are integers ranging from 0 to \(2^{127} - 1\).

**To set up the Token Generating Tool:**
1. Using a text editor, create a new file named tokengentool for your token generator program.
2. Go to https://raw.github.com/riptano/ComboAMI/2.2/tokentoolv2.py.
3. Copy and paste the program into the tokengentool file.
4. Save and close the file.
5. Make it executable:
   ```
   chmod +x tokengentool
   ```
6. Run the program:
   ```
   ./tokengentool <nodes_in_dcl1> <nodes_in_dcl2> ...
   ```
   The Token Generating Tool calculates the token values.
7. Enter the corresponding value for each node in the initial_token property of the node's cassandra.yaml file.

**Calculating tokens for a single data center**
For a single data center, DataStax recommends always using the NetworkTopologyStrategy and the RandomPartitioner. The NetworkTopologyStrategy is as easy to use as SimpleStrategy and allows for expansion to multiple data centers in the future. It is much easier to configure the most flexible replication strategy initially, than to reconfigure replication after you have already loaded data into your cluster. Be sure to configure the strategy_options for your replication strategy.

For a single data center, enter the number of nodes in Token Generating Tool. For example, for 6 nodes in a single data center, you enter:
```
./tokengentool 6
```
The tool displays the token for each node:
```
{
  "0": {
    "0": 0,
    "1": 28356863910078205288614550619314017621,
    "2": 56713727820156410577229101238628035242,
    "3": 8507059173023465843651857942052864,
```
Multiple data center deployment

Calculating tokens for multiple racks in a single data center

If you have multiple racks in single data center, enter the number of nodes in the Token Generating Tool and then assign the tokens to nodes to alternating racks. For example: rack1, rack2, rack3, rack1, rack2, rack3, and so on. Replica placement and partitioner is the same as with Calculating tokens for a single data center.

As a best practice, each rack should have the same number of nodes so you can alternate the rack assignments. For example:

./tokengentool 8

The tool displays the token for each node. The image shows the rack assignments:

Calculating tokens for a multiple data center cluster

In multiple data center deployments, replica placement must be calculated per data center using the NetworkTopologyStrategy for your custom keyspaces (DataStax Enterprise system keyspaces excluded). This strategy determines replica placement independently within each data center. The first replica is placed according to the partitioner. Additional replicas in the same data center are determined by walking the ring clockwise until a node in a different rack from the previous replica is found. If no such node exists, additional replicas are placed in the same rack. Do not use SimpleStrategy for this type of cluster and be sure to configure the strategy_options for your replication strategy.

There are different methods you can use when calculating multiple data center clusters. The important point is that the nodes within each data center manage an equal amount of data. The distribution of the nodes within the cluster is not as
Multiple data center deployment

important. DataStax recommends using DataStax Enterprise OpsCenter to re-balance a cluster.

**Alternating token assignments**

Calculate tokens for each data center using the Token Generating Tool and then alternate the token assignments so that the nodes for each data center are evenly dispersed around the ring.

```
./tokengentool 3 3
```

The tool displays the token for each node in each data center:

```
{
  "0": {  
    "0": 56713727820156410577229101238628035242,
    "1": 113427455640312821154458202477256070485,
    "2": 113427455640312821154458202477256070485
  },
  "1": {  
    "0": 28356863910078205288614550619314017621,
    "1": 85070591730234615865843651857942052863,
    "2": 141784319550391026443072753096570088106
  }
}
```

The following image shows the token position and data center assignments:

![Token Position Diagram](image)

**Avoiding token collisions**

To avoid token collisions, offset the values for each token. Although you can increment in values of 1, it is better to use a larger offset value, such as 100, to allow room to replace a dead node.

The following shows an example of a cluster with two 3 node data centers and one 2 node data center.
Expanding a DataStax AMI cluster

For instructions on expanding the DataStax AMI (Amazon Machine Image), see the latest AMI documentation.
Integrated Solutions

DSE Analytics with Hadoop

Getting Started with Analytics and Hadoop in DataStax Enterprise

In DataStax Enterprise, you can run analytics on your Cassandra data via the platform’s built-in Hadoop integration. The Hadoop component in DataStax Enterprise is not meant to be a full Hadoop distribution, but rather enables analytics to be run across DataStax Enterprise’s distributed, shared-nothing architecture. Instead of using the Hadoop Distributed File System (HDFS), DataStax Enterprise uses Cassandra File System (CassandraFS) keyspaces for the underlying storage layer. This provides replication, data location awareness, and takes full advantage of Cassandra’s peer-to-peer architecture.

DataStax Enterprise supports running analytics on Cassandra data with the following Hadoop components:

- MapReduce
- Hive for running MapReduce queries on Cassandra data.
- Pig for data analysis.
- Apache Mahout for machine learning applications.

Starting and stopping a DSE Analytics node

The way you start up a DSE Analytics node depends on the type of installation, tarball or packaged.

Tarball installation

From the install directory, use this command to start the analytics node:

```
bin/dse cassandra -t
```

The analytics node starts up.

From the install directory, use these commands to stop the analytics node:

1. Get the dse process ID from the top of output from this command:
   
   `ps auwx | grep dse`

2. Run the cassandra-stop command using the process ID (PID) from the top of the output.
   
   `bin/dse cassandra-stop <PID>`

Packaged installation

1. Enable Hadoop mode by setting this option in `/etc/default/dse`: `HADOOP_ENABLED=1`

2. Start the dse service `<start-dse>` using this command:
   
   `sudo service dse start`

   The analytics node starts up.

You stop an analytics node using this command:

```
sudo service dse stop
```
Integrated Solutions

**Hadoop getting started tutorial**

In this tutorial, you download a text file containing a State of the Union speech and upload the file to the CassandraFS. Next, you run a classic MapReduce job that counts the words in the file and creates a sorted list of word/count pairs as output. The mapper and reducer are provided in a JAR file. Download the State of the Union speech now.

This tutorial assumes you started an DataStax Enterprise 3.0.7 analytics node on Linux. Also, the tutorial assumes you have permission to perform Hadoop and other DataStax Enterprise operations, for example, that you preface commands with sudo if necessary.

**Setup**

1. Unzip the downloaded obama.txt.zip file into a directory of your choice on your file system. This file will be the input for the MapReduce job.

2. Create a directory in the CassandraFS for the input file using the `dse command version` of the familiar hadoop fs command.

   ```bash
   cd <dse-install>
   bin/dse hadoop fs -mkdir /user/hadoop/wordcount/input
   ```

3. Copy the input file that you downloaded to the CassandraFS.

   ```bash
   bin/dse hadoop fs -copyFromLocal <path>/obama.txt /user/hadoop/wordcount/input
   ```

4. Check the version number of the hadoop-examples-<version>.jar. On tarball installations, the JAR is in the DataStax Enterprise resources directory. On packaged and AMI installations, the JAR is in the /usr/dse/hadoop directory.

5. Get usage information about how to run the MapReduce job from the jar file. For example:

   ```bash
   bin/dse hadoop jar /<install_location>/resources/hadoop/hadoop-examples-1.0.4.8.jar wordcount
   ```

   The output is:

   ```
   2013-10-02 12:40:16.983 java[9505:1703] Unable to load realm info from SCDynamicStore
   Usage: wordcount <in> <out>
   ```

   If you see the SCDynamic Store message, just ignore it. The internet provides information about the message.

6. Run the Hadoop word count example in the JAR.

   ```bash
   bin/dse hadoop jar /<install_location>/resources/hadoop/hadoop-examples-1.0.4.8.jar wordcount
   /user/hadoop/wordcount/input
   /user/hadoop/wordcount/output
   ```

   The output is:

   ```
   13/10/02 12:40:36 INFO input.FileInputFormat: Total input paths to process : 0
   13/10/02 12:40:36 INFO mapred.JobClient: Running job: job_201310020848_0002
   13/10/02 12:40:37 INFO mapred.JobClient: map 0% reduce 0%
   . . .
   13/10/02 12:40:55 INFO mapred.JobClient: FILE_BYTES_WRITTEN=19164
   13/10/02 12:40:55 INFO mapred.JobClient: Map-Reduce Framework
   . . .
   ```
7. List the contents of the output directory on the CassandraFS.

```bash
bin/dse hadoop fs -ls /user/hadoop/wordcount/output
```

The output looks something like this:

```
Found 3 items
-rwxrwxrwx  1 root wheel      0 2013-10-02 12:58 /user/hadoop/wordcount/output/_SUCCESS
drwxrwxrwx  - root wheel      0 2013-10-02 12:57 /user/hadoop/wordcount/output/_logs
-rwxrwxrwx  1 root wheel  24528 2013-10-02 12:58 /user/hadoop/wordcount/output/part-r-00000
```

8. Using the output file name from the directory listing, get more information about the output file using the `dsetool` utility.

```bash
bin/dsetool checkcfs /user/hadoop/wordcount/output/part-r-00000
```

The output is:

```
Path: cfs://127.0.0.1/user/hadoop/wordcount/output/part-r-00000
INode header:
  File type: FILE
  User: root
  Group: wheel
  Permissions: rwxrwxrwx (777)
  Block size: 67108864
  Compressed: true
  First save: true
  Modification time: Wed Oct 02 12:58:05 PDT 2013
INode:
  Block count: 1
  Blocks:       subblocks    length   start     end
     (B) f2fa9d90-2b9c-11e3-9ccb-73ded3cb6170: 1 24528 0 24528
                  f3030200-2b9c-11e3-9ccb-73ded3cb6170: 24528 0 24528
  Block locations:
     f2fa9d90-2b9c-11e3-9ccb-73ded3cb6170: [localhost]
Data:
  All data blocks ok.
```

9. Finally, look at the output of the MapReduce job—the list of word/count pairs using the dse version of the familiar `hadoop fs -cat` command.

```bash
bin/dse hadoop fs -cat /user/hadoop/wordcount/output/part-r-00000
```

The output is:

```
"D."  1
"Don't" 1
"I"  4
...
```

**Hadoop demos**

After starting Hadoop, you can run these additional Hadoop demos:

- **Portfolio Manager demo**: Demonstrates a hybrid workflow using DataStax Enterprise.
- **Hive Demo**: Demonstrates using Hive to access data in Cassandra.
- **Mahout Demo**: Demonstrates Mahout support in DataStax Enterprise by determining which entries in the sample input data file remained statistically in control and which have not.
Pig Demo: Create a Pig relation, perform a simple MapReduce job, and put the results back into CassandraFS or into a Cassandra column family.

Sqoop Demo: Migrates data from a MySQL database containing information from the North American Numbering Plan.

Setting the replication factor

The default replication for system keyspaces is 1. This replication factor is suitable for development and testing of a single node, not for a production environment. For production increase the replication factors to at least 2. This ensures resilience to single-node failures. For example:

```
[default@unknown] UPDATE KEYSPACE cfs
  WITH placement_strategy = 'org.apache.cassandra.locator.NetworkTopologyStrategy'
  AND strategy_options={Analytics:3};
```

For more information, see Changing replication settings.

Configuration for running jobs on a remote cluster

This information is intended for advanced users.

To connect to external addresses:

1. Make sure that the hostname resolution works properly on the localhost for the remote cluster nodes.

2. Copy the dse-core-default.xml and dse-mapred-default.xml files from any working remote cluster node to your local Hadoop conf directory.

3. Run the job with dse hadoop.

4. If you need to override the JT location or if DataStax Enterprise cannot automatically detect the JT location, before running the job, define the HADOOP_JT environment variable:

   ```plaintext
   HADOOP_JT=<jobtracker host>:<jobtracker port> dse hadoop jar ....
   ```

5. If you need to connect to many different remote clusters from the same host:
   a. Before starting the job, copy the remote Hadoop conf directories fully to the local node (into different locations).
   b. Select the appropriate location by defining HADOOP_CONF_DIR.

About Portfolio Manager demo application

Your DataStax Enterprise (DSE) installation contains a demo application that shows a sample mixed workload on a DSE cluster. The use case is a financial application where users can actively create and manage a portfolio of stocks.

On the Cassandra 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 where each portfolio updates based on its overall value and the percentage of gain or loss compared to the purchase price. This 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 Hive MapReduce 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.

Prerequisites

- A single-node or multiple node instance of DataStax Enterprise is installed.
Your cluster is configured and running with one analytics node:

- **Binary install**
- **Package install**

**Running the demo**

To run the demo:

1. On the analytics node, verify that DataStax Enterprise is running in analytics mode:
   - **Binary install:** `<install_location>/bin/nodetool ring -h localhost`
   - **Package install:** `nodetool ring -h localhost`

2. Go to the portfolio manager demo directory.
   - **Binary install:** `cd <install_location>/demos/portfolio_manager`
   - **Packaged install:** `cd /usr/share/dse-demos/portfolio_manager`

**Note**

You must run the pricer utility from a directory where you have write permissions (such as your home directory), or else run it as root or using sudo.

3. Run the `bin/pricer` utility to generate stock data for the application:
   - To see all of the available options for this utility:
     ```bash
     bin/pricer --help
     ```
   - If running on a single node cluster on localhost:
     ```bash
     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.

4. Start the web service:
   ```bash
   cd website
   ./start
   ```
5. Open a browser and go to http://localhost:8983/portfolio to see the real-time Portfolio Manager demo application.

![Portfolio Manager Demo](image)

6. Open another terminal.

7. Start Hive and run the MapReduce job for the demo in Hive.
   - Binary
     
     `<install_location>/bin/dse hive -f <install_location>/demos/portfolio_manager/10_day_loss.q`
   - Packaged install:
     
     `dse hive -f /usr/share/dse-demos/portfolio_manager/10_day_loss.q`
   
   The MapReduce job takes several minutes to run.

8. Open the URL http://localhost:50030/jobtracker.jsp in a browser to watch the progress in the job tracker.
9. After the job completes, refresh the Portfolio Manager web page to see the results of the Largest Historical 10 day Loss for each portfolio.

**Using the job tracker node**

For each MapReduce job submitted to the job tracker, DataStax Enterprise schedules a series of tasks on the analytics nodes. One task tracker service per node handles the map and reduce tasks scheduled for that node. Within a data center, the job tracker monitors the execution and status of distributed tasks that comprise a MapReduce job.

**Using multiple job tracker services**

DataStax Enterprise 2.1 and later can use multiple job tracker nodes in a cluster, one per data center. In deployments having multiple data centers far away from each other, using multiple job trackers and multiple file systems can improve performance by taking advantage of data locality on each cluster.

Tasks related to the job tracker are:

- Setting the Job Tracker Node
- Managing the Job Tracker Using dsetool Commands
- Changing the Job Tracker Client Port

**Setting the job tracker node**

There are several ways to set the job tracker node:

- Configure the Cassandra seeds list in cassandra.yaml. DataStax Enterprise designates the first analytics node from the seeds list as the job tracker node.

```
dse cassandra -t -j
```

or in a binary distribution:

```
<install_location>/bin/dse cassandra -t -j
```

- Start up an analytics node using the -j option.

```
dse cassandra -t -j
```

- Use the dsetool movejt command.

If you list any IP addresses in the seeds list of the cassandra.yaml file, DataStax Enterprise nominates a node from the list in each data center to be the job tracker.
About the reserve job tracker

DataStax Enterprise 2.1 and later nominates a node in the cluster as a reserve job tracker for a data center. The reserve job tracker becomes the job tracker when, for some reason, there is no local node in the data center that can function as job tracker.

When you upgrade from DataStax Enterprise 2.0 and earlier to DataStax Enterprise 2.2, the job tracker node from the old release is automatically designated as the temporary, reserve job tracker. After migration, the local job tracker election process runs in each data center to determine permanent, reserve job trackers.

Managing the job tracker using dsetool commands

Several dsetool commands are useful for managing job tracker nodes:

- `dsetool jobtracker`
  Returns the job tracker hostname and port to your location in the data center where you issued the command.

- `dsetool movejt <data center>-<workload> <node IP>`
  Moves the job tracker and notifies the task tracker nodes.

- `dsetool movejt <node IP>`
  In DataStax Enterprise 2.1 and later, if you do not specify the data center name, the command moves the reserve job tracker.

- `dsetool listjt`
  Lists all job tracker nodes grouped by their local data center.

- `dsetool ring`
  Lists the nodes and types of the nodes in the ring.

Listing job trackers example

If you are not sure which nodes in your DSE cluster are job tracker, run the following command:

```
dsetool jobtracker
```

or in a binary distribution:

```
<install_location>/bin/dsetool jobtracker
```

Moving the job tracker node example

If your primary job tracker node fails, you can use `dsetool movejt` to move the job tracker to another analytics node in the cluster.

1. Log in to a DataStax Enterprise analytics node.
2. Run the `dsetool movejt` command and specify the data center name, hyphen, Analytics (for the workload), and the IP address of the new job tracker node in your DataStax Enterprise cluster. For example, to move the job tracker to node 110.82.155.4 in the DC1 data center:

   ``
dsetool movejt DC1-Analytics 110.82.155.4
```

   or in a binary distribution:

   ```
<install_location>/bin/dsetool movejt DC1-Analytics 110.82.155.4
```
3. Allow 20 seconds for all of the analytics nodes to detect the change and restart their task tracker processes.
4. In a browser, connect to the new job tracker and confirm that it is up and running. For example (change the IP to reflect your job tracker node IP):

   http://110.82.155.4:50030

5. If you are running Hive or Pig MapReduce clients, you must restart them to pick up the new job tracker node information.

**Changing the job tracker client port**

By default, the job tracker listens on port 8012 for client messages. You can use another port by configuring the mapred.job.tracker property.

**To change the job tracker client port:**

1. Open the mapred-site.xml file for editing. The location of this file is:
   - **Packaged installations:** /etc/dse/hadoop
   - **Binary installations:** /<install_location>/resources/hadoop/conf

2. Locate the mapred.job.tracker property.

   ```
   <!-- Auto detect the dse job tracker -->
   <property>
   <name>mapred.job.tracker</name>
   <value>${dse.job.tracker}</value>
   <description>
   The address of the job tracker
   </description>
   
   2. In the mapred.job.tracker property, change the placeholder `${dse.job.tracker}` value to the port number you want to use. For example, change the port number from the default to 8013.

   ```

   ```
   <!-- Auto detect the dse job tracker -->
   <property>
   <name>mapred.job.tracker</name>
   <value>8013</value>
   <description>
   The address of the job tracker
   </description>
   
   About the Cassandra File System

   The **Cassandra File System (CassandraFS)** replaces the Hadoop Distributed File System (HDFS). It is designed to simplify the operational overhead of Hadoop by removing the single points of failure in the Hadoop NameNode and to offer easy Hadoop integration for Cassandra users. When an analytics node starts up, DSE creates a default CassandraFS rooted at cfs:/ and an archive file system named cfs-archive.

   **Configuring a CFS superuser**

   A CFS superuser is the DSE daemon user, the user who starts DataStax Enterprise. A cassandra superuser, set up using the `CQL CREATE USER ... SUPERUSER command`, is also a CFS superuser.

   A CFS superuser can modify files in the CassandraFS without any restrictions. Files that a superuser adds to the CassandraFS are password-protected.

   **Using multiple Cassandra File Systems**
DataStax Enterprise 2.1 and later support multiple CassandraFS’s. Some typical reasons for using an additional CassandraFS are:

- To isolate hadoop-related jobs
- To configure keyspace replication by job
- To segregate file systems in different physical data centers
- To separate Hadoop data in some other way

**Creating multiple Cassandra File Systems**

To create an additional CassandraFS:

1. Open the core-site.xml file for editing. This file is located in:
   - Packaged installations: /etc/dse/hadoop
   - Binary installations: /<install_location>/resources/hadoop/conf
2. Add one or more property elements to core-site.xml using this format:
   
   ```
   <property>
   <name>fs.cfs-<filesystem name>.impl</name>
   <value>com.datastax.bdp.hadoop.cfs.CassandraFileSystem</value>
   </property>
   ```
3. Save the file and restart Cassandra.

DSE creates the new CassandraFS.

To access the new CassandraFS, construct a URL using the following format:

```
cfs-<filesystemname>:<path>
```

For example, assuming the new file system name is NewCassandraFS:

```
hadoop fs -copyFromLocal /tmp/giant_log.gz cfs-NewCassandraFS://cassandrahost/tmp
```
```
hadoop fs distcp hdfs:/// cfs-NewCassandraFS:///
```

**Using Hive**

DataStax Enterprise includes a Cassandra-enabled Hive MapReduce client. Hive is a data warehouse system for Hadoop that projects a relational structure onto data stored in Hadoop-compatible file systems. You query the data using a SQL-like language called HiveQL. You can start the Hive client on any analytics node and run MapReduce queries directly on data already stored in Cassandra. DataStax maps any existing column families into Hive tables. You do not need to run a stand-alone Hive MetaStore. To define new Hive tables and load data into the Cassandra File System (CassandraFS), use HiveQL just as you would use it in an HDFS-based Hadoop implementation.

HiveQL is extensible by virtue of its user defined types. You upload custom user-defined functions to manipulate the data in your queries.

**About the Hive metastore**

Metadata about the objects you define in Hive is stored in a database called the metastore. In HDFS-based Hive, when you run Hive on your local machine, your DDL commands create objects in a local metastore that is not available to other Hive clients. DataStax Enterprise implements the Hive metastore as a keyspace within Cassandra. Therefore, the metastore is shared and requires no configuration except increasing the default replication factor.
Note
The default replication for system keyspaces is 1. This replication factor is suitable for development and testing, not for a production environment. For production increase the replication factors for the HiveMetaStore and cfs keyspaces to at least 2; see Changing replication settings.

Using Hive
This section provides information about:

- Setting the Job Tracker node for Hive
- Starting a Hive client
- Starting the Hive server
- Creating Hive CassandraFS tables
- Changing Hive storage properties on the fly
- ODBC driver for Hive

Setting the Job Tracker node for Hive
Hive generates MapReduce jobs for most of its queries. Hive MapReduce jobs are submitted to the job tracker node for the DataStax Enterprise cluster. Hive clients automatically select the correct job tracker node upon startup. You set the job tracker node for Hive as you would for any analytics node. Use the dsetool commands to manage the job tracker.

Starting the Hive server
To connect to Hive via the JDBC driver, start Hive on one of the Hadoop nodes as follows:

dse hive --service hiveserver

or in a binary distribution:

<install_location>/bin/dse hive --service hiveserver

Starting a Hive client
When you install DataStax Enterprise using the packaged or AMI distributions, you can start Hive as follows:

dse hive

or in a binary distribution:

<install_location>/bin/dse hive

Creating Hive CassandraFS tables
Use Hive with CassandraFS as you would use it in an HDFS-based Hadoop implementation. Create Hive tables using the CREATE TABLE command.

For example:

hive> CREATE TABLE invites (foo INT, bar STRING)  
     PARTITIONED BY (ds STRING);

Load data into a table using the LOAD DATA command. The HiveQL Manual provides more information about the HiveQL syntax. The loaded data resides in the cfs keyspace. Your Hive metadata store also resides in Cassandra in its own keyspace.
Integrated Solutions

For example:

```
hive> LOAD DATA LOCAL INPATH '<install_location>/resources/hive/examples/files/kv2.txt'
OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');

hive> LOAD DATA LOCAL INPATH '<install_location>/resources/hive/examples/files/kv3.txt'
OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-08');

hive> SELECT count(*), ds FROM invites GROUP BY ds;
```

**Note**
The paths to the Hive example files shown in the example LOAD commands above are for the binary distribution.

**Using the count function**
Using cqlsh, set the consistency level to ALL before issuing a Hive SELECT expression containing the count function. Using ALL ensures that when you ping one node for a scan of all keys, the node is fully consistent with the rest of the cluster. Using a consistency level other than ALL can return resultsets having fewer rows than expected because replication has not finished propagating the rows to all nodes. A count that is higher than expected can occur because tombstones have not yet been propagated to all nodes.

To get accurate results from the count function using a consistency level other than ALL:

- Repair all nodes.
- Prevent new data from being added or deleted.

**Changing Hive storage properties on the fly**
You can change Hive storage properties listed in the serdeproperties and tblproperties on the fly using DataStax Enterprise 2.1 and later. Using the Hive SET command, set properties in the hive session. The settings become effective for the next query. Using DataStax Enterprise 2.0 and earlier, you had to use ALTER TABLE to change the storage properties.

**Running the Hive demo**
The Hive demo shows you how to access data in Cassandra. DataStax Enterprise uses a custom storage handler to allow direct access to data stored in Cassandra through Hive.

**Mapping a Hive database to a Cassandra keyspace**
To access data stored in Cassandra, first define a database in Hive that maps to a keyspace in Cassandra. One way you can map them is by making sure that the name is the same in both Hive and Cassandra. For example:

```
hive> CREATE DATABASE PortfolioDemo;
```

Optionally, if your Hive database and Cassandra keyspace use different names (or the Cassandra keyspace does not exist), you can declare keyspace properties in your external table definition using the TBLPROPERTIES clause. If the keyspace does not yet exist in Cassandra, Hive will create it.

For example, in the case where the keyspace exists in Cassandra but under a different name:

```
hive> CREATE DATABASE MyHiveDB;

hive> CREATE EXTERNAL TABLE MyHiveTable(row_key string, col1 string, col2 string) STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
```
Or if the keyspace does not exist in Cassandra yet and you want to create it:

```
hive> CREATE EXTERNAL TABLE MyHiveTable(row_key string, col1 string, col2 string)
STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
TBLPROPERTIES ( "cassandra.ks.name" = "MyCassandraKS",
"cassandra.ks.repfactor" = "2",
"cassandra.ks.strategy" = "org.apache.cassandra.locator.SimpleStrategy");
```

You use the `cassandra.ks.repfactor` property to define the replication factor for a keyspace that uses the SimpleStrategy replication strategy.

To create a keyspace that uses the NetworkTopologyStrategy replication strategy, use the `cassandra.ks.stratOptions` property to define the replication factors for data centers:

```
hive> CREATE EXTERNAL TABLE MyHiveTable(row_key string, col1 string, col2 string)
STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
TBLPROPERTIES ( "cassandra.ks.name" = "MyCassandraKS",
"cassandra.ks.stratOptions" = "DC1:1, DC2:2, DC3:1",
"cassandra.ks.strategy" = "org.apache.cassandra.locator.NetworkTopologyStrategy");
```

### Hive to Cassandra table mapping

An external table in Hive maps to a column family in Cassandra. In DataStax Enterprise 2.0 and earlier, all automatically created hive tables relied on the SERDE property to map typed data in the Cassandra column family to strings. Hive did not store Cassandra data in a typed manner.

In DataStax Enterprise 2.1 and later, automatically created hive tables use the following logic for mapping Cassandra Types to Hive:

<table>
<thead>
<tr>
<th>Cassandra Type</th>
<th>Hive Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTF8Type</td>
<td>string</td>
</tr>
<tr>
<td>AsciiType</td>
<td>string</td>
</tr>
<tr>
<td>DateType</td>
<td>timestamp</td>
</tr>
<tr>
<td>LongType</td>
<td>bigint</td>
</tr>
<tr>
<td>Int32Type</td>
<td>int</td>
</tr>
<tr>
<td>DoubleType</td>
<td>double</td>
</tr>
<tr>
<td>FloatType</td>
<td>float</td>
</tr>
<tr>
<td>BooleanType</td>
<td>boolean</td>
</tr>
<tr>
<td>UUIDType</td>
<td>binary</td>
</tr>
<tr>
<td>TimeUUIDType</td>
<td>binary</td>
</tr>
<tr>
<td>all other types</td>
<td>binary</td>
</tr>
</tbody>
</table>

### Validating types

The `STORED BY` clause specifies the storage handler to use, which for Cassandra is `org.apache.hadoop.hive.cassandra.CassandraStorageHandler`. The `WITH SERDEPROPERTIES` clause specifies the properties used when serializing/deserializing data passed between the Hive table and Cassandra.

Validate types using the `cassandra.cf.validatorType`. Set the value of the `validatorType` to the Cassandra types that map to Hive types.
hive> DROP TABLE IF EXISTS StockHist;
create external table StockHist(row_key string, column_name string, value double)
STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
WITH SERDEPROPERTIES ("cassandra.ks.name" = "PortfolioDemo",
"cassandra.cf.validatorType" = "UTF8Type,UTF8Type,DoubleType"
);

This forces the columns to be deserialized from CassandraTypes into Strings.

Specifying CassandraFS and MapReduce properties
The TBLPROPERTIES clause specifies CassandraFS and MapReduce properties for the table. For example:

hive> CREATE EXTERNAL TABLE Users(userid string, name string,
email string, phone string)
STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
WITH SERDEPROPERTIES ("cassandra.columns.mapping" = ":key,user_name,primary_email,home_phone")
TBLPROPERTIES ("cassandra.range.size" = "100",
"cassandra.slice.predicate.size" = "100");

For static Cassandra column families that model objects (such as users), mapping them to a relational structure is straightforward. In the example above, the first column of the Hive table (userid) maps to the row key in Cassandra. The row key in Cassandra is similar to a PRIMARY KEY in a relational table and should be the first column in your Hive table. If you know what the column names are in Cassandra, you can map the Hive column names to the Cassandra column names as shown above.

However, for dynamic column families (such as time series data), all rows likely have a different set of columns, and in most cases you do not know what the column names are. To convert this type of column family to a Hive table, you would convert a wide row in Cassandra to a collection of short rows in Hive using a special set of column names (row_key, column_name, value). For example:

hive> CREATE EXTERNAL TABLE PortfolioDemo.Stocks
(row_key string, column_name string, value double)
STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler';

Mapping column names to Cassandra row and column names
Optionally, you can add a WITH SERDEPROPERTIES clause to map meaningful column names in Hive to the Cassandra row key, column names and column values. For example:

hive> CREATE EXTERNAL TABLE PortfolioDemo.PortfolioStocks
(portfolio string, ticker string, number_shares int)
STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
WITH SERDEPROPERTIES ("cassandra.columns.mapping" = ":key,:column,:value");

Using cassandra.columns.mapping, you can use a mapping of meaningful column names you assign in the Hive table to Cassandra row key, column/subcolumn names and column/subcolumn values. In the mapping, :key is a special name reserved for the column family row key, :column for column names, :subcolumn for subcolumn names (in super column families), and :value for column (or subcolumn) values. If you do not provide a mapping, then the first column of the Hive table is assumed to be the row key of the corresponding Cassandra column family.

Once you have defined your external tables in Hive, you can query the database to select from the Hive table. For example:

hive> SELECT * FROM PortfolioDemo.Stocks;

Any other query besides a SELECT * in Hive runs as a MapReduce job.

Inserting data into Cassandra via Hive
Once you have defined an external table object in Hive that maps to a Cassandra column family, you can move the results of MapReduce queries back into Cassandra using the `INSERT OVERWRITE TABLE` command. For example:

```
hive> CREATE EXTERNAL TABLE PortfolioDemo.HistLoss
    (row_key string, worst_date string, loss string)
STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler';
```

```
hive> INSERT OVERWRITE TABLE PortfolioDemo.HistLoss
    SELECT a.portfolio, rdate, cast(minp as string)
    FROM (SELECT portfolio, MIN(preturn) as minp
    FROM portfolio_returns
    GROUP BY portfolio )
a JOIN portfolio_returns b ON (a.portfolio = b.portfolio and a.minp = b.preturn);
```

**SERDEPROPERTIES reference**

The `SERDEPROPERTIES` clause specifies the properties used when serializing/deserializing data passed between the Hive table and Cassandra. You can add a `WITH SERDEPROPERTIES` clause to map meaningful column names in Hive to the Cassandra row key, column names and column values.

The following properties can be declared in a `WITH SERDEPROPERTIES` clause:

- `cassandra.columns.mapping` - Mapping of Hive to Cassandra columns
- `cassandra.cf.name` - Column family name in Cassandra
- `cassandra.host` - IP of a Cassandra node to connect to
- `cassandra.port` - Cassandra RPC port - default 9160
- `cassandra.partitioner` - Partitioner - default RandomPartitioner

**TBLPROPERTIES reference**

The `TBLPROPERTIES` clause specifies CassandraFS and MapReduce properties for the table. The following properties can be declared in a `TBLPROPERTIES` clause:

- `cassandra.ks.name` - Cassandra keyspace name.
- `cassandra.ks.repfactor` - Cassandra replication factor - default 1.
- `cassandra.ks.strategy` - Replication strategy - default SimpleStrategy.
- `cassandra.input.split.size` - MapReduce split size - default 64 * 1024. This property dictates how many rows are processed per mapper (that is, 64k rows per split).
- `cassandra.range.size` - MapReduce key range size - default 1000. This property specifies the number of rows fetched at a time over the split. For example, if a mapper is processing a total of 64k rows, it pulls 1000 rows at a time 64 times.
- `cassandra.slice.predicate.size` - MapReduce slice predicate size - default 1000. This property describes which columns to fetch from each row and how many columns per row are fetched. For example, for a wide row in Hive, this is the paging size for columns across a row. This means that a row with 10,000 columns is fetched 1000 columns at a time.

**MapReduce performance tuning**

You can change performance settings in the following ways:

- In your external table definitions, using the `TBLPROPERTIES` or `SERDEPROPERTIES` clauses.
- Using the `set` Hive command. For example: `set mapred.reduce.tasks=32;`
In the `mapred-site.xml` file.

Packaged installations: /etc/dse/hadoop/mapred-site.xml

Binary installations: `<install_location>/resources/hadoop/conf/mapred-site.xml`

**Note**
This is a system setting so if you change it you must restart the analytics nodes.

**Speeding up map reduce jobs:**
Increase your mappers to one per CPU core by setting `mapred.tasktracker.map.tasks.maximum` in `mapred-site.xml`.

**Accessing rows with 100,000 columns or more:**
In the `TBLPROPERTIES` clause, set the `cassandra.range.size` and `cassandra.slice.predicate.size` to fetch one row with 100,000 columns at once. Although this requires more disk usage and scan runs, it is better to fetch one row with 100,000 columns at once than fetching 1000 rows with 100,000 columns at a time.

**Increasing the number of map tasks to maximize performance:**

- Turn off map output compression, in `mapred-site.xml`, to lower memory usage.
- The `cassandra.input.split.size` property (in `TBLPROPERTIES`) sets how many rows are processed per mapper. The default size is 64k rows per split. You can decrease the split size to create more mappers.

**Improving Counter Performance:**
For example, when performing `select count(1) from <column family>;`, you can improve the speed of the counter by setting `cassandra.enable.widerow.iterator=false`. This setting causes all columns after the 1000th column to be ignored for each row, thus improving the speed of the counter.

**Out of Memory Errors:**
When your mapper or reduce tasks fail, reporting Out of Memory (OOM) errors, turn the `mapred.map.child.java.opts` setting in Hive to:

```
SET mapred.child.java.opts="-server -Xmx512M"
```
You can also lower memory usage by turning off map output compression in `mapred-site.xml`.

**ODBC driver for Hive**
DataStax provides an [ODBC driver for Hive on Windows](https://datastax.github.io/datastax-odbc/)

**Using the DataStax ODBC driver for Hive on Windows**
The DataStax ODBC Driver for Hive provides Windows users access to the information stored in the Hadoop distribution bundled into DataStax Enterprise. This driver allows you to access the data stored on your DataStax Enterprise Hadoop 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 HiveQL.

**Prerequisites**
- Windows® 7 Professional or Windows® 2008 R2. Both 32- and 64-bit editions are supported.
- Microsoft Visual C++ 2010 runtime.
- A cluster with a Hadoop node running the Hive server. See [Starting the Hive server](https://datastax.github.io/datastax-odbc/).

**Installing the driver**
To install the DataStax ODBC driver on a Windows platform:

1. Download the driver from Client Libraries and CQL Drivers.
2. Double-click the downloaded file and follow the wizard's instructions.

**Configuring the driver**

Set up the DataStax ODBC driver for access by your BI tool.

1. Click **Start > Program Files > DataStax Hive ODBC Connector > ODBC Driver Manager**.
2. Click the **Drivers** tab to verify that the driver is present.

![ODBC Data Source Administrator](image)

3. Create either a User or System DSN (data source name) for your BI tool connection.
   
   a. Click the **User DSN** or **System DSN** tab.
   
   b. Click **Add**, select **DataStax Hive ODBC Connector**, and then click **Finish**.
   
   c. In **DataStax Hive ODBC Connector 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.</td>
</tr>
<tr>
<td>Host</td>
<td>IP or hostname of your Hive server.</td>
</tr>
<tr>
<td>Port</td>
<td>Listening port for the Hive service.</td>
</tr>
<tr>
<td>Database</td>
<td>By default, all tables reside within the default database. To check for the appropriate database, use the <code>show databases</code> Hive command.</td>
</tr>
</tbody>
</table>

   d. Click **Test**. The test results are displayed.

   **Note**
   
   If your DataStax Enterprise cluster is on Amazon EC2, you must open the listing port for the Hive Server. For more information, refer to the latest AMI documentation.

   4. To configure the advanced options, see Appendix C in the *DataStax Hive ODBC Connector User Guide for Windows*:

   **Start > Program Files > DataStax Hive ODBC Connector > User's Guide**

**Using the DataStax ODBC driver for Hive**
After configuring the ODBC data source for Hive, you can connect and pull data from Hive using any compliant BI tool. For example, to retrieve data using Microsoft Excel:

1. Use the data connection wizard to select your new ODBC data source:

![Data Connection Wizard](image)

2. In **Connect to OBDC Data Source**, select **DSE2 Hive**, and then click **Next**.
3. Select one or more data objects (or construct a query) to retrieve the data, and then click **Finish**.

After the ODBC query is executed and the data is retrieved, a Hive MapReduce job runs on the server:

```
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201208230939_0006,
Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201208230939_0006
Kill Command = ./dse hadoop job
 -Dmapred.job.tracker=127.0.0.1:8012 -kill job_201208230939_0006
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2012-08-23 12:44:39,795 Stage-1 map = 0%, reduce = 0%
2012-08-23 12:44:42,824 Stage-1 map = 100%, reduce = 0%
2012-08-23 12:44:44,833 Stage-1 map = 100%, reduce = 100%
Ended Job = job_201208230939_0006
MapReduce Jobs Launched:
Job 0: Map: 1   HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
```

**Using Mahout**

DataStax 2.1 and later support **Apache Mahout**, a Hadoop component that offers machine learning libraries. Mahout facilitates building intelligent applications that learn from data and user input. Machine learning use cases are many and some, such as the capability of web sites to recommend products to visitors based on previous visits, are notorious.
Currently, Mahout jobs that use Lucene features are not supported.

**Running the Mahout demo**

The DataStax Enterprise installation includes a Mahout demo. The demo determines with some percentage of certainty which entries in the input data remained statistically in control and which have not. The input data is time series historical data. Using the Mahout algorithms, the demo classifies the data into categories based on whether it exhibited relatively stable behavior over a period of time. The demo produces a file of classified results.

**To run the Mahout demo**

1. After installing DataStax Enterprise, start an analytics node.
2. Go to the demos directory in one of these locations:
   - **Tarball install:** `cd <install_location>/demos/mahout`
   - **Packaged install:** `cd /usr/share/dse-demos/mahout`
3. Run the script in the demo directory. For example, on Linux:
   
   `./run_mahout_example.sh`

   If you are running OpsCenter, view the Hadoop job progress:

   ![Hadoop Job Progress](image)

   When the demo completes, a message appears on the standard output about the location of the output file. For example:

   ```
   The output is in /tmp/clusteranalyze.txt
   ```

**Using Mahout commands in DataStax Enterprise**

You can run Mahout commands on the *dse command line*. For example, on Mac OSX to get a list of which commands are available:

```
   cd ~/dse-3.0
   bin/dse mahout
```

The list of commands appears.

**Mahout command line help**

You use one of these commands as the first argument plus the help option.

```
   cd ~/dse-3.0
   bin/dse mahout arff.vector --help
```

The output is help on the arff.vector command.
Add Mahout classes to the class path, execute Hadoop command

You can use Hadoop commands to work with Mahout. Using this syntax first adds Mahout classes to the class path, and then executes the Hadoop command.

```
dse mahout hadoop <hadoop command> <options>
```

For example, a Mahout file as input to this command, converts the file to text, so you can read it:

```
cd ~/dse-3.0
bin/dse mahout hadoop fs -text <mahout file> | more
```

The Apache web site offers an in-depth tutorial.

Using Pig

DataStax Enterprise (DSE) includes a CassandraFS-enabled Apache Pig Client. Pig is a platform for analyzing large data sets that uses a high-level language (called Pig Latin) for expressing data analysis programs. Pig Latin lets developers specify a sequence of data transformations such as merging data sets, filtering them, and applying functions to records or groups of records. Pig comes with many built-in functions and allows you to develop your own functions for special-purpose processing. Documentation for Pig Latin is available from Apache.

Pig Latin programs run in a distributed fashion on a DSE cluster. These programs are compiled into MapReduce jobs and executed using Hadoop. When using Pig with DSE, all jobs can be run in MapReduce mode, even on a single-node cluster. Since all Hadoop nodes are peers in DSE (no Name Node), there is no concept of local mode for Pig. DSE Pig includes a custom storage handler for Cassandra that allows you to run Pig programs directly on data stored in Cassandra. The native Pig storage handler stores data in CassandraFS, the Cassandra-enabled Hadoop distributed file system.

Setting the Job tracker node for Pig

Pig Latin programs are compiled into sequences of MapReduce jobs that are run in parallel. Jobs are submitted to the job tracker node for the DataStax Enterprise cluster. Pig clients will automatically select the correct job tracker node on startup. You set the job tracker node for Pig as you would for any analytics (Hadoop) node and use the dsetool commands to manage the job tracker.

Starting Pig

DataStax Enterprise must be running as an analytics (Hadoop) node. See Starting and stopping DataStax Enterprise.

Start the Pig shell as follows:

**Packaged installs:** dse pig

**Binary installs:** `<install_location>/bin/dse pig`

After Pig starts the prompt changes to grunt. Be sure to use ";" for the shell commands.

Working in DSE Pig

DataStax Enterprise allows you to use Pig with data stored in CassandraFS just as you would in a regular Hadoop implementation (using the default Pig storage handler). Pig Latin statements work with relations. A Pig relation is defined as follows:

- A relation is a bag (more specifically, an outer bag).
- A bag is a collection of tuples.
- A tuple is an ordered set of fields.
- A field is a piece of data.
A Pig relation is similar to a table in a relational database, where the tuples in the bag correspond to the rows in a table. However, unlike a relational table, Pig relations do not require that every tuple contain the same number of fields or that the fields in the same position (column) be of the same type. Pig relations resemble Cassandra tables more than relational tables. The Apache Pig documentation contains more information on defining and working with Pig relations.

Using Pig to access data in Cassandra

DSE uses a custom storage handler, CassandraStorage() that allows direct access to data stored in Cassandra through Pig. To access data in Cassandra, the target keyspace and table must already exist. Pig can read and write data in a Cassandra table, but will not create the table.

The Pig LOAD command pulls data into a Pig relation from Cassandra via the CassandraStorage handler. You do not need to specify type information as it is automatically inferred. For a regular table, the format of the Pig LOAD command is as follows:

\[
\text{<pig_relation_name>} = \text{LOAD} \ 'cassandra://<\text{keyspace}>/<\text{column_family}>' \\
\text{USING} \ \text{CassandraStorage}();
\]

The Pig STORE command pushes data from a Pig relation to Cassandra via the CassandraStorage handler:

\[
\text{STORE} \ <\text{relation_name}> \ \text{INTO} \ 'cassandra://<\text{keyspace}>/<\text{column_family}>' \\
\text{USING} \ \text{CassandraStorage}();
\]

Running the Pig demo

Pig operates on data stored in the Hadoop distributed file system (or CassandraFS in DSE). Your DSE installation contains sample data for running the Pig demo examples. The sample data file contains tuples of two fields each (name and score). Using Pig, the examples demonstrate creating a Pig relation and performing a simple MapReduce job that calculates the total score for each user. Result output can then be stored back into CassandraFS or into a Cassandra table.

Loading Pig sample data into CassandraFS

The Pig sample data file is located in /usr/share/dse-demos/pig/files/example.txt for packaged installations or <install_location>/demos/pig/files/example.txt for binary installations.

1. Load the Pig sample data file into CassandraFS:
   - Packaged install: dse hadoop fs -put files/example.txt /
   - Binary installs: <install_location>/bin/dse hadoop fs -put files/example.txt /
2. Verify that the file is present:
   
   dse hadoop fs -ls /

Creating a Pig relation from a data file

In this section you create a relation called score_data, which defines a schema of two fields (or columns) - named name and score. The LOAD command loads the relation with the data from the example.txt file stored in CassandraFS. The USING PigStorage() clause is optional, since this is already the default storage handler for Pig.

1. Start Pig.
2. Load the relation with data from the example.txt file:
   
   grunt> score_data = LOAD 'cfs:///example.txt' USING PigStorage() \\
   \text{AS} (\text{name:chararray}, \text{score:long});
3. View the tuples stored in the relation:
   
   grunt> DUMP score_data;
Running a MapReduce job in Pig

In this example, you take the raw data you loaded into the score_data relation and perform a number of calculations on the data using the Pig built-in relational operators. Intermediate results are also stored in Pig relations.

1. **GROUP** the tuples in the score_data relation by the name field, and store the results in a relation called name_group:
   
   `grunt> name_group = GROUP score_data BY name PARALLEL 3;`

   The PARALLEL keyword controls how many reducers are used.

2. Use the **FOREACH** operator to calculate the total score for each user grouping in the name_group relation, and store the results in a relation called name_total:
   
   `grunt> name_total = FOREACH name_group GENERATE group, COUNT(score_data.name), LongSum(score_data.score) AS total_score;`

3. Order the results in descending order by total score and store the results in a relation called ordered_scores:
   
   `grunt> ordered_scores = ORDER name_total BY total_score DESC PARALLEL 3;`

4. You can either output the results to standard output (DUMP) or to a file in CassandraFS (STORE):
   
   `grunt> DUMP ordered_scores;`
   `grunt> STORE ordered_scores INTO 'cfs:///final_scores.txt' USING PigStorage();`

   The USING clause is optional in this case, since PigStorage() is already the default storage handler.

Creating the PigDemo keyspace in Cassandra

For Pig to access data in Cassandra, the target keyspace and table must already exist. (Pig can read and write data from/to a table in Cassandra, but it will not create the table).

To create the PigDemo keyspace and Scores table used in the following examples, run the following commands in the cassandra-cli utility.

1. Start the cassandra-cli utility:
   
   `cassandra-cli`

   or in a binary distribution:
   
   `<install_location>/resources/cassandra/bin/cassandra-cli`

2. Connect to a node in your DSE cluster on port 9160:
   
   `[default@unknown] connect <server_-_ip_address>/9160`

   Examples:
   
   `[default@unknown] connect 110.82.155.4/9160`
   `[default@unknown] connect localhost/9160`
3. Create the PigDemo keyspace.

```
[default@unknown] CREATE KEYSPACE PigDemo
    WITH placement_strategy = 'org.apache.cassandra.locator.SimpleStrategy'
    AND strategy_options = [{replication_factor:1}];
```

**Note**
The default replication for system keyspaces is 1. This replication factor is suitable for development and testing, not for a production environment. For more information, see Changing replication settings.

4. Connect to the PigDemo keyspace you just created.

```
[default@unknown] use PigDemo;
```

5. Create the Scores column family.

```
[default@unknown] create column family Scores with comparator = 'LongType';
```

6. Exit cassandra-cli:

```
[default@unknown] exit;
```

**Writing data to a Cassandra table**

In this example, the scores example data is loaded into CassandraFS (see Loading Pig sample data into CassandraFS). This data consists of tuples containing two fields (name and score). For a Cassandra table, you must store three fields: the row key (name), the column name (score), and the column value (empty).

To calculate the total score for each user in the same manner as in the Running a MapReduce job in Pig example, you have to account for the empty field.

Run these commands from the Pig shell.

1. If you have not already created the score_data relation from the example.txt file stored in CassandraFS:

   ```
   grunt> score_data = LOAD 'cfs:///example.txt' AS (name:chararray, score:long);
   ```

2. Create a relation called cassandra_tuple:

   ```
   grunt> cassandra_tuple = FOREACH score_data GENERATE name, score, '' AS value;
   ```

   This command defines a tuple of three fields for Cassandra (row key, column name, column value). The column value must be an empty string as using null is the equivalent to deleting.

3. Group by name and store the results into a relation called group_by_name:

   ```
   grunt> group_by_name = GROUP cassandra_tuple BY name PARALLEL 3;
   ```

   The PARALLEL keyword controls how many reducers are used.
4. Create an aggregated row for each user containing tuples of their scores and store the results in a relation called aggregate_scores:

```
grunt> aggregate_scores = FOREACH group_by_name GENERATE group,
  cassandra_tuple.(score, value);
grunt> DUMP aggregate_scores;
```

Notice how the data was aggregated for input into Cassandra. A tuple was constructed for each Cassandra row. In Pig notation, a tuple is enclosed in parentheses ( ). Within each row tuple is a bag of column tuples where each column tuple represents an individual score. An inner bag is enclosed in curly brackets { }, so a Pig tuple that represents a row in a table is structured as:

```
(<row_key>,{{<column_name1>,<value1>},{<column_name2>,<value2>}})
```

Note that the value is empty to create a value-less column in Cassandra:

```
(jdoe,{{36}, {128}})
```

5. Now that the data is in a format that can map to the Cassandra table, store the Pig results into Cassandra using the CassandraStorage handler:

```
grunt> STORE aggregate_scores INTO 'cassandra://PigDemo/Scores' USING CassandraStorage();
```

The INTO clause specifies where to store the data in Cassandra. The format is `cassandra://<keyspace>/<column_family>`.

**Reading data from a Cassandra table**

You must have previously completed the examples in *Writing data to a Cassandra table* to run this example. In this example, you calculate the total scores for each user and group the raw score data into rows by user:

1. Create a Pig relation called cassandra_data by loading rows from the Cassandra table:

```
grunt> cassandra_data = LOAD 'cassandra://PigDemo/Scores' USING CassandraStorage()
  AS (name, columns: bag {T: tuple(score, value)});
```

2. Use the FOREACH operator to calculate the total score for each user, and store the results in a relation called total_scores:

```
grunt> total_scores = FOREACH cassandra_data GENERATE name, COUNT(columns.score),
  SUM(columns.score) as total PARALLEL 3;
```

3. Order the results in descending order by total score, and store the results in a relation called ordered_scores:

```
grunt> ordered_scores = ORDER total_scores BY total DESC PARALLEL 3;
grunt> DUMP ordered_scores;
```

**About Sqoop**

Sqoop is an Apache Software Foundation tool for transferring data between an RDBMS data source and Hadoop or between other data sources, such as NoSQL.

DataStax Enterprise support for Sqoop empowers you to import data from an external data source to Hadoop, Hive, or Cassandra column families. A DSE node runs the Hadoop/Analytics workload, and the Hadoop job imports data from a data source using Sqoop.

**Running the Sqoop demo**
To get started using Sqoop, first run the Sqoop demo to import data from a MySQL table to text files in the Cassandra File System (CFS).

**Importing data**

You can import data from any JDBC-compliant data source. For example:

- DB2
- MySQL
- Oracle
- SQL Server
- Sybase

You need a JDBC driver for the RDBMS or other type of data source.

**Migrating data to a Cassandra table**

After importing data into text files in Cassandra, take a look at how to expand the basic dse sqoop import command used by the demo to migrate data to a Cassandra column family.

Finally, glance at the extent of the Sqoop commands listed in the online help and the Cassandra additions.

**Running the Sqoop demo**

The Sqoop demo migrates the data from a MySQL table to text files in CFS. The Sqoop data migration demo uses the MySQL database and data from the North American Numbering Plan. This data consists of the area-code (NPA) and telephone number (Nxx) for the USA and Canada.

**Demo requirements**

To run the demo, you need:

- An installation of JDK 1.6.x. The JRE alone will not work.
- An installation of MySQL
- Sufficient MySQL database privileges to create database objects
- A JDBC driver for MySQL in the sqoop/lib directory
- The connection string that is appropriate for the JDBC driver
- One or more DSE nodes running the Analytics workload to run the Hadoop job that actually imports data from the external data source
- A PATH environment variable that includes the bin directory of the DSE installation

**Step-by-step procedure**

To run the Sqoop demo on a single node on a Mac, for example, follow these steps.

1. Check the JDK 1.6.x is installed.
2. Install MySQL and download the JDBC driver for MySQL from the MySQL site. This example uses mysql-connector-java-5.0.8-bin.jar.
3. Copy the JDBC driver for MySQL to the sqoop/lib directory.
   - RHEL or Debian installations
     /usr/share/dse-demos/sqoop/lib
   - Tar distribution, such as Mac
     <install_location>/demos/sqoop/lib

4. Put the connector in a directory included in the Sqoop classpath, such as the resources/sqoop subdirectory of your DataStax Enterprise (DSE) installation.

5. On the command line, start the MySQL daemon. For example:
   
   ```
   sudo ./mysqld_safe --user=mysql
   ```

6. Start MySQL and create the demo database:
   
   ```
   sudo ./mysql
   mysql> CREATE DATABASE npa_nxx_demo;
   ```

7. Then connect to the database and create the table:
   
   ```
   mysql> USE npa_nxx_demo;
   mysql> CREATE TABLE npa_nxx (npa_nxx_key varchar(16) NOT NULL,
                                npa varchar(3) DEFAULT NULL,
                                nxx varchar(3) DEFAULT NULL,
                                lat varchar(8) DEFAULT NULL,
                                lon varchar(8) DEFAULT NULL,
                                linetype varchar(1) DEFAULT NULL,
                                state varchar(2) DEFAULT NULL,
                                city varchar(36) DEFAULT NULL,
                                PRIMARY KEY (npa_nxx_key)
                  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
   ```

8. Locate the demos/sqoop directory.
   The location of the demo directory depends on your platform:
   - RHEL or Debian installations
     /usr/share/dse-demos/sqoop
   - Tar distribution, such as Mac
     <install_location>/demos/sqoop

9. Populate the table by loading the CSV file in the demos/sqoop directory.
   
   ```
   mysql> LOAD DATA LOCAL INFILE 'npa_nxx.csv'
       INTO TABLE npa_nxx_demo.npa_nxx
       FIELDS TERMINATED BY ','
       ENCLOSED BY '"'
       LINES TERMINATED BY '\n';
   ```

MySQL returns the following message:

Query OK, 105291 rows affected (1.01 sec) Records: 105291 Deleted: 0 Skipped: 0 Warnings: 0
10. Start DSE as an analytics node. The method you use depends on your platform:
   - RHEL or Debian installations
     Edit /etc/default/dse, set HADOOP_ENABLED=1, and start the dse service.
   - Tar distribution, such as Mac
     Make the bin directory in the DSE installation directory, the current directory and run the dse cassandra command using the -t option to start dse. The -t option starts Hadoop and marks the node for Analytics.

11. Use the dse command in the bin directory to migrate the data from the MySQL table to text files in the CFS directory, npa_nxx.

   Use the database username and password or -P instead of --password to be prompted for the database password. If the database account is not password-protected, just omit the password option.

   ```
   sudo ./dse sqoop import --connect jdbc:mysql://127.0.0.1/npa_nxx_demo \
   --username root \
   --password <password> \
   --table npa_nxx \
   --target-dir /npa_nxx
   ```

   DSE returns this message: INFO mapreduce.ImportJobBase: Retrieved 105291 records.

   To run this demo and import data to nodes in a cluster, the database permissions must be granted to the nodes. For example, use the GRANT ALL command to grant MySQL access to the hosts.

Cassandra Log4j appender solutions

DataStax Enterprise allows you to stream your web and application log information into a database cluster via Apache log4j.

About the log4j Utility

Apache log4j is a Java-based logging framework that provides runtime application feedback. It provides the ability to control the granularity of log statements using an external configuration file (log4j.properties).

With the Cassandra Appender, you can store the log4j messages in a column family where they're available for in-depth analysis using the Hadoop and Solr capabilities provided by DataStax Enterprise. For information about Cassandra logging, see Logging Configuration. Additionally, DataStax provides a Log4j Search Demo.

The log4j utility has three main components: loggers, appenders, and layouts. Loggers are logical log file names. They are the names known to the Java application. Each logger is independently configurable for the level of logging. Outputs are controlled by Appenders. Numerous Appenders are available and multiple Appenders can be attached to any Logger. This makes it possible to log the same information to multiple outputs. Appenders use Layouts to format log entries. In the example below, messages show the level, the thread name, the message timestamp, the source code file, the line number, and the log message.

Log levels

The available levels are:

- All - turn on all logging
- OFF - no logging
- FATAL - severe errors causing premature termination
- ERROR - other runtime errors or unexpected conditions
Cassandra Log4j appender solutions

- **WARN** - use of deprecated APIs, poor use of API, near errors, and other undesirable or unexpected runtime situations
- **DEBUG** - detailed information on the flow through the system
- **TRACE** - more detailed than DEBUG
- **INFO** - highlight the progress of the application at a coarse-grained level

Datastax does not recommend using TRACE or DEBUG in production due to verbosity and performance.

**Log messages**

As mentioned above, the messages that appear in the log are controlled via the `conf/log4j.properties` file. Using this properties file, you can control the granularity to the Java package and class levels. For example, DEBUG messages from a particular class can be included in the log while messages from others remain at a higher level. This is helpful to reduce clutter and to identify messages. The log is most commonly a file and/or stdout. The format, behavior (such as file rolling), and so on is also configurable at runtime.

Below are sample log messages from a Cassandra node startup:

```
INFO [main] 2012-02-10 09:15:33,112 DatabaseDescriptor.java (line 495)  
  Found table data in data directories. Consider using the CLI to define your schema.
INFO [main] 2012-02-10 09:15:33,135 CommitLog.java (line 166)  
  No commitlog files found; skipping replay
INFO [main] 2012-02-10 09:15:33,150 StorageService.java (line 400)  
  Cassandra version: 1.0.7
INFO [main] 2012-02-10 09:15:33,150 StorageService.java (line 401)  
  Thrift API version: 19.20.0
INFO [main] 2012-02-10 09:15:33,150 StorageService.java (line 414)  
  Loading persisted ring state
...
```

**Storing log4j messages in a column family**

The Cassandra Appender provides the capability to store log4j messages in a Cassandra column family.

**To enable the Cassandra Appender:**

1. Add `resources/log4j-appender/lib/` to your application classpath.
2. Modify the `conf/log4j.properties` file, as shown in the example below:

```properties
# Cassandra Appender
log4j.appender.CASS=com.datastax.logging.appender.CassandraAppender
log4j.appender.CASS.hosts = 127.0.0.1
log4j.appender.CASS.port = 9160
#log4j.appender.CASS.appName = "myApp"
#log4j.appender.CASS.keyspaceName = "Logging"
#log4j.appender.CASS.columnFamily = "log_entries"
#log4j.appender.CASS.placementStrategy =
   "org.apache.cassandra.locator.NetworkTopologyStrategy"
#log4j.appender.CASS.strategyOptions = {"DC1" : "1", "DC2" : "3" }
#log4j.appender.CASS.replicationFactor = 1
#log4j.appender.CASS.consistencyLevelWrite = ONE
#log4j.appender.CASS.maxBufferedRows = 256

log4j.logger.com.foo.bar= INFO, CASS
```

Commented lines are included for reference and to show the default values.

*`log4j.appender.CASS=com.datastax.logging.appender.CassandraAppender`* specifies the CassandraAppender class and assigns it the CASS alias. This alias is referenced in the last line.

*`log4j.appender.CASS.hosts = 127.0.0.1`* allows using a comma delimited list of Cassandra nodes (in case a node goes down).

Specify replication options in lines:

*`log4j.appender.CASS.placementStrategy = "org.apache.cassandra.locator.NetworkTopologyStrategy"
log4j.appender.CASS.strategyOptions = {"DC1" : "1", "DC2" : "3" }.*

*`log4j.logger.com.foo.bar= INFO, CASS`* specifies that all log messages of level INFO and higher, which are generated from the classes and sub-packages within the `com.foo.bar` package, are sent to the Cassandra server by the Appender.

By default, the CassandraAppender records log messages in the Column Family `log_entries` in the `Logging` keyspace. The definition of this Column Family is as follows:

```cql
CREATE COLUMNFAMILY log_entries (    KEY uuid PRIMARY KEY,    app_start_time bigint,    app_name text,    class_name text,    file_name text,    level text,    line_number text,    log_timestamp bigint,    logger_class_name text,    host_ip text,    host_name text,    message text,    method_name text,    ndc text,    thread_name text,    throwable_str_rep text )    WITH    comment=’’ AND    comparator=text AND
```

Cassandra Log4j appender solutions
Cassandra Log4j appender solutions

```java
row_cache_provider='ConcurrentLinkedHashCacheProvider' AND
key_cache_size=200000.000000 AND
row_cache_size=0.000000 AND
read_repair_chance=1.000000 AND
gc_grace_seconds=864000 AND
default_validation=text AND
min_compaction_threshold=4 AND
max_compaction_threshold=32 AND
row_cache_save_period_in_seconds=0 AND
key_cache_save_period_in_seconds=14400 AND
replication_on_write=True;
```

**Example**

Consider the following log snippet:

```
09:20:55,470  WARN SchemaTest:68 - This is warn message #163
09:20:55,470  INFO SchemaTest:71 - This is info message #489
java.io.IOException: Danger Will Robinson, Danger!
at com.datastax.logging.SchemaTest.testSavedEntries(SchemaTest.java:58)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
...```

Note that the ERROR entry above includes the stack trace associated with an Exception. The associated rows in the `log_entries` Column Family appear as follows (queried using cqlsh):

```
KEY,eea1256e-db24-4cef-800b-843b3b2fb72c | app_start_time,1328894454774 | level,WARN |
   log_timestamp,1328894455391 | logger_class_name,org.apache.log4j.Category | message,This is warn message #163 | thread_name,main |

KEY,f7283a71-32a2-43cf-888a-0c1d3328548d | app_start_time,1328894454774 | level,INFO |
   log_timestamp,1328894455064 | logger_class_name,org.apache.log4j.Category | message,This is info message #489 | thread_name,main |

KEY,37ba6b9c-9fd5-4dba-8fbc-51c1696bd235 | app_start_time,1328894454774 | level,ERROR |
   log_timestamp,1328894455392 | logger_class_name,org.apache.log4j.Category | message,Test exception. | thread_name,main | throwable_str_rep,java.io.IOException: Danger Will Robinson, Danger!
ad com.datastax.logging.SchemaTest.testSavedEntries(SchemaTest.java:58)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
...
```

Not all columns have values because the set of values in logging events depends on the manner in which the event was generated, that is, which logging method was used in the code and the configuration of the column family.

Storing logging information in Cassandra provides the capability to do in-depth analysis via the DataStax Enterprise platform using Hadoop and Solr.

**Log4j search demo**

The Log4j Search Demo shows an example of searching and filtering log4j messages generated by a standard Java application. In the demo, a Hadoop pi calculation is run with a `log4j.properties` file set to use the CassandraAppender that comes with DataStax Enterprise. As the logs are generated, they are indexed in real time by Solr and made available for searching in the demo user interface.

For information on configuring log4j, see [Cassandra Log4j appender solutions](#).
Before starting this demo, be sure that you have started DataStax Enterprise and Solr on a single node. See Starting DSE and DSE Search.

**Running the demo**

1. Open a shell window or tab and make the `log_search` directory your current directory. The location of the demo directory depends on your platform:

   **RHEL or Debian installations**
   ```bash
cd /usr/share/dse-demos/log_search
   ```

   **Tar distribution, such as Mac**
   ```bash
cd $DSE_HOME/demos/log_search
   ```

2. Open another shell window or tab and add the schema:

   ```bash
   ./1-add-schema.sh
   ```

   The script posts `solrconfig.xml` and `schema.xml` to these locations:
   ```
   ```

3. Start a Hadoop job using demo's log4j settings:

   ```bash
   ./2-run-hadoop-test.sh
   ```

4. Open the results in a web browser, where you can view and search for messages:

   ```
   http://localhost:8983/demos/log_search/
   ```

5. Use the search/filter feature to view the log messages.

**Search Solutions**
**What's new in DSE Search/Solr 4.0**

DataStax Enterprise features secure enterprise search support using the GA version of Apache Solr 4.0 and Lucene. You can run DSE Search/Solr on a secure cluster using Kerberos or SSL Options, as described in *Running the demo on a secure cluster.*

DataStax 3.0 and later include Java client and server components to enable Kerberos Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO) authentication. To use the SolrJ API against DSE Search clusters with Kerberos authentication, client applications should use the SolrJ-Auth library and the DataStax Enterprise SolrJ component as described in the `solrj-auth-README.md` file.

In addition to the security features, the most important new DSE Search features in DataStax Enterprise 3.0 are:

- Improved indexing, re-indexing, and schema reloading prevent schema disagreements.
- Support for the Solr copyField using CQL, the Solr HTTP API, and the Thrift API.
- Supported Solr 4 improvements include:
  -- Pseudo join (Solr 2272)
  -- Pivot faceting (Solr 792):
    Pseudo join and pivot faceting are only supported for single node clusters and clusters with the replication factor set to the cluster size. Pseudo join and pivot faceting don't fully support Solr/DSE distributed mode.
  -- Better performance when using the DirectSolrSpellChecker (Lucene 2507)
  -- AutomatonQuery (Lucene 1606)
  -- Pseudo fields and parameters related to the fl parameter
  -- Configuration of codecs per field
  -- Including Shard information in the Solr response (Solr 3134)
  -- A new indexing status section in the Core Admin panel tells users when/If re-indexing is in progress. DSE Search/Solr UI controls that are not supported have been removed.

**Introduction to DSE Search/Solr**

Coming from the Apache Lucene project, Solr is the most popular open source enterprise search platform in use today. Solr’s primary features include robust free-text search, hit highlighting, and rich document (PDF, Microsoft Word, and so on) handling. Solr also provides more advanced features like aggregation, grouping, and geo-spatial search. Today, Solr powers the search and navigation features of many of the world's largest Internet sites. With the inclusion of Solr 4.0, near real-time indexing can be performed.

The unique combination of Cassandra, Solr, and Hadoop in DSE bridges the gap between online transaction processing (OLTP) and online analytical processing (OLAP). DSE Search in Cassandra offers a way to aggregate and look at data in many different ways in real-time. Cassandra speed compensates for typical MapReduce performance problems. By integrating Solr into the DataStax Enterprise big data platform, DataStax extends Solr’s capabilities and overcomes the shortcomings of Open Source Solr (OSS) mentioned in *the next section.*
DSE Search is easily scalable. You *add search capacity* to your cluster in the same way as you add Hadoop or Cassandra capacity to your cluster. You can have a hybrid cluster of nodes, some running Cassandra, some running search, and some running Hadoop. If you don't need Cassandra or Hadoop, migrate to DSE strictly for Solr and create an exclusively Solr cluster. The DSE cluster configuration improves upon the master-slave configuration supported by OSS.

OSS tools and APIs are supported, simplifying migration from Solr to DSE Search for Solr users.

**Sources of information about OSS**

Covering all the features of OSS is beyond the scope of DataStax Enterprise documentation. Because DSE Search/Solr supports all Solr tools and APIs, refer to Solr documentation for information about topics, such as how to construct Solr query strings to retrieve indexed data.

- Apache Solr documentation
- Solr Tutorial on the Solr site
- Solr Tutorial on Apache Lucene site
- Solr data import handler
- Comma-Separated-Values (CSV) file importer
- JSON importer
- Solr cell project, which includes a tool for importing data from PDFs

**Benefits of using Solr in DataStax Enterprise**

DataStax Enterprise Search 3.0 and later is built on top of the released version of Solr 4.0. Solr offers real-time querying of files. Search indexes remain tightly in line with live data. There are significant benefits of running your enterprise search functions through DataStax Enterprise instead of OSS, including:

- A fully fault-tolerant, no-single-point-of-failure search architecture
- Linear performance scalability that comes from adding new search nodes online
- Automatic indexing of data ingested into Cassandra
- Automatic and transparent data replication
- Isolation of all real-time, Hadoop, and search/Solr workloads to prevent competition between workloads for either compute resources or data
- The capability to read/write to any Solr node, which overcomes the Solr write bottleneck
- Selective updates of one or more *individual fields* supported (a full re-index operation is still required)
- Search indexes that can span multiple data centers (OSS cannot)
• Limited CQL support for Solr/search queries (Solr HTTP API recommended)
• Creation of Solr indexes from existing column families created with CQL 2
DSE Search takes secondary indexes to a new level: data added to Cassandra is locally indexed in Solr and data added to Solr is locally indexed in Cassandra.

Unsupported features
DSE Search does not support:

• Cassandra super columns
• Cassandra counter columns
• Cassandra timeseries type rows
• Creation of Solr indexes from existing column families created with CQL 3. Column families created with CQL 2 are supported.
• Cassandra composite columns, Solr fields must be strings.

DSE Search/Solr versus Open Source Solr
By virtue of its integration into DataStax Enterprise, differences exist between DSE Search/Solr and Open Source Solr (OSS).

Major differences
The major differences in capabilities are:

<table>
<thead>
<tr>
<th>Capability</th>
<th>DSE</th>
<th>OS Solr</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Includes a database</td>
<td>yes</td>
<td>no</td>
<td>A user has to create an interface to add a database to OSS.</td>
</tr>
<tr>
<td>Indexes real-time data</td>
<td>yes</td>
<td>no</td>
<td>Cassandra ingests real-time data and Solr indexes the data.</td>
</tr>
<tr>
<td>Provides an intuitive way update data</td>
<td>yes</td>
<td>no</td>
<td>DataStax provides a SQL-like language and command-line shell, CQL, for loading and updating data. Data added to Cassandra shows up in Solr.</td>
</tr>
<tr>
<td>Indexes Hadoop output without ETL</td>
<td>yes</td>
<td>no</td>
<td>Cassandra ingests the data, Solr indexes the data, and you run MapReduce against that data in one cluster.</td>
</tr>
<tr>
<td>Balances loads on nodes/shards</td>
<td>yes</td>
<td>no</td>
<td>Unlike OSS and Solr Cloud loads can be rebalanced efficiently.</td>
</tr>
<tr>
<td>Spans indexes over multiple data centers</td>
<td>yes</td>
<td>no</td>
<td>A cluster can have more than one data center for different types of nodes.</td>
</tr>
<tr>
<td>Automatically re-indexes Solr data</td>
<td>yes</td>
<td>no</td>
<td>The only way to re-index data in OSS is to have the client re-ingest everything.</td>
</tr>
<tr>
<td>Stores data added through Solr in Cassandra</td>
<td>yes</td>
<td>no</td>
<td>Data updated using the Solr API shows up in Cassandra.</td>
</tr>
<tr>
<td>Makes durable updates to data</td>
<td>yes</td>
<td>no</td>
<td>Updates are durable and written to the Cassandra commit log regardless of how the update is made.</td>
</tr>
</tbody>
</table>

[1] This option is not available in OSS.
<table>
<thead>
<tr>
<th>Upgrades of Lucene preserve data</th>
<th>yes</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataStax integrates Lucene upgrades periodically and when you upgrade DSE, data is preserved. OSS users must re-ingest all their data when upgrading to Lucene.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security</th>
<th>yes</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataStax has extended SolrJ to protect internal communication and HTTP access. Solr data can be encrypted and audited.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Minor differences**

Minor differences between DSE Search and OSS include:

- You launch DSE Search by starting a DataStax Enterprise node in DSE Search mode. You start Solr using `java -jar start.jar`
- DSE Search terminology used to describe objects differs from OSS terminology. The DataStax Enterprise vs Solr concepts section lists the differences.
- *Delete by query* in DSE Search differs from OSS. Deletions begin immediately. You do not need to post a commit after posting the delete command.
- The process for *creating an index* and *reloading a schema* differs.
- DSE Search has removed the Optimize button from the Core Admin UI.
- In the DSE Search schema, if you do not configure the uniqueKey field as stored (stored="true"), DataStax Enterprise forces that flag to be true.

  This change is necessary to make distributed search work.

- Behavior differs between DSE Search and OSS when you configure a non-unique field as not stored.

  In OSS, the data is lost, whereas in DSE Search, the data is stored in Cassandra. The field does not show up in the search results of DSE Search or OSS.

- DataStax provides a real-time caching directory factory flag, *DSENRTCachingDirectoryFactory*, that you can use to specify where files are cached.
- The autoCommit element in the Solrconfig.xml is removed in DSE Search/Solr and the autoSoftCommit element is uncommented.

  In OSS the autoCommit element is present and uncommented. The autoSoftCommit is commented out.

- OSS supports relative paths set by the `<lib>` property in the solrconfig.xml, but DSE Search/Solr does not. Configuring Solr library paths describes a workaround for this issue that DataStax Enterprise will address in a future release.

Pseudo join and pivot faceting, not fully supported by DataStax Enterprise, do not belong in the differences list because OSS does not support these, or any other OSS features, in distributed mode. OSS does not distribute data in a scalable, peer-to-peer system like DataStax Enterprise does.

**DataStax Enterprise vs Solr concepts**

In a distributed environment, such as DataStax Enterprise and Cassandra, the column family data is spread over multiple nodes. In Solr, there are several names for an index of documents and configuration on a single node:

- A core
- A collection
- One shard of a collection

Each document in a core/collection is considered unique and contains a set of fields that adhere to a user-defined schema. The schema lists the field types and how they should be indexed. DSE Search maps Solr cores/collections to Cassandra column families. Each column family has a separate Solr core/collection on a particular node. Solr documents are mapped to Cassandra rows, and document fields to columns. The shard is analogous to a partition of...
the column family. The Cassandra keyspace is a prefix for the name of the Solr core/collection and has no counterpart in Solr.

This table shows the relationship between Cassandra and Solr concepts:

<table>
<thead>
<tr>
<th>Cassandra</th>
<th>Solr--single node environment</th>
<th>Solr--distributed environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column family</td>
<td>Core or collection</td>
<td>Collection</td>
</tr>
<tr>
<td>Row</td>
<td>Document</td>
<td>Document</td>
</tr>
<tr>
<td>Row key</td>
<td>Unique key</td>
<td>Unique key</td>
</tr>
<tr>
<td>Column</td>
<td>Field</td>
<td>Field</td>
</tr>
<tr>
<td>Node</td>
<td>N/A</td>
<td>Node</td>
</tr>
<tr>
<td>Partition</td>
<td>N/A</td>
<td>Shard</td>
</tr>
<tr>
<td>Keyspace</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

With Cassandra replication, a Cassandra node or Solr core contains more than one partition (shard) of column family (collection) data. Unless the replication factor equals the number of cluster nodes, the Cassandra node or Solr core contains only a portion of the data of the column family or collection.

**Configuring Solr**

A Solr schema defines the relationship between data in a column family and a Solr core. The schema identifies the columns to index in Solr and maps column names to Solr types. This document describes the Solr schema at a high level. For details about all the options and Solr schema settings, see the Solr wiki.

**Wikipedia Sample Schema Elements**

The sample schema.xml for the Wikipedia demo represents a typical schema. It specifies a tokenizer that determines the parsing of the wiki text. The set of fields specifies what Solr indexes and stores. In this example, these name, body, title, and date fields are indexed.

```xml
<schema name="wikipedia" version="1.1">
    <types>
        <fieldType name="string" class="solr.StrField"/>
        <fieldType name="text" class="solr.TextField">
            <analyzer><tokenizer class="solr.WikipediaTokenizerFactory"/></analyzer>
        </fieldType>
    </types>
    <fields>
        <field name="id" type="string" indexed="true" stored="true"/>
        <field name="name" type="text" indexed="true" stored="true"/>
        <field name="body" type="text" indexed="true" stored="true"/>
        <field name="title" type="text" indexed="true" stored="true"/>
        <field name="date" type="string" indexed="true" stored="true"/>
    </fields>
    <defaultSearchField>body</defaultSearchField>
    <uniqueKey>id</uniqueKey>
</schema>
```

The example schema.xml meets the requirement to have a unique key and no duplicate rows. The unique key maps to the row key and is necessary for DSE to route documents to cluster nodes. This unique key is like a primary key in SQL. The last element in the schema.xml example designates that the unique key is id. In a DSE Search/Solr schema, the value of the stored attribute of non-unique fields needs to be true; True causes the field to stored in Cassandra. The field does not show up in search results.

1 Requires using Zookeeper.
Changing a schema

Changing the Solr schema makes reloading the Solr core necessary. Re-indexing can be disruptive. Users can be affected by performance hits caused by re-indexing. Changing the schema is recommended only when absolutely necessary. Also, changing the schema during scheduled down time is recommended.

About column family metadata

After indexing the Wikipedia articles, Cassandra columns in the column family contain metadata corresponding to the fields listed in the demo schema. The output of the CLI command, DESCRIBE wiki, shows this metadata:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Validation Class</th>
<th>Index Name</th>
<th>Index Type</th>
<th>Index Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>org.apache.cassandra.db.marshall.UTF8Type</td>
<td>wiki_solr_body_index</td>
<td>CUSTOM</td>
<td>(class_name=com.datastax.bdp.cassandra.index.solr.SolrSecondaryIndex)</td>
</tr>
<tr>
<td>date</td>
<td>org.apache.cassandra.db.marshall.UTF8Type</td>
<td>wiki_solr_date_index</td>
<td>CUSTOM</td>
<td>(class_name=com.datastax.bdp.cassandra.index.solr.SolrSecondaryIndex)</td>
</tr>
<tr>
<td>name</td>
<td>org.apache.cassandra.db.marshall.UTF8Type</td>
<td>wiki_solr_name_index</td>
<td>CUSTOM</td>
<td>(class_name=com.datastax.bdp.cassandra.index.solr.SolrSecondaryIndex)</td>
</tr>
<tr>
<td>solr_query</td>
<td>org.apache.cassandra.db.marshall.UTF8Type</td>
<td>wiki_solr_solr_query_index</td>
<td>CUSTOM</td>
<td>(class_name=com.datastax.bdp.cassandra.index.solr.SolrSecondaryIndex)</td>
</tr>
<tr>
<td>title</td>
<td>org.apache.cassandra.db.marshall.UTF8Type</td>
<td>wiki_solr_title_index</td>
<td>CUSTOM</td>
<td>(class_name=com.datastax.bdp.cassandra.index.solr.SolrSecondaryIndex)</td>
</tr>
</tbody>
</table>

Compaction Strategy: org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy

Column metadata matches each field in the schema except the id field because id is the unique key. The column metadata example shows some of the Cassandra Validator types in the Validation Class attribute.

Solr type mapping

DataStax Enterprise 3.0 and earlier releases use legacy mapping of Solr types to Cassandra validator types. In DataStax Enterprise 3.0.1 and later, this mapping is used:

<table>
<thead>
<tr>
<th>Solr Type</th>
<th>Cassandra Validator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCDIntField</td>
<td>Int32Type</td>
<td>Binary-coded decimal (BCD) integer. BCD is a relatively inefficient encoding that offers the benefits of quick decimal calculations and quick conversion to a string.</td>
</tr>
<tr>
<td>BCDLongField</td>
<td>LongType</td>
<td>BCD long integer</td>
</tr>
<tr>
<td>BCDStrField</td>
<td>UTF8Type</td>
<td>BCD string</td>
</tr>
<tr>
<td>BinaryField</td>
<td>BytesType</td>
<td>Binary data</td>
</tr>
<tr>
<td>Solr Type</td>
<td>Cassandra Validator</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>TextField</td>
<td>UTF8Type</td>
<td></td>
</tr>
<tr>
<td>StrField</td>
<td>UTF8Type</td>
<td></td>
</tr>
<tr>
<td>LongField</td>
<td>LongType</td>
<td></td>
</tr>
<tr>
<td>IntField</td>
<td>Int32Type</td>
<td></td>
</tr>
<tr>
<td>TrieField</td>
<td>see description</td>
<td></td>
</tr>
<tr>
<td>TrieIntField</td>
<td>Int32Type</td>
<td></td>
</tr>
<tr>
<td>TrieLongField</td>
<td>LongType</td>
<td></td>
</tr>
<tr>
<td>UUIDField</td>
<td>UUIDType</td>
<td></td>
</tr>
<tr>
<td>LatLonType</td>
<td>UTF8Type</td>
<td></td>
</tr>
<tr>
<td>PointType</td>
<td>UTF8Type</td>
<td></td>
</tr>
<tr>
<td>GeoHashField</td>
<td>UTF8Type</td>
<td></td>
</tr>
</tbody>
</table>

**Legacy Mapping of Solr Types to Cassandra Validators**

In DataStax Enterprise 3.0 and earlier, Solr types map to these Cassandra validator types:
<table>
<thead>
<tr>
<th>Field Type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FloatField</td>
<td>FloatType</td>
</tr>
<tr>
<td>DoubleField</td>
<td>DoubleType</td>
</tr>
<tr>
<td>DateField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>ByteField</td>
<td>BytesType</td>
</tr>
<tr>
<td>BinaryField</td>
<td>BytesType</td>
</tr>
<tr>
<td>BoolField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>UUIDField</td>
<td>UUIDType</td>
</tr>
<tr>
<td>TrieDateField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>TrieDoubleField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>TrieField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>TrieFloatField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>TrieIntField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>TrieLongField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>All Others</td>
<td>UTF8Type</td>
</tr>
</tbody>
</table>

For efficiency in operations such as range queries, using Trie types is recommended.

**Configuring Solr type mapping**

By default, DataStax Enterprise 3.0.x enables legacy type mapping (dseTypeMappingVersion is set to 0).

To make the new Solr type mappings effective, add the following line to the Solr config:

```
<dseTypeMappingVersion>1</dseTypeMappingVersion>
```

Switching between the two versions is not supported. Attempting to load a solrconfig with a different dseTypeMappingVersion configuration and reloading the core causes an error.

**Configuring Solr library paths**

Contrary to the examples shown in the solrconfig.xml indicating that relative paths are supported, DataStax Enterprise does not support the relative path values set for the `<lib>` property. DSE Search/Solr fails to find files placed in directories defined by the `<lib>` property. The workaround is to place custom code or Solr contrib modules in these directories:

- Packaged installs: `/usr/share/dse`
- Binary installs: `<install_location>/resources/dse/lib`

**Creating a search index**

A minimal Solr installation requires these files:

- `Schema.xml`
  
  Describes the fields to index in Solr and types associated with them. These fields map to Cassandra columns. To route search requests to the appropriate nodes, the schema needs a unique key.

- `Solrconfig.xml`
  
  Holds configuration information for query handlers and Solr-specific caches.

After **writing a schema.xml** you HTTP-post the solrconfig.xml and the schema.xml to a Solr node in your DataStax Enterprise cluster. Next, you create a new Solr core (or reload an existing core) to create (or recreate) an index on a
column family for searching Cassandra data.

When users post schema or configuration files simultaneously, schema disagreements can occur. This causes Solr errors.

**Note**
Do not make schema changes on hot production systems.

**To create a Solr index for searching Cassandra data:**

1. Post the configuration file using the cURL utility:
   
   ```
   --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8'
   ```

2. Post the schema file:
   
   ```
   --data-binary @schema.xml -H 'Content-type:text/xml; charset=utf-8'
   ```

3. Create or reload a Solr core. Do not perform this step before performing steps 1 and 2.
   
   **Create a Solr core**
   
   ```
   curl "http://localhost:8983/solr/admin/cores?action=CREATE&name=<keyspace.columnfamily>"
   ```

   Creating a Solr core on one node automatically creates the core other Solr nodes, and DSE Search stores the files on all the Cassandra nodes.

   **Reload an existing Solr core**
   
   ```
   curl "http://localhost:8983/solr/admin/cores?action=RELOAD&name=<keyspace.columnfamily>"
   ```

   Reload a Solr core instead of creating a new one when you need to modify the schema.xml or solrconfig.xml. You can use options with the RELOAD command to re-index and keep or delete the Lucene index.

**Checking indexing status**

If you HTTP post the files to a pre-existing column family, DSE Search starts indexing the data. If you HTTP post the files to a non-existent column keyspace or column family, DSE Search creates the keyspace and column family, and then starts indexing the data. For example, you can change the stopwords.txt file, repost the schema, and the index updates.

To check the indexing status, open the Solr Admin and click Core Admin.
Using RELOAD command options

When you make a change to the schema, the compatibility of the existing index and the new schema is questionable. If the change to the schema made changes to a field’s type, the index and schema will certainly be incompatible. Changes to a field’s type can actually occur in subtle ways, occasionally without a change to the schema.xml file itself. For example, a change to other configuration files, such as synonyms, can change the schema. If such an incompatibility exists, a full re-index, which includes deleting all the old data, of the Solr data is required. In these cases, anything less than a full re-index renders the schema changes ineffective. Typically, a change to the Solr schema requires a full re-indexing.

Use these RELOAD command options to specify the level of re-indexing that occurs:

- distributed
  True, the default, distributes an index to nodes in the cluster. False re-indexes the Solr data on one node.

  curl -v "http://localhost:8983/solr/admin/cores?action=RELOAD&name=<keyspace.columnfamily>&distributed=true"

- reindex and deleteAll
  Re-indexes data in place or re-indexes in full. The default for both options is false.

Re-indexing in place

Setting reindex=true and deleteAll=false re-indexes data and keeps the existing lucene index. During the uploading process, user searches yield inaccurate results. To perform an in-place re-index, use this syntax:
&name=<keyspace.columnfamily>&reindex=true&deleteAll=false"

**Re-indexing in full**

Setting reindex=true and deleteAll=true deletes the Lucene index and re-indexes the dataset. User searches initially return no documents as the Solr cores reload and data is re-indexed.

Setting reindex=false and deleteAll=true does nothing.

**Checking a schema**

After creating a schema and indexing documents, you can check that the Solr index is working by using the Solr Admin UI in this location:

http://localhost:8983/solr/

If the UI appears, the index is working. The UI looks something like this:

![Solr Admin UI](image-url)

**Adding and viewing index resources**

DSE Search includes a REST API for viewing and adding resources associated with an index. You can look at the contents of the existing Solr resource by loading its URL in a web browser or using HTTP get. Retrieving and viewing resources returns the last uploaded resource, even if the resource is not the one currently in use. If you upload a new schema, and then *before* reloading, request the schema resource, Solr returns the new schema even though the core continues to use the old schema.

Use this format:

http://<host>[:<port>]/solr/resource/<keyspace>.<columnfamily>/<filename>.<ext>

Generally, you can post any resource required by Solr to this URL. For example, stopwords.txt and elevate.xml are optional, frequently-used Solr configuration files that you post using this URL.
Running the demo

You need to start a DataStax Enterprise node as a Solr node to run the demo. You can run the demo in these ways:

- With 3,000 wikipedia articles
- Using your legacy data
- On a secured cluster

You can run Solr on one or more nodes. DataStax does not support running Solr and Hadoop on the same node, although it’s possible to do so in a development environment. In production environments, run Solr and Hadoop on separate nodes.

Starting a Solr node

Follow these steps to start DSE Search/Solr on a single node.

1. **Start DSE as a Solr node.**
2. In another shell, check that your Cassandra ring is up and running. For example, on a Mac:
   
   - RHEL or Debian installations
     
     dsetool ring -h localhost
   
   - Tar distribution, such as Mac
     
     cd <install_location>/bin
     ./dsetool ring -h localhost

   A table of information appears showing the state of the node and identifying it as a Solr node.

Now, set up and run the DSE search demo.

Running the demo

After starting DSE as a Solr node, open a shell window or tab, and follow these steps to run the demo.

1. Make the wikipedia demo directory your current directory. The location of the demo directory depends on your platform:
   
   - RHEL or Debian installations
     
     cd /usr/share/dse-demos/wikipedia
   
   - Tar distribution
     
     cd <install_location>/demos/wikipedia

2. Upload the schema.

   ./1-add-schema.sh

The script posts solrconfig.xml and schema.xml to these locations:


wiki.solr in the URL creates the keyspace (wiki) and the column family (solr) in Cassandra. The script also creates the Solr index and core.
3. Index the articles contained in the wikipedia-sample.bz2 file in the demo directory:

   ./2-index.sh --wikifile wikipedia-sample.bz2

   Three thousand articles load.

4. To see a sample Wikipedia search UI, open your web browser and go to:

   http://localhost:8983/demos/wikipedia
5. Inspect the index keyspace, wiki, using the Solr Admin tool:

http://localhost:8983/solr/

Be sure to enter the trailing "/".

6. Inspect the column family, solr. In the Solr Admin tool, click wiki.solr to inspect the schema.
Running DSE Search/Solr demos using legacy data

DataStax 3.0.x adds an the dseTypeMappingVersion element to the solrconfig.xml of the DSE Search/Solr demos:

- Solr wikipedia demo
- Log search demo
- Solr stress demo

To use data from an earlier release, you need to use the default legacy type mapping. Disable the new Solr type mappings in the solrconfig.xml files of the demos by following these steps.

To run demos using data from an earlier release

1. Make the default legacy type mapping effective by commenting out the dseTypeMappingVersion element.
   ```xml
   <!-- <dseTypeMappingVersion>1</dseTypeMappingVersion> -->
   ```
   You can also use 0 instead of 1 for the version.

2. Delete the wikipedia data in Cassandra database and the Solr index.

3. Upload the solrconfig.xml and schema.xml.
   ```bash
```

4. Reload the core
   ```bash
curl "http://localhost:8983/solr/admin/cores?action=RELOAD&name=wiki.solr"
```
   Attempting to load a solrconfig with a different dseTypeMappingVersion configuration and reloading the core causes an error.

5. Index the articles contained in the wikipedia-sample.bz2 file in the demo directory:
   ```bash
./2-index.sh --wikifile wikipedia-sample.bz2
```

Running the demo on a secure cluster

You can run the wikipedia, stress, and log search demos on a secure cluster. Additional options you need to set are:

- Kerberos Options
  - -a enable Kerberos authentication
  - -h <hostname> server hostname (not required if server hostname resolution is correctly set up)
- HTTP Basic Authentication (not recommended for production)
  - -u username
  - -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

Using your own data in a search

Ways to insert your own data into DSE Search/Solr include:

- Executing CQL 2 statements on the command line or from a client. Use the syntax described in DataStax 1.0 documentation and shown in the following example.
Using any Thrift API, such as Pycassa or Hector.

**Example using CQL 2**

1. After starting DSE as a Solr node, open a shell window or tab, go to the bin directory on Linux for example, and start CQL in CQL 2 mode:

   ```
   ./cqlsh
   ```

2. Create a keyspace, a table (column family), and insert some data to be indexed by DSE Search.

   ```
   CREATE KEYSPACE mykeyspace WITH strategy_class = SimpleStrategy
   AND strategy_options:replication_factor = 1;
   USE mykeyspace;
   CREATE TABLE mysolr (
     KEY text PRIMARY KEY,
     solr_query text,
     name text,
     title text,
     body text
   );
   INSERT INTO mysolr (KEY, name, title, body) VALUES (123, 'Christopher Morley', 'Life', 'Life is a foreign language; all men mispronounce it.');
   INSERT INTO mysolr (KEY, name, title, body) VALUES (124, 'Daniel Akst', 'Life', 'In matters of self-control as we shall see again and again, speed kills. But a little friction really can save lives.');
   INSERT INTO mysolr (KEY, name, title, body) VALUES (125, 'Abraham Lincoln', 'Success', 'Always bear in mind that your own resolution to succeed is more important than any one thing.');
   INSERT INTO mysolr (KEY, name, title, body) VALUES (126, 'Albert Einstein', 'Success', '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.');
   ```

3. Make the wikipedia demo directory your current directory.

4. Copy the schema.xml to schema.save.xml.

5. Change schema.xml to contain this schema:

   ```
   <schema name="my_search_demo" version="1.1">
   <fields>
     <field name="id" type="string" indexed="true" stored="true"/>
     <field name="body" type="text" indexed="true" stored="true"/>
     <field name="name" type="text" indexed="true" stored="true"/>
     <field name="title" type="text" indexed="true" stored="true"/>
   </fields>
   </schema>
   ```

6. Post the configuration file using the cURL utility:

   ```
   --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8'
   ```

7. Post the schema file:

   ```
   --data-binary @schema.xml -H 'Content-type:text/xml; charset=utf-8'
   ```

8. Create a Solr core.

   ```
curl "http://localhost:8983/solr/admin/cores?action=CREATE&name=mykeyspace.mysolr"
   ```

If you are recreating the mykeyspace.mysolr core, use the RELOAD instead of the CREATE command.
9. Search Cassandra using the Solr HTTP API to find titles like Succ*.

   http://localhost:8983/solr/mykeyspace.mysolr/
   select?q=title%3ASucc*&wt=json&indent=on&omitHeader=on

   The response is:

   {  
      "response":{"numFound":2,"start":0,"docs":[  
         {  
            "id":"126",
            "body":"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",
            "title":"Success"},
         {  
            "id":"125",
            "body":"Always bear in mind that your own resolution to succeed is more important than any one thing.",
            "name":"Abraham Lincoln",
            "title":"Success"}
      ]
   }

**Querying search results**

DSE Search hooks into the Cassandra Command Line Interface (CLI), Cassandra Query Language (CQL) library, the cqlsh tool, existing Solr APIs, and Thrift APIs.

**Using SolrJ and other Solr clients**

Solr clients work with DSE 2.0 and later. If you have an existing Solr application, and you want to use DSE, it is straightforward. Create a schema, then import your data and query using your existing Solr tools. The Wikipedia demo is built and queried using SolrJ. The query is done using pure Ajax. No Cassandra API is used for the demo.

You can also use any Thrift API, such as Pycassa or Hector, to access DSE-Search. Pycassa supports secondary indexes. You can use secondary 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 a DSE-Search node.

**Using the Solr HTTP API**

There is no difference between using the Solr HTTP API in OSS and in DSE Search.

**Solr HTTP API example**

To find the titles in the solr column family that begin with the letters natio in a format available through the Solr HTTP API, such as JSON, use this URL:
    title%3Anation*&fl=title&wt=json&indent=on&omitHeader=on

The response, sorted by relevance, is returned in JSON format:

```
{
    "response": {"numFound": 7, "start": 0, "docs": [
        {
            "title": "Bolivia national football team 1999"},
        {
            "title": "Bolivia national football team 2000"},
        {
            "title": "Kenya national under-20 football team"},
        {
            "title": "Israel men's national inline hockey team"},
        {
            "title": "Bolivia national football team 2001"},
        {
            "title": "Bolivia national football team 2002"},
        {
            "title": "List of French born footballers who have played for other national teams"}
    ]
}
```

**Querying multiple column families**

To map multiple Cassandra column families to a single Solr core, use the Solr API. Specify multiple column families using the shards parameter. For example:

```
http://<host>:<port>/solr/<keyspace1>.<cf1>/select?q=*&shards=
    <host>:<port>/solr/<keyspace1>.<cf1>,<host>:<port>/solr/<keyspace2>.<cf2>
```

Using the Solr API, you can query multiple column families simultaneously if they have same schema.

**Delete by query**

In DSE Search, the delete by query triggers a different process from OSS that includes a commit. After you issue a delete by query, documents start getting deleted immediately and deletions continue until all documents are removed. For example:

1. Delete the wikipedia data in Cassandra database and the Solr index. On the command line:

   ```
    "<delete><query>*:*</query></delete>" -H
    "Content-type:text/xml; charset=utf-8"
   ```

   You do not have to post a commit command after posting the delete command as you do in OSS, and doing so is ineffective.
2. Check that the data has been deleted by searching for the titles in the solr column family that begin with the letters \textit{natio}. After running the wikipedia demo, you know that this query returns some results. Enter this URL in the browser:

\begin{verbatim}
&fl=title&wt=json&indent=on&omitHeader=on
\end{verbatim}

Output:
The output shows no documents found.

\begin{verbatim}
{
  "response":{
    "numFound":0,"start":0,"docs":[]
  }
}\end{verbatim}

\textbf{Using CQL}

You can use a solr\_query expression in a SELECT statement to retrieve Solr data from Cassandra. CQL Solr queries are suitable for simple, brief, and occasional testing and simple administrative tasks, but not recommended for production-level queries which are better suited for the \textit{Solr HTTP API}. Using Solr HTTP API is faster than using CQL. Using the Solr HTTP API, the read request goes directly to Cassandra. Using CQL, the read request first goes to Solr. A document ID, an unordered bit set, is returned. Next, the request goes to Cassandra.

\textbf{Synopsis}

\begin{verbatim}
SELECT <select expression>
  FROM <column family>
  [USING CONSISTENCY <level>]
  [WHERE solr\_query = '<search expression>' [LIMIT <n>]]
\end{verbatim}

<search expression> syntax is a Solr query string that conforms to the \textit{Lucene syntax} and \textit{Solr query syntax}. You enclose the Solr query string in single quotation marks. For example:

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
\textbf{Type of Query} & \textbf{Example} & \textbf{Description} \\
\hline
Field search & 'title:natio* AND Kenya' & You can use multiple fields defined in the schema: 'title:natio* AND body:Carlos Aragonés' \\
Wildcard search & 'Ken?a' & Use ? or * for single or multi-character searches. \\
Fuzzy search & 'Kenya~' & Use with caution, many hits can occur. \\
Phrase search & ''American football player'' & Searches for the phrase enclosed in double quotation marks. \\
Proximity search & ''football Bolivia''~10' & Searches for football and Bolivia within 10 words of each other. \\
Range searches & 'title:[football TO soccer]' & Supports both inclusive and exclusive bounds using square brackets and curly braces, respectively. \\
Term boosting & ''football''^4 ''soccer'' & By default, the boost factor is 1. Must be a positive number. \\
Boolean operator & '+Macedonian football' & AND, +, OR, NOT and - can be used. \\
Grouping & '(football OR soccer) AND' & Use parentheses to group clauses. \\
\hline
\end{tabular}
\end{center}
A **SELECT** expression reads one or more records from a Cassandra column family and returns a result-set of rows. Each row consists of a row key and a collection of columns corresponding to the query. Unlike the projection in a SQL `SELECT`, there is no guarantee that the results will contain all of the columns specified because Cassandra is schema-optional. An error does not occur if you request non-existent columns.

**CQL Example**

To query the Wikipedia demo search results:

1. Connect to the cqlsh in CQL 2 or 3 mode. On the Mac, for example:

   ```
   cd <install_location>/bin
   ./cqlsh -3
   ```

2. Use the wiki keyspace and include the solr_query expression in a CQL select statement to find the titles in the solr column family that begin with the letters *natio*:

   ```
   use wiki;
   SELECT title FROM solr
   WHERE solr_query='title:natio*';
   ```

   The output, sorted in lexical order, appears:

   ```
   title
   --------------------------------------------------------------------------
   Bolivia national football team 2002
   List of French born footballers who have played for other national teams
   Bolivia national football team 1999
   Bolivia national football team 2001
   Bolivia national football team 2000
   Israel men's national inline hockey team
   Kenya national under-20 football team
   ```

**About Solr shard selection**

Previously, for each queried token range, Cassandra selected the first closest node to the node issuing the query within that range. Equally distant nodes were always tried in the same order, so that resulted in one or more nodes being hotspotted and often selecting more shards than actually needed. In DataStax Enterprise 2.2 and later, an improved algorithm uses a shuffling technique to balance the load, and also attempts to minimize the number of shards queried as well as the amount of data transferred from non-local nodes.

**Viewing the status of the Solr core**

You can use the Solr API to view the status of the core. For example, to view the status of the wiki.solr core after running the wikipedia demo, use this URL:

```
```
Viewing the status of all cores

To view the status of all cores use this URL:


For example, the URL for viewing the status of all cores in json format and the output is:
&wt=json&indent=on&omitHeader=on

{
  "defaultCoreName":"default.1367873019536859000",
  "initFailures":{},
  "status":{
    "wiki.solr":{
      "indexing":false,
      "name":"wiki.solr",
      "isDefaultCore":false,
      "instanceDir":"solr/",
      "dataDir":"/var/lib/cassandra/data/solr.data/wiki.solr/",
      "config":"solrconfig.xml",
      "schema":"schema.xml",
      "startTime":"2013-05-06T20:43:41.527Z",
      "uptime":2908719,
      "index":{
        "numDocs":3579,
        "maxDoc":3579,
        "version":241,
        "segmentCount":8,
        "current":true,
        "hasDeletions":false,
        "directory":"org.apache.lucene.store.NRTCachingDirectory
          :NRTCachingDirectory
          (org.apache.lucene.store.NIOFSDirectory@
           /private/var/lib/cassandra/data/solr.data/wiki.solr/index
           lockFactory=org.apache.lucene.store.NativeFSLockFactory@43824d2c;
           maxCacheMB=48.0 maxMergeSizeMB=4.0)",
        "userData":{"commitTimeMSec":"1367869224465"},
        "lastModified":"2013-05-06T19:40:24.465Z",
        "sizeInBytes":9760545,
        "size":"9.31 MB"}}}}

**Viewing the status of field cache memory**

The Solr field cache caches values for all indexed documents, which if left unchecked, can result in out-of-memory errors. To ensure that the jvm heap can accommodate the cache, monitor the status of the field cache. To view the status of the field cache memory usage, append &memory=true to the URL used to view the status of cores. For example, to view the field cache memory usage of the core, wiki.solr, after running the wikipedia demo, use this URL:


For example, the URL for viewing the field cache memory usage in json format and the output is:
&wt=json&indent=on&omitHeader=on
&memory=true

<core status> . . .

"memory":{
  "unInvertedFields":{
    "totalSize":0,
    "totalReadableSize":"0 bytes"},
  "multiSegment":{
    "multiSegment":"StandardDirectoryReader(segments_4:241 _34(4.0.0.2):
C521 _35(4.0.0.2):C519 _33(4.0.0.2):C484 _36(4.0.0.2):C501 _37(4.0.0.2):
C388 _38(4.0.0.2):C419 _39(4.0.0.2):C370 _3a(4.0.0.2):C377)",
    "fieldCache":{
      "entriesCount":0},
    "totalSize":0},
  "segments":{
    "_34":{
      "segment":"_34",
      "docValues":{
        "norms":{
          "body":{
            "size":56,
            "readableSize":"56 bytes"},
          "date":{
            "size":56,
            "readableSize":"56 bytes"},
          "id":{
            "size":56,
            "readableSize":"56 bytes"},
          "title":{
            "size":56,
            "readableSize":"56 bytes"}},
        "fieldCache":{
          "entriesCount":0},
        "totalSize":224},
    "_35":{
      "segment":"_35",
      "docValues":{
        "norms":{
          "body":{
            "size":56,
            "readableSize":"56 bytes"},
          "date":{
            "size":56,
            "readableSize":"56 bytes"},
          "id":{
            "size":56,
            "readableSize":"56 bytes"},
          "title":{
            "size":56,
            "readableSize":"56 bytes"}}}}
  ...

134
Expiring a DSE Search column

You can update a DSE Search column to set a column expiration date using CQL. Eventually, this action causes removal of the column from the database.

To set a DSE Search column to expire, add a field named ttl_expire to the schema. Next, update the column using CQL to set the time-to-live (TTL) option. The following section shows you the step-by-step procedure.

To expire a DSE Search column

This procedure builds upon the Wikipedia demo to expire a DSE Search column.

1. Make the *wiki demo* directory your current directory. Modify the sample schema.xml file of the Wikipedia demo to add the ttl_expire field:

   ```xml
   <field name="ttl_expire" type="string" indexed="true" stored="true"/>
   ```

2. Post the schema and Solr configuration file for the Wikipedia demo by rerunning the demo script. On Linux, for example:

   ```bash
   sudo ./1-add-schema.sh
   ```

3. Index the articles contained in the wikipedia-sample.bz2 file in the demo directory. For example:

   ```bash
   sudo ./2-index.sh --wikifile wikipedia-sample.bz2
   
   Three thousand articles load.
   ```

4. Start cqlsh.

To test expiration of a DSE Search column

1. On the cqlsh command line, use the wiki keyspace, and then alter the Solr column to set gc_grace_seconds to 0.

   ```cql
   USE wiki;
   ALTER TABLE solr WITH gc_grace_seconds = 0;
   ```

   By setting *gc_grace_seconds* to 0, the column will be removed as soon as the TTL seconds expire.

2. Use the CQL UPDATE command to update, or create if the column doesn't exist, the Solr column. For example, set TTL values on two, non-existent rows.

   ```cql
   UPDATE solr USING TTL 10
   SET title='testtitle', body='solr body',
   WHERE KEY='key1';
   
   UPDATE solr USING TTL 3600
   SET title='testtitle2', body='solr body',
   WHERE KEY='key2';
   ```

3. After 10 seconds, query the database to check that the column entitled testtitle was removed from the database, but the column entitled testtitle2 has not yet been removed.

   ```cql
   SELECT * FROM solr WHERE solr_query='title:testtitle';
   
   SELECT * FROM solr WHERE solr_query='title:testtitle2';
   ```

   The first query returns no results after 10 seconds. The second query returns the key2 if an hour (3600 seconds) has not elapsed.

Managing expired columns

---

135
After Cassandra expires a column using the time-to-live (TTL) mechanism, DSE Search/Solr can still find the expired column. The column data remains in the index until one of the following conditions is met:

- Re-indexing occurs due to a DSE Search ttl rebuild timeout.
  Set the `ttl rebuild timeout properties` in the dse.yaml file.
- All columns in a row expire due to the Cassandra time-to-live (TTL) mechanism, triggering removal of the entire row/Solr document from the index.

Setting the ttl rebuild timeout properties is recommended for managing expired columns.

**Forcing re-indexing**

To force a re-indexing operation, you can periodically poll the column family and re-index Solr when there is any expired column. To poll the column family, you add an expiring time field to the index document, so you can search on that expiring field to re-index the expired documents. You can configure one scheduler per Solr core to search the expired documents periodically and re-index them. The re-index of the Solr secondary index, which is a per-row type of secondary index, actually re-indexes the whole row.

TTL re-indexing does consume resources, such as cpu, memory and read Cassandra column families. You can set the re-indexing frequency to a longer time if there’s not much TTL data in Cassandra column families. Typically, you need check the frequency of your compaction and assign the re-indexing frequency to a value less than compaction frequency but suitable for your business requirement.

**Configuring a scheduler**

To configure a scheduler, set the `ttl_index configuration parameters` in the dse.yaml file.

**Mixing workloads in a cluster**

A common question is how to use real-time (Cassandra), analytics (Hadoop), or search (Solr) nodes in the same cluster. To mix workloads in a cluster, you need to segregate the real-time, analytics, or search nodes into separate data centers.

**Note**

Within the same data center, attempting to run Solr on some nodes and real-time queries or analytics on other nodes does not work.

Do not run Solr and Hadoop on the same node in either production or development environments.

A DSE data center (DC) can be physical or virtual. In this diagram, nodes in data centers 1 and 2 (DC 1 and DC 2) run a mix of:

- Real-time queries (Cassandra and no other services)
- Analytics (Cassandra and Hadoop)

Data centers 3 and 4 (DC 3 and DC 4) are dedicated to search.
In separate data centers for different types of nodes, you can make some of your DSE nodes handle search while others handle MapReduce, or just act as ordinary Cassandra nodes. Cassandra ingests the data, Solr indexes the data, and you run MapReduce against that data, all in one cluster without having to do any manual extract, transform, and load (ETL) operations. Cassandra handles the replication and isolation of resources.

The Solr nodes run HTTP and hold the indexes for the column family data. If a Solr node goes down, the commit log replays the Cassandra inserts, which correspond to Solr inserts, and the node is restored automatically.

**Deploying multiple data centers**

To set up a mixed workload cluster, which is a cluster that has more than one data center to accommodate different types of nodes, see *Multiple data center deployment*.

**Replicating data across data centers**

You set up replication for Solr nodes exactly as you do for other nodes in a Cassandra cluster, by creating or altering a keyspace to define the replication strategy.

You can use the pre-release CQL 3 `CREATE KEYSPACE` and `ALTER KEYSPACE` statements to set up replication.

**Common operations**

Common DSE Search/Solr operations are:

- Adding a new Solr node
- Deleting Solr data
- Decommissioning and repairing a node
- Rebuilding an index
- Managing the location of Solr data
- Changing the Solr connector port

**Adding a new Solr node**

To increase the number of nodes in a Solr cluster, you can add or bootstrap a DSE node to the cluster. If you want to increase capacity of your search, you need to bootstrap the node, then optionally, rebalance the cluster. To bootstrap a Solr node, use the same method you use to bootstrap a Cassandra node. Using the default DSESimpleSnitch automatically puts all the Solr nodes in the same data center. Use OpsCenter Enterprise to rebalance the cluster.

**Deleting Solr data**

To delete a column family and its data, including the indexed data, from a Solr node drop the column family using the Cassandra Query Language (CQL) or the Command Line Interface (CLI). The following example, which assumes you ran the Wikipedia demo, lists the Solr files on the file system, drops the solr column family that the demo created, and then verifies that the files have been deleted from the file system:
1. List the Solr data files on the file system.
   - Packaged install:
     \[ls /usr/local/var/lib/dse5/data/solr.data/wiki.solr/index/\]
   - Tarball install:
     \[ls /var/lib/cassandra/data/solr.data/wiki.solr/index\]
   The output looks something like this:
   
   _33.fdt  _35_nrm.cfe  _38_Lucene40_0.tim
   _33.fdx  _35_nrm.cfs  _38_Lucene40_0.tip
   _33.fnm  _36.fdt     _38_nrm.cfe
   . . .

2. Launch cqlsh and execute the CQL command to drop the solr column family.

   use wiki;
   drop columnfamily solr;

3. Exit cqlsh and check that the files have been deleted on the file system. For example:

   \[ls /var/lib/cassandra/data/solr.data/wiki.solr/index\]
   The output is:
   
   \[ls: /var/lib/cassandra/data/solr.data/wiki.solr/index: No such file or directory\]

**Updating Solr data**

Using the CQL, the CLI, or Solr APIs, you can modify Solr and column family data. When you update a column family using CQL or CLI, the Solr document is updated. When you update a Solr document using the Solr API, the column family is updated. Re-indexing occurs automatically after an update.

Writes are durable. A Solr API client writes data to Cassandra first, and then Cassandra updates secondary indexes. All writes to a replica node are recorded both in memory and in a commit log before they are acknowledged as a success. If a crash or server failure occurs before the memory tables are flushed to disk, the commit log is replayed on restart to recover any lost writes.

The Solr index update operation is similar to a Cassandra secondary index update. If the old column value was still in the Cassandra memtable, Cassandra removes the index entry; otherwise, the old entry remains to be purged by compaction. If a read sees a stale index entry before compaction purges it, the reader thread invalidates it. You can also trigger the expiration of search data.

**Updating individual fields using the Solr API**

You can use the Solr HTTP REST API to insert into, modify, or delete data from a Solr node. When you update only a single field, the document is re-indexed in full. After writing the field modifications to the Solr document, use a URL in the following format to update the document:

\[curl http://<host>:<port>/solr/<keyspace>.<column family>/update?replacefields=false\]

The Solr convention is to use curl for issuing update commands instead of using a browser.

When you use CQL or CLI to update a field, DSE Search implicitly sets replacefields to false and updates individual fields in the Solr document. The re-indexing of data occurs automatically.

**Re-indexing using the Core Admin UI**
You can re-index manually using the UI or command-line tools. In the Core Admin screen of the Solr Admin UI, the Reload, Reindex and Full Reindex buttons perform functions that correspond to RELOAD command options.

**Warning about using the optimize command**

Do not use the optimize command. This warning appears in the system log when you use the optimize:

```
WARN [http-8983-2] 2013-03-26 14:33:04,450 CassandraDirectUpdateHandler2.java (line 697)
Calling commit with optimize is not recommended.
```

The Lucene merge policy is very efficient. Using the optimize command is no longer necessary and using the optimize command in a URL can cause nodes to fail.

**Decommissioning and repairing a node**

You can decommission and repair a Solr node in the same manner as you would a Cassandra node.

**Rebuilding an index**

To rebuild the index, *reload the Solr core.*

**Managing the location of Solr data**

Solr has its own set of data files. Like Cassandra data files, you can control where the Solr data files are saved on the server. By default, the data is saved in `<Cassandra data directory>/solr.data`. You can change the location from the `<Cassandra data directory>` to another directory, from the command line. For example:

```
cassandra -s -Ddse.solr.data.dir=/opt
```

In this example, the data in solr.data is saved in the `/opt` directory.

**Accessing the validation Log**

DSE Search stores validation errors that arise from non-indexable data sent from non-Solr nodes in this log:

```
/var/log/cassandra/solrvalidation.log
```

For example, if a Cassandra node that is not running Solr puts a string in a date field, an exception is logged for that column when the data is replicated to the Solr node.

**Changing the Solr connector port**

To change the Solr port from the default, 8983, change the http.port setting in the catalina.properties file installed with DSE in `<dse-version>/resources/tomcat/conf`.

**Tuning DSE Search performance**

In the event of a performance degradation, high memory consumption, or other problem with DataStax Enterprise Search nodes, try:

- Using column family compression
- Configuring the SearchHandler
- Configuring the solrconfig.xml update handler flag
- Managing caching
- Tuning the solrconfig.xml to specify cache locations
- Adding replicas to increase read performance
Changing the replication factor
Managing the data consistency level on replicas
Configuring the available indexing threads

**Using column family compression**

Search nodes typically engage in read-dominated tasks, so maximizing storage capacity of nodes, reducing the volume of data on disk, and limiting disk I/O can improve performance. In Cassandra 1.0 and later, you can configure data compression on a per-column family basis to optimize performance of read-dominated tasks.

Configuration affects the compression algorithm for compressing SSTable files. For read-heavy workloads, such as those carried by Enterprise Search, Snappy compression is recommended. Compression using the Snappy compressor is enabled by default when you create a column family in Cassandra 1.1 and later. You can change compression options using CQL. Developers can also implement custom compression classes using the org.apache.cassandra.io.compress.ICompressor interface. You can configure the compression chunk size for read/write access patterns and the average size of rows in the column family.

**Configuring the Search Handler**

The wikipedia demo solrconfig.xml configures the SearchHandler as follows:

```xml
<requestHandler name="search" class="solr.SearchHandler" default="true">

DataStax recommends using this configuration for the SearchHandler.

**Configuring the update handler and autoSoftCommit**

You need to configure the solrconfig.xml to use near real-time capabilities in Solr by setting the default high-performance update handler flag.

For example, the Solr configuration file for the Wikipedia demo sets this flag as follows and uncomment the autoSoftCommit element:

```xml
<!-- The default high-performance update handler -->
<updateHandler class="solr.DirectUpdateHandler2">

The autoCommit element has been removed to prevent hard commits that hit the disk and flush the cache. The soft commit forces uncommitted documents into internal memory. When data is committed, is it immediately available after the commit.

The autoSoftCommit element uses the maxTime update handler attribute. The update handler attributes enable near real-time performance and trigger a soft commit of data automatically, so checking synchronization of data to disk is not necessary. This table describes both update handler options.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxDocs</td>
<td>No default</td>
<td>Maximum number of documents to add since the last soft commit before automatically triggering a new soft commit.</td>
</tr>
<tr>
<td>maxTime</td>
<td>1000</td>
<td>Maximum expired time in milliseconds between the addition of a document and a new, automatically triggered soft commit.</td>
</tr>
</tbody>
</table>
For more information about the update handler and modifying SolrConfig.xml, see the Solr documentation.

**Changing the stack size and memtable space**

Some Solr users have reported that increasing the stack size improves performance under Tomcat. To increase the stack size, uncomment and modify the default `-Xss128k` setting in the cassandra-env.sh file. Also, decreasing the memtable space to make room for Solr caches might improve performance. Modify the memtable space using the `memtable_total_space_in_mb` property in the cassandra.yaml file.

**Managing caching**

DataStax Enterprise 3.0 defaults to using NRTCachingDirectoryFactory, which is recommended for real-time performance. These non-settable defaults specify where files are cached and files are managed:

- `maxMergeSizeMB = 4.0 MB`
- `maxCachedMB = 48.0 MB`

You can configure the solrconfig.xml to specify where files are cached, in RAM or on the file system, by setting the DSE near real-time caching directory factory flag. By changing directory factory attributes, you can manage where files are cached.

To manage caching operations:

1. Open `solrconfig.xml` for editing.
2. Add a `directoryFactory` element to `solrconfig.xml` of type `DSENRTCachingDirectoryFactory`. For example:

   ```xml
   <directoryFactory name="DirectoryFactory"
   class="com.datastax.bdp.cassandra.index.solr.DSENRTCachingDirectoryFactory">
   <double name="maxmergesizemb">5.0</double>
   <double name="maxcachedmb">32.0</double>
   </directoryFactory>
   ```

3. Set the `DirectoryFactory` attributes:
   - `maxmergesizemb`
     The threshold (MB) for writing a merge segment to a RAMDirectory or to the file system. If the estimated size of merging a segment is less than `maxmergesizemb`, the merge segment is written to the RAMDirectory; otherwise, it is written to the file system.
   - `maxcachedmb`
     The maximum value (MB) of the RAMDirectory.

**Increasing read performance by adding replicas**

You can increase DSE Search read performance by configuring replicas just as you do in Cassandra. You define a replica placement strategy and the number of replicas you want. For example, you can add replicas using the `NetworkTopologyStrategy` replica placement strategy. To configure this strategy, you can use CQL. For example, if you are using a `PropertyFileSnitch`, perform these steps:

1. Check the data center names of your nodes using the nodetool command.

   ```
   ./nodetool -h localhost ring
   ```

**Note**

The data center names, DC1 and DC2 in this example, must match the data center name configured for your snitch.
2. Start CQL on the DSE command line and create a keyspace that specifies the number of replicas you want.

```
CREATE KEYSPACE test
    WITH strategy_class = 'NetworkTopologyStrategy'
    AND strategy_options:DC1 = 1
    AND strategy_options:DC2 = 3;
```

The strategy options set the number of replicas in data centers, one replica in data center 1 and three in data center 2. For more information about adding replicas, see Choosing Keyspace Replication Options.

---

### Changing the replication factor for a Solr keyspace

When you post the solrconfig.xml and schema.xml and create or reload a Solr core, DSE Search creates a keyspace and column family in Cassandra. The default replication factor for this keyspace is 1. If you need more than one replica of the keyspace in your cluster, you need to update the replication factor of the keyspace.

The following procedure builds on the [wikipedia demo example](https://www.wikipedia.org). Assume the solrconfig.xml and schema.xml files have already been posted using wiki.solr in the URL, which creates a keyspace named wiki that has a default replication factor of 1. You want three replicas of the keyspace in the cluster, so you need to update the Solr keyspace replication factor.

**To change the Solr keyspace replication factor**

1. Check the name of the data center of the Solr/Search nodes.
   ````
   ./nodetool -h localhost ring
   ```
   
   The output tells you that the name of the data center for your node is, for example, datacenter1.

2. Use the pre-release version of CQL 3 (included with DataStax Enterprise 3.0) or Cassandra CLI to change the replication factor of the keyspace. Set a replication factor of 3 using CQL 3, for example:

```
ALTER KEYSPACE wiki
    WITH strategy_class = NetworkTopologyStrategy
    AND strategy_options:Solr = 3
```

If you have data in a keyspace and then change the replication factor, run `nodetool repair` to avoid having missing data problems or data unavailable exceptions.

---

### Managing the consistency level

Consistency refers to how up-to-date and synchronized a row of data is on all of its replicas. Like Cassandra, DSE-Search extends Solr by adding an HTTP parameter, `cl`, that you can send with Solr data to tune consistency. The format of the URL is:

```
http://<host>:<port>/solr/<keyspace>.<column family>/update?cl=ONE
```

The `cl` parameter specifies the consistency level of the write in Cassandra on the client side. The default consistency level is QUORUM, but you can change the default globally, on the server side using Cassandra's drivers and client libraries.

**Setting the consistency level using SolrJ**

SolrJ does not allow setting the consistency level parameter using a Solr update request. To set the consistency level parameter:

```
HttpSolrServer httpSolrServer = new HttpSolrServer(url);
httpSolrServer.getInvariantParams().add("cl", "ALL");
```

For more information, see the [Data Consistency in DSE Search blog](https://www.datastax.com/blog/data-consistency-in-dse-search).
**Configuring the available indexing threads**

DSE Search provides a new multi-threaded indexing implementation to improve performance on multi-core machines. All index updates are internally dispatched to a per-core indexing thread pool and executed asynchronously: this allows for greater concurrency and parallelism, but as a consequence, index requests will return a response before the indexing operation is actually executed. The number of available indexing threads per core is by default equal to number of available cores times 2; it can be configured by editing the max_solr_concurrency_per_core parameter in the dse.yaml configuration file; if set to 1, DSE Search will go back to the synchronous indexing behavior of the previous release.

Also, DSE Search provides advanced, JMX-based, configurability and visibility through the IndexPool-ks.cf (where ks.cf is the name of a DSE Search Solr core) MBean under the com.datastax.bdp namespace.

**Frequently asked questions**

1. **Can you run Solr and Hadoop on the same node?**
   Only in development environments. In production environments, running Solr and Hadoop on the same node will cause a failure.

2. **How do you add a file to the Solr index?**
   HTTP post the file using this URL format:
   
   `http://<host>:<port>/solr/resource/<keyspace>.<columnfamily>/<filename>.<ext>`

3. **Why does the Solr schema need to have a unique key?**
   To route cluster documents.

4. **Does DSE Search support composite columns?**
   No.

5. **How can you query data in Solr?**
   Cassandra Command Line Interface (CLI), Cassandra Query Language (CQL) library, the cqlsh tool, existing Solr APIs, and Thrift APIs.

6. **As an existing Solr user, how much trouble is it to transition to DSE Search?**
   Making the change is straight-forward. If you have an existing Solr application, and you want to use DSE 2.0 or later, create a schema, then import your data and query using your existing Solr tools.

7. **Is it possible to search data using fuzzy matching?**
   Yes, you can include a `solr_query expression` to CQL that offers extensive query options, such as fuzzy matching.

8. **When you search for non-existent Solr data, why doesn’t an error occur?**
   Unlike the projection in a SQL SELECT, there is no guarantee that the results will contain all of the columns specified because Cassandra is schema-optional. An error does not occur if you request non-existent columns. In a production environment, you must search using the LOCAL_QUORUM consistency. The default consistency level in DataStax Enterprise 3.0.1 and later is QUORUM.

9. **Can you run Hadoop on some nodes and Solr on others in a virtual data center?**
   All nodes in that dedicated virtual data center must be running Solr. Attempting to run Solr on some nodes and Cassandra or Hadoop on others within the same virtual data center does not work.

10. **How do you increase Solr capacity in a cluster?**
    Bootstrap one or more additional nodes, using the `bootstrap` method.

11. **How do you modify or remove data from a Solr node?**
    Use CQL, CLI, or Solr APIs as described in *Inserting into, Modifying, and Deleting Data from a Solr Node*.

12. **How do you rebuild an index?**
**Migrating from other databases**

DataStax offers two solutions for migrating from other databases:

- The **COPY command**, which mirrors what the PostgreSQL RDBMS uses for file/export import.
- **Apache Sqoop**, which is a tool designed to transfer data between an RDBMS and Hadoop. DataStax Enterprise modified Sqoop so you can not only transfer data from an RDBMS to a Hadoop node in a DataStax Enterprise cluster, but also move data directly into Cassandra as well.

**About the COPY command**

You can use COPY in Cassandra’s CQL shell to load flat file data into Cassandra (nearly all RDBMS’s have unload utilities that allow table data to be written to OS files) as well as data to be written out to OS files.

**ETL Tools**

If you need more sophistication applied to a data movement situation (more than just extract-load), then you can use any number of extract-transform-load (ETL) solutions that now support Cassandra. These tools provide excellent transformation routines that allow you to manipulate source data in literally any way you need and then load it into a Cassandra target. They also supply many other features such as visual, point-and-click interfaces, scheduling engines, and more.

Many ETL vendors who support Cassandra supply community editions of their products that are free and able to solve many different use cases. Enterprise editions are also available that supply many other compelling features that serious enterprise data users need.

You can freely download and try ETL tools from Jaspersoft, Pentaho, and Talend that all work with DataStax Enterprise and community Cassandra.
Reference

dse commands and dsetool

The table of dse commands describes each command, which include dse versions of Hadoop File System Shell commands. You also use dse commands to start Hive, Mahout, and Pig.

A dsetool utility for CassandraFS- and Hadoop-related tasks is also available for checking the CassandraFS and listing subranges in addition to managing the job tracker, discussed earlier.

Using dse commands

Use the dse commands from the bin directory of the DataStax Enterprise Linux installation or from the command line of a packaged or AMI distribution.

Synopsis

```
dse [-v] | cassandra [options] | hadoop [options] | hive [options]
   | mahout [options] | pig [options] | sqoop [options]
```

This table describes the key dse commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Option</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dse</td>
<td>-v</td>
<td>Send the DSE version number to standard output.</td>
<td>none</td>
</tr>
<tr>
<td>dse</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cassandra</td>
<td></td>
<td>Start up a real-time Cassandra node in the</td>
<td>link to example</td>
</tr>
<tr>
<td>cassandra</td>
<td>-s</td>
<td>background.</td>
<td></td>
</tr>
<tr>
<td>cassandra</td>
<td>-t</td>
<td>Start up an analytics node in the background.</td>
<td>link to example</td>
</tr>
<tr>
<td>cassandra</td>
<td>-t -j</td>
<td>Start up an analytics node as the job tracker.</td>
<td>link to example</td>
</tr>
<tr>
<td>cassandra</td>
<td>-f</td>
<td>Start up a real-time Cassandra node in the</td>
<td>none</td>
</tr>
<tr>
<td>cassandra</td>
<td>-f -t</td>
<td>foreground.</td>
<td>none</td>
</tr>
<tr>
<td>cassandra</td>
<td>-f -s</td>
<td>Start up a DSE Search/Solr node in the</td>
<td>none</td>
</tr>
<tr>
<td>cassandra-stop</td>
<td>-p &lt;pid&gt;</td>
<td>Stop the DataStax Enterprise process number</td>
<td>link to example</td>
</tr>
<tr>
<td>hadoop</td>
<td>version</td>
<td>Sends the version of the Hadoop component to</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>standard output.</td>
<td></td>
</tr>
<tr>
<td>hadoop</td>
<td>fs</td>
<td>Invoke the Hadoop FileSystem shell.</td>
<td>link to example</td>
</tr>
<tr>
<td></td>
<td>-options&gt;</td>
<td>Send Apache Hadoop fs command descriptions to</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>standard output.</td>
<td></td>
</tr>
<tr>
<td>hive</td>
<td></td>
<td>Start the Hive client.</td>
<td>link to example</td>
</tr>
</tbody>
</table>

145
dse hive --service <name>  
Start Hive by connecting through the JDBC driver.  
link to example

dse mahout  
Describe Mahout commands.  
link to example

dse mahout <mahout command> <options>  
Run the Mahout command.  
link to example

dse mahout hadoop <hadoop command> <options>  
Add Mahout classes to classpath and execute the hadoop command.  
link to example

dse pig  
Start Pig.  
link to example

dse sqoop -help  
Send Apache Sqoop command line help to standard output.  
link to example

Hadoop, hive, mahout, and pig commands must be issued from an analytics node. The hadoop fs options, which DSE Analytics supports with one exception (-moveToLocal), are described in the HDFS File System Shell Guide on the Apache Hadoop web site. DSE Analytics has not yet implemented the -moveToLocal option, but you can use the -copyToLocal.

**Checking the CassandraFS using dsetool**

Use the dsetool commands from the bin directory of the DataStax Enterprise Linux installation or from the command line in a packaged or AMI distribution. Use the dsetool checkcfs command to scan the CassandraFS for corrupted files. For example:

```
dsetool checkcfs cfs:/// 
```

Use the dsetool to get details about a particular file that has been corrupted. For example:

```
dsetool checkcfs /tmp/hadoop/mapred/system/jobtracker.info 
```

**Listing sub-ranges using dsetool**

The dsetool command syntax for listing subranges of data in a keyspace is:

```
dsetool [-h] [hostname] list_subranges <keyspace> <table> <rows per subrange> <start token> <end token> 
```

- `<rows per subrange>` is the approximate number of rows per subrange.
- `<start partition range>` is the start range of the node.
- `<end partition range>` is the end range of the node.

**Note**

You run nodetool repair on a single node using the output of list_subranges. The output must be partition ranges used on that node.

**Example**

```
dsetool list_subranges Keyspace1 Standard1 10000 113427455640312821154458202477256070485 0 
```

**Output**

The output lists the subranges to use as input to the nodetool repair command. For example:
Nodetool repair command options

You need to use the nodetool utility when working with sub-ranges. The start partition range (-st) and end partition range (-et) options specify the portion of the node needing repair. You get values for the start and end tokens from the output of dsetool list_subranges command. The new nodetool repair syntax for using these options is:

```
nodetool repair <keyspace> <table> -st <start_token> -et <end_token>
```

Example

```
nodetool repair Keyspace1 Standard1 -st 113427455640312821154458202477257997936583470460 -et 1324254427956245212271516664615147681247
nodetool repair Keyspace1 Standard1 -st 1324254427956245212271516664615147681247 -et 151409576048389227347257997936583470460
nodetool repair Keyspace1 Standard1 -st 151409576048389227347257997936583470460 -et 0
```

These commands begins an anti-entropy node repair from the start partition range to the end partition range.

Installing glibc on Oracle Linux 6.x and later

Oracle Enterprise Linux 6.x and later do not, by default install the 32-bit versions of the glibc libraries. You need to install the libraries in order to install DataStax Enterprise.

To install glibc on Oracle Linux:

1. Make the yum.repos.d your current directory.
   ```
cd /etc/yum.repos.d
   
```
2. Download the public-yum-ol6.repo package from the repository.
   ```
   
```
3. Check that glibc.i686 is ready for installation and install it.
   ```
yum list
   yum install glibc.i686
   
```

File locations

This section lists the locations of configuration files, packaged install directories, and tarball install directories.

Locations of configuration files

The configuration and property files, such as cassandra.yaml, and dse.yaml are located in the following directories:

- Packaged installs
  - /etc/dse/cassandra/cassandra.yaml
  - /etc/dse/dse.yaml
- Binary installs
  - <install_location>/resources/cassandra/conf/cassandra.yaml
  - <install_location>/resources/cassandra/conf/dse.yaml
Packaged install locations

The DataStax Enterprise CentOS/RHEL/Oracle Linux and Debian/Ubuntu packages are installed into the following directories:

Cassandra directories
- /var/lib/cassandra (Cassandra and CassandraFS data directories)
- /var/log/cassandra
- /var/run/cassandra
- /usr/share/dse/cassandra (Cassandra environment settings)
- /usr/share/dse/cassandra/lib
- /usr/share/dse/demos (Portfolio, Solr, Sqoop demos)
- /usr/bin
- /usr/sbin
- /etc/dse/cassandra (Cassandra configuration files)
- /etc/init.d
- /etc/security/limits.d
- /etc/default/
- /usr/share/doc/dse-libcassandra (Notices and .cqlshrc samples)

Hadoop directories
- /usr/share/dse/hadoop (Hadoop environment settings)
- /etc/dse/hadoop (Hadoop configuration files)

Hive directories
- /usr/share/dse/hive (Hive environment settings)
- /etc/dse/hive (Hive configuration files)

Mahout directories
- /usr/share/dse/mahout (Mahout properties files)
- /etc/dse/mahout (Mahout JAR files)

Pig directories
- /usr/share/dse/pig (Pig environment settings)
- /etc/dse/pig (Pig configuration files)

Solr directories
- /usr/share/dse-demos (Search - Wikipedia demo)
- /usr/share/dse/solr/web/solr/WEB-INF (SPENGO configuration)

Sqoop directories
- /usr/share/dse/sqoop (Sqoop environment settings)
- /etc/dse/sqoop
- /usr/share/dse-demos (Sqoop demo)

Log4j directories
- /etc/dse/log4j (log4j configuration file)
DataStax Enterprise configuration (dse.yaml)

- /usr/share/dse-demos (Log Search demo)

**DataStax OpsCenter directories**

- /var/lib/opscenter (SSL certificates for encrypted agent/dashboard communications)
- /var/log/opscenter (log directory)
- /var/run/opscenter (runtime files)
- /usr/share/opscenter (JAR, agent, web application, and binary files)
- /etc/opscenter (configuration files)
- /etc/init.d (service startup script)
- /etc/security/limits.d (OpsCenter user limits)

**Binary tarball install locations**

The tar installation creates the following directories in the `<install_location>` directory:

**DataStax Enterprise directories**

- bin (DataStax Enterprise start scripts)
- demos (Portfolio Manager Demo)
- interface
- javadoc
- lib
- resources/cassandra/bin (Cassandra utilities)
- resources/cassandra/conf (Cassandra configuration files)
- resources/hadoop (Hadoop installation)
- resources/hive (Hive installation)
- resources/log4j-appender (log4j logging)
- resources/mahout (mahout installation)
- resources/pig (Pig installation)
- resources/solr (Solr installation)
- resources/sqoop (Sqoop installation)
- resources/cassandra (Notices)
- resources/solr/web/solr/WEB-INF (SPENGO configuration)

**DataStax Enterprise configuration (dse.yaml)**

The dse.yaml file is the configuration file for setting the delegated snitch, Kerberos authentication, and purging of expired data from the Solr indexes. It is located in the following directories:

- **Packaged installs:** /etc/dse/cassandra
- **Binary installs:** `<install_location>/resources/dse/conf`

For cassandra.yaml configuration, see Node and cluster configuration (cassandra.yaml).

**Snitch settings**
The following property sets which snitch is delegated:

**delegated_snitch**

( Default: com.datastax.bdp.snitch.DseSimpleSnitch) The snitch to be delegated by the DseDelegateSnitch. The DseSimpleSnitch places Cassandra, Analytics, and Solr nodes into separate data centers. See DseSimpleSnitch.

For more information, see About snitches in the Cassandra documentation.

**Kerberos support**

The following properties set the QOP (Quality of Protection) and encryption options:

**kerberos_options**

- **keytab**: resources/dse/conf/dse.keytab
- **service_principal**: dse/_HOST@REALM
- **http_principal**: HTTP/_HOST@REALM
- **qop**: auth

A comma-delimited list of Quality of Protection values that clients and servers can use for each connection. The valid values are:

- **auth**: (Default) Authentication only.
- **auth-int**: Authentication plus integrity protection for all transmitted data.
- **auth-conf**: Authentication plus integrity protection and encryption of all transmitted data.

Encryption using auth-conf is separate and completely 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 one and using it for both encryption and authentication.

**client_encryption_options**

The options are:

- **enabled**: (Default: false)
- **keystore**: resources/dse/conf/.keystore
- **keystore_password**: cassandra
- **keystore_type**: JKS
- **truststore**: resources/dse/conf/.truststore
- **truststore_password**: cassandra
- **truststore_type**: JKS
- **protocol**: ssl
- **cipher_suites**: [TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA]

**Scheduler settings for Solr indexes**

These settings control the schedulers in charge of querying for and removing expired data.

**ttl_index_rebuild_options**

The options are:
Starting and stopping DataStax Enterprise

- **fix_rate_period**: (Default: 300 seconds) Schedules how often to check for expired data.
- **initial_delay**: (Default: 20 seconds) Delays the first checks to speed up startup.

Each Solr core must be checked individually. The following settings define which threadpool executes those checks concurrently:

- **thread_pool_initial_size**: (Default: 3)
- **thread_pool_max_size**: (Default: 20)
- **thread_pool_keep_alive_time**: (Default: 60 seconds) How long an idle thread is kept in the active pool.
- **thread_pool_blocking_queue_size**: (Default: 20)
- **max_solr_search_result_count_per_page**: (Default: 200) Sets the maximum Solr search result count per page. Use this setting to throttle the resources consumed by the scheduled checks.

There is one scheduler thread per Solr core. Configure the settings accordingly for best resources usage.

**Multi-threaded indexing**

DSE Search provides multi-threaded indexing implementation 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.

**max_solr_concurrency_per_core**

(Default: number of available cores times 2) Configures the number of available indexing threads per core. If set to 1, DSE Search returns to the synchronous indexing behavior.

**Starting and stopping DataStax Enterprise**

After you have installed and configured DSE on one or more nodes, you are ready to start your cluster starting with the seed nodes. In a mixed-workload DSE cluster, you must start the analytics seed node first.

Packaged installations include startup and stop scripts for running DSE as a service. Binary packages do not.

- **Starting DataStax Enterprise as a stand-alone process**
- **Starting DataStax Enterprise as a service**
- **Stopping a DataStax Enterprise node**

**Starting DataStax Enterprise as a stand-alone process**

If running a mixed-workload cluster, determine which nodes to start as analytics, Cassandra, and search nodes. Begin with the seed nodes first - analytics seed node, followed by the Cassandra seed node - then start the remaining nodes in the cluster one at a time. For additional information, see *Multiple data center deployment*.

**To start DataStax Enterprise as a stand-alone process:**

From the install directory:

- **Analytics node**: `bin/dse cassandra -t`
- **Real-time Cassandra node**: `bin/dse cassandra`
Starting and stopping DataStax Enterprise

- **Solr node:** `bin/dse cassandra -s`

**Note**
DataStax does not support using the `-t` search tracker option in combination with the `-s` option to mark the node for Hadoop analytics and search.

To check that your ring is up and running (from the install directory):

```
$ bin/nodetool ring -h localhost
```

**Starting DataStax Enterprise as a service**

Packaged installations provide startup scripts in `/etc/init.d` for starting DSE as a service.

For mixed-workload clusters, nodes that are Cassandra-only can simply start the DSE service (skip step 1).

**To start DataStax Enterprise as a service:**

1. Edit the `/etc/default/dse` file, and then add the appropriate line to this file, depending on the type of node you want:
   - `HADOOP_ENABLED=1` - Designates the node as DataStax Enterprise analytics and starts the Hadoop Job Tracker and Task Tracker services.
   - `SOLR_ENABLED=1` - Starts the node as DSE Enterprise Search/Solr node.

**Note**
DataStax does not support using the `SOLR_ENABLED` and `HADOOP_ENABLED` options to mark the same node for both search and Hadoop analytics.

2. Start the DSE service:
   ```
sudo service dse start
   ```

3. To check if your cluster is up and running:
   ```
nodetool ring -h localhost
   ```

On RHEL and CentOS, the DSE service runs as a Java process. On Debian and Ubuntu systems, the DSE service runs as a jsvc process.

**Stopping a DataStax Enterprise node**

To speed up the restart process, before stopping the dse service or the Cassandra or DataStax Enterprise process, run `nodetool drain`. This step writes the current memtables to disk. When you restart the node, Cassandra does not need to read through the commit log. 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 the service on a node:**

```
nodetool drain -h <host name>
sudo service dse stop
```

**To stop the stand-alone process on a node:**

To stop a node, find the DataStax Enterprise Java process ID (PID) and stop the process using its PID number. From the install location:
bin/nodetool drain -h <host name>
ps auwx | grep dse
bin/dse cassandra-stop -p <PID>  ## Use sudo if necessary.

Troubleshooting tips

This section lists some common problems experienced with DataStax Enterprise and solutions or workarounds.

**Mahout Jobs that Use Lucene Not Supported**

DataStax does not currently support Mahout jobs, such as built-in support for creating vectors from Lucene indexes, that use Lucene features. Attempting to run Mahout jobs that use Lucene features results in this type of error message:


**MX4J warning message during installation**

When Cassandra loads, you may notice a message that MX4J will not load and that mx4j-tools.jar is not in the classpath.

You can ignore this message. MX4J provides an HTML and HTTP interface to JMX and is not necessary to run Cassandra. DataStax recommends using OpsCenter. It has more monitoring capabilities than MX4J.

**DSE Search/Solr cannot find custom files**

Open Source Solr (OSS) supports relative paths set by the <lib> property in the solrconfig.xml, but DSE Search/Solr does not. *Configuring Solr library paths* describes a workaround for this issue.
Release notes

DataStax Enterprise 3.0.9
In DataStax Enterprise 3.0.9, the Cassandra component has been upgraded.

Components

- Apache Cassandra 1.1.12 (updated)
- Apache Hadoop 1.0.4.8
- Apache Hive 0.9.0.1
- Apache Pig 0.9.2
- Apache Solr 4.0.2.3
- Apache log4j 1.2.16
- Apache Sqoop 1.4.2.1
- Apache Mahout 0.6
- Apache Tomcat 6.0.32
- Apache Thrift 0.7.0
- Apache Commons

Issue Resolved
This release fixes the issue that caused a Java-level deadlock. (DSP-2579)

Issue
During a DataStax Enterprise 2.2.3 to 3.0.9 upgrade, you might encounter the following error in conjunction with a CQL `solr_query`: (DSP-2699)

```
ERROR [Finalizer] 2013-12-02 16:43:12,343 CoreContainer.java (line 478) CoreContainer was not shutdown prior to finalize(), indicates a bug -- POSSIBLE RESOURCE LEAK!!! instance=1618047676
```
DataStax Enterprise 3.0.8

In DataStax Enterprise 3.0.8, the following issues have been resolved:

- Fixed error where setting max_solr_concurrency_per_core parameter to 1 caused error. (DSP-2321)
- Fixed problem of secondary indexes not working reliably when row cache was enabled. (DSP-2551)
- HadoopTrackerPlugin now waits for ring to stabilize before checking for keyspace. (DSP-2555)
- Overlapping Solr shard ranges. Empty subsets returned duplicating results. (DSP-2563)
- Include and set up the old snappy for older distributions. (DSP-2567)
- Fixed deadlock between SystemClassLoader and ModuleClassLoader. (DSP-2579)

DataStax Enterprise 3.0.7

DataStax Enterprise 3.0.7 updates Cassandra and Hadoop components and includes a new dsetool command for checking for checking the Cassandra File System (CassandraFS).

Components

- Apache Cassandra 1.1.9.9
- Apache Hadoop 1.0.4.8
- Apache Hive 0.9.0.1
- Apache Pig 0.9.2
- Apache Solr 4.0.2.3
- Apache log4j 1.2.16
- Apache Sqoop 1.4.2.1
- Apache Mahout 0.6
- Apache Tomcat 6.0.32
- Apache Thrift 0.7.0
- Apache Commons

Issues resolved

- This release solves the issue preventing Hadoop from accessing libraries in Hive. (DSP-1495)
- The issue that prevented MapReduce Jobs from running longer than 24 hours on kerberized clusters is resolved. (DSP-2402)
- Fixed race condition between CFS compaction and nodetool scrub. (DSP-2425)
- Fixed the problem causing repeated Hive queries to impact CassandraFS performance. (DSP-2441)
- When starting a large cluster, a node could see all the other nodes repeatedly going down and coming back up. This problem has been resolved. (DSP-2534)
- The issue causing DEBUG messages to appear when running certain jobs, such as the portfolio demo, has been resolved. (DSP-2550)
DataStax Enterprise 3.0.6

In DataStax Enterprise 3.0.6, the following issues have been resolved:

- The fix for issue Cassandra-5529, "thrift_max_message_length_in_mb makes long-lived connections error out" has been backported. (DSP-2380)
- Removed redundant versions of Jetty from the distribution files. (DSP-2378)
- Fixed an issue that occurred under certain circumstances when a query selected multiple partitioned columns in a hive table. No results were returned. (DSP-2374)
- Fixed broken hive Views. (DSP-2369)
- Fixed the nodetool enablethrift and nodetool disablethrift commands. (DSE_2390)

DataStax Enterprise 3.0.5

DataStax Enterprise 3.0.5 fixes the issue entitled, "Filter query of +_token_lhs:* being created". Resolution of this issue improves performance on certain Solr queries, such as filter queries that use an asterisk (*). (DSP-2298)

DataStax Enterprise 3.0.4

DataStax Enterprise 3.0.4 fixes this issue: Removed excessive AssertionError warnings in SetCoverFinder. (DSP-2062)

DataStax Enterprise 3.0.3

DataStax Enterprise 3.0.3 includes enhancements and a number of resolved issues.

Enhancements

- Enable internal security in DataStax Enterprise without downtime (DSP-2096). See Enable internal security in DataStax Enterprise 3.0.3 without downtime.
- DataStax Enterprise 3.0.3 adds the ability to run MapReduce jobs on remote clusters (DSP-2116). See Configuration for running jobs on a remote cluster.

Issues resolved

- Analytics cluster running 3 or more nodes list command (dse hadoop fs -ls /) produces "No such file or directory". (DSP-787)
- The dsetool ring command now properly shows Job Tracker annotation (JT) for all data centers. (DSP-2028)
- Improve performance of CQL queries. (DSP-2054)
- Backport CASSANDRA-4049: (DSP-2077)
- Backport CASSANDRA-5361: Enable ThreadLocal allocation in the JVM. (DSP-2091)
- On error, JobTracker now properly shut downs and releases its bound port. (DSP-2135)
- Hadoop clusters now show a warning that exceptions will occur because the ring isn't fully up after about 10 seconds. (DSP-2137)
- When TaskTracker fails to start, prevents retrying on the next attempt. (DSP-2140)
- The dseTypeMapping version now includes a force option (for use by experts only). (DSP-2163)
DataStax Enterprise 3.0.2

DataStax Enterprise 3.0.2 includes updated components, enhancements, and changes. These release notes list issues and resolved issues.

Components

- Apache Cassandra 1.1.9.7
- Apache Hadoop 1.0.4.7
- Apache Hive 0.9.0.1
- Apache Pig 0.9.2
- Apache Solr 4.0.2.3
- Apache log4j 1.2.16
- Apache Sqoop 1.4.2.1
- Apache Mahout 0.6
- Apache Tomcat 6.0.32
- Apache Thrift 0.7.0
- Apache Commons

Enhancements and changes

- Improved Lucene/Solr concurrency
  Some users may experience performance gains.
- Removal of JNA jars from DataStax Enterprise tarball installation
  Warnings appear in the system log about the absence of the jars. For information about installing JNA, see Installing JNA.
- Access to the CassandraFS
  A Cassandra File System (CFS) superuser can modify files in the CFS without any restrictions. Files that a superuser adds to the Cassandra File System are password-protected.
- DSE Search/Solr support for copy fields
  If stored=false in the copyField directive:
    - Ingested data is copied by the copyField mechanism to the destination field for search, but data is not stored in Cassandra.
    - When you add a new copyField directive to the schema.xml, pre-existing and newly ingested data is re-indexed when copied as a result of the new directive.
  If stored=true in the copyField directive (backward compatibility mode):
    - Ingested data is copied by the copyField mechanism and data is stored in Cassandra.
    - When you add a new copyField directive to the schema.xml, pre-existing data is re-indexed as the result of an old copyField directive, but not when copied as the result of a new copyField directive. To be re-indexed, data must be re-ingested after you add a new copyField directive to the schema.
Support for changing the stored attribute value of copyField directives

To change the stored attribute value of a copyField directive from true to false:

1. Change the values of stored in copyField directives to false.
2. Post the solrconfig.xml and the modified schema.xml.
3. Reload the core, specifying an in-place re-index.

Previously stored copies of data are not automatically removed from Cassandra.

Changing the stored attribute value from false to true is not directly supported. The workaround is:

1. Remove the copyField directives that have stored=false.
2. Reload the solrconfig.xml and schema.xml. Use the reindex=true option.
3. Add back the copyField directives you removed in step 1 to the schema.xml and set stored=true.
4. Post the solrconfig.xml and the modified schema.xml.
5. Reload the core, specifying an in-place re-index.
6. Re-ingest the data.

Stored values are not automatically removed from Cassandra.

Issues

- Associating hostname with IPv6 loopback address in /etc/hosts breaks Hadoop. (DSP-2003)

Issues resolved

- In earlier releases, when a nodetool drain operation occurred during the DSE Search/Solr shutdown process, a call to stop tomcat caused a node to hang. The shutdown now occurs without hanging. (DSP-1994)

- Classpath problems that affected running Hadoop jobs have been fixed. The way classes and libraries are loaded has changed and dependencies set by the CLASSPATH have been minimized. (DSP-1810)

- Cassandra-5098 has been backported to DataStax Enterprise 3.0.2 to fix a problem in Pig that incorrectly decoded row keys in widerow mode has been backported. (C*-5098)

- The reference to the fair-scheduler.xml file in the mapred-site.xml that caused problems with the fair scheduling assignment of resources to Hadoop jobs has been fixed. To enable the fair scheduler you uncomment a section in the mapred-site.xml that looks something like this:

  <!-- FairScheduler is included. Uncomment to enable. -->
  <property>
    <name>mapred.jobtracker.taskScheduler</name>
    <value>org.apache.hadoop.mapred.FairScheduler</value>
  </property>
  ...
  <value>dse-3.0.2/dse/resources/hadoop/conf/fair-scheduler.xml</value>
  </property>

  You might need to change the value element shown here. Check for the presence of a file named fair-scheduler.xml in the Hadoop conf directory. If the file has a different name, change the name of the file to fair-scheduler.xml. Specify the absolute path to the file. (DSP-1971)

- After upgrading DataStax Enterprise 2.x to 3.0.2, a Solr-indexed field containing an empty date would cause a parse exception when encountered in search results. This problem has been resolved. (DSP-1944)
DataStax Enterprise 3.0.1

- In DataStax Enterprise 3.0, before compaction and after all columns in a row were expired by the time-to-live (TTL) mechanism, you could still search for and find expired columns. This issue has been resolved: Expired columns are no longer returned in search results after all columns in a row/Solr document are expired. (DSP-1884)
- DataStax Enterprise would not stop when issuing the cassandra-stop command. This problem has been resolved. (DSP-1998)
  - Fixed an issue where Solr field deletes were not being distributed to all Solr nodes. (DSP-1979)

DataStax Enterprise 3.0.1

DataStax Enterprise 3.0.1 includes updated components, enhancements, and changes. These release notes list issues and resolved issues.

Components

- Apache Cassandra 1.1.9.3
- Apache Hadoop 1.0.4.3
- Apache Hive 0.9.0.1
- Apache Pig 0.9.2
- Apache Solr 4.0.2.2
- Apache log4j 1.2.16
- Apache Sqoop 1.4.2.1
- Apache Mahout 0.6
- Apache Tomcat 6.0.32
- Apache Thrift 0.7.0
- Apache Commons

Enhancements and changes

DataStax Enterprise 3.0.1 has been enhanced or changed in the following ways:

- The default consistency level has changed from ONE to QUORUM for reads and writes to resolve a problem finding a CassandraFS block when using consistency level ONE on a Hadoop node. (DSP-1809)
- Solr type mapping to Cassandra validator types has been refactored in this release. (DSP-1876)
- The configuration files for these DSE Search/Solr demos have been modified to use new type mapping:
  - Solr wikipedia demo
  - Log search demo
  - Solr stress demo

Running DSE Search/Solr demos using legacy data describes how to use data from an earlier release.
DSE Search provides a new multi-threaded indexing implementation to improve performance on multi-core machines. All index updates are internally dispatched to a per-core indexing thread pool and executed asynchronously: this allows for greater concurrency and parallelism, but as a consequence, index requests will return a response before the indexing operation is actually executed. The number of available indexing threads per-core is by default equal to number of available cores times 2: it can be configured by editing the max_solr_concurrency_per_core parameter in the dse.yaml configuration file; if set to 1, DSE Search will go back to the synchronous indexing behavior of the earlier release. (DSP-1644)

Also, DSE Search provides advanced, JMX-based, configurability and visibility through the IndexPool-ks.cf (where ks.cf is the name of a DSE Search Solr core) MBean under the com.datastax.bdp namespace.

- On the cqlsh command line, Tab completion now reveals user names when you type a CQL security command that takes a known user name as an option. (DSP-1371)
- This release includes a plugin API for Solr updates and a plugin to the CassandraDocumentReader. The plugin API transforms data from the secondary indexing API before it is submitted to Solr. The plugin to the CassandraDocumentReader transforms the results data from Cassandra to Solr. (DSP-1493)
- The deprecated Solr document cache is now disabled. (DSP-1794)
- Solr/Cassandra range manipulation and token filtering algorithms have been rewritten to improve performance and internal maintenance. This change is backward compatible with previous releases. (DSP 1708)
- This release includes two features for performing an anti-entropy node repair on a subrange of data instead of all the data in a keyspace. (DSP-1661)
  - A new dsetool command, list_subranges, estimates subranges of data in a keyspace based on a specified number of rows.
  - New nodetool repair options, start token (-st) and end token (et), designate subranges of data for distribution within those ranges.

Using these commands, DSE Search now performs a partial re-index instead of a full re-index of Solr data after an anti-entropy repair.

- You can now track memory usage of internal Lucene and Solr data structures using OpsCenter. These metrics are among those you can track: (DSP-1617)
  - Doc values (including norms)
  - Field caches
  - Doc-set caches
  - Terms index (FST) caches
  - NRTCachingDirectory (internal RAMDirectory)
  - IndexWriter RAM usage
- Cassandra 5155 has been backported to the Cassandra component included in this release, Cassandra 1.1.9.3. With the enhancement, you can configure an Ec2Region data center name. In the same EC2 region, you can now run a real-time Cassandra data center and a DSE Search/Solr cluster. (DSE-1685)
- The Query Elevation search component now functions correctly if you upload the elevate.xml to Cassandra like you upload the solrconfig.xml. Alternatively, put elevate.xml in a directory on all the nodes. (DSP-1652)

**Issues**

- To insert data using CQL or Thrift that will be indexed by Solr, run the inserts on a Solr node. (DSP-2007)
• Use a single CQL statement or batch operation in thrift to insert data in fields that the Solr schema declares a `copyField` having a multi-value destination. Otherwise any, subsequent writes to those fields overwrite any values in the field being copied to. (DSP-1882)

For example, using pycassa:

```python
import pycassa
from pycassa.pool import ConnectionPool
from pycassa.columnfamily import ColumnFamily

pool = ConnectionPool('test')
col_fam = pycassa.ColumnFamily(pool, 'copy')
b = col_fam.batch()
b.insert('thrift_test',
    {'text1': 'textval1',
     'text2': 'textval2'})
b.insert('thrift_test', {'multi_text_coll': "solrjson:['foo', 'bar']"})
b.send()

b2 = col_fam.batch()
```

• During upgrading, you might see warnings when initially starting up an Analytics/Hadoop node. To avoid making concurrent changes to the schema, which are not fully supported in this release, nodes coordinate the configuration of the system keyspaces. When the node designated to update the schema is not fully initialized or a user runs dsetool before the schema update occurs, this type of warning occurs: (1804)

```
Found CFS filesystem in Hadoop config: cfs-archive
TrackerManager.java (line 201) JobTracker location query failed with consistency level QUORUM, retrying with level ONE
CassandraJobConf.java (line 358) Unable to retrieve JobTracker primary and reserve locations, will set local address as JT for Analytics-Analytics
TrackerManager.java (line 157) Error writing JT location
InvalidRequestException(why:Keyspace dse_system does not exist)
```

You can ignore these warnings. (DSP-1916)

• Solr can return duplicated results because Solr improperly indexes Cassandra when all of these conditions exist:
  - Primary key values contain special characters, such as #, @ and $.
  - The unique key field in the Solr schema is of field type "text" (solr.TextField).
  - The field has a tokenizer that treats such special characters as white space.

  The solution is to ensure that the Solr unique key field is of type string (solr.StrField). (DSP 882 and 839)

• When using Async or HsHa, Hadoop users may see an error that a connection failed (Failed to open server transport) and a different transport will be used (Falling back to TFramedTransport). This error is benign. To remove the error, in mapred-site.xml, set the property cassandra.client.transport.factory to org.apache.cassandra.thrift.TFramedTransportFactory". You may also need to fix the property in the dsetool, nodetool, and cassandra-cli scripts. (DSP-1844)

• The RPC Thrift server doesn't support Async or HsHa when using Kerberos. You must either change the settings in cassandra.yaml to rpc_server_type: sync, or disable Kerberos and restart the server. (DSP-1844)

**Issues resolved**

• Range query performance problems and inconsistency across nodes. (DSP-1577)

• The Query Elevation search component now works correctly. You can upload the elevate.xml to C* like solrconfig.xml or have it reside in a directory on all the nodes. (DSP-1652)

• Solr queries return inconsistent values after nodes are rebalanced. (DSP-1672)
• Solr delete by query, when the key is a UUID, creates a corrupt tombstone. (DSP-1709)
• Cassandra-5301 has been backported to the Cassandra 1.1.9.3 included in this release. This fix relaxes the consistency level for authentication queries for non-default users. The default super user, "cassandra", requires reading the dse_auth keyspace at QUORUM. Other superusers read and write at a consistency level of ONE.
• For backward compatibility, the Thrift set_keyspace behavior has been altered to allow a call to set_keyspace followed by a call to login. (DSP-1878)
• You no longer need to enable Hadoop to connect to external addresses. DSE automatically sets the listen_address:rpc_port. (DSP-1139)
• An extraneous copy of the Cassandra Command Line Interface (CLI) in <install location>/resources/cassandra/bin in a tarball installation was not correctly configured. This copy has been removed. Use the CLI utility located in <install location>/bin. (DSP-1832)

DataStax Enterprise 3.0
DataStax Enterprise 3.0 includes updated components enhancements, and changes.

Components

• Apache Cassandra 1.1.9
• Apache Hadoop 1.0.4.2
• Apache Hive 0.9.0.1
• Apache Pig 0.9.2
• Apache Solr 4.0 GA
• Apache Thrift 0.7.0
• Apache log4j 1.2.16
• Apache Sqoop 1.4.2.1
• Apache Mahout 0.6

Enhancements
DataStax Enterprise 3.0 has been enhanced in the following ways:

• Kerberos security added to all components (Cassandra, Hadoop, Solr)
• Cassandra-based password authentication backported (Cassandra only)
• Full Cassandra-level data auditing
• At rest data encryption
• Integration of the final Solr 4.0 release
• Various Solr stability improvements:
  • Concurrent solr core reload
  • Improved handling of bad schemas
  • Improved stability under concurrent access situations
  • Proper de-duplication of search results
• Other Solr improvements
**Changes and requirements**

### Changes to the Solr demo script

In this release, changes have been made to the Solr demo script. The scripts to run the wikipedia demo have been updated. For example, the `1_add_schema.sh` script has been updated to include these lines:

```bash
CREATE_URL="http://$host:8983/solr/admin/cores?action=CREATE&name=${KS}.${CF}"
curl -X POST $CREATE_URL
echo "Created index."
```

### Disk full alert

DataStax assumes that the Customer's operation team monitors cluster resources to ensure that enough disk space exists. In the event of an oversight, Cassandra marks the node to be decommissioned when the disk is approaching full. The server should stop serving when the disk is almost full, the node is removed from the ring, and Cassandra issues an alert.

### Java requirements

DataStax recommends using the latest 64-bit version of Java 6.

- **DSE and Cassandra**: JRE not supported below 1.6.0_29.
- **Kerberos**: The JRE should be later than 1.6.0_26 due to a Kerberos bug.

### Resolved issues

This release fixes the following issues:

- Fields in FieldInfos that are in the SolrInputDocument when the document is sent to Solr were neither stored in Solr nor indexed. (DSP-1352)
- On node restart, errant Solr documents in commit log prevented the node from starting. (DSP-1297)
- After storing data on one Solr node, the data replication was inconsistent. Now, the data shows up in searches from other nodes. (DSP-1121)

### Issues

This release has the following issues:

- An outdated file, SECURITY_NOTES, is included in the installation directory. This file should have been removed. For security information and procedures, use this document.
- The Cassandra log4j appender doesn't support multiple hosts. (DSP-1601)
- The sstableloader tool does not work in an environment using Kerberos. The workaround is to run sstableloader in a ring that doesn't use Kerberos. (DSP-1168)
- When you have a multiple data center cluster with Kerberos enabled, all keyspaces accessed from Hadoop must be configured with NetworkTopologyStrategy. If configured with SimpleStrategy, Hadoop jobs will hang. (DSP-1630)
- MapReduce jobs hang before completing or finishing cleanup with older versions of Hadoop (MAPREDUCE-4560, MAPREDUCE-4299). The workaround is remove the mapred.reduce.slowstart.completed.maps parameter and restart. (DSP-1154)
• The `nodetool repair -pr` command does not completely repair a keyspace unless it is in every datacenter. (CASSANDRA-5424)

• In earlier releases, the authenticator allowed various Cassandra clients, such as Hector, to set a keyspace, and then login. In this release, the `org.apache.cassandra.auth.PasswordAuthenticator` requires that the client login and then set the keyspace. (Cassandra-5423, DSP-1878)

• Open Source Solr (OSS) supports relative paths set by the `<lib>` property in the `solrconfig.xml`, but DSE Search/Solr does not. `Configuring Solr library paths` describes a workaround for this issue that DataStax Enterprise will address in a future release. (DSP-1840)

• After upgrading DataStax Enterprise 2.x to 3.x, a Solr-indexed field containing an empty date causes a parse exception when encountered in search results. (DSP-1944)