# Contents

## About DataStax Enterprise
- New features in DataStax Enterprise 3.1 ......................................................... 1
- Key features of DataStax Enterprise ................................................................. 1

## Installation
- 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 other Linux OS ................. 7
- Installing the DataStax Enterprise Tarball on SUSE Enterprise .............................. 8
- Installing on cloud providers .............................................................................. 9

## Upgrading to DataStax Enterprise 3.1
- General upgrade procedure ................................................................................. 15
- Upgrade instructions for installing the DSE tarball on any Linux distribution .......... 16
- Upgrade installation instructions for Debian-based distributions ......................... 17
- Upgrade installation instructions for RHEL-based distributions ............................ 17
- Version specific upgrade instructions .................................................................. 18
  - Security ............................................................................................................. 21
  - Security ............................................................................................................. 22
- Upgrading the DataStax AMI ............................................................................... 25

## Security
- Security management ........................................................................................... 27
- Authenticating a DataStax Enterprise cluster with Kerberos ................................. 28
- Client-to-node encryption ...................................................................................... 31
- Node-to-node encryption ....................................................................................... 32
- Preparing server certificates .................................................................................. 33
- Installing the cqlsh security packages ................................................................... 33
- Transparent data encryption .................................................................................. 35
- Configuring and using data auditing ..................................................................... 39
- Configuring and using internal authentication ....................................................... 44
- Managing object permissions using internal authorization ..................................... 46
- Configuring system_auth keyspace replication ..................................................... 47
- Configuring firewall port access .......................................................................... 48

## Deployment
- Production deployment planning .......................................................................... 50
- Configuring replication ......................................................................................... 50
- Single data center deployment ............................................................................... 53
- Multiple data center deployment ......................................................................... 56
- Expanding a DataStax AMI cluster ..................................................................... 59
### Integrated Solutions

- DSE Analytics with Hadoop .......................................................... 60
- Cassandra Log4j appender solutions ................................................. 95
- Search Solutions ............................................................................. 99
- Migrating from other databases ...................................................... 148

### Reference

- dse commands and dsetool .............................................................. 150
- Installing glibc on Oracle Linux 6.x and later ................................. 152
- File locations .................................................................................. 152
- DataStax Enterprise configuration (dse.yaml) ................................. 154
- Starting and stopping DataStax Enterprise .................................... 156
- Troubleshooting tips ...................................................................... 158

### Release notes

- DataStax Enterprise 3.1.6 release notes .......................................... 159
- DataStax Enterprise 3.1.5 release notes .......................................... 161
- DataStax Enterprise 3.1.4 release notes .......................................... 163
- DataStax Enterprise 3.1.3 release notes .......................................... 165
- DataStax Enterprise 3.1.2 release notes .......................................... 166
- DataStax Enterprise 3.1.1 release notes .......................................... 167
- DataStax Enterprise 3.1 release notes ............................................ 167
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 3.1

Key new features are:

- Support for virtual nodes (vnodes) in Cassandra real-time clusters. Not supported in clusters containing Solr and Hadoop clusters.
- Support for mixed architecture. You can run clusters with virtual node-enabled and non-virtual node data centers.
- Support for the Murmur3 partitioner.
- Includes Cassandra 1.2, which supports the released version of CQL 3.
- Hadoop and Hive support for CQL 3.
- Many DSE Search/Solr enhancements, including:
  - Solr 4.3 support
  - Per segment filters
  - Improved performance on facets
  - Multivalue field support
  - Support for docValues in the schema field definition
  - Per-segment caching for facets improves performance
  - Configurable TTL for a field or document using the Solr HTTP API
  - Configurable number of columns loaded from Cassandra for dynamic field queries.
- Support for audit logging of queries and prepared statements submitted to the DataStax Java Driver, which uses the CQL binary protocol.

Key features of DataStax Enterprise

The key features of DataStax Enterprise include:

- **Security** - *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.

- **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 modifies this architecture, 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.
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.
About DataStax Enterprise
Installation

Installing the DataStax Enterprise package on RHEL-based distributions

DataStax provides a packaged release for installing DataStax Enterprise and OpsCenter 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.

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 on the install machine.
- Install the latest version of Oracle Java SE Runtime Environment (JRE) 6 or 7.
- Java Native Access (JNA) is required for production installations. See Installing the JNA on RHEL or CentOS Systems.
- If you are installing DataStax Enterprise on a 64-bit Oracle Linux, first install 32-bit versions of glibc libraries.

Also see Recommended production settings.

Installation steps

DataStax provides Yum repositories for CentOS, Oracle Linux, and RHEL systems.

From a terminal window:

1. Check which version of Java is installed:

   # java -version

   Use the latest version of Oracle Java 6 or 7 on all nodes. If you need help installing Java, see Installing Oracle JRE on RHEL-based 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 DataStax Enterprise and OpsCenter using Yum:

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

    For example:

    yum install dse-full-3.1.4-1

    **Note**

    dse-full installs only DataStax Enterprise. dse-full opscenter installs both DataStax Enterprise and OpsCenter.

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.

**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.

**Prerequisites**

- Before you can install, you must register with DataStax to get a username and password.
- Advanced Package Tool is installed.
- Root or sudo access on the install machine.
- Install the latest version of Oracle Java SE Runtime Environment (JRE) 6 or 7.
- Java Native Access (JNA) is required for production installations. See Installing the JNA on Debian or Ubuntu Systems.
Note
If you are using Ubuntu 10.04 LTS, you need to update to JNA 3.4. See the steps describing this procedure in Installing the JNA on Debian or Ubuntu Systems.

Also see Recommended production settings.

Installation steps
DataStax provides Debian package repositories for Debian and Ubuntu systems.

From a terminal window:

1. Check which version of Java is installed:
   
   # java -version

   Use the latest version of Oracle Java 6 or 7 on all nodes. If you need help installing Java, see Installing the Oracle JRE and JNA.

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 or getting the packages, use http instead of https.

4. Install the packages:

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

   For example:

   $ sudo apt-get install dse-full=3.1.4-1 dse=3.1.4-1 dse-hive=3.1.4-1 dse-pig=3.1.4-1 ...

   Note
   dse-full installs only DataStax Enterprise. dse-full opscenter installs both DataStax Enterprise and OpsCenter.

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.

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.
Installing the DataStax Enterprise tarball on Mac OSX or other 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.

Prerequisites

Before installing DataStax Enterprise make sure you have met the following prerequisites:

- Before you can install, you must register with DataStax to get a username and password.
- Install the latest version of Oracle Java SE Runtime Environment (JRE) 6 or 7.
- Java Native Access (JNA) is required for production installations. See Installing the JRE and the JNA.

Note

If you are using Ubuntu 10.04 LTS, you need to update to JNA 3.4. See the steps describing this procedure in Installing the JNA on Debian or Ubuntu Systems.

Also see Recommended production settings.

Installation steps

From a terminal window:

1. Check which version of Java is installed:

   # java -version

   Use the latest version of Oracle Java 6 or 7 on all nodes. If you need help installing Java, see Installing the Oracle JRE and JNA.

2. Download the distributions. Installing OpsCenter is optional.

   $ curl -OL http://<username>:<password>@downloads.datastax.com/enterprise/dse.tar.gz
   $ curl -OL http://<username>:<password>@downloads.datastax.com/enterprise/opscenter.tar.gz

3. Unpack the distributions:

   $ tar -xzvf dse.tar.gz
   $ tar -xzvf opscenter.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 to 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.
- 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 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.
- Install the latest version of Oracle Java SE Runtime Environment (JRE) 6 or 7. See Installing the Oracle JRE and JNA.
- Java Native Access (JNA) is required for production installations. See Installing the JNA on SUSE Systems. Also see Recommended production settings.

**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
- Connecting to the server
- Install the JRE and JNA
- Install DataStax Enterprise
- Configure DataStax Enterprise
- Configuring OpsCenter and Agents

Note

Links to some HP documents require that you are logged into the HP Cloud Console to open.

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.

Note

For multiple availability zones, use the same key pair in each zone. If you used the HP Cloud console to create the key pair, you can retrieve the public key using the REST API. You must first create an authorization token to execute the API calls, then use the List Key Pairs command to retrieve the 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.

Note
After making any change to a security group, you must restart the nodes. You cannot change which security group is associated with an instance after the instance is created.

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 --ipproto 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. Set the Internet Control Message Protocol port:

<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>

   b. Set the ports described in the Firewall port table. For these ports set the IP Protocol to tcp.

   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.

Installing on cloud providers
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**.

---

**Running Instances**

<table>
<thead>
<tr>
<th>Status</th>
<th>Instance</th>
<th>Flavor</th>
<th>Image</th>
<th>Key Pair</th>
<th>Private IP</th>
<th>Fixed Public IP</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️ Building</td>
<td>483481</td>
<td>standard.large</td>
<td>8645 - Ubuntu Oneiric 11.10 Server 64-bit 20120311</td>
<td>DataStaxKey</td>
<td>10.7.185.209</td>
<td>15.185.171.82</td>
<td>less than a minute</td>
</tr>
</tbody>
</table>
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 (\(<\text{keyname}\).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.

   ```
   Instance: 483461
   X
   
   To access via SSH:
   
   1. Locate the private key for the key pair named **DataStaxKey**
   2. For SSH to work, ensure your key is protected:
      ```bash
      chmod 400 DataStaxKey.pem
      ```
   3. Open your SSH client (such as Terminal).
   4. Connect via your server’s IP address:
      ```
      15.185.171.82
      ```
   
   **Example**
   ```
   ssh -i DataStaxKey.pem root@15.185.171.82
   ```
   
   Replace with **ubuntu**

**Install the JRE and JNA**

Oracle Java SE Runtime Environment (JRE) 6 or 7 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 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 Install Oracle JRE on Debian or Ubuntu Systems.

3. Install the JNA as described in Instalising the JNA on Debian or Ubuntu Systems.

**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 on Debian or Ubuntu.

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
Installing on cloud providers

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 3.1

General upgrade procedure

The upgrade process for DataStax Enterprise is aimed at providing minimal downtime (ideally zero). With a few exceptions, the cluster will continue to work as though it were on the older version until all of the nodes in the cluster have been upgraded.

To perform an upgrade with zero downtime, we recommend performing the upgrade as a rolling restart. The procedure for the rolling upgrade is:

1. Make a backup of the data by taking a snapshot of the node to be upgraded.
2. Stop the node.
3. Install the new product.
4. Configure the new product.
5. Start the node.
6. Check the logs for warnings, errors and exceptions.
7. Repeat these steps for each node in the cluster.

While the cluster is in a partially upgraded state, please observe the general limitations.

Most version upgrades also have special limitations or require special steps. These details are listed in the Version specific upgrade instructions. The steps outlined in these pages are generally cumulative, so the special instructions for all versions between your current version and the version you’re upgrading to need to be followed.

The following sections examine the above steps in more detail:

Stop the node

Before stopping the node, run nodetool drain to flush the commit log of the old installation:

```bash
nodetool drain -h <hostname>
```

If you are upgrading a DSE Search/Solr node, this step is mandatory to prevent having to re-index data. This step is recommended for upgrading other nodes because it saves time when the node starts up.

Then stop the node as appropriate for your installation type.

- For packaged installs (Debian and RHEL), stop DataStax Enterprise as a service.
- For binary tarball installs, stop DataStax Enterprise as a stand-alone process.

Install the new product

Before installing the new product, we recommend you back up any configuration files you have modified. Depending on how you install the product, these files may be overwritten with default values during the installation.

After backing up your configuration, follow the appropriate installation instructions depending on your current installation type:

- Binary tarball installs
- Debian-based installs
- RHEL-based installs
Configure the new product

Using the backups you made of your configuration files, merge any modifications you have previously made into the new configuration files for the new version. Configuration options change often, so be sure to double check the Version specific upgrade instructions for additional steps and changes regarding configuration.

Start the node

Follow the steps appropriate for your installation type:

- For packaged installs (Debian and RHEL), start DataStax Enterprise as a service.
- For binary tarball installs, start DataStax Enterprise as a stand-alone process.

Check the logs for warnings, errors and exceptions

Many times warnings, errors and exceptions will be found in the logs on starting up an upgraded node. Some of these are informational and will help you execute specific upgrade related steps. Be sure to check the Version specific upgrade instructions to identify which of these is expected, and for instructions on using these warnings and errors to complete your upgrade. If you find any unexpected warnings, errors or exceptions, contact Support.

Repeat on each node in the cluster

The order of upgrading nodes matters. Please upgrade nodes in the following order:

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

General limitations while cluster is in a partially upgraded state

- Do not run nodetool repair.
- Do not use new features.
- Do not issue these types of queries during a rolling restart: DDL, TRUNCATE
- Hadoop specific limitations
  - Do not run mapreduce jobs.
- Solr specific limitations
  - Do not update schemas.
  - Do not re-index Solr unless you are following an instruction in these upgrade procedures to re-index.
  - Do not issue these types of queries during a rolling restart: DDL, TRUNCATE, and Solr queries.
- Security limitations
  - Do not change security credentials or permissions until the upgrade is complete.
  - Do not attempt to set up Kerberos authentication. First upgrade the cluster, and then set up Kerberos.

Upgrade instructions for installing the DSE tarball on any Linux distribution
This procedure shows how to install a new version of the DataStax Enterprise binary tarball to replace an existing installation.

Before performing the installation, be sure to back up your configuration files for future reference.

**Upgrading a node and migrating the data**

1. Download the DataStax Enterprise tarball using your username and password:
   
   ```bash
   $ curl -OL http://<username>:<password>@downloads.datastax.com/enterprise/dse.tar.gz
   
   Get the `<username>` and `<password>` from your DataStax registration confirmation email. If you don’t have the email, register on the DataStax web site.
   ```

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

3. Unpack the DataStax Enterprise 3.1 tarball:
   
   ```bash
   tar -xzvf <dse-3.1 tarball name>
   ```

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

**Upgrade installation instructions for Debian-based distributions**

This procedure shows how to install a new version of DataStax Enterprise using the Advanced Package Tool, apt-get.

Please be sure to backup your configuration files before starting this process as apt-get overwrites any modifications you have made to them.

1. If you were previously using a version of DataStax Community, add the DataStax repository to the `/etc/apt/sources.list` using your username and password:
   
   ```bash
   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.
   ```

2. Upgrade the node:
   
   ```bash
   sudo apt-get update
   sudo apt-get install dse-full
   ```

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

**Upgrade installation instructions for RHEL-based distributions**

This procedure shows how to install a new version of DataStax Enterprise using the Yum Package Manager.

Please be sure to backup your configuration files before starting this process. However, Yum may also back them up in place using a .rpmsave extension. For example, cassandra.yaml.rpmsave.

1. Open the Yum repository file for DataStax Enterprise in `/etc/yum.repos.d` for editing:
   
   ```bash
   sudo vi /etc/yum.repos.d/datastax.repo
   ```
2. Replace the contents of the file with the following lines using your username and password:

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

Get the `<username>` and `<password>` from your DataStax registration confirmation email. If you don’t have the email, register on the DataStax web site.

2. Upgrade the node:

   ```
sudo yum clean all
sudo yum install dse-full
```

3. If a prompt informs you of the download size and asks for confirmation to continue, type Y to continue.

## Version specific upgrade instructions

### Upgrading from any Datastax Community version

#### Uninstall DataStax Community first

If you installed the DataStax Community Debian or RPM packages, you must remove DataStax Community after backing up your configuration files and before setting up and installing from the appropriate repository.

- For Debian packages:
  ```
sudo apt-get remove dsc cassandra
sudo apt-get autoremove
```
  This action also shuts down Cassandra on the node if you haven’t done so already.

- For RPM packages:
  ```
rpm -e apache-cassandra1 –noscripts
```

The old Cassandra configuration file is renamed to cassandra.yaml.rpmsave:

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

#### Converting snitches

The snitch is set in the dse.yaml file instead of cassandra.yaml file. The dse.yaml is located in the `<install_location>/resources/dse/conf directory.`

The following table describes how to convert these properties:

<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 as set in the cassandra.yaml file and leave the default delegated_snitch in the new dse.yaml file unchanged.</td>
</tr>
</tbody>
</table>
**Version specific upgrade instructions**

| org.apache.cassandra.locator.PropertyFileSnitch | Copy/paste the cassandra-topology.properties file from the old installation to <install_location>/resources/cassandra/conf, overwriting 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.

**Upgrading to version 2.2.0**

**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.

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>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>ALTER TABLE</td>
</tr>
<tr>
<td>INSERT</td>
<td>CREATE TABLE</td>
</tr>
<tr>
<td>SELECT</td>
<td>CREATE INDEX</td>
</tr>
<tr>
<td>UPDATE</td>
<td>CREATE KEYSPACE</td>
</tr>
<tr>
<td></td>
<td>DROP TABLE</td>
</tr>
<tr>
<td></td>
<td>DROP INDEX</td>
</tr>
<tr>
<td></td>
<td>DROP KEYSPACE</td>
</tr>
<tr>
<td></td>
<td>TRUNCATE</td>
</tr>
</tbody>
</table>

Perform these steps to resolve the schema disagreement:

1. Using the Command Line Interface (CLI), run the DESCRIBE CLUSTER command. For example:

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

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

   ```bash
   [default@unknown] describe cluster;
   Cluster Information:
   Snitch: com.datastax.bdp.snitch.DseDelegateSnitch
   Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
   Schema versions:
   UNREACHABLE: [10.202.205.203, 10.80.207.102, 10.116.138.23]
   ```

2. Restart the unreachable nodes.

3. Repeat the previous steps 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.
Compaction strategy

If you created tables having the CompactionStrategy configured to be LeveledCompactionStrategy, you need to scrub the SSTables that store those tables.

Before scrubbing the SSTables, upgrade all the nodes.

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 mytable
   ```

   To get help about sstablescrub:
   ```
   usage: sstablescrub -h
   ```

   If you do not scrub the affected SSTables, you might encounter the following error during compactions on tables 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 to version 3.0.0

Hadoop

The ownership of the Hadoop mapred staging directory in the CassandraFS has 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 3.1 as root, use the following command on Linux:

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

Solr

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.

Solr configuration files from previous versions of Datastax Enterprise will be invalidated by the new version of Solr included in this release. Follow these steps to update your Solr config file on the first solr node you upgrade, before upgrading any other nodes:

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. You will also need to issue the following command to recover indexes on each solr node upgraded. On the first node upgraded, this process should happen after the Solr configuration file has been uploaded. Note that in the command below you will need to substitute the name of your solr core.

   ```
   curl -v "http://localhost:8983/solr/admin/cores?action=CREATE&<solr core>.solr&recovery=true"
   ```
The following is an example of how to perform these steps using our Solr-based demos. If you wish to do this on a test cluster, first run the solr, wiki and logging demos on a test cluster running the older version of DSE.

1. Go to the directory containing your Solr application. For example, go to the demos directory:
   - Binary installation
     ```bash
cd <install_location>/demos
   ```
   - Package installation
     ```bash
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:
     ```bash
curl -v --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8'
   ```
   - From the wikipedia directory:
     ```bash
curl -v --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8'
   ```
   - From the log_search directory:
     ```bash
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. After each node is upgraded, run the CREATE command with the recovery option set to true, and the distributed option set to false:

   ```bash
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"
```

**Security**

If you wish to use security you must upgrade the entire cluster before setting up security and then do another rolling restart.

**Upgrading to version 3.1.x**

**Version restrictions**

Upgrading directly from versions other than these does not work:

- DataStax Enterprise 2.2 or later
- Cassandra 1.1.5 - 1.2.10
- DataStax Community 1.1 (Cassandra 1.1.9)

To upgrade from earlier versions, first upgrade to DataStax Enterprise 2.2, and then upgrade the SSTables.

To upgrade from a Brisk release, contact Support.
**Partitioner**

Merge your partitioner setting from the old to the new file. Do not attempt to use the Cassandra 1.2 default partitioner option, *Murmur3Partitioner*, in the new file unless you were already using it.

**CQL 3**

Do not issue any CQL 3 queries until all nodes are upgraded and schema disagreements are resolved.

**Security**

The client_encryption_options for enabling client-to-node SSL have been removed from dse.yaml in 3.1.2 and later. To enable client-to-node SSL, set the option in the cassandra.yaml file.

Before upgrading, if you use these DataStax Enterprise security features, adjust the replication strategy and options in the cassandra.yaml file to configure a replication factor for the dse_auth keyspace greater than 1:

- Kerberos
- Object permission management (internal authorization)
- Internal authentication

Adjust the replication factor for dse_auth on each node in the cluster. After updating the cassandra.yaml file and restarting the node, run nodetool repair to repair the first range returned by the partitioner for the keyspace:

```
nodetool repair dse_auth -pr
```

This should only take a few seconds to complete.

The new version of Cassandra updates the security options. First simply merge the following settings into the new configuration files:

- authenticator
- authorizer
- auth_replication_strategy
- auth_replication_options
- any other diffs

Use the old settings while you are upgrading the cluster so that backward compatibility is maintained. For example, the new file contains the old, Cassandra 1.1 authenticator and authorizer options at this point:

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

If you are upgrading a secure cluster, there may be a significant delay to each node's first startup as the security migration takes place (up to 1 minute). The delay is due to ensuring that the ring is fully connected before the migration starts. During the upgrade of a secure cluster, you may see a security related error message (documented below). You will see the following message in the log when the node has completed the migration:

```
Auth.java (line 208) Migration of legacy auth data is complete.
You should now switch to org.apache.cassandra.auth implementations in cassandra.yaml.
```

After all nodes have been upgraded, change these options to the new Cassandra 1.2 values and perform a rolling restart as explained below.
### Note
If using Kerberos authentication, there are no credentials data to migrate, but user records must still be updated. Merge the related diffs from the old to the new file.

1. Edit the cassandra.yaml to switch to the official Apache versions of PasswordAuthenticator and CassandraAuthorizer:
   ```
   authenticator: org.apache.cassandra.auth.PasswordAuthenticator
   authorizer: org.apache.cassandra.auth.CassandraAuthorizer
   ```

2. Remove these options from the cassandra.yaml file:
   - auth_replication_strategy
   - auth_replication_options

3. Optionally, adjust the replication factor of the system_auth keyspace. The amount of data in this keyspace is typically very small, so leaving it replicated across the cluster is relatively cheap.

### SSTable upgrades
After restarting each node, consider upgrading sstables. Upgrading SSTables is highly recommended under these conditions:

- If you use counter columns
- If you are upgrading from Cassandra 1.1.x or earlier
- If you are upgrading from a DataStax Enterprise version having Cassandra 1.1.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, DataStax recommends upgrading SSTables now to prevent possible future SSTable incompatibilities:

   - [Tarball](#): `<install_location>/bin/nodetool -h upgradesstables`
   - [Package or AMI](#): `nodetool -h upgradesstables`

### Virtual nodes
DataStax recommends using virtual nodes only on data centers running purely Cassandra workloads. You should disable virtual nodes on data centers running either Hadoop or Solr workloads by setting num_tokens to 1 in the cassandra.yaml.

### Solr
If you are upgrading a Solr node, make the default legacy type mapping effective by commenting out the `dseTypeMappingVersion` element:

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

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.

Solr configuration files from previous versions of Datastax Enterprise will be invalidated by the new version of Solr included in this release. Follow these steps to update your Solr config file on the first solr node you upgrade, before upgrading any other nodes:
1. Open the `system.log` and look for the message about the Solr error.

2. The error message briefly describes the changes you need to make. Correct these errors in your `solrconfig.xml` files, then post the corrected files.

3. Existing cores cannot be loaded until the `solrconfig.xml` errors are resolved. Issue the following command to recover indexes on each upgraded Solr node. On the first node upgraded, this process should happen after the Solr configuration file has been uploaded. Note that in the command below you will need to substitute the name of your Solr core.

   ```bash
curl -v "http://localhost:8983/solr/admin/cores?action=CREATE&solr core.solr&recovery=true"
   ``

The following is an example of how to perform these steps using our Solr-based demos. If you wish to do this on a test cluster, first run the solr, wiki and logging demos on a test cluster running the older version of DSE.

Go to the directory containing your Solr application. For example, go to the `demos` directory:

- **Binary installation**
  ```bash
cd install_location/demos
  ```

- **Package installation**
  ```bash
cd /usr/share/dse-demos
  ```

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:**
  ```bash
curl -v --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8' http://localhost:8983/solr/resource/demo.solr/solrconfig.xml
  ```

- **From the `wikipedia` directory:**
  ```bash
  ```

- **From the `log_search` directory:**
  ```bash
curl -v --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8' http://localhost:8983/solr/resource/Logging.log_entries/solrconfig.xml
  ```

After running each curl command, a SUCCESS message appears.

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

After each node is upgraded, run the CREATE command with the the recovery option set to true, and the distributed option set to false:

```bash
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"
```

**Expected error messages**

If you are upgrading from DataStax Enterprise 3.0.x, an exception that looks something like this might appear in logs during a rolling upgrade. Ignore these error messages:
Upgrading the DataStax AMI

ERROR 15:36:54,908 Exception in thread Thread[GossipStage:1,5,main]
    java.lang.NumberFormatException: For input string: "1276087595351923798765477786913079296"

When upgrading Cassandra 1.2 nodes, you can ignore the following error messages related to when a node is attempting to push mutations to the new system_auth keyspace:

    error writing to /192.168.123.11
    java.lang.RuntimeException: Can't serialize ColumnFamily ID 2d324e48-3275-3517-8dd5-9a2c5b0856c5
to be used by version 5, because int <-> uuid mapping could not be established
    (CF was created in mixed version cluster).
    at org.apache.cassandra.db.ColumnFamilySerializer cfIdSerializedSize(ColumnFamilySerializer.java:196)

When upgrading a Solr node, you can ignore the following error:

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 01:29:55,145 Cannot activate core: ks.cf_10000_keys_50_cols
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")))

Recommissioning a node

If you decommissioned a node in the last 72 hours:

- Do not recommission the node until 72 hours has passed.
- If you wish to recommission the node after 72 hours, run "nodetool gossipinfo". Check the STATUS line for the token of the decommissioned node and verify that it does not exist. If it does not exist, then the node has been deleted and it is safe to recommission the node.
- If you need to bring the node into the cluster, contact Support for detailed information on how to assassinate the node.

Update the dse_system keyspace to use the EverywhereStrategy

If you are upgrading to DSE 3.1.5, manually update the dse_system keyspace to the EverywhereStrategy. In cqlsh enter:

ALTER KEYSPACE dse_system WITH replication = {'class': 'EverywhereStrategy'};

Then enter the following command on all nodes:

nodetool repair dse_system

Upgrading the DataStax AMI
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 *Converting snitches*.

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.

The DataStax Java Driver 1.0.1-dse, available on Maven Central, enables Kerberos support on DataStax Enterprise 3.1 and supports SSL for client/server communication with Cassandra 1.2.1 and later. For more information, see Java Driver documentation.

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>Object permission management</td>
<td>Yes</td>
<td>Partial [1]</td>
<td>Partial [1]</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 tables 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.
Cassandra commit log data is not encrypted, only at rest data is encrypted.

Data in DSE/Search Solr tables 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.

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.

**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**.
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.

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
Because JCE-based products are restricted for export to certain countries by the U.S. Export Administration Regulations, DataStax Enterprise does not ship with the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy. DataStax recommends installing the JCE Unlimited Strength Jurisdiction Policy Files:

1. Go to the Oracle Java SE download page.
   - For Java 6, click Previous Releases > Java Platform Technologies > Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6.
   - For Java 7, under Additional Resources, download the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files.
2. Unzip the downloaded file.
3. Copy local_policy.jar and US_export_policy.jar to the $JAVA_HOME/jre/lib/security directory overwriting the existing JARS.

If you choose not to use AES-256, you must remove the AES-256 settings as an allowed cypher for each principal and regenerate the keys for the krbtgt principal. Remove AES-256 settings in one of the following ways:

- If you have not created the principles, use the -e flag to specify encryption:salt type pairs. For example: -e "arcsfour-hmac:normal des3-hmac-sha1:normal" This method requires Kerberos 5-1.2 on the KDC.
- If you have already created the principles, 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.

Alternately, you can 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. Then change the keys for the krbtgt principal.

Note
If the KDC is used by other applications, changing the krbtgt principal's keys invalidates any existing tickets. To prevent this, use the -keepold option when executing the change_password command. For example: 'cpw -randkey krbtgt/krbtgt/REALM@REALM'
Authenticating a DataStax Enterprise cluster with Kerberos

**Securing DataStax Enterprise nodes**

Do not upgrade DataStax Enterprise and set up Kerberos at the same time; see *Security*.

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:

   ```
   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:

   ```
   authenticator: com.datastax.bdp.cassandra.auth.KerberosAuthenticator
   ```

4. Change the replication strategy and default replication factor for the system_auth keyspace. See *Configuring system_auth keyspace replication*.

   DataStax recommends configuring system_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 system_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>`). In DataStax Enterprise 3.1.3 and later, the `cqlshrc` file is in your `~/.cassandra` directory. In earlier releases, the file named `.cqlshrc` is in your home directory.

- Set `<REALM>` to the name of your Kerberos realm.

- Leave `_HOST` as is. 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, see Running the demo on a secure cluster in the Getting started guide.

- The keytab file must contain the credentials for both of the fully resolved principal names, which replace `_HOST` with the FQDN of the host in the `service_principal` and `http_principal` settings. The UNIX user running DSE must also have read permissions on the keytab.

- 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 `~/.cassandra` or 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. Where you set them depends on the version:

- In 3.1.2 and later, configure the client_encryption_options only in the cassandra.yaml file. If necessary, remove them from the dse.yaml.
- In prior 3.1 version, configured them identically in both the dse.yaml and cassandra.yaml files.

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>
  protocol: ssl
  cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA]
```

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, 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.
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 leaving the key password the same as the keystore password:
   
   ```
   keytool -genkey -alias <dse_node0> -keyalg RSA -keystore .keystore
   ```
   
2. Repeat the previous step on each node using a different alias for each one.
3. 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
   ```

4. 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 <dsenodeN> -file <dse_nodeN>.cer -keystore .truststore
   ```

5. 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 dependancies.

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 untarred 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, in DataStax Enterprise 3.1.3 and later you need to create a cqlshrc file in your ~/.cassandra directory. In earlier releases, you need to name the file .cqlshrc and place it 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 gops is not specified the default (auth) is used. On the client side, the gops 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. 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.

You must create a pem key which is used in the cqlshrc file. For example:

```
keytool -importkeystore -srckeystore .keystore -destkeystore <user>.p12 -deststoretype PKCS12
openssl pkcs12 -in <user>.p12 -out <user>.pem -nodes
```

Kerberos and SSL

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

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 or alter an existing table.
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>
Transparent data encryption

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 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 -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 users table:

```cql
ALTER TABLE users
  WITH compression =
  ( 'sstable_compression' : 'DeflateCompressor',
    'chunk_length_kb' : 64 );
```

To set encryption options in the users table, for example:

```cql
ALTER TABLE users
  WITH compression =
  ( 'sstable_compression' : 'Encryptor',
    'cipher_algorithm' : 'AES/ECB/PKCS5Padding',
    'secret_key_strength' : 128
    '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 CQL statement is:

```cql
ALTER TABLE users
  WITH compression =
  ( 'sstable_compression' : 'EncryptingSnappyCompressor',
    'cipher_algorithm' : 'AES/ECB/PKCS5Padding',
    'secret_key_strength' : 128
    '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
- EncryptingDeflateCompressor
- EncryptingSnappyCompressor
- DeflateCompressor
- SnappyCompressor (default)

**Note**
If defining a table with the Encryptor encryptor, set the young generation heap (-Xmn) parameter to a larger space to improve garbage collection. For example if running cassandra-stress, set : -Xmn1600M.

**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'
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.

```sql
ALTER TABLE users
    WITH compression =
    {
        'sstable_compression' : 'EncryptingSnappyCompressor',
        'cipher_algorithm' : 'AES/ECB/PKCS5Padding',
        'secret_key_strength' : 128
        'iv_length' : 16
    };
```

**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:bxegm8vh4wE3S2hO9J36RL2gldBLx0O46J/QmoC3W3U=
- AES/CBC/PKCS5Padding:256:FUhaiy7NG8oeSfe7cOo3hhvovjVl2iJlwBGFH6hsE=
- RC2/CBC/PKCS5Padding:128:5Iw8EW3GqE6y/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

The SolrJ-Auth code is now public.
Auditing is implemented as a log4j-based 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.

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.

DataStax 3.1 supports audit logging of queries and prepared statements submitted to the DataStax Java Driver, which uses the CQL binary protocol.

**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.appendender.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.appendender.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.appendender.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.
Configuring and using data auditing

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 keyspace 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>
<tr>
<td>category</td>
<td>DML/DDL/QUERY for example</td>
<td>no</td>
</tr>
<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**

USE dsp904;

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**

USE dsp904;

host:/192.168.56.1|source:/192.168.56.101|user:#<User allow_all groups=[]> |timestamp:13510004648848|category:DML|type:SET KS|ks:dsp904

**CQL query**

SELECT * FROM t0;

**CQL BATCH**

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;

| 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' ) |

| 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 |
| 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 |
| batch:7d13a423-4c68-4238-af06-a779697088a9 | ks:Keyspace1 | cf:Standard1 |
| operation:insert into Keyspace1.Standard1 (id, name) VALUES (?, ?) [id=1,name=vic] |

DataStax Java Driver queries

| host:ip-10-85-22-245.ec2.internal/10.85.22.245 | source:/127.0.0.1 | user:anonymous |
| timestamp:1370537557052 | category:DDL | type:ADD_KS |
| ks:test | operation:create keyspace test with replication = {'class': 'NetworkTopologyStrategy', 'Analytics': 1}; |

| host:ip-10-85-22-245.ec2.internal/10.85.22.245 | source:/127.0.0.1 | user:anonymous |
| timestamp:1370537557208 | category:DDL | type:ADD_CF |
| ks:test | cf:new_cf | operation:create COLUMNFAMILY test.new_cf ( id text PRIMARY KEY , col1 int, col2 ascii, col3 int); |

| host:ip-10-85-22-245.ec2.internal/10.85.22.245 | source:/127.0.0.1 | user:anonymous |
| timestamp:1370537557236 | category:DML | type:CQL_UPDATE |
| ks:test | cf:new_cf | operation:insert into test.new_cf (id, col1, col2, col3) values ('test1', 42, 'blah', 3); |
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.

```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

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
- DataStax Java and C# Drivers
- Hector
- pycassa

Internal authentication stores user names and bcrypt-hashed passwords in the system_auth.credentials table.

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

Configuring internal authentication
To configure Cassandra to use internal authentication, first you make a few changes to the cassandra.yaml. Next, you start up the client using the default user name and password (cassandra/cassandra). Finally, you change the superuser user name and password to secure these credentials, and set up user accounts.

1. Change the authenticator option in the cassandra.yaml to the native Cassandra PasswordAuthenticator by uncommenting only the PasswordAuthenticator:
   
   ```yaml
   authenticator: org.apache.cassandra.auth.PasswordAuthenticator
   ```

2. **Configure the replication strategy for the system_auth keyspace.**

3. Restart DataStax Enterprise. The syntax for starting up the Cassandra client for the first time after configuring internal 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 -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.

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.

Now, that the superuser password is secure, set up user accounts and authorize users to access the database objects by using CQL to grant them permissions on those objects.

CQL 3 supports the following statements for setting up user accounts:

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

---

### Enable internal security in DataStax Enterprise 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 org.apache.cassandra.auth.PasswordAuthenticator.
   - Change the authorizer to org.apache.cassandra.auth.CassandraAuthorizer.
5. After the restarts have completed, remove the default superuser and create at least one new superuser.

Logging in with cqlsh

To avoid having to pass credentials for every login using cqlsh, in DataStax Enterprise 3.1.3 and later you need to create a cqlshrc file in your ~/.cassandra directory. In earlier releases, you need to name the file .cqlshrc and place it in your home directory. When present, it passes default login information to cqlsh. For example:

```
[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 the 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**

CassandraAuthorizer is one of many possible IAuthorizer implementations, and the one that stores permissions in the system_auth.permissions table to support all authorization-related CQL 3 statements. Configuration consists mainly of changing the authorizer option in the cassandra.yaml to use the CassandraAuthorizer.

To configure internal authorization for managing object permissions:

1. In the cassandra.yaml, comment out the default AllowAllAuthorizer and add the CassandraAuthorizer as shown here:

   ```yaml
   #authorizer: org.apache.cassandra.auth.AllowAllAuthorizer
   authorizer: org.apache.cassandra.auth.CassandraAuthorizer
   ```

   You can use any authenticator except AllowAll.

2. **Configure the system_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.

4. Restart the node after changing the cassandra.yaml file.

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

- GRANT
- LIST PERMISSIONS
- REVOKE

You can **enable internal authorization on existing clusters with no downtime.**

**Configuring system_auth keyspace replication**

Cassandra uses the system_auth keyspace for storing security authentication and authorization information. If you use the following authenticator/authorizer, you must set the replication factor with a keyspace command such as `ALTER KEYSACE` to prevent a potential problem logging into a secure cluster:

- `authenticator: org.apache.cassandra.auth.PasswordAuthenticator: the users’ hashed passwords in system_auth.credentials table`
- `authorizer: org.apache.cassandra.auth.CassandraAuthorizer: the users’ permissions in system_auth.permissions table`

**Setting the replication factor**

Do not use the default replication factor of 1 for the system_auth keyspace. 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 most system_auth queries, Cassandra uses a consistency level of ONE and uses QUORUM for the default cassandra superuser; see Configuring data consistency.

**SimpleStrategy example:**

```
ALTER KEYSACE "system_auth"
   WITH REPLICAION ={
   'class' : 'SimpleStrategy', 'replication_factor' : 3 
};
```

**NetworkTopologyStrategy example:**
ALTER KEYSPACE "system_auth"
    WITH REPLICACTION = {'class' : 'NetworkTopologyStrategy', 'dc1' : 3, 'dc2' : 2};

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. See Setting the job tracker node.</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 opscnterd 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 opscnterd daemon. See OpsCenter and OpsCenter agent ports.</td>
</tr>
<tr>
<td><strong>OpsCenter Specific</strong></td>
<td></td>
</tr>
<tr>
<td>8888</td>
<td>OpsCenter website. The opscnterd 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. Please read the description for port 7199.</td>
</tr>
<tr>
<td>7000</td>
<td>Cassandra inter-node cluster communication. See storage_port.</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+). Open this port only if you want to remotely connect to the node via JMX. Running nodetool or opscnterd locally does not require these ports to be open. See JMX options in Tuning Java resources.</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 opscnterd daemon to make Thrift requests to each node in the cluster. See rpc_port.</td>
</tr>
<tr>
<td><strong>DataStax Enterprise Specific</strong></td>
<td></td>
</tr>
<tr>
<td>7001</td>
<td>Encrypted node-to-node communication port. See ssl_storage_port.</td>
</tr>
<tr>
<td>9042</td>
<td>CQL native clients port. See native_transport_port.</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 opscnterd daemon.</td>
</tr>
<tr>
<td><strong>OpsCenter Specific</strong></td>
<td></td>
</tr>
<tr>
<td>Port</td>
<td>Description</td>
</tr>
<tr>
<td>--------</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>

1(1, 2, 3, 4, 5) See OpsCenter and OpsCenter agent ports.
Deployment

Production deployment planning

The Cassandra 1.2 topic Planning a cluster deployment provides guidance for planning a DataStax Enterprise cluster. The following guidelines are also recommended:

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

Configuring replication

Cassandra performs replication to store multiple copies of data on multiple nodes for reliability and fault tolerance. To configure replication, you need to choose a data partitioner and replica placement strategy. Data partitioning determines how data is placed across the nodes in the cluster. For information about how this works, see Data distribution and replication. Nodes communicate with each other about replication and other things using the gossip protocol. Be sure to configure gossip, as described in About internode communications (gossip).

Virtual nodes

DataStax Enterprise 3.1 and above can use virtual nodes. Virtual nodes simplify many tasks in Cassandra, such as eliminating the need to determine the partition range (calculate and assign tokens), rebalancing the cluster when adding or removing nodes, and replacing dead nodes. For a complete description of virtual nodes and how they work, see About virtual nodes, and the Virtual nodes in Cassandra 1.2 blog.

Using virtual nodes

In the cassandra.yaml file uncomment num_tokens and leave the initial_token parameter unset. Guidelines for using virtual nodes include:

- **Determining the num_tokens value:** The initial recommended value for num_tokens is 256. For more guidance, see Setting up virtual nodes.
- **Mixed architecture:** Cassandra supports using virtual node-enabled and non-virtual node data centers. For example, a single cluster could have a vanilla cassandra data center with vnodes enabled and a search data center without vnodes.
- **Migrating:** To upgrade existing clusters to virtual nodes, see Enabling virtual nodes on an existing production cluster.

Caution!

Currently, DataStax recommends using virtual nodes only on data centers running purely Cassandra workloads. You should disable virtual nodes on data centers running either Hadoop or Solr workloads.

Using the single-token-per-node architecture in DSE 3.1 and above
If you don't use virtual nodes, you must make sure that each node is responsible for roughly an equal amount of data. To do this, assign each node an initial-token value and calculate the tokens for each data center as described in Generating tokens located in the DataStax Enterprise 3.0 documentation. You can also use the Murmur3Partitioner and calculate the tokens as described in Cassandra 1.2 Generating tokens.

**Partitioner settings**

You can use either the Murmur3Partitioner or RandomPartitioner with virtual nodes.

The Murmur3Partitioner (org.apache.cassandra.dht.Murmur3Partitioner) is the default partitioning strategy for new Cassandra clusters (1.2 and above) and the right choice for new clusters in almost all cases. You can only use Murmur3Partitioner for new clusters; you cannot change the partitioner in existing clusters. If you are switching to the 1.2 cassandra.yaml, be sure to change the partitioner setting to match the previous partitioner.

The RandomPartitioner (org.apache.cassandra.dht.RandomPartitioner) was the default partitioner prior to Cassandra 1.2. You can continue to use this partitioner when migrating to virtual nodes.

**Snitch settings**

A snitch determines which data centers and racks are written to and read from. It informs Cassandra about the network topology so that requests are routed efficiently and allows Cassandra to distribute replicas by grouping machines into data centers and racks. All nodes must have exactly the same snitch configuration.

The following sections describe three commonly-used snitches. All available 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. This snitch logically configures each type of node in separate data centers to segregate the analytics, real-time, and search workloads. You can use the DseSimpleSnitch for mixed-workload DSE clusters located in one physical data center or for multiple physical data centers. When using multiple data centers, place each type of node (Cassandra, Hadoop, and Solr) in a separate physical data center.

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

**SimpleSnitch**

For a single data center (or single node) cluster, the 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 use a rack and data center aware snitch from the start, such as the RackInferringSnitch.

**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 the cassandra-topology.properties configuration file.

- Packaged installations: /etc/dse/cassandra/cassandra-topology.properties
- Tarball installations: <install_location>/resources/cassandra/conf/cassandra-topology.properties
Deployment

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

For example, suppose 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, you would specify them as follows:

```
# 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 definitions.

### Choosing keyspace replication options

When you create a keyspace, you must define the replica placement strategy class and the number of replicas you want. DataStax recommends choosing `NetworkTopologyStrategy` for single and multiple data center clusters. This strategy is as easy to use as the `SimpleStrategy` and allows for expansion to multiple data centers in the future. 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:

```
CREATE KEYSPACE test
    WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'us-east' : 6};
```

Or for a multiple data center cluster:

```
CREATE KEYSPACE test2
    WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'dc1' : 3, 'dc2' : 3};
```

When creating the keyspace, 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 and later, 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.
   - **Packaged installs**: nodetool status
   - **Binary installs**: `<install_location>/bin/nodetool status`
   
   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:

   ```
   ALTER KEYSPACE cfs
   WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'dc1' : 3);
   
   ALTER KEYSPACE cfs_archive
   WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'dc1' : 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. Alter 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.

**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.
Single data center deployment

center deployment.

**Prerequisites**

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

- Installing DataStax Enterprise on each node.
- Choosing a name for the cluster.
- For a mixed-workload cluster, knowing the purpose of each node.
- Getting the IP address of each node.
- Determining which nodes are seed nodes. (DataStax Enterprise nodes use this host list to find each other and learn the topology of the ring.)
- Other possible configuration settings are described in The cassandra.yaml configuration file.

In DataStax Enterprise 3.0.1 and later, 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.

**Configuration example**

This example describes installing a 6 node cluster spanning 2 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. Suppose 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. If the nodes are behind a firewall, open the required ports for internal/external communication. See Configuring firewall port access.
3. If DataStax Enterprise is running, stop the nodes and clear the data:

   - Packaged installs:
     $ sudo service dse stop (stops the service)
     $ sudo rm -rf /var/lib/cassandra/* (clears the data from the default directories)
   - Binary installs:
     From the install directory:
     $ sudo bin/dse cassandra-stop (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:

   - num_tokens: 256 See Setting up virtual nodes in About virtual nodes.
   - -seeds: <internal IP_address of each seed node>
   - listen_address: <localhost IP_address>
   - auto_bootstrap: false (Add this setting only when initializing a fresh cluster with no data.)
   - Comment out auth_replication_options and replication_factor:

     # auth_replication_options:
     # replication_factor: 1

node0:

cluster_name: 'MyDemoCluster'
num_tokens: 256
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

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

<table>
<thead>
<tr>
<th>Node</th>
<th>listen address</th>
</tr>
</thead>
<tbody>
<tr>
<td>node1</td>
<td>110.82.155.1</td>
</tr>
<tr>
<td>node2</td>
<td>110.82.155.2</td>
</tr>
<tr>
<td>node3</td>
<td>110.82.155.3</td>
</tr>
<tr>
<td>node4</td>
<td>110.82.155.4</td>
</tr>
<tr>
<td>node5</td>
<td>110.82.155.5</td>
</tr>
<tr>
<td>node6</td>
<td>110.82.155.6</td>
</tr>
<tr>
<td>node7</td>
<td>110.82.155.7</td>
</tr>
</tbody>
</table>
5. 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*.

6. Check that your cluster is up and running:

- Packaged installs: `nodetool status`
- Binary installs: `<install_location>/bin/nodetool status`

---

**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:

- Installing DataStax Enterprise on each node.
- Choosing a name for the cluster.
- For a mixed-workload cluster, knowing the purpose of each node.
- Getting the IP address of each node.
- Determining which nodes are seed nodes. (DataStax Enterprise nodes use this host list to find each other and learn the topology of the ring.)
- Developing a naming convention for each data center and rack, for example: DC1, DC2 or 100, 200 and RAC1, RAC2 or R101, R102.
- Other possible configuration settings are described in *The cassandra.yaml configuration file.*
In DataStax Enterprise 3.0.1 and later, 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.

**Configuration example**

This example describes installing a 6 node cluster spanning 2 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.

**Property file locations in packaged installations:**

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

**Property files locations 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:

   node0 10.168.66.41 (seed1)
   node1 10.176.43.66
   node2 10.168.247.41
   node3 10.176.170.59 (seed2)
   node4 10.169.61.170
   node5 10.169.30.138

2. If the nodes are behind a firewall, open the required ports for internal/external communication. See *Configuring firewall port access*.

3. If DataStax Enterprise is running, stop the nodes and clear the data:

   - Packaged installs:
     
     $ sudo service dse stop (stops the service)
     $ sudo rm -rf /var/lib/cassandra/* (clears the data from the default directories)
   
   - Binary installs:
     
     From the install directory:
     
     $ sudo bin/dse cassandra-stop (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:

- num_tokens: 256  See Setting up virtual nodes in About virtual nodes.
- seeds: <internal IP_address of each seed node>
- listen_address: <localhost IP address>
- auto_bootstrap: false  (Add this setting only when initializing a fresh cluster with no data.)
- Comment out auth_replication_options and replication_factor:

  # auth_replication_options:
  # replication_factor: 1

node0:

  cluster_name: 'MyDemoCluster'
  num_tokens: 256
  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 listen_address:

<table>
<thead>
<tr>
<th>Node</th>
<th>Listen address</th>
</tr>
</thead>
<tbody>
<tr>
<td>node1</td>
<td>10.176.43.66</td>
</tr>
<tr>
<td>node2</td>
<td>10.168.247.41</td>
</tr>
<tr>
<td>node3</td>
<td>10.176.170.59</td>
</tr>
<tr>
<td>node4</td>
<td>10.169.61.170</td>
</tr>
<tr>
<td>node5</td>
<td>10.169.30.138</td>
</tr>
</tbody>
</table>

5. If necessary, change the dse.yaml file on each node 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:

    delegated_snitch: org.apache.cassandra.locator.PropertyFileSnitch
6. In the cassandra-topology.properties file, use your naming convention to 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:

```
# 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
```

7. 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*.

8. Check that your cluster is up and running:

- Packaged installs: `nodetool status`
- Binary installs: `<install_location>/bin/nodetool status`

![nodetool status output](image)

**More information about configuring data centers**

Links to more information about configuring a data center:

- *Choosing keyspace replication options*
- *Replication in a physical or virtual data center* (Applies only to the single-token-per-node architecture.)

**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 HiveQL queries on Cassandra data
- **Pig** for exploring very large data sets
- **Apache Mahout** for machine learning applications

Before starting an analytics/Hadoop node on a production cluster or data center, it is important to disable the virtual node configuration. You can skip this step to run the Hadoop getting started tutorial.

**Disabling virtual nodes**

DataStax recommends using virtual nodes only on data centers running Cassandra real-time workloads. You should disable virtual nodes on data centers running either Hadoop or Solr workloads.

To disable virtual nodes:

1. In the cassandra.yaml file, set num_tokens to 1.
   ```
   num_tokens = 1
   ```
2. Uncomment the initial_token property and set it to 1 or to the value of a generated token for a multi-node cluster.

**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 this command to stop the analytics node:

```
bin/dse cassandra-stop
```

Check that the dse process has stopped:

```
ps auwx | grep dse
```

If the dse process stopped, the output should be minimal, for example:
If the output indicates that the dse process is not stopped, rerun 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
   ```

**Hadoop getting started tutorial**

In this tutorial, you download a text file containing a State of the Union speech and 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 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.

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

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

   ```
   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:

```
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.

```
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%

```

7. List the contents of the output directory on the CassandraFS.

```
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.

   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:
   (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.

   bin/dse hadoop fs -cat /user/hadoop/wordcount/output/part-r-00000

The output is:

   "D." 1
   "Don't 1
   "I 4
   ...

**Hadoop demos**

You can run these Hadoop demos in DataStax Enterprise:

- **Portfolio Manager demo**: Demonstrates a hybrid workflow using DataStax Enterprise.
- **Hive example**: Shows how to use 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**: Creates a Pig relation, performs a MapReduce jobs, and stores results in a Cassandra table.
- **Sqoop Demo**: Migrates data from a MySQL database containing information from the North American Numbering Plan.

**Setting the replication factor**

The default replication for the HiveMetaStore, dse_system, cfs, and cfs_archive system keyspaces is 1. A replication factor of 1 is suitable for development and testing of a single node, not for a production environment. For production clusters, increase the replication factor to at least 2. The higher replication factor ensures resilience to single-node
failures. For example:

```
ALTER KEYSPACE cfs
    WITH REPLICATION = {
        'class': 'NetworkTopologyStrategy',
        'dc1': 3
    };
```

### 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:
   ```
   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`.

### 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.
• Start up an analytics node using the \texttt{-j} option.

\begin{verbatim}
dse cassandra -t -j
\end{verbatim}

or in a binary distribution:

\begin{verbatim}
<install_location>/bin/dse cassandra -t -j
\end{verbatim}

• Use the \texttt{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.

\textbf{About the reserve job tracker}

DataStax Enterprise 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.

\textbf{Using common hadoop commands}

Use familiar hadoop \texttt{fs} commands to perform functions in the CassandraFS that correspond to open source, HDFS file system shell commands in HDFS:

• Packaged or AMI distributions:

\begin{verbatim}
dse hadoop fs <option>
\end{verbatim}

• Tarball installation:

\begin{verbatim}
<install_location>/bin/dse hadoop fs <option>
\end{verbatim}

For example, using this syntax, you can load MapReduce input from the local file system into the Cassandra File System on Linux.

\begin{verbatim}

dse hadoop fs -mkdir /user/hadoop/wordcount/input

dse hadoop fs -copyFromLocal $HADOOP_EXAMPLE/data/state_of_union/state_of_union.txt /user/hadoop/wordcount/input
\end{verbatim}

To list all options for performing command hadoop HDFS commands:

\begin{verbatim}

dse hadoop fs -help
\end{verbatim}

A \textit{DSE command reference} lists other commands.

\textbf{Managing the job tracker using dsetool commands}

Several \texttt{dsetool commands} are useful for managing job tracker nodes:

• \texttt{dsetool jobtracker}

  Returns the job tracker hostname and port to your location in the data center where you issued the command.

• \texttt{dsetool movejt <data center>-<workload> <node IP>}

  Moves the job tracker and notifies the task tracker nodes.

• \texttt{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>
</property>

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>
</property>

### About the Cassandra File System

A Hive or Pig analytics job requires a Hadoop file system to function. DataStax Enterprise provides a replacement for the Hadoop Distributed File System (HDFS) called the Cassandra File System (CassandraFS), which serves this purpose. When an analytics node starts up, DataStax Enterprise 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.

### Deleting files from the Cassandra File System

Cassandra does not immediately remove deleted data from disk when you use the `dse hadoop fs -rm <file>` command. Instead, Cassandra treats the deleted file like any data deleted from Cassandra. A tombstone is written to indicate the new column status. Columns marked with a tombstone exist for a configured time period (defined by the `gc_grace_seconds` value set on the table). When the grace period expires, the compaction process permanently deletes the column. You do not have to manually remove expired data. A deleted column can reappear if you do not run node repair routinely.

To force deletion of data after using `dse hadoop fs -rm <file>` command, use the `nodetool flush` command. This command forces tombstones in the memtable to flush to disk and compact together with the data to remove the files.

### 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:

   ```xml
   <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 use `dse commands` to copy data to the new CassandraFS site:

```
dse hadoop fs -copyFromLocal /tmp/giant_log.gz cfs-NewCassandraFS://cassandrahost/tmp

dse hadoop fs distcp hdfs:/// cfs-NewCassandraFS:///
```

Using Hive

DataStax Enterprise includes Hive, a data warehouse system for Hadoop that converts most HiveQL queries into MapReduce jobs for execution on a DataStax Enterprise analytics node. HiveQL is a SQL-like language that organizes data into table structures for storage in the Cassandra File System (CassandraFS). DataStax Enterprise analytics nodes store Hive table structures in the CassandraFS instead of in the Hadoop Distributed File System (HDFS). The table metadata is stored in the metastore database.

Unlike open source Hive, there is no need to run the metastore as a standalone database to support multiple users. DataStax Enterprise implements the Hive metastore as a keyspace within Cassandra. The metastore supports multiple
users and requires no configuration except increasing the default replication factor of the keyspace.

**Setting the Job Tracker node for Hive**

Hive clients automatically select the correct job tracker node upon startup. You *change the job tracker node* for Hive as you would for any analytics node, and you use the *dsetool commands* to manage the job tracker.

**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.

**Hive storage handlers**

You use a Hive client and custom storage handlers to access data in Cassandra. DataStax Enterprise maps existing Cassandra tables into Hive tables. To store Hive tables in the Cassandra file system (CassandraFS) as CQL tables, DataStax Enterprise 3.1 and later provides the CQL 3 storage handler: `org.apache.hadoop.hive.cassandra.cql3.CqlStorageHandler`.

For information about using Hive with legacy tables, such as those created using Thrift or the CLI, see *DataStax Enterprise 3.0 documentation*.

**Hive to Cassandra type mapping**

This table shows CQL, Cassandra internal storage engine, and Hive data type mapping:

<table>
<thead>
<tr>
<th>CQL 3</th>
<th>Cassandra Internal</th>
<th>Hive</th>
</tr>
</thead>
<tbody>
<tr>
<td>text/varchar</td>
<td>UTF8Type</td>
<td>string</td>
</tr>
<tr>
<td>ascii</td>
<td>AsciiType</td>
<td>string</td>
</tr>
<tr>
<td>timestamp</td>
<td>DateType</td>
<td>timestamp</td>
</tr>
<tr>
<td>bigint</td>
<td>LongType</td>
<td>bigint</td>
</tr>
<tr>
<td>int</td>
<td>Int32Type</td>
<td>int</td>
</tr>
<tr>
<td>double</td>
<td>DoubleType</td>
<td>double</td>
</tr>
<tr>
<td>float</td>
<td>FloatType</td>
<td>float</td>
</tr>
<tr>
<td>boolean</td>
<td>BooleanType</td>
<td>boolean</td>
</tr>
<tr>
<td>uuid</td>
<td>UUIDType</td>
<td>binary</td>
</tr>
<tr>
<td>timeuuid</td>
<td>TimeUUIDType</td>
<td>binary</td>
</tr>
<tr>
<td>other</td>
<td>other</td>
<td>binary</td>
</tr>
</tbody>
</table>

**TBLPROPERTIES and SERDEPROPERTIES**

The TBLPROPERTIES clause specifies CassandraFS and MapReduce properties for the table. 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 partition key, column names and column values.

The following properties can be declared in the TBLPROPERTIES or SERDEPROPERTIES clause or both. You can change Hive storage properties listed in the following table 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.
<table>
<thead>
<tr>
<th>Property</th>
<th>Used in Hive Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cassandra.cf.name</td>
<td>TBL and SERDE</td>
<td>Cassandra table name</td>
</tr>
<tr>
<td>cassandra.columns.mapping</td>
<td>TBL</td>
<td>Mapping of Hive to legacy Cassandra columns</td>
</tr>
<tr>
<td>cassandra.consistency.level</td>
<td>TBL and SERDE</td>
<td>Consistency level - default LOCAL_ONE</td>
</tr>
<tr>
<td>cassandra.cql3.type</td>
<td>TBL and SERDE</td>
<td>CQL types</td>
</tr>
<tr>
<td>cassandra.host</td>
<td>TBL and SERDE</td>
<td>IP of a Cassandra node to connect to</td>
</tr>
<tr>
<td>cassandra.ks.name</td>
<td>TBL and SERDE</td>
<td>Cassandra keyspace name</td>
</tr>
<tr>
<td>cassandra.ks.repfactor</td>
<td>TBL and SERDE</td>
<td>Cassandra replication factor - default 1</td>
</tr>
<tr>
<td>cassandra.ks.strategy</td>
<td>TBL and SERDE</td>
<td>Replication strategy class</td>
</tr>
<tr>
<td>cassandra.ks.stratOptions</td>
<td>TBL and SERDE</td>
<td>Strategy options shown in the NetworkTopologyStrategy example</td>
</tr>
<tr>
<td>cassandra.page.size</td>
<td>SERDE</td>
<td>Fetch tables with many columns by page size</td>
</tr>
<tr>
<td>cassandra.partitioner</td>
<td>TBL and SERDE</td>
<td>Partitioner - default is your configured partitioner</td>
</tr>
<tr>
<td>cassandra.port</td>
<td>TBL and SERDE</td>
<td>Cassandra RPC port - default 9160</td>
</tr>
<tr>
<td>cassandra.input.split.size</td>
<td>TBL and SERDE</td>
<td>MapReduce split size, rows processed per mapper (64k rows per split) - default 64 * 1024</td>
</tr>
</tbody>
</table>

In DataStax Enterprise 3.1.5, the default consistency level for cassandra.consistency.level is LOCAL_ONE.

When you create an external table in Hive, you need to specify these properties:

- cassandra.ks.name
- cassandra.cf.name
- cassandra.ks.repfactor (if SimpleStrategy is used)
- cassandra.ks.strategy
- cassandra.ks.stratOptions (if NetworkTopologyStrategy is used)

You do not need to specify cassandra.partitioner. Your configured partitioner is used by Hive. For example, Hive uses this property value if you use the Cassandra 1.2 default partitioner:

"cassandra.partitioner" = "org.apache.cassandra.dht.Murmur3Partitioner"

Mapping a Hive database to a Cassandra keyspace and MapReduce performance tuning show examples of using some of these properties.

**Running Hive**

You can run Hive as a server or as a client. To perform the demo examples, run Hive as a client.
Starting a Hive client

Use a Hive client on a node in the cluster under these conditions:

- To connect to the Hive server running on another node
- To use Hive in a single-node cluster

You can start a Hive client on any analytics node and run MapReduce queries directly on data already stored in Cassandra.

To start a Hive client:

For packaged or AMI distributions, you can start the Hive client as follows:

```
dse hive
```

For a tarball, binary distribution, start Hive as follows:

```
<install_location>/bin/dse hive
```

A warning about not being able to access history indicates a permissions problem. Change file system permissions or use sudo to start hive.

Creating a table

If the table exists in Cassandra, you can create a corresponding table to access data using Hive. Use the CREATE EXTERNAL TABLE command. If the table does not already exist in Cassandra, create one using CQL 3, and then use the CREATE EXTERNAL TABLE command.

The HiveQL Manual provides information about the HiveQL syntax.

1. Start cqlsh. For example, on Linux:

   ```
   ./cqlsh
   ```

2. Create a keyspace and a table and insert some data using CQL 3.

   ```
cqlsh> CREATE KEYSPACE cql3ks WITH replication =
   { 'class': 'NetworkTopologyStrategy',
   'Analytics': '1' };
   
cqlsh> CREATE TABLE cql3ks.test
   (m text, n text, o text, p text, PRIMARY KEY (m));
   
cqlsh> INSERT INTO cql3ks.test (m, n, o, p)
   VALUES ('abc', 'def', 'hij', 'klm');
   ```

   Alternatively, you can create legacy tables because the Hive implementation in DataStax Enterprise 3.1 is backward compatible.

3. Start the Hive client. For example, from the DataStax Enterprise `<install_home>/bin` directory on Linux:

   ```
   ./dse hive
   ```

4. Create a table in Hive that corresponds to the test table you created in CQL 3.

   ```
hive> CREATE EXTERNAL TABLE MyHiveTable
   (m string, n string, o string, p string)
   STORED BY 'org.apache.hadoop.hive.cassandra.cql3.CqlStorageHandler'
   TBLPROPERTIES ( "cassandra.ks.name" = "cql3ks",
   "cassandra.cf.name" = "test",
   "cassandra.cql3.type" = "text, text, text, text");
   ```
5. Issue a Hive Query to get some information about the keyspace/Hive database.

    hive> describe database cql3ks;

Output is:

    OK
    cql3ks cfs://127.0.0.1/user/hive/warehouse/cql3ks.db

6. Retrieve the data in the table using a Hive query.

    hive> select * from MyHiveTable;

Output is:

    OK
    abc def hij klm

**Automatic recognition of Cassandra keyspaces**

If a keyspace exists in Cassandra, you can use the keyspace in Hive. For example, create a keyspace in Cassandra using cqlsh. Add some data to the table using cqlsh, and then access the data by simply issuing a USE command in Hive.

    cqlsh> CREATE KEYSPACE cassandra_keyspace WITH replication =
           ( 'class': 'SimpleStrategy', 'replication_factor': 1 );
    cqlsh> USE cassandra_keyspace;
    cqlsh:cassandra_keyspace> CREATE TABLE exampleTable
                               ( key int PRIMARY KEY , data text );
    cqlsh:cassandra_keyspace> INSERT INTO exampletable (key, data )
                               VALUES ( 1, 'This data can be read automatically in hive');
    cqlsh:cassandra_keyspace> quit;

At this point, you can use the keyspace in Hive without manually specifying it.

    hive> USE cassandra_keyspace;
    hive> SHOW TABLES;
    OK
    exampleTable
    hive> SELECT * FROM exampletable;
    OK
    1 This data can be read automatically in hive

**Mapping a Hive database to a Cassandra keyspace**

You can map a Hive database to an existing Cassandra keyspace by naming them the same in the CREATE EXTERNAL TABLE definition. Optionally, if your Hive database and Cassandra keyspace use different names, you can declare keyspace properties in your external table definition using the TBLPROPERTIES clause.

**SimpleStrategy keyspace example**

    hive> CREATE EXTERNAL TABLE MyHiveTable
               (m string, n string, o string, p string)
           STORED BY 'org.apache.hadoop.hive.cassandra.cql3.CqlStorageHandler'
           TBLPROPERTIES ("cassandra.ks.name" = "MyCassandraKS",
                          "cassandra.cf.name" = "mycasstable",
                          "cassandra.ks.repfactor" = "2");
"cassandra.ks.strategy" =
"org.apache.cassandra.locator.SimpleStrategy");

NetworkTopologyStrategy keyspace example

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
    (m string, n string, o string, p string)
    STORED BY 'org.apache.hadoop.hive.cassandra.cql3.CqlStorageHandler'
    TBLPROPERTIES ("cassandra.ks.name" = "MyCassandraKS",
    "cassandra.cf.name" = "mycasstable",
    "cassandra.ks.stratOptions" = "DC1:1, DC2:2, DC3:1",
    "cassandra.ks.strategy" =
    "org.apache.cassandra.locator.NetworkTopologyStrategy");

Accessing tables with many columns

In the TBLPROPERTIES clause, set the cassandra.page.size to fetch a table with many columns at once by page size.

Handling schema changes

A runtime exception can occur if you create an external table in Hive, change the Cassandra table that is mapped to the Hive table, and then query the Hive table. Changes that occur to the Cassandra table get out of synch with the Hive table, and naturally, this causes problems. The workaround is:

1. In Hive, drop any external table after changing the corresponding Cassandra table.
2. Run SHOW TABLES:

   hive> DROP TABLE mytable;
   hive> SHOW TABLES;

   Now, the table in Hive contains the updated data.

Creating a Hive managed table

To use Hive with CassandraFS as you would use it in an HDFS-based Hadoop implementation except saving the tables to CassandraFS instead of HDFS, follow examples in this section. Create a Hive managed table 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.

For example, on the Mac OSX:

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.

**Using an external file system in Hive**
In DataStax Enterprise 3.1.5, you can map a file in an external file system, such as **S3 native file system** to a table in Hive. The DSE Analytics/Hadoop cluster continues to use the CassandraFS file system. The data source is external to Hive, located in S3 for example. You create a Hive external table for querying the data in an external file system. When you drop the external table, only the table metadata stored in the HiveMetaStore keyspace in the CassandraFS is removed. The data persists in the external file system.

First, you set up the `hive-site.xml` and `core-site.xml` files, and then create an external table as described in this procedure.

1. Open the `hive-site.xml` for editing. This file is located in:
   - Packaged installations: `/etc/dse/hadoop`
   - Binary installations: `/<install_location>/resources/hive/conf`
2. Add a property to hive-site.xml to set the default file system to be the native S3 block file system. Use fs.default.name as the name of the file system and the location of the bucket as the value. For example, if the S3 bucket name is dsp2377:

```xml
<property>
  <name>fs.default.name</name>
  <value>s3n://dsp2377</value>
</property>
```

3. Save the file.

4. 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

5. Add these properties to core-site.xml to specify the access key ID and the secret access key credentials for accessing the native S3 block filesystem:

```xml
<property>
  <name>fs.s3n.awsAccessKeyId</name>
  <value>ID</value>
</property>

<property>
  <name>fs.s3n.awsSecretAccessKey</name>
  <value>Secret</value>
</property>
```

6. Save the file and restart Cassandra.

7. Start Hive, and on the Hive command line, create an external table for the data on S3. Specify the S3 file name as shown in this example:

```
DSE creates the new CassandraFS.
```

```
hive> CREATE EXTERNAL TABLE mytable (key STRING, value INT)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '=' STORED AS TEXTFILE LOCATION 's3n://dsp2377/2377/data';
```

Now, having the S3 data in Hive, you can query the data using Hive.

### Starting the Hive server

A node in the analytics cluster can act as the Hive server. Other nodes connect to Hive through the JDBC driver. To start the Hive server, choose a node in the Hadoop cluster and run this command:

```
dse hive --service hiveserver
```

or in a binary distribution:

```
<install_location>/bin/dse hive --service hiveserver
```

### 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 Hive SET 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`.

### 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 specifies rows to be processed per mapper. The default size is 64k rows per split. You can decrease the split size to create more mappers.

### 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:

```bash
SET mapred.child.java.opts="-server -Xmx512M"
```

You can also lower memory usage by turning off map output compression in `mapred-site.xml`.

### 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](#).

#### 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 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:

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:

```plaintext
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.

```bash
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:

```bash
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 data flow language and platform for exploring big data sets. The language, Pig Latin, uses relations, which are similar to tables. The tuples in a relation correspond to the rows in a table. However, unlike a relational database table, Pig relations do not require every tuple to contain the same number of fields or fields in the same position (column) to be of the same type. Using Pig, you can devise logic for data transformations, such as filtering data and grouping relations. The transformations occur during the MapReduce phase. The Apache Pig documentation contains more information about defining and working with Pig relations.

Configure the job tracker node for the node running Pig as you would for any analytics (Hadoop) node. Use the dsetool commands to manage the job tracker. After configuration, Pig clients automatically select the correct job tracker node on startup. Pig programs are compiled into MapReduce jobs, executed in parallel by Hadoop, and run in a distributed fashion on a local or remote cluster.

Support for CQL collections

In DataStax Enterprise 3.1.4 and later, support for CQL collections has been added. Pig-supported types must be used. For example, a list cannot include columns of type decimal because the version of Pig integrated in DataStax 3.1.4 does not support this type. A list of strings or ints works fine.

CQL 3 pushdown filter

DataStax 3.1.4 includes a CqlStorage URL option, use_secondary. Setting the option to true optimizes the processing of the data by moving filtering expressions in Pig as close to the data source as possible. To use this capability:

- Create a secondary index for the Cassandra table.
  For Pig pushdown filtering, the secondary index must have the same name as the column it is indexing.
- Include the use_secondary option with a value of true in the url format for CqlStorage. For example:

  ```sql
  newdata = LOAD 'cql://ks/cf_300000_keys_50_cols?use_secondary=true'
  USING CqlStorage();
  ```

Formatting Pig data

These examples include how to format Pig data, which differs slightly from DataStax Enterprise 3.1.2-3.1.3. The syntax in 3.1.4 has been simplified and .value is no longer needed to specify the values to be formatted:

- DataStax Enterprise 3.1.2-3.1.3

  ```bash
  grunt> insertformat= FOREACH morevalues GENERATE
      TOTUPLE(TOTUPLE('a',x.value)),TOTUPLE(y.value);
  ```
DataStax Enterprise 3.1.4

grunt> insertformat= FOREACH morevalues GENERATE
TOTUPLE(TOTUPLE('a',x)),TOTUPLE(y);

Running the Pig demo

Three examples demonstrate how to use Pig on DataStax Enterprise 3.1.4 and later to work with CQL 3 tables in Cassandra 1.2.6.x and later.

- How to save Pig relations from/to Cassandra
  Pig uses a single tuple.

- How to work with a Cassandra compound primary key in Pig
  Pig uses three tuples, one for the partition key and two for the two clustering columns.

- How to use Pig to set up logic for exploring library data
  This example from the Cassandra and Pig tutorial shows how to copy public library data into Cassandra, add logic to save the data to a Pig relation, execute programs by running MapReduce jobs, and view results in a Cassandra table.

Start Pig

1. First, start DataStax Enterprise as an analytics (Hadoop) node.
   - Packaged installations
     Set HADOOP_ENABLED=1 in the /etc/default/dse file, and then use this command to start an analytics node:
     
     ```
     sudo service dse start
     ```
   - Binary installations
     
     ```
     <DSE install directory>bin/dse cassandra -t
     ```

2. Start the Pig shell:
   - Packaged installations
     
     ```
     dse pig
     ```
   - Binary installations
     
     ```
     From the installation directory:
     bin/dse pig
     ```

   The Pig grunt prompt appears, and you can now enter Pig commands.

Example: Save Pig relations from/to Cassandra

For Pig to access data in Cassandra, the target keyspace and table must already exist. Pig can save data from a Pig relation to a table in Cassandra and from a Cassandra table to a Pig relation, but it cannot create the table.

1. Start cqlsh.
2. Using cqlsh, create and use a keyspace named, for example, cql3ks.

```
cqlsh> CREATE KEYSPACE cql3ks WITH replication =
    ('class': 'SimpleStrategy', 'replication_factor': 1 );
```

```
cqlsh> USE cql3ks;
```

3. Create a two-column (a and b) Cassandra table named test and another two-column (x and y) table named moredata. Insert data into the tables.

```
cqlsh:cql3ks> CREATE TABLE test (a int PRIMARY KEY, b int);
cqlsh:cql3ks> CREATE TABLE moredata (x int PRIMARY KEY, y int);
cqlsh:cql3ks> INSERT INTO test (a,b) VALUES (1,1);
cqlsh:cql3ks> INSERT INTO test (a,b) VALUES (2,2);
cqlsh:cql3ks> INSERT INTO test (a,b) VALUES (3,3);
cqlsh:cql3ks> INSERT INTO moredata (x, y) VALUES (4,4);
cqlsh:cql3ks> INSERT INTO moredata (x, y) VALUES (5,5);
cqlsh:cql3ks> INSERT INTO moredata (x, y) VALUES (6,6);
```

4. Using Pig, add logic to load the data (4, 5, 6) from the Cassandra moredata table into a Pig relation.

```
grunt> moretestvalues= LOAD 'cql://cql3ks/moredata/' USING CqlStorage;
```

5. Add the test table data to the relation. The key column is a chararray, 'a'.

```
grunt> insertformat= FOREACH moretestvalues GENERATE
           TOTUPLE(TOTUPLE('a',x)),TOTUPLE(y);
```

6. Save the relation to the Cassandra test table.

```
grunt> STORE insertformat INTO
      'cql://cql3ks/test?output_query=UPDATE+cql3ks.test+set+b+%3D+%3F'
      USING CqlStorage;
```

Pig uses a URL-encoded prepared statement to store the relation to Cassandra. The cql:// URL is followed by an output_query, which specifies which key should be used in the command. The rest of the arguments, the "?"s, for the prepared statement are filled in by the values related to that key in Pig.

7. On the cqlsh command line, check that the test table now contains its original values plus the values from the moredata table:

```
cqlsh:cql3ks> SELECT * from test;
```

```
a | b
--+-
5 | 5
1 | 1
2 | 2
4 | 4
6 | 6
3 | 3
```

**Example: Handle a compound primary key**
1. Create a four-column (a, b, c, d) Cassandra table named compotable and another five-column (id, x, y, z, data) table named compmore. Insert data into the tables.

```cql
CREATE TABLE compotable (
  a int,
  b int,
  c text,
  d text,
  PRIMARY KEY (a,b,c)
);
```

```cql
INSERT INTO compotable (a, b, c, d)
VALUES (1,1,'One','match');
```

```cql
INSERT INTO compotable (a, b, c, d)
VALUES (2,2,'Two','match');
```

```cql
INSERT INTO compotable (a, b, c, d)
VALUES (3,3,'Three','match');
```

```cql
INSERT INTO compotable (a, b, c, d)
VALUES (4,4,'Four','match');
```

```cql
create table compmore (
  id int PRIMARY KEY,
  x int,
  y int,
  z text,
  data text
);
```

```cql
INSERT INTO compmore (id, x, y, z, data)
VALUES (1,5,6,'Fix','nomatch');
```

```cql
INSERT INTO compmore (id, x, y, z, data)
VALUES (2,6,5,'Sive','nomatch');
```

```cql
INSERT INTO compmore (id, x, y, z, data)
VALUES (3,7,7,'Seven','match');
```

```cql
INSERT INTO compmore (id, x, y, z, data)
VALUES (4,8,8,'Eight','match');
```

```cql
INSERT INTO compmore (id, x, y, z, data)
VALUES (5,9,10,'Ninen','nomatch');
```

3. Using Pig, add logic to load the data from the Cassandra compmore table to a Pig relation.

```pig
moredata = load 'cql://cql3ks/compmore' USING CqlStorage;
```

4. Add the fields from the table to a relation.

```pig
insertformat = FOREACH moredata GENERATE TOTUPLE
  (TOTUPLE('a',x),TOTUPLE('b',y),
   TOTUPLE('c',z)),TOTUPLE(data);
```

The data processed later during the MapReduce phase is formatted as follows:

`[((PartitionKey_Name,Value),(ClusteringKey_1_name,Value))...(ArgValue1,ArgValue2,ArgValue3,...)]`

4. Save the Pig relation to the Cassandra test table.

```pig
STORE insertformat INTO 'cql://cql3ks/compotable?output_query=
UPDATE%20cql3ks.compotable%20SET%20d%20%3D%20%3F' USING CqlStorage;
```

The STORE statement needs to be on one line with no space after output_query=. The cql:// URL includes a prepared statement, described later.
**Example: Explore library data**

This example uses library data from the Institute of Library and Museum Services, encoded in UTF-8 format. Download the formatted data for this example now. Two files are available to copy/paste code and run a pig script instead of stepping through this example manually. The files are located in these directories:

- Packaged installation
  /usr/share/dse-demos/pig
- Tarball installation
  <install-location>/demos/pig/cql

Using these files, you can:

- Copy/paste the commands in steps 2-3 from this document or from the library-populate-cql.txt file.
- Execute steps 7-10 automatically by running the library-cql.pig script.

1. Unzip libdata.csv.zip and give yourself permission to access the downloaded file. On the Linux command line, for example:

   chmod 777 libdata.csv

2. Create and use a keyspace called libdata.

   cqlsh:libdata> CREATE KEYSPACE libdata WITH replication =
   
   {'class': 'SimpleStrategy', 'replication_factor': 1 };

   cqlsh:libdata> USE libdata;

3. Create a table for the library data that you downloaded.

   cqlsh:libdata> CREATE TABLE libout
   
   ("STABR" TEXT, "FSCSKEY" TEXT, "FSCS_SEQ" TEXT, "LIBID" TEXT, "LIBNAME" TEXT, "ADDRESS" TEXT, "CITY" TEXT,
   "ZIP" TEXT, "ZIP4" TEXT, "CNTY" TEXT, "PHONE" TEXT, "C_OUT_TY" TEXT, "C_MSA" TEXT, "SQ_FEET" INT, "F_SQ_FT" TEXT, "L_NUM_BM" INT,
   "F_BKMOB" TEXT, "HOURS" INT, "F_HOURS" TEXT, "WKS_OPEN" INT,
   "F_WKSOPEN" TEXT, "YR_SUB" INT, "STATSTRU" INT, "STATNAME" INT,
   "STATADDR" INT, "LONGITUD" FLOAT, "LATITUDE" FLOAT, "FIPSST" INT,
   "FIPSVO" INT, "FIPSPLAC" INT, "CNTYPOP" INT, "LOCATE" TEXT, "CENTRACT" FLOAT, "CENBLOCK" INT, "CDCODE" TEXT, "MAT_CENT" TEXT,
   "MAT_TYPE" INT, "CBSA" INT, "MICROF" TEXT,
   PRIMARY KEY ("FSCSKEY", "FSCS_SEQ");

4. Import data into the libout table from the libdata.csv file that you downloaded.

   cqlsh:libdata> COPY libout
   
   "F_HOURS", "WKS_OPEN", "F_WKSOPEN", "YR_SUB", "STATSTRU", "STATNAME",
   "STATADDR", "LONGITUD", "LATITUDE", "FIPSST", "FIPSVO", "FIPSPLAC",
   "CNTYPOP", "LOCATE", "CENTRACT", "CENBLOCK", "CDCODE", "MAT_CENT",
   "MAT_TYPE", "CBSA", "MICROF") FROM 'libdata.csv' WITH HEADER=TRUE;

In the FROM clause of the **COPY command**, use the path to libdata.csv in your environment.
5. Check that the libout table contains the data you copied from the downloaded file.

   cqlsh:libdata> SELECT count(*) FROM libdata.libout LIMIT 20000;

   count
       -----
        17598

6. Create a table to hold results of Pig relations.

   cqlsh:libdata> CREATE TABLE libsqft (year INT, state TEXT, sqft BIGINT,
            PRIMARY KEY (year, state));

7. Using Pig, add a plan to load the data from the Cassandra libout table to a Pig relation.

   grunt> libdata = LOAD 'cql://libdata/libout' USING CqlStorage();

8. Add logic to remove data about outlet types other than books-by-mail (BM). The C_OUT_TY column uses BM and
   other abbreviations to identify these library outlet types:
      
   - CE–Central Library
   - BR–Branch Library
   - BS–Bookmobile(s)
   - BM–Books-by-Mail Only

   grunt> book_by_mail = FILTER libdata BY C_OUT_TY == 'BM';
   grunt> DUMP book_by_mail;

9. Add logic to filter the data based on library buildings and sum square footage of the buildings. Group data by state.
   The STABR column contains the state codes.

   grunt> libdata_buildings = FILTER libdata BY SQ_FEET > 0;
   grunt> state_flat = FOREACH libdata_buildings GENERATE
            STABR AS State, SQ_FEET AS SquareFeet;
   grunt> state_grouped = GROUP state_flat BY State;
   grunt> state_footage = FOREACH state_grouped GENERATE
            group as State, SUM(state_flat.SquareFeet)
            AS TotalFeet:int;
   grunt> DUMP state_footage;

   The MapReduce job completes successfully and the output shows the square footage of the buildings.

10. Add logic to filter the data by year, state, and building size, and save the relation to Cassandra using the cql://
    URL. The URL includes a prepared statement, described later.

    grunt> insert_format= FOREACH state_footage GENERATE
            TOTUPLE(TOTUPLE('year',2011), TOTUPLE('state',State)), TOTUPLE(TotalFeet);
    grunt> STORE insert_format INTO 'cql://libdata/libsqft?output_query=UPDATE%20libdata.libsqft%20SET%20sqft%20=%20%3F' USING CqlStorage;

    When the MapReduce job completes, a message appears that the records were written successfully.
11. In CQL, query the libsqft table to see the Pig results now stored in Cassandra.

```sql
SELECT * FROM libdata.libsqft;
```

<table>
<thead>
<tr>
<th>year</th>
<th>state</th>
<th>sqft</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>AK</td>
<td>570178</td>
</tr>
<tr>
<td>2011</td>
<td>AL</td>
<td>2792246</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2011</td>
<td>WV</td>
<td>1075356</td>
</tr>
<tr>
<td>2011</td>
<td>WY</td>
<td>724821</td>
</tr>
</tbody>
</table>

**Data access using Pig**

DataStax Enterprise includes a custom storage handler for Cassandra that you use to execute Pig programs directly on data stored in Cassandra. The DataStax Enterprise Pig driver uses the Cassandra File System (CassandraFS) instead of the Hadoop distributed file system (HDFS). Apache Cassandra, on the other hand, includes a Pig driver that uses the Hadoop Distributed File System (HDFS).

In DataStax Enterprise 3.1.2 and later, use one of these storage handlers and urls to transform Cassandra data using Pig:

<table>
<thead>
<tr>
<th>Table Format</th>
<th>Storage Handler</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQL</td>
<td>CqlStorage()</td>
<td>cql://</td>
</tr>
<tr>
<td>storage engine</td>
<td>CassandraStorage()</td>
<td>cassandra://</td>
</tr>
</tbody>
</table>

**Working with legacy Cassandra tables**

You use the CassandraStorage() handler and cfs:// url instead of CqlStorage() and cql:// to work with Cassandra tables that are in the storage engine (CLI/Thrift) format in Pig. Legacy tables are created using the WITH COMPACT STORAGE directive in CQL or are created using Thrift or the CLI.

**Using the CqlStorage handler**

To use data in CQL tables created by any version of Cassandra, use the CqlStorage() handler and cql:// url. To access data in the CassandraFS, the target keyspace and table must already exist. Data in a Pig relation can be stored in a Cassandra table, but Pig will not create the table.

DataStax Enterprise supports these Pig data types:

- int
- long
- float
- double
- boolean
- chararray

The Pig LOAD statement pulls Cassandra data into a Pig relation through the storage handler. The format of the Pig LOAD statement is:

```
<pig_relation_name> = LOAD 'cql://<keyspace>/<table>'
    USING CqlStorage();
```

The Pig demo examples include using the LOAD command.
**LOAD schema**

The LOAD Schema is:

(colname:colvalue, colname:colvalue, ...) where each colvalue is referenced by the Cassandra column name.

**Saving a Pig relation to Cassandra**

The Pig STORE command pushes data from a Pig relation to Cassandra through the CqlStorage handler:

STORE <relation_name> INTO 'cql://<keyspace>/<column_family>?<prepared statement>'
USING CqlStorage();

**URL format**

The url format for CqlStorage is:

cql://[username:password@]<keyspace>/<columnfamily>[?[
  [page_size=<size>]
  [columns=<col1,col2>]
  [output_query=<prepared_statement_query>]
  [where_clause=<clause>]
  [split_size=<size>]
  [partitioner=<partitioner>]
  [use_secondary=true|false]]

Where:

- page_size -- the number of rows per page
- columns -- the select columns of CQL query
- where_clause -- the where clause on the index columns, which needs url encoding
- split_size -- number of rows per split
- partitioner -- Cassandra partitioner
- use_secondary -- to enable pig filter partition push down
- output_query -- the CQL query for writing in a prepared statement format

**Store schema**

The input schema for Store is:

(value, value, value) where each value schema has the name of the column and value of the column value.

The output schema for Store is:

(((name, value), (name, value)), (value ... value), (value ... value)) where the first tuple is the map of partition key and clustering columns. The rest of the tuples are the list of bound values for the output in a prepared CQL query.

**Creating a URL-encoded prepared statement**

The Pig demo examples show the steps required for setting up a prepared CQL query:
1. Format the data for the query.

2. Construct the prepared output_query portion of the cql:// URL, and execute the query. First, format the Cassandra data for the prepared query, and then construct and execute the query.

**Format the data**

The example of saving Pig relations from/to Cassandra shows the output schema: the name of the test table primary key 'a', represented as a chararray in the relation is paired with a value in the moredata table. In this case, the key for test table is only a partitioning key, and only a single tuple is needed.

The Pig statement to add (moredata) fields to the relation is:

```
grunt> insertformat= FOREACH morevalues GENERATE TOTUPLE(TOTUPLE('a',x)),TOTUPLE(y);
```

The example of exploring library data works with more complicated data, a partition key and clustering column:

```
grunt> insertformat = FOREACH moredata GENERATE TOTUPLE(TOTUPLE('a',x),TOTUPLE('b',y),TOTUPLE('c',z)),TOTUPLE(data);
```

**Construct the prepared query**

The output query portion of the cql:// URL is the prepared statement. The prepared statement must be url-encoded to make special characters readable by Pig.

The example of saving Pig relations from/to Cassandra shows how to construct a prepared query:

```
'cql://cql3ks/test?output_query=UPDATE+cql3ks.test+set+b+%3D+%3F'
```

The key values of the test table are automatically transformed into the 'WHERE (key) =' clause to form the output_query portion of a prepared statement.

**Execute the query**

To update the test table using the values in the moredata table (4-6), the prepared statement is executed using these WHERE clauses when the MapReduce job runs:

```
... WHERE a = 5
... WHERE a = 4
... WHERE a = 6
```

This output_query in Pig statement forms the '...' url-encoded portion of the prepared statement:

```
grunt> STORE insertformat INTO
    'cql://cql3ks/test?output_query=UPDATE+cql3ks.test+set+b+%3D+%3F'
    USING CqlStorage;
```

Unencoded the UPDATE statement is:

```
UPDATE cql3ks.test set b = ?
```

The prepared statement represents these queries:

```
UPDATE cql3ks.test set b = 5 WHERE a = 5;
UPDATE cql3ks.test set b = 4 WHERE a = 4;
UPDATE cql3ks.test set b = 6 WHERE a = 6;
```

**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 tables. 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 table*.

Finally, glance at the extent of the Sqoop commands listed in the *online help* and the Cassandra additions.

**Getting information about the sqoop command**

Use the help option of the sqoop import command to get online help on Sqoop command line options. For example, on the Mac:

```
cd <install_location>/bin
./dse sqoop import --help
```

**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 or later. 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
Step-by-step procedure

To run the Sqoop demo on a single node on a Mac, for example, follow these steps.

1. Install MySQL and download the JDBC driver for MySQL from the MySQL site. This example uses mysql-connector-java-5.0.8-bin.jar.

2. Copy the JDBC driver for MySQL to the sqoop/lib directory.
   - RHEL or Debian installations
     /usr/share/dse/sqoop/lib
   - Tar distribution, such as Mac
     <install_location>/resources/sqoop/lib

3. On the command line, start the MySQL daemon. For example:
   
   sudo ./mysqld_safe --user=mysql

4. Start MySQL and create the demo database:

   sudo ./mysql

   mysql> CREATE DATABASE npa_nxx_demo;

5. 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;

6. 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
7. 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

8. 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.

9. 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.

### Checking imported data

DataStax Enterprise provides a SQL-like language called CQL that is similar to the DDL, DML, and SELECT syntax in SQL. CQL lessens the learning curve for those coming from RDBMS systems. You can use familiar syntax for all object creation and data access operations. You can use the Cassandra Query Language (CQL) utility to confirm the success of the Sqoop import. Alternatively, you can use the Cassandra Command Line Interface (CLI) to perform the same type of queries.

### Using CQL to check imported data

To check the data in the example of importing data into a table, you can use CQL. For example, to check the number of rows imported into the table:

```
./cqlsh -2

USE newKS;

SELECT count(*) FROM npa_nxx_cf limit 200000;
```

The number of records appears.
SELECT * FROM npa_nxx_cf where key IN (626794,212524,512538);

Records appear for Pasadena, New York, and Austin.

<table>
<thead>
<tr>
<th>KEY</th>
<th>city</th>
<th>lat</th>
<th>linetype</th>
<th>lon</th>
<th>npa</th>
<th>nxx</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>626794</td>
<td>Pasadena</td>
<td>34.17</td>
<td>L</td>
<td>118.13</td>
<td>626</td>
<td>794</td>
<td>CA</td>
</tr>
<tr>
<td>212524</td>
<td>New York</td>
<td>40.71</td>
<td>L</td>
<td>074.01</td>
<td>212</td>
<td>524</td>
<td>NY</td>
</tr>
<tr>
<td>512538</td>
<td>Austin</td>
<td>30.27</td>
<td>L</td>
<td>097.74</td>
<td>512</td>
<td>538</td>
<td>TX</td>
</tr>
</tbody>
</table>

Validating import results in a cluster

Use this command to view the results in the Cassandra File System:

```
./dse hadoop fs -ls /npa_nxx
```

Depending on the number of DataStax Enterprise analytics nodes and task tracker configuration, the output shows a number of files in the directory, part-m-0000n, where 'n' ranges from 0 to the number of tasks that were executed as part of the Hadoop job.

The contents of these files can be viewed using this command:

```
./dse hadoop fs -cat /npa_nxx/part-m-00000
```

By varying the number of tasks (the 00000), the output looks something like this:

```
361991,361,991,27.73,097.40,L,TX,Corpus Christi
361992,361,992,27.73,097.40,L,TX,Corpus Christi
361993,361,993,27.73,097.40,L,TX,Corpus Christi
361994,361,994,27.73,097.40,L,TX,Corpus Christi
361998,361,998,27.79,097.90,L,TX,Agua Dulce
361999,361,999,27.80,097.40,W,TX,Padre Island National Seashore
```

As shown in the output, the CSV file format that Sqoop requires does not include optional spaces in the delimiter.

Migrating Data to a Cassandra table

Suppose you have a MySQL data source, npa_nxx_demo, and you want to migrate its table and data to a Cassandra table. You execute the dse command in the bin directory of the DataStax Enterprise installation, using cassandra options. For example, if you already ran the Sqoop demo, delete the npa_nxx.java file in $DSE_HOME/bin, then run this command:

```
sudo ./dse sqoop import --connect jdbc:mysql://127.0.0.1/npa_nxx_demo \
  --username root \ 
  --cassandra-keyspace newKS \ 
  --cassandra-column-family npa_nxx_cf \ 
  --cassandra-row-key npa_nxx_key \ 
  --cassandra-thrift-host 127.0.0.1 \ 
  --cassandra-create-schema
```

Cassandra options to the import command

First, the dse command passes the external data source parameters:

- IP address of the server, the connection string for the external data source
- The database username
- Optional password (not shown in the example)
The name of the external data source table to migrate

Next, the dse command passes the Cassandra parameters:

- Name of the new Cassandra keyspace to use
- Name of the new table object
- Primary key of the table
- IP address of the Cassandra node to connect to
- The cassandra-create-schema parameter, which tells Sqoop to create my a new keyspace. You can also use existing keyspaces.

DSE interprets these command options as follows:

```
--cassandra-row-key <key>
```

The values of the key column of the source table become the partition keys in Cassandra.

**Note**

No detection of duplicates is performed, so using a unique key in the source table is strongly recommended to prevent overwriting and losing data.

```
--cassandra-thrift-host <cassandra-host(s)>
```

A comma-separated list of IP addresses identifies Cassandra nodes.

```
--cassandra-create-schema
```

An option that creates a Cassandra keyspace and table during the import operation.

**Note**

The table is created with no column metadata and all data is imported as strings.

**Usage notes**

- Using this option when the keyspace already exists is valid.
- Using this option when the table already exists causes an error that aborts the import operation.
- Using this option when the keyspace or table, or both, do not exist is required; otherwise, the import operation is aborted.
- Using this option creates a table with no column metadata and all data is imported as strings. Do not use this option if you want column metadata.

**Note**

To generate column metadata for imported data, create a table, setting column values and names as string types, and then import data.

**About the generated Sqoop JAR file**

After running the `dse sqoop import` command, a Java class appears in the DSE installation bin directory. This file, by default named `npa_nxx.java` after the demo table, can encapsulate one row of the imported data. You can specify the name of this JAR file, the output directory, and the class package using Sqoop command line options. For more information, see [Sqoop documentation](#).
Snippets of the output and end result of the import operation look like this:

INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
INFO tool.CodeGenTool: Beginning code generation
INFO manager.SqlManager: Executing SQL statement:
```sql
SELECT t.*
FROM `npa_nxx` AS t
LIMIT 1
```
INFO orm.CompilationManager: HADOOP_HOME is /Users/robin/dev/dse-2.2/resources/hadoop/bin/..
INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-robin/compile/2e2b8b85fba83ccf1f52a8ea77c3b12f/npa_nxx.jar

INFO mapreduce.ImportJobBase: Beginning import of npa_nxx

INFO config.DseConfig: Load of settings is done.

INFO mapreduce.ImportJobBase: Transferred 0 bytes in 10.724 seconds (0 bytes/sec)
INFO mapreduce.ImportJobBase: Retrieved 105291 records

In this version of Sqoop, there is a Map, but no Reduce, phase.

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 table 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
Cassandra Log4j appender solutions

- **FATAL** - severe errors causing premature termination
- **ERROR** - other runtime errors or unexpected conditions
- **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:

```java
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 table**

The Cassandra Appender provides the capability to store log4j messages in a Cassandra table.

**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:

```properties
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 table `log_entries` in the `Logging` keyspace. The definition of this table is as follows:

```
cqlsh:Logging> DESCRIBE TABLE log_entries;
CREATE TABLE 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=''
    comparator=text
```

Cassandra Log4j appender solutions
**Cassandra Log4j appender solutions**

```java
row_cache_provider='ConcurrentLinkedHashCacheProvider' AND
key_cache_size=200000.00 AND
row_cache_size=0.00 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` table appear as follows (queried using cqlsh):

```
KEY,eea1256e-db24-4cef-800b-843b3b2fb72c | app_start_time,1328894454774 | level,WARN | log_timestamp,13288944555391 | 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,13288944555064 | 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,13288944555392 | logger_class_name,org.apache.log4j.Category | message, Test exception. | thread_name,main | throwable_str_rep,java.io.IOException: Danger Will Robinson, Danger!
at 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 table.

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 DSE Search/Solr on a single node.
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**
   
   cd /usr/share/dse-demos/log_search

   **Tar distribution, such as Mac**
   
   cd $DSE_HOME/demos/log_search

2. Open another shell window or tab and add the schema:
   
   ./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:
   
   ./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

**Getting started with Solr in DataStax Enterprise**
DataStax Enterprise supports Open Source Solr (OSS) tools and APIs, simplifying migration from Solr to DataStax Enterprise. DataStax Enterprise Search 3.1 and later is built on top of Solr 4.3. Before starting a DSE Search/Solr node on a production cluster or data center, it is important to disable virtual nodes. You can skip this step to run the Solr getting started tutorial.

**Disabling virtual nodes**

DataStax recommends using virtual nodes only on data centers running Cassandra real-time workloads. You should disable virtual nodes on data centers running either Hadoop or Solr workloads.

To disable virtual nodes:

1. In the cassandra.yaml file, set num_tokens to 1.

   num_tokens = 1

2. Uncomment the initial_token property and set it to 1 or to the value of a generated token for a multi-node cluster.

**Introduction to Solr**

The Apache Lucene project, Solr features 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. DataStax extends Solr’s capabilities that are compared 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, provided the Solr nodes are in a separate data center, 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.

**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
Benefits of using Solr in DataStax Enterprise

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--add 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 tables created with CQL/CLI/Thrift

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
- Cassandra composite columns, Solr fields must be strings.

Defining key Solr terms

In a distributed environment, such as DataStax Enterprise and Cassandra, the 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 tables. Each table 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 table. 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:
With Cassandra replication, a Cassandra node or Solr core contains more than one partition (shard) of table (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 table or collection.

### Installing Solr nodes

To install a Solr node, use the same installation procedure as you use to install any other type of node. To use real-time (Cassandra), analytics (Hadoop), or search (Solr) nodes in the same cluster, segregate the different nodes into separate data centers. Using the default DSESimpleSnitch automatically puts all the Solr nodes in the same data center. Use OpsCenter Enterprise to rebalance the cluster when you add a node to the cluster.

#### Starting and stopping a Solr node

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

**Tarball installation**

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

```bash
bin/dse cassandra -s
```

The Solr node starts up.

From the install directory, use this command to stop the Solr node:

```bash
bin/dse cassandra-stop
```

**Packaged installation**

1. Enable Solr mode by setting this option in `/etc/default/dse`: `SOLR_ENABLED=1`
2. Start the dse service `<start-dse>` using this command:

   ```bash
   sudo service dse start
   ```

   The Solr node starts up.

You stop a Solr node using this command:

```bash
sudo service dse stop
```

### Solr getting started tutorial

Setting up Cassandra and Solr for this tutorial involves the same basic steps as setting up a typical DSE Search/Solr application:

- Create a Cassandra table.
Setup

This setup assumes you started DataStax Enterprise 3.1 in DSE Search/Solr mode and downloaded the sample data.

1. Unzip the files you downloaded in the DataStax Enterprise installation home directory. The solr_tutorial directory is created in the installation directory that contains the following files.
   - The CSV (comma separated value) data, nhanes52.csv
   - Cassandra table definition, create_nhanes.cql
   - The copy command, copy_nhanes.cql
   - The Solr schema, schema.xml

   You can take a look at these files by using your favorite editor.

2. Copy the solrconfig.xml from the DataStax Enterprise 3.1 <install-location>/demos/wikipedia directory (tarball installations) or /usr/share/dse-demos (packaged installations) to the solr_tutorial directory.

Create a Cassandra table

1. Start cqlsh, and create a keyspace. Use the keyspace.

   ```
   cqlsh> CREATE KEYSPACE nhanes_ks WITH REPLICACTION = 
        { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
   cqlsh> USE nhanes_ks;
   ```

2. Copy the CQL table definition from the downloaded create_nhanes.cql file, and paste it on the cqlsh command line.

   This action creates the nhanes table in the nhanes_ks keyspace. The table uses the WITH COMPACT STORAGE directive.

Import data

1. Copy the cqlsh COPY command from the downloaded copy_nhanes.cql file.

2. Paste the COPY command on the cqlsh command line, change the FROM clause to match the path to /solr_tutorial/nhanes52.csv in your environment, and then run the command.

   This action imports the data from the CSV file into the nhanes table in Cassandra.

In a production environment, you would likely use a hefty tool, such as the Cassandra bulk loader or sqoop for importing data. An alternative to importing the data into the pre-existing Cassandra table is to import the data into Solr and let DataStax Enterprise create the table after search indexing.

Create a search index

On the command line in the solr_tutorial directory, upload the solrconfig.xml and schema.xml to Solr, and create the Solr core named after the Cassandra table and keyspace, nhanes_ks.nhanes.
Now, the searching can begin.

**Exploring the Solr Admin**

After creating the Solr core, you can check that the Solr index is working by using the browser-based Solr Admin:

http://localhost:8983/solr/

To explore the Solr Admin:

1. Click Core Admin. Unless you loaded other cores, the path to the default core, nhanes_ks.nhanes, appears.

   At the top of the Solr Admin console, the Reload, Reindex and Full Reindex buttons perform functions that correspond to **RELOAD command options**. If you modify the schema.xml or solrconfig.xml, you use these controls to re-index the data.

2. Check that the numDocs value is 20,050. The number of Solr documents corresponds to the number of rows in the CSV data and nhanes table you created in Cassandra.
3. In Core Selector, select the name of the core, nhanes_ks.nhanes. Selecting the name of the core brings up additional items, such as Query, in the vertical navigation bar.

You can learn more about the Solr Admin from the Overview of the Solr Admin UI.

Running a simple search
To search the database, experienced users run Solr HTTP API queries in a browser or on the command line with the curl utility. You can also use the Solr Admin query form. Using the query form has some advantages for those new to Solr. The form contains text entry boxes for constructing a query and can provide query debugging information.

To get started searching the nhanes database:
1. In the Solr Admin, click Query.
   
   A query form appears.

2. Notice that the form has a number of query defaults set up, including the select URL in Request-Handler and *:* in the main query parameter entry box--q.
3. Scroll down the form and click Execute.
   
The defaults select all the fields in all the documents, starting with row 0 and ending at row 10. The wt parameter specifies the output format, xml by default.

---

**Running a faceted search**

Now, run a complex query and include facet parameters in the request.

1. In the Solr Admin query form, specify a family size of 9 in the main query parameter text entry box—q:
   
   family_size:9

2. In sort, specify sorting by age in ascending order, youngest to oldest:
   
   age asc

3. In fl (filter list), specify returning only age and family size in results:
   
   age family_size

Results from the main query will include only data about families of 9.
4. Click the facet option.
   Text entry boxes for entering facet parameter values appear.

5. In facet.field, type this value:
   age

   The number of people in each age group will appear toward the end of the query results.
6. Click Execute.

The numfound value shows that 186 families having nine members were found. The query results include only results from the fields in the filter list, age and family_size.
7. Scroll to the end of the query form to see the facet results. The facet results show 11 people of age 17, 10 of age 34, and so on.

You can learn more about faceting from Solr documentation.

**Using the Solr HTTP API**

For serious searching, use the Solr HTTP API. The Solr Admin query form is limited, but useful for learning about Solr, and can even help you get started using the Solr HTTP API. The form shows the queries in Solr HTTP format at the top of the form. After looking at a few URLs, you can try constructing queries in Solr HTTP format.

To get started using the Solr HTTP API:
1. Scroll to the top of the form, and click the greyed out URL.

A page of output independent of the query form appears that you can use to examine and change the URL. The URL looks like this:

http://localhost:8983/solr/nhanes_ks.nhanes/select?
q=family_size%3A9&sort=age+asc&fl=age+family_size
&wt=xml&indent=true&facet=true&facet.field=age

2. In the URL in the address bar, make these changes:

FROM:
q=family_size%3A9
&fl=age+family_size

TO:
q=age:[20+TO+40]
&fl=age+family_size+num_smokers

The modified URL looks like this:

http://localhost:8983/solr/nhanes_ks.nhanes/select?
q=age:[20+TO+40]&sort=age+asc&fl=age+family_size+num_smokers
&wt=xml&indent=true&facet=true&facet.field=age

In the Solr Admin query form, you can use spaces in the range [20 TO 40], but in the URL, you need to use URL encoding for spaces and special characters. For example, use + or %20 instead of a space, [20+TO+40].
3. Use the modified URL to execute the query. Move to the end of the URL, and press ENTER.

The number of hits increases from 186 to 7759. Results show the number of smokers and family size of families whose members are 20-40 years old. Facets show how many people fell into the various age groups.

```xml
<doc>
  <int name="age">20</int>
  <int name="family_size">4</int>
  <int name="num_smokers">1</int>
</doc>
</result>

4. Experiment with different Solr HTTP API URLs by reading documentation on the internet and trying different queries using this sample database.

This tutorial introduced you to DSE Search/Solr basic setup and searching. Next, delve into DataStax Enterprise documentation and the recommended Solr documentation.

**Configuring Solr**

DataStax Enterprise 3.1 includes improvements in the mapping of Solr types to Cassandra validators. If you are running applications from DataStax Enterprise 3.0.x and earlier, configure legacy mapping of Solr types.

**Latest mapping of Solr types**

This table shows the DataStax Enterprise 3.1 mapping of Solr types to Cassandra validators.

<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</td>
</tr>
<tr>
<td>Field Type</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>--------------------------------------------------</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>BoolField</td>
<td>BooleanType</td>
<td>True (1, t, or T) or False (not 1, t, or T)</td>
</tr>
<tr>
<td>ByteField</td>
<td>Int32Type</td>
<td>Contains an 8-bit number value.</td>
</tr>
<tr>
<td>DateField</td>
<td>DateType</td>
<td>Point in time with millisecond precision</td>
</tr>
<tr>
<td>DoubleField</td>
<td>DoubleType</td>
<td>Double (64-bit IEEE floating point)</td>
</tr>
<tr>
<td>ExternalFileField</td>
<td>UTF8Type</td>
<td>Values from disk file</td>
</tr>
<tr>
<td>FloatField</td>
<td>FloatType</td>
<td>32-bit IEEE floating point</td>
</tr>
<tr>
<td>IntField</td>
<td>Int32Type</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>LongField</td>
<td>LongType</td>
<td>Long integer (64-bit signed integer)</td>
</tr>
<tr>
<td>RandomSortField</td>
<td>UTF8Type</td>
<td>Dynamic field in random order</td>
</tr>
<tr>
<td>ShortField</td>
<td>Int32Type</td>
<td>Short integer</td>
</tr>
<tr>
<td>SortableDoubleField</td>
<td>DoubleType</td>
<td>Numerically sorted doubles</td>
</tr>
<tr>
<td>SortableFloatField</td>
<td>FloatType</td>
<td>Numerically sorted floating point</td>
</tr>
<tr>
<td>SortableIntField</td>
<td>Int32Type</td>
<td>Numerically sorted integer</td>
</tr>
<tr>
<td>SortableLongField</td>
<td>LongType</td>
<td>Numerically sorted long integer</td>
</tr>
<tr>
<td>StrField</td>
<td>UTF8Type</td>
<td>String (UTF-8 encoded string or Unicode)</td>
</tr>
<tr>
<td>TextField</td>
<td>UTF8Type</td>
<td>Text, usually multiple words or tokens</td>
</tr>
<tr>
<td>TrieDateField</td>
<td>DateType</td>
<td>Date field for Lucene TrieRange processing</td>
</tr>
<tr>
<td>TrieDoubleField</td>
<td>DoubleType</td>
<td>Double field for Lucene TrieRange processing</td>
</tr>
<tr>
<td>TrieField</td>
<td>see description</td>
<td>Same as any Trie field type</td>
</tr>
<tr>
<td>TrieFloatField</td>
<td>FloatType</td>
<td>Floating point field for Lucene TrieRange processing</td>
</tr>
<tr>
<td>TrieIntField</td>
<td>Int32Type</td>
<td>Int field for Lucene TrieRange processing</td>
</tr>
<tr>
<td>TrieLongField</td>
<td>LongType</td>
<td>Long field for Lucene TrieRange processing</td>
</tr>
<tr>
<td>UUIDField</td>
<td>UUIDType</td>
<td>Universally Unique Identifier (UUID)</td>
</tr>
<tr>
<td>LatLonType</td>
<td>UTF8Type</td>
<td>Latitude/Longitude 2-D point, latitude first</td>
</tr>
<tr>
<td>PointType</td>
<td>UTF8Type</td>
<td>Arbitrary n-dimensional point for spatial search</td>
</tr>
<tr>
<td>GeoHashField</td>
<td>UTF8Type</td>
<td>Geohash lat/lon pair represented as a string</td>
</tr>
</tbody>
</table>

For efficiency in operations such as range queries, using Trie types is recommended. Notes about some of the types are:

- **BCD**
  A relatively inefficient encoding that offers the benefits of quick decimal calculations and quick conversion to a string.

- **SortableDoubleField/DoubleType**
  If you use the plain types (DoubleField, IntField, and so on) sorting will be lexicographical instead of numeric.

- **TrieField**
  Used with a type attribute and value: integer, long, float, double, date.
Legacy mapping of Solr Types

DataStax Enterprise 3.0.x and earlier use the legacy type mapping by default.

<table>
<thead>
<tr>
<th>Solr Type</th>
<th>Cassandra Validator</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>StrField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>LongField</td>
<td>LongType</td>
</tr>
<tr>
<td>IntField</td>
<td>Int32Type</td>
</tr>
<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>

Configuring Solr type mapping

By default, DataStax Enterprise 3.1 enables the latest type mapping whereas DataStax Enterprise 3.0.x enables legacy type mapping. In DataStax Enterprise 3.1, to use the legacy type mapping or to revert to the new Solr type mapping, configure dseTypeMappingVersion in the Solr config:

Set the value to 1 to enable the latest type mapping:

```xml
<dseTypeMappingVersion>1</dseTypeMappingVersion>
```

Set the value to 0 to enable the legacy type mapping:

```xml
<dseTypeMappingVersion>0</dseTypeMappingVersion>
```

Switching between the two versions after adding data is not supported. Attempting to load a solrconfig with a different dseTypeMappingVersion configuration and reloading the core causes an error.

Changing a Solr type mapping

Changing a Solr type mapping is rarely if ever done and is not recommended; however, for particular circumstances, DataStax Enterprise 3.1 includes the capability to convert new Solr type mappings, such as the Solr LongField to another type, such as TrieLongField. You configure the dseTypeMappingVersion using the force option.

The Cassandra internal validation classes of the types you are converting to and from must be compatible and the conversion must be to/from valid types. For example, converting a legacy Trie type to a new Trie type is invalid. The output of the CLI command, DESCRIBE keyspace_name, shows the validation classes assigned to columns.
For example, the org.apache.cassandra.db.marshal.LongType column validation class is mapped to solr.LongType. You can force this column to be of the TrieLongField type by using force="true" in the solrconfig.xml, and then running a core reload with re-indexing.

```xml
<dseTypeMappingVersion force="true">1</dseTypeMappingVersion>
```

Use this option only if you are an expert and have confirmed that the Cassandra internal validation classes of the types involved in the conversion are compatible.

**Configuring the schema**

A Solr schema defines the relationship between data in a table 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.

**Sample schema**

The schema.xml for the example of using DSE Search/Solr represents a typical schema. It specifies a tokenizer that determines the parsing of the example text. The set of fields specifies the data that Solr indexes and stores. The id, body, name, and title fields are indexed.

```xml
<schema name="my_search_demo" version="1.1">
  <types>
    <fieldType name="string" class="solr.StrField"/>
    <fieldType name="text" class="solr.TextField">
      <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
      </analyzer>
    </fieldType>
  </types>
  <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>
  <defaultSearchField>body</defaultSearchField>
  <uniqueKey>id</uniqueKey>
</schema>
```

This schema.xml meets the requirement to have a unique key and no duplicate rows. The unique key maps to the Cassandra partition key and is necessary for DataStax Enterprise 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 be stored in Cassandra. The indexed="true" gives the field the potential to show up in search results.

**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.

**Configuring the Solr library path**

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
Creating a Solr 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 table for searching Cassandra data.

Indexing guidelines

You can avoid problems by following these recommendations:

1. When users post schema or configuration files simultaneously, schema disagreements can occur. This causes Solr errors.
   Do not make schema changes on hot production systems

2. While Solr is indexing, do not truncate a table that Solr has indexed.
   You can check indexing status using the Solr Admin.

Uploading the schema and configuration

This procedure describes how to create a Solr index by posting the solrconfig.xml and schema.xml and creating the Solr core. In Example Using CQL you can follow similar steps to actually create a Solr index and insert data into Solr and Cassandra.

1. Post the configuration file using the cURL utility:
   ```bash
   --data-binary @solrconfig.xml -H "Content-type:text/xml; charset=utf-8"
   ```

2. Post the schema file:
   ```bash
   --data-binary @schema.xml -H "Content-type:text/xml; charset=utf-8"
   ```

Creating a Solr core

You cannot create a Solr core unless you first upload the schema and configuration files.

Use this command create a Solr core.

```bash
curl "http://localhost:8983/solr/admin/cores?action=CREATE&name=<keyspace.table>"
```

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.
Using DSE Search/Solr

When you update a table using CQL or CLI, the Solr document is updated. When you update a Solr document using the Solr API, the table 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 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 commit log replaces the Solr updatelog, which is not supported in DSE Search/Solr. Consequently, atomic updates and real-time get that require the updateLog are not supported. In Cassandra, a write is atomic at the row-level, meaning inserting or updating columns in a row is treated as one write operation.

The Solr index update operation is similar to a Cassandra 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.

Inserting, deleting, and searching data

The show you how to perform basic operations:

- Inserting, deleting, and searching data
- Deleting Solr data
- Using copy fields
- Using docValues and copy fields for faceting

You can insert data into Solr in several ways:

- Executing CQL statements on the command line or from a client. Use the syntax described in the DataStax Apache Cassandra documentation and shown in the following example.
- Using the Solr HTTP API update command
- Using any Thrift API, such as Pycassa or Hector
These examples include the first two methods, using CQL and the Solr HTTP API.

**Example Using CQL**

1. If you did not already do this, create a directory named solr_tutorial. Copy the schema.xml and solrconfig.xml from the wikipedia demos directory to the solr_tutorial directory.

2. 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:

   ```bash
   ./cqlsh
   ```

3. Create a keyspace and a table, and then, insert some data for DSE Search to index. You need use the WITH COMPACT STORAGE directive when defining the table.

   ```sql
   CREATE KEYSPACE mykeyspace
       WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
   USE mykeyspace;
   CREATE TABLE mysolr
       ( id text PRIMARY KEY,
       name text,
       title text,
       body text
       ) WITH COMPACT STORAGE;
   INSERT INTO mysolr (id, name, title, body) VALUES ('123', 'Christopher Morley', 'Life', 'Life is a foreign language; all men mispronounce it.');
   INSERT INTO mysolr (id, 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 (id, 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 (id, 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.');
   ```

4. Change the schema.xml file to contain the schema shown in the [Sample schema section](#).

5. On the command line in the solr_tutorial directory, post the configuration file using the cURL utility.

   ```bash
   --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8'
   ```

6. Post the schema file:

   ```bash
   --data-binary @schema.xml -H 'Content-type:text/xml; charset=utf-8'
   ```

8. Create a Solr core.

   ```bash
   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*.

\[\text{http://localhost:8983/solr/mykeyspace.mysolr/}
\text{select?q=title%3ASucc*\&wt=json\&indent=on\&omitHeader=on}\

The response is:

\[
\{
  "response":{
    "numFound":2,"start":0,"docs":[
      {
        "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"},
      {
        "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"}
    ]
  }
\}

**Example using the Solr HTTP 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:

\[\text{curl http://<host>:<port>/solr/<keyspace>.<table>/update?}
\text{replacefields=false -H 'Content-type: application/json' -d}
\text{'}<\text{json string}>\]

Using this format to insert data into the Cassandra table and Solr index created in the previous example, the curl command is:

\text{ -H 'Content-type: application/json' -d}
\text{'}[\{"id":"130", "body":"Life is a beach.", "name":"unknown", "title":"Life"}]\]

The Solr convention is to use curl for issuing update commands instead of using a browser. You do not have to post a commit command in the update command as you do in OSS, and doing so is ineffective.

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.

**Warning about using the optimize command**

Do not include the optimize command in URLs to update Solr data. This warning appears in the system log when you use the optimize:

\[\text{WARN [http-8983-2] 2013-03-26 14:33:04,450 CassandraDirectUpdateHandler2.java (line 697)}
\text{Calling commit with optimize is not recommended.}\]

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.

**Deleting Solr data**

119
To delete a Cassandra table and its data, including the data indexed in Solr, from a Solr node drop the table 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 table named solr 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:
     ```shell
     ls /usr/local/var/lib/dse5/data/solr.data/wiki.solr/index/
     ```
   - Tarball install:
     ```shell
     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 table named solr.
   ```sql
   USE wiki;
   DROP TABLE solr;
   ```

3. Exit cqlsh and check that the files have been deleted on the file system. For example:
   ```shell
   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
   ```

### Using copy fields

The way DSE Search/Solr handles copy fields depends on the value of the stored attribute.

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.

### Using a copy field and multivalue field

When you use copy fields to copy multiple values into a field, CQL comes in handy because you do not need to format the data in json, for example, when you insert it. Using the Solr HTTP API update command, the data must be formatted.

Use the CQL BATCH command to insert column values in a single CQL statement to prevent overwriting. This process is consistent with Solr HTTP APIs, where all copied fields need to be present in the inserted document. You need to use BATCH to insert the column values whether or not the values are stored in Cassandra.
Using docValues and copy fields for faceting

Using docValues can improve performance of faceting, grouping, filtering, sorting, and other operations described on the Solr Wiki.

For faceting to use docValues, the schema needs to specify multiValued=“true” even if the field is a single-value facet field. The field also needs to include docValues=“true”. You also need to use a field type that supports being counted by Solr. The text type, which tokenizes values, cannot be used, but the string type works fine.

Example of using copy fields and docValues

This example uses copy fields to copy various aliases, such as a twitter name and email alias, to a multivalue field. You can then query the multivalue field using any alias as the term to get the other aliases in the same row or rows as the term. This example also uses docValues

1. If you did not already do this, create a directory named solr_tutorial. Copy the schema.xml and solrconfig.xml from the wikipedia demos directory to the solr_tutorial directory.

2. Using CQL, create a keyspace and a table to store user names, email addresses, and their skype, twitter, and irc names. The all field will exist in the Solr index only, so you do not need an all column in the table.

   CREATE KEYSPACE user_info
   WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };

   CREATE TABLE user_info.users (    id text PRIMARY KEY,
   name text,
   email text,
   skype text,
   irc text,
   twitter text
   ) WITH COMPACT STORAGE;

3. Run a CQL BATCH command, as explained earlier, if the schema includes a multivalue field.

   BEGIN BATCH
   INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
   ('user1', 'john smith', 'jsmith@abc.com', 'johnsmith', 'smitty', '@johnsmith')

   INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
   ('user2', 'elizabeth doe', 'lizzy@swbell.net', 'roadwarriorliz', 'elizdoe', '@edoe576')

   INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
   ('user3', 'dan graham', 'etnaboy1@aol.com', 'danielgra', 'dgraham', '@dannyboy')

   INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
   ('user4', 'john smith', 'jonsmit@fyc.com', 'johnsmith', 'jsmith345', '@johnrsmith')

   INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
   ('user5', 'john smith', 'jds@adeck.net', 'jdsmith', 'jdansmith', '@smithjd999')

   INSERT INTO user_info.users (id, name, email, skype, irc, twitter) VALUES
   ('user6', 'dan graham', 'hacker@legalb.com', 'dangrah', 'dgraham', '@graham222')

   APPLY BATCH;
4. Use a schema that contains the multivalued field--all, copy fields for each alias plus the user id, and a docValues option.

```xml
<schema name="my_search_demo" version="1.1">
  <types>
    <fieldType name="string" class="solr.StrField"/>
    <fieldType name="text" class="solr.TextField">
      <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
      </analyzer>
    </fieldType>
  </types>
  <fields>
    <field name="id"  type="string" indexed="true"  stored="true"/>
    <field name="name"  type="string" indexed="true"  stored="true"/>
    <field name="email" type="string" indexed="true"  stored="true"/>
    <field name="skype" type="string" indexed="true"  stored="true"/>
    <field name="irc"  type="string" indexed="true"  stored="true"/>
    <field name="twitter" type="string" indexed="true"  stored="false" multiValued="true"/>
    <field name="all" type="string" docValues="true" indexed="true" stored="false" multiValued="true"/>
  </fields>
  <defaultSearchField>name</defaultSearchField>
  <uniqueKey>id</uniqueKey>
  <copyField source="id" dest="all"/>
  <copyField source="email" dest="all"/>
  <copyField source="skype" dest="all"/>
  <copyField source="irc" dest="all"/>
  <copyField source="twitter" dest="all"/>
</schema>
```

5. On the command line in the solr_tutorial directory, upload the schema and solrconfig.xml to Solr. Create the Solr core for user_info.users.

```bash
curl http://localhost:8983/solr/resource/user_info.users/solrconfig.xml
   --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8'
curl http://localhost:8983/solr/resource/user_info.users/schema.xml
   --data-binary @schema.xml -H 'Content-type:text/xml; charset=utf-8'
curl "http://localhost:8983/solr/admin/cores?action=CREATE&name=user_info.users"
```

6. Search Solr to identify the user, aliases, and id of users having an alias smitty.

```
http://localhost:8983/solr/user_info.users/select?q=all%3Asmitty&wt=xml&indent=true
```

Output is:

```xml
<result name="response" numFound="1" start="0">
  <doc>
    <str name="id">user1</str>
    <str name="email">jsmith@abc.com</str>
    <str name="irc">smitty</str>
    <str name="name">john smith</str>
    <str name="skype">johnsmith</str>
    <str name="twitter">@johnsmith</str>
  </doc>
</result>
```
7. Run this query:

   http://localhost:8983/solr/user_info.users/select/?q=*:*&facet=true&facet.field=name&facet.mincount=1&indent=yes

   At the bottom of the output, the facet results appear. Three instances of john smith, two instances of dan graham, and one instance of elizabeth doe:

   
   ...
   </result>
   <lst name="facet_counts">
     <lst name="facet_queries"/>
     <lst name="facet_fields">
       <lst name="name">
         <int name="john smith">3</int>
         <int name="dan graham">2</int>
         <int name="elizabeth doe">1</int>
       </lst>
     </lst>
   </lst>
   ...

8. Now, you can view the status of the field cache memory to see the RAM usage of docValues per Solr field. Results look something like the example shown in Example 2.

### Changing the value of a stored copyField attribute

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.

### Reloading a Solr core

*Reload a Solr core* instead of creating a new one when you modify the schema.xml or solrconfig.xml.

   curl "http://localhost:8983/solr/admin/cores?action=RELOAD&name=<keyspace.table>"

You can use options with the RELOAD command to re-index and keep or delete the Lucene index.

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.

  ```bash
  curl -v "http://localhost:8983/solr/admin/cores?action=RELOAD&name=<keyspace.table>&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:

  ```bash
  curl "http://localhost:8983/solr/admin/cores?action=RELOAD&name=<keyspace.table>&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.

  **Rebuilding an index using the 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.

  **Checking indexing status**

  If you HTTP post the files to a pre-existing table, DSE Search starts indexing the data. If you HTTP post the files to a non-existent column keyspace or table, DSE Search creates the keyspace and table, 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.
You can also check the logs to get the indexing status. For example, you can check information about the plugin initializer:

INDEXING / REINDEXING -
INFO SolrSecondaryIndex plugin initializer. 2013-08-26 19:25:43,347 SolrSecondaryIndex.java (line 403) Reindexing 439171 keys for core wiki.solr

Or you can check the SecondaryIndexManager.java information:

INFO Thread-38 2013-08-26 19:31:28,498 SecondaryIndexManager.java (line 136) Submitting index build of wiki.solr for data in SSTableReader(path='/mnt/cassandra/data/wiki/solr/wiki-solr-ic-6-Data.db')
INFO Thread-38 2013-08-26 19:38:10,701 SecondaryIndexManager.java (line 156) Index build of wiki.solr complete

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>.<table>/<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.

Using legacy data

To use DSE Search/Solr data from an 3.0 release or earlier, you need to use the legacy type mapping. A solrconfig.xml option, dseTypeMappingVersion element, enables or disables the new Solr type mappings. By default, the new
mappings are enabled by the solrconfig.xml of these DSE Search/Solr demos:

- Solr wikipedia demo
- Log search demo
- Solr stress demo

To use data from an earlier release in these demos, you need to 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. Enable the default legacy type mapping 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
   ```

**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 straight-forward. 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 Cassandra indexes. You can use indexes in Pycassa just as you use the solr_query expression in DSE Search.

DataStax has extended SolrJ to protect internal Solr communication and HTTP access using SSL. You can also use SolrJ to change the consistency level of a DSE-Search node.

**Using the Solr HTTP API**

You can use the Solr HTTP API to query data indexed in DSE Search/Solr just as you would search for data indexed in OSS. After creating a keyspace in Cassandra using CQL, for example, you can HTTP post the files to a pre-existing or non-existent table. DSE Search creates the table if it doesn't exist, and then starts indexing the data. You can use the HTTP API to query the database.

**Solr HTTP API example**
Assuming you performed the previous example, to find the titles in the mykeyspace.mysolr table that begin with the letters Succ in XML, use this URL:

http://localhost:8983/solr/mykeyspace.mysolr/
   select?q=%20title%3ASucc*&fl=title

The response is:

```
<response>
   <lst name="responseHeader">
      <int name="status">0</int>
      <int name="QTime">2</int>
      <lst name="params">
         <str name="fl">title</str>
         <str name="q">title:Succ*</str>
      </lst>
   </lst>
   <result name="response" numFound="2" start="0">
      <doc>
         <str name="title">Success</str>
      </doc>
      <doc>
         <str name="title">Success</str>
      </doc>
   </result>
</response>
```

**Using CQL**

You can use a solr_query expression in a SELECT statement to retrieve Solr data from Cassandra. In this release, CQL Solr queries are suitable for clusters having a single node, but not recommended for production-level queries which are better suited for the Solr HTTP API. Using the 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.

**Synopsis**

```
SELECT <select expression>
   FROM <table>
      [WHERE solr_query = '<search expression>' ] [LIMIT <n>]
```

<search expression> syntax is a Solr query string that conforms to the Lucene syntax and Solr query syntax. You enclose the Solr query string in single quotation marks. For example, after running the wikipedia demo you can use these Solr query strings:

<table>
<thead>
<tr>
<th>Type of Query</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field search</td>
<td>'title:natio* AND Kenya'</td>
<td>You can use multiple fields defined in the schema: 'title:natio* AND body:Carlos Aragonés'</td>
</tr>
<tr>
<td>Wildcard search</td>
<td>Ken?a</td>
<td>Use ? or * for single or multi-character searches.</td>
</tr>
<tr>
<td>Fuzzy search</td>
<td>'Kenya~'</td>
<td>Use with caution, many hits can occur.</td>
</tr>
<tr>
<td>Phrase search</td>
<td>&quot;American football player&quot;</td>
<td>Searches for the phrase enclosed in double quotation marks.</td>
</tr>
</tbody>
</table>
A `SELECT` expression reads one or more records from a Cassandra table and returns a result-set of rows. Each row consists of a partition 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. 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. On the Mac, for example:
   ```bash
cd <install_location>/bin
./cqlsh
```

2. Use the wiki keyspace and include the `solr_query` expression in a CQL select statement to find the titles in the table named `solr` that begin with the letters `natio`:
   ```cql
   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
   ```

**Delete by query**

After you issue a delete by query, documents start getting deleted immediately and deletions continue until all documents are removed. For example:

Delete the `mykeyspace.mysolr` data that you inserted in the Using DSE Search/Solr example. On the command line:

```bash
  "<delete><query>*:*</query></delete>" -H
  'Content-type:text/xml; charset=utf-8'
```
You do not have to post a commit command in the update command as you do in OSS, and doing so is ineffective.

**Limiting columns indexed and returned by a query**

When using dynamic fields, the default column limit controls the maximum number of indexed columns overall, not just dynamic field columns. The column limit also controls the maximum number of columns returned during queries. This column limit prevents out of memory errors caused by using too many dynamic fields. If dynamic fields are not used, the column limit has no effect.

To change the default column limit, which is 1024, configure the `dseColumnLimit` element in the `solrconfig.xml` file. You can override the default configuration using the `column.limit` parameter in a query to specify a different value, for example 2048.

```
http://localhost:8983/solr/<keyspace>.<table>/select?q=
title%3Amytitle*&fl=title&column.limit=2048
```

**Querying multiple tables**

To map multiple Cassandra tables to a single Solr core, use the Solr API. Specify multiple tables 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 tables simultaneously if they have same schema.

**About Solr shard selection**

Previously, for each queried partition 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.

**Querying using spellcheck**

By default, the `solrconfig.xml` does not include configuration for the Solr suggestor. After creating a request handler in the `solrconfig.xml` for `/suggest`, you can issue a query specifying the autocomplete/spellcheck behavior using the `shards.qt=` parameter. For example, to test the suggestor:

```
```

**Using the ExtendedDisMax query parser**

The traditional Solr query parser (defType=lucene) is the default query parser and intended for compatibility with traditional Solr queries. The ExtendedDisMax Query Parser (eDisMax) includes more features than the traditional Solr query parser, such as multi-field query (Disjunction Max Query), relevancy calculations, full query syntax, and aliasing. Edismax is essentially a combination of the traditional Solr query parser and the dismax query parser, plus a number of other functional and usability enhancements. It is the most powerful query parser for Solr that is offered out of the box. For more information, see *Solr 4.x Deep Dive* by Jack Krupansky.

eDisMax supports phrase query features, such as phrase fields and phrase slop. You can use full query syntax to search multiple fields transparently, eliminating the need for inefficient copyField directives.

**eDisMax example**

To query the title and body fields of the `mykeyspace.mysolr` table from the previous example, specify the edismax deftype, the title and body query fields, and boost factors in your query:
http://localhost:8983/solr/mykeyspace.mysolr/
  select?q=life+Life
  &defType=edismax&qf=title^10.0+body^0.2
  &wt=json&indent=on&omitHeader=on

Output in json format is:

```json
{
  "response": {"numFound": 3, "start": 0, "docs": [
    {
      "id": "123",
      "body": "Life is a foreign language; all men mispronounce it.",
      "name": "Christopher Morley",
      "title": "Life"
    },
    {
      "id": "124",
      "body": "In matters of self-control as we shall see again and again, speed kills. But a little friction really can save lives.",
      "name": "Daniel Akst",
      "title": "Life"
    },
    {
      "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"
    }
  ]
}
```

If you change the boost factors in the query to make matches in the body more significant than matches in the title field (qf=title^0.2+body^10.0 for example), the response changes to list the quotations from Morley and Einstein before the quotation of Akst:

```json
{
  "response": {"numFound": 3, "start": 0, "docs": [
    {
      "id": "123",
      "body": "Life is a foreign language; all men mispronounce it.",
      "name": "Christopher Morley",
      "title": "Life"
    },
    {
      "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": "124",
      "body": "In matters of self-control as we shall see again and again, speed kills. But a little friction really can save lives.",
      "name": "Daniel Akst",
      "title": "Life"
    }
  ]
}
```

Configuring the default parser and query fields

You can set default values for most Solr request parameters in the search request handler in solrconfig.xml. To simplify queries you can make eDisMax the default query parser and also specify the query fields in the solrconfig.xml. After configuring the default parser, you no longer need to specify &defType=edismax on the Solr query request.

To modify the default query parser:

1. Navigate to the demos/wikipedia directory and open the solrconfig.xml, for example, for editing.
2. Locate the solr.SearchHandler and add edismax as the defType.

This step eliminates the need to use defType in a query.
3. Add a line to the solr.SearchHandler that specifies different default query fields and field boosting.
   For example:

   ```xml
   <requestHandler name="search" class="solr.SearchHandler" default="true">
   <!-- default values for query parameters can be specified, these will be overridden by parameters in the request -->
   <lst name="defaults">
     <str name="echoParams">explicit</str>
     <int name="rows">10</int>
     <str name="defType">edismax</str>
     <str name="qf">body^.2 title^10.0</str>
   </lst>
   </requestHandler>
   ``

   This step eliminates the need to use qf in a query.

4. Post the configuration and schema files using the cURL utility:

   ```bash
       --data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8'
       --data-binary @schema.xml -H 'Content-type:text/xml; charset=utf-8'
   ```

5. Reload the Solr core for the keyspace and table.

   ```bash
   curl "http://localhost:8983/solr/admin/cores?action=RELOAD&name=mykeyspace.mysolr"
   ```
6. Test the configuration using this simplified query:

http://localhost:8983/solr/mykeyscape.mysolr/
   select?q=life+Life
   &wt=json&indent=on&omitHeader=on

Output is:

```json
{
  "response": {
    "numFound": 3, "start": 0, "docs": [
    {
      "id": "123",
      "body": "Life is a foreign language; all men mispronounce it.",
      "name": "Christopher Morley",
      "title": "Life"
    },
    {
      "id": "124",
      "body": "In matters of self-control as we shall see again and again, speed kills. But a little friction really can save lives."
    },
    {
      "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."
    }
  ]
}
```

The default query field boost factors make matches in the title more significant than matches in the body, so the output lists the quotations from Morley and Akst before the quotation of Einstein.

**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 status of the wiki.solr core looks like this:

```
http://localhost:8983/solr/admin/cores?action=STATUS&wt=json&indent=on&omitHeader=on
```

```
{
  "defaultCoreName":"default.1371321667755813000",
  "initFailures":{},
  "status":{
    "wiki.solr":{
      "name":"wiki.solr",
      "isDefaultCore":false,
      "instanceDir":"solr/",
      "dataDir": "/var/lib/cassandra/data/solr.data/wiki.solr/",
      "config": "solrconfig.xml",
      "schema": "schema.xml",
      "startTime": "2013-06-16T21:05:54.894Z",
      "uptime": 7212565,
      "index":{
        "numDocs": 3579,
```
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. For example, when performing faceted queries using multi-valued fields the multiValued fields are multi-segmented (as opposed to single segmented single-valued fields), resulting in an inefficient near real time (NRT) performance. In DataStax Enterprise 3.1 and later, you can use densely packed DocValue field types (introduced in Solr 4.2) and per-segment docsets. Facet queries will be per-segment, which improves real-time search performance problems.

To ensure that the jvm heap can accommodate the cache, monitor the status of the field cache and take advantage of the Solr 4.3 option for storing the cache on disk or on the heap. 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 DSE Search quick start example after running a few facet queries, use this URL:


Example 1

For example, the URL for viewing the field cache memory usage in json format and the output is:

http://localhost:8983/solr/admin/cores?action=STATUS&wt=json&indent=on&omitHeader=on&memory=true

...
Example 2

After running a few sort by query functions, the output looks something like this:

```
"fieldCache":{
  "entriesCount":0,
  "totalSize":51600,
  "totalReadableSize":"50.4 KB"},
"totalSize":619200,
"totalReadableSize":"604.7 KB"},
"totalMemSize":619200,
"totalReadableMemSize":"604.7 KB"}
```

Expiring a DSE Search column

You can update a DSE Search column to set a time when data expires in these ways:

- Configuring the high-performance update handler
- Using the Solr HTTP API
Using CQL to set TTL

If you configure TTL in the solrconfig.xml, and then use the Solr HTTP API to set TTL, the latter takes precedence. If you configure TTL in the solrconfig.xml and then use CQL to set TTL, the latter also takes precedence.

After configuring a column to expire, Cassandra eventually removes the data from the database and the Solr index.

You can set a time-to-live (TTL) on an entire document or just a field. Configuring per-document TTL causes removal of the entire document. Configuring per-field TTL causes removal of the field only.

Configuring TTL in the update handler

You can configure the solrconfig.xml to include the TTL per-document or per-field on data added to the Solr index or Cassandra database. You construct a Solr HTTP API query to search the Solr index using a ttl component. Depending on the configuration, TTL then applies to the entire document or just to a named field.

To configure per-document or per-field TTL in the update handler:

1. Configure the high-performance update handler section of the solrconfig.xml.
   - For per-document TTL, add these lines to the high-performance updateHandler section:

   <-- The default high-performance update handler -->
   <updateHandler class="solr.DirectUpdateHandler2">
     ...
     <lst name="defaults">
       <int name="ttl">1</int>
     </lst>
   </updateHandler>

   - For per-field TTL, add these lines to the updateHandler section:

   <int name="defaults">
       <int name="ttl.<column/field name1>">1</int>
       <int name="ttl.<column/field name2>">1</int>
       <int name="ttl.<column/field name3>">1</int>
       <int name="ttl.<column/field name4>">1</int>
     ...
   </lst>

2. Re-index the data by uploading the schema.xml and solrconfig.xml and reloading the core.

Configuring TTL using the Solr HTTP API

Use the Solr HTTP API update command to set a time-to-live (TTL). You can construct a URL to update data that includes the TTL per-document or per-field parameter:

- Using the ttl parameter
  Specifies per-document TTL. For example:
  

- Using the ttl.<field name> parameter
  Specifies per-field TTL. For example:
  

Configuring TTL using CQL

Using a CQL INSERT or UPDATE operation, you can set the TTL property. For example, continuing with the example in Using DSE Search/Solr, insert a 5 minute (300 seconds) TTL property on the all columns of the Einstein data:
Search Solutions

```
INSERT INTO mysolr (id, 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.'),
USING TTL 300;
```

Check the remaining time-to-live on the data:

```
select TTL (name) from mykeyspace.mysolr where id = '126';
```

After the remaining time has passed, because all columns of data were set to expire, querying the data returns no results.

**Managing expired columns**

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 the recommended method for managing expired columns.

**Capacity planning**

Using DSE Search/Solr is memory-intensive. This discovery process is intended to help you, the DSE Search/Solr administrator, develop a plan for having sufficient memory resources to meet the needs of your users.

**Overview**

First, you estimate how large your Solr index will grow by indexing a number of documents on a single node, executing typical user queries, and then examining the `field cache memory` usage for heap allocation. Repeat this process using a greater number of documents until you get a feel for the size of the index for the maximum number of documents that a single node can handle. You can then determine how many servers to deploy for a cluster and the optimal heap size. The index should be stored on SSDs or should fit in the system IO cache.

**Setup**

You need to have the following hardware and data:

A node with:
- GB of RAM
- SSD or spinning disk

Input data:
- N documents indexed on a single test node
- A complete set of sample queries to be executed
- The total number of documents system should support

**Step-by-step process**

1. Create a schema.xml and solrconfig.xml.
2. Start a node.
3. Add N docs.
4. Run a range of queries that simulate those of users in a production environment.
5. View the status of field cache memory to discover the memory usage.
6. View the size of the index (on disk) included in the status information about the Solr core.
7. Based on the server's system IO cache available, set a maximum index size per-server.
8. Based on the memory usage, set a maximum heap size required per-server.
9. Calculate the maximum number of documents per node based on #6 and #7.

When the system is approaching the maximum docs per-node, add more nodes.

**Results**

- Optimal heap size per node
- Number of nodes needed for your application, where the replication factor can be increased for more queries per second.

**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. Within the same data center, attempting to run Solr on some nodes and real-time queries or analytics on other nodes does not work. The answer is to organize the nodes running different workloads into virtual data centers.

**Creating a virtual data center**

Virtual data centers are a convenient way to organize workloads within clusters. When you create a keyspace using CQL, you can set up virtual data centers, independent of what physical data center the individual nodes are in. You assign analytics nodes to one data center, search nodes to another, and Cassandra real-time nodes to yet another data center. The separate, virtual data centers for different types of nodes segregate workloads running Solr from those running Cassandra real-time or Hadoop analytics applications. Segregating workloads ensures that only one type of workload is active per data center.

In separate data centers, different types of nodes can handle search while others handle MapReduce, or just act as ordinary Cassandra nodes. 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.

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.

**Workload segregation**

The batch needs of Hadoop and the interactive needs of Solr are incompatible from a performance perspective, so these workloads need to be segregated. Cassandra real-time applications and DSE Search/Solr applications or Hadoop are also incompatible, but for a different reason—dramatically distinct access patterns:

- A Cassandra real-time application needs very rapid access to Cassandra data.
  
  The real-time application accesses data directly by key, large sequential blocks, or sequential slices.
- A DSE Search/Solr application needs a broadcast or scatter model to perform full-index searching.
  
  Virtually every Solr search needs to hit a large percentage of the nodes in the virtual data center (depending on the RF setting) to access data in the entire Cassandra table. The data from a small number of rows are returned at a time.

To deploy a mixed workload cluster, see *Multiple data center deployment*.

**Restrictions**

Do not run Solr and Hadoop on the same node in either production or development environments.

In multiple data centers having clusters that are not running Solr, do not attempt to insert data to be indexed by Solr using CQL or Thrift from these Hadoop or Cassandra real-time clusters. Run the CQL or Thrift inserts on a Solr node.

**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 `CREATE KEYSPACE` to set up replication.

**Common operations**

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 real-time (Cassandra), analytics (Hadoop), or DSE Search (Solr) nodes on separate nodes and in separate data centers.

Common DSE Search/Solr operations are:

- Performing anti-entropy node repair
- Handling inconsistencies in query results
- Adding a new Solr node
- Decommissioning and repairing a node
- Managing the location of Solr data
- Viewing the Solr log
- Changing the Solr logging level
- Accessing the validation Log
- Changing the Solr connector port
- Excluding hosts from Solr-distributed queries

**Fast repair**
Repairing subranges of data in a cluster is faster than running a nodetool repair operation on entire ranges because all the data replicated during the nodetool repair operation has to be re-indexed. When you repair a subrange of the data, less data has to be re-indexed.

**To repair a subrange**

Perform these steps as a rolling repair of the cluster, one node at a time.

1. Run the `dsetool list_subranges command`, using the approximate number of rows per subrange, the beginning of the partition range (token), and the end of the partition range of the node.

   ```
   dsetool list_subranges my_keyspace my_table 10000 113427455640312821154458202477256070485 0
   ```

   The output lists the subranges.

<table>
<thead>
<tr>
<th>Start Token</th>
<th>End Token</th>
<th>Estimated Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>113427455640312821154458202477256070485</td>
<td>132425442795624521227151664615147681247</td>
<td>11264</td>
</tr>
<tr>
<td>132425442795624521227151664615147681247</td>
<td>151409576048389227347257997936583470460</td>
<td>11136</td>
</tr>
<tr>
<td>151409576048389227347257997936583470460</td>
<td>0</td>
<td>11264</td>
</tr>
</tbody>
</table>

2. Use the output of the previous step as input to the `nodetool repair` command.

   ```
   nodetool repair my_keyspace my_table -st 113427455640312821154458202477256070485 -et 132425442795624521227151664615147681247
   nodetool repair my_keyspace my_table -st 132425442795624521227151664615147681247 -et 151409576048389227347257997936583470460
   nodetool repair my_keyspace my_table -st 151409576048389227347257997936583470460 -et 0
   ```

   The anti-entropy node repair runs from the start to the end of the partition range.

**Handling inconsistencies in query results**

Due to the nature of a distributed system, the DSE Search/Solr consistency level of ONE, and other factors, Solr queries can return inconsistent results. For example, Solr might return different numFound counts from consecutive queries.

An efficient way of achieving consistent results is to repair nodes using the **subrange repair method**.

**Adding a new Solr node**

To increase the number of nodes in a Solr cluster, you can add a DSE node to the cluster. If you want to increase capacity of your search, add the node, then optionally, rebalance the cluster. To add a Solr node, use the same method you use to add 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.

**Decommissioning and repairing a node**

You can decommission and repair a Solr node in the same manner as you would a Cassandra node. The efficient and recommended way to repair a node, or cluster, is to use the **subrange repair method**.

**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, on Linux:

```bash
cd <install_directory>

bin/dse cassandra -s -Ddse.solr.data.dir=/opt
```
In this example, the Solr data is saved in the /opt directory.

**Viewing the Solr log**

DSE Search logs Solr log messages in the Cassandra system log:

```
/var/log/cassandra/system.log
```

**Changing the Solr logging level**

Assuming you configured and are using the Apache log4j utility, you can control the granularity of Solr log messages, and other log messages, in the Cassandra system.log file by configuring the log4j-server.properties file. The `log4j-server.properties` file is located in:

- Packaged installations: `/etc/dse/cassandra`
- Binary installations: `/resources/cassandra/conf/`

To set log levels, configure the `log4j.rootLogger` value, specifying one of these values:

- **All** - turn on all logging
- **OFF** - no logging
- **FATAL** - severe errors causing premature termination
- **ERROR** - other runtime errors or unexpected conditions
- **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

For example, open the `log4j-server.properties` file and change the log level by configuring the `log4j.rootLogger` value:

```
# output messages into a rolling log file as well as stdout
log4j.rootLogger=INFO,stdout
```

**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`.

**Securing a Solr cluster**

DataStax Enterprise supports secure enterprise search using Apache Solr 4.3 and Lucene. 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 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.

HTTP Basic Authentication
When you enable Cassandra’s internal authentication by specifying authenticator: org.apache.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:

1. Comment AllowAllAuthenticator and uncomment the PasswordAuthenticator in cassandra.yaml to enable HTTP Basic authentication for Solr.

   #authenticator: org.apache.cassandra.auth.AllowAllAuthenticator
   authenticator: org.apache.cassandra.auth.PasswordAuthenticator
   #authenticator: com.datastax.bdp.cassandra.auth.PasswordAuthenticator
   #authenticator: com.datastax.bdp.cassandra.auth.KerberosAuthenticator

2. Configure the replication strategy for the system_auth keyspace.

3. Start the server.

4. Open a browser, and go to the service web page. For example, assuming you ran the wikipedia demo, go to http://localhost:8983/demos/wikipedia/.

   The browser asks you for a Cassandra username and password.

Excluding hosts from Solr-distributed queries
You can exclude hosts from Solr-distributed queries in DataStax Enterprise 3.1.2 and later. To exclude hosts from queries, perform these steps on each node that you want to send queries to.

1. In DataStax Enterprise 3.1.2 and later, navigate to the Solr/conf directory:
   - Packaged installations: /usr/share/dse/solr/conf
   - Tarball installations: <dse install>/resources/solr/conf

2. Open the exclude.hosts file, and add the list of nodes to be excluded. Each name must be separated by a newline character.
3. Update the list of routing endpoints on each node, by calling the JMX operation refreshEndpoints() on the com.datastax.bdp:type=ShardRouter mbean.

**Using the ShardRouter Mbean**

DataStax Enterprise 3.1.2 exposes the com.datastax.bdp:type=ShardRouter Mbean, providing the following operations:

- `getShardSelectionStrategy(String core)` Retrieves the name of the shard selection strategy used for the given core.
- `getEndpoints(String core)` Retrieves the list of endpoints that can be queried for the given core.
- `getEndpointLoad(String core)` Retrieves the list of endpoints with related query load for the given core; the load is computed as a 1-minute, 5-minutes and 15-minutes exponentially weighted moving average, based on the number of queries received by the given node.
- `refreshEndpoints()` Manually refreshes the list of endpoints to be used for querying cores.

**Tuning DSE Search performance**

In the event of a performance degradation, high memory consumption, or other problem with DataStax Enterprise Search nodes, try:

- Using Cassandra table 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 table 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-table 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 table 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 table.

**Configuring re-indexing and repair**

When running the RELOAD command using the reindex or deleteAll options, a long delay might indicate that tuning is needed. Tune the performance of re-indexing and index rebuilding by making a few changes in the solrconfig.xml file.
1. Increase the size of the RAM buffer, which is set to 100MB by default, to 125, for example.

```xml
<indexConfig>
    <useCompoundFile>false</useCompoundFile>
    <ramBufferSizeMB>125</ramBufferSizeMB>
    <mergeFactor>10</mergeFactor>

2. Increase the soft commit time, which is set to 1000 ms by default, to a larger value. For example, increase the time to 15-16 minutes:

```xml
    <autoSoftCommit>
        <maxTime>1000000</maxTime>
    </autoSoftCommit>
```

The downside of changing the autoSoftCommit attribute is that newly updated rows take longer than usual (1000 ms) to appear in search results.

**Configuring update performance**

If updates take too long and you changed the default autoSoftCommit from the default 1000 ms to a higher value, reset autoSoftCommit in the solrconfig.xml to its default value.

**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 uncomments the autoSoftCommit element:

```xml
<!-- The default high-performance update handler -->
<updateHandler class="solr.DirectUpdateHandler2">

    <autoSoftCommit>
        <maxTime>1000</maxTime>
    </autoSoftCommit>

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, it is 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>
</table>

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 and later 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 NetworkToplogyStrategy 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

   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 data center names and number of replicas you want.

   Set the number of replicas in data centers. For example, 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**

The following procedure builds on the example in Using DSE Search/Solr. Assume the solrconfig.xml and schema.xml files have already been posted using mykeyspace.mysolr in the URL, which creates a keyspace named mykeyspace 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 CQL 3 to change the replication factor of the keyspace. Set a replication factor of 3, for example:

   ALTER KEYSPACE mykeyspace WITH REPLICAION = { 'class' :
   'NetworkTopologyStrategy', 'datacenter1' : 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:

   curl "http://<host>:<port>/solr/<keyspace>.<table>/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:

   HttpServletRequest httpSolrServer = new HttpSolrServer(url);
   httpSolrServer.getInvariantParams().add("cl", "ALL");

   For more information, see the Data Consistency in DSE Search blog.

**Configuring the available indexing threads**

DSE Search provides a 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
implementation allows for greater concurrency and parallelism, but as a consequence, index requests return a response before the indexing operation is actually executed. The number of available indexing threads per Solr core is by default equal to number of available CPU cores times 2. The available threads can be configured by editing the `max_solr_concurrency_per_core` parameter in the `dse.yaml` configuration file; if set to 1, DSE Search uses the legacy synchronous indexing implementation.

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.

**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>
<tr>
<td>Upgrades of Lucene preserve data</td>
<td>yes</td>
<td>no</td>
<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>
</tr>
<tr>
<td>Security</td>
<td>yes</td>
<td>no</td>
<td>DataStax has extended SolrJ to protect internal communication and HTTP access. Solr data can be encrypted and audited.</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 Defining key Solr terms 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. You do not need to use the commit attribute in a Solr HTTP update command.

  In OSS the autoCommit element is present and uncommented. The autoSoftCommit is commented out. You need to use the commit attribute in a Solr HTTP update command.

- OSS supports relative paths set by the `<lib>` property in the solrconfig.xml, but DSE Search/Solr does not. Configuring the Solr library path describes a workaround for this issue that DataStax Enterprise will address in a future release.

- When using dynamic fields, the default column limit controls the maximum number of indexed columns (overall, not just dynamic field ones), as well as columns returned during queries. The column limit prevents out of memory errors caused by using too many dynamic fields.

- DSE Search/Solr uses the Cassandra commit log instead of the Solr updateLog.

- In OSS, you set the data location for saving Solr data files by configuring the `<dataDir>xxx</dataDir>` element in solrconfig.xml. DSE Search ignores this element and saves data in `<Cassandra data directory>/solr.data` unless you specify another location during start up.

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.

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.

  Requires using Zookeeper.
• **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.
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 cassandra</td>
<td></td>
<td>Start up a real-time Cassandra node in the</td>
<td>link to example</td>
</tr>
<tr>
<td></td>
<td>-s</td>
<td>background.</td>
<td></td>
</tr>
<tr>
<td>dse cassandra</td>
<td>-s</td>
<td>Start up a DSE Search/Solr node in the</td>
<td>link to example</td>
</tr>
<tr>
<td></td>
<td>-Ddse.solr.data.dir=&lt;path&gt;</td>
<td>Use &lt;path&gt; to store Solr data.</td>
<td></td>
</tr>
<tr>
<td>dse cassandra</td>
<td>-t</td>
<td>Start up an analytics node in the background.</td>
<td>link to example</td>
</tr>
<tr>
<td>dse 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>dse cassandra</td>
<td>-f</td>
<td>Start up a real-time Cassandra node in the</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>-f -t</td>
<td>foreground.</td>
<td>none</td>
</tr>
<tr>
<td>dse cassandra</td>
<td>-f -s</td>
<td>Start up a DSE Search/Solr node in the</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>foreground.</td>
<td></td>
</tr>
<tr>
<td>dse 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></td>
<td></td>
<td>pid.</td>
<td></td>
</tr>
<tr>
<td>dse 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>dse hadoop</td>
<td>fs &lt;options&gt;</td>
<td>Invoke the Hadoop FileSystem shell.</td>
<td>link to example</td>
</tr>
<tr>
<td>dse hadoop</td>
<td>fs -help</td>
<td>Send Apache Hadoop fs command descriptions to</td>
<td>link to example</td>
</tr>
<tr>
<td></td>
<td></td>
<td>standard output.</td>
<td></td>
</tr>
</tbody>
</table>
Start the Hive client.  
Start Hive by connecting through the JDBC driver.

Describe Mahout commands.

Run the Mahout command.

Add Mahout classes to classpath and execute the hadoop command.

Start Pig.

Send Apache Sqoop command line help to standard output.

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 option.

### 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:

```bash
dsetool checkcfs cfs:///  
```

Use the dsetool to get details about a particular file that has been corrupted. For example:

```bash
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:

```bash
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**

```bash
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 113427455640312821154458202477256070485 -et 132425442795624521227151664615147681247 11264
nodetool repair Keyspace1 Standard1 -st 132425442795624521227151664615147681247 -et 151409576048389227347257997936583470460 11136
nodetool repair Keyspace1 Standard1 -st 151409576048389227347257997936583470460 -et 0 11264
```

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/cassandra-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, Hadoop, 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

- **initial delay**: (Default: 20 seconds) Speeds up startup by delaying the first TTL checks.
- **fix_rate_period**: (Default: 300 seconds) Schedules how often to check for expired data.
- **max_docs_per_batch**: (Default: 200) The maximum number of documents deleted per batch by the TTL rebuild thread.

**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.

**Solr indexing performance tuning**

Back-pressure aims at reducing the rate of inserts to relieve memory pressure and cpu/IO usage so that the flush/commit can happen faster. Back-pressure takes effect when a flush/commit is requested. On overloaded systems experiencing an high write rate and slow index/commit performance, try tuning the back-pressure threshold to improve Solr indexing performance by adding this option to the dse.yaml file and configuring it:

```
back_pressure_threshold_per_core
```

By default, the back-pressure threshold is based on the total number of queued requests, rather than the average.

**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
- Verify DataStax Enterprise is running

**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:

- **Real-time Cassandra node**: `bin/dse cassandra`
- **Analytics node**: `bin/dse cassandra -t`
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:

```
$ cd <install_location>
$ bin/nodetool status
```

**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 edit 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 designate 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 status
   ```

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, 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.

Running `nodetool drain` before using the `cassandra-stop` command to stop a stand-alone process is pointless because the command drains the node before stopping it.

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:

From the install location:
troubleshooting tips

```
bin/dse cassandra-stop ## Use sudo if necessary
```

In the unlikely event that the cassandra-stop command fails because it cannot find the process DataStax Enterprise Java process ID (PID), the output instructs you to find the DataStax Enterprise Java process ID (PID) manually, and stop the process using its PID number.

```
ps auwx | grep dse
bin/dse cassandra-stop -p <PID>
```

**Verify DataStax Enterprise is running**

Use `nodetool status` as it is much less verbose than `nodetool ring`.

- Packaged installs: `nodetool status`
- Binary install: `<install_location>/bin/nodetool status`

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 the Solr library path* describes a workaround for this issue.
Release notes

DataStax Enterprise 3.1.x release notes cover these releases:

- DataStax Enterprise 3.1.6 release notes and updated component
- DataStax Enterprise 3.1.5 release notes and the updated components
- DataStax Enterprise 3.1.4 release notes and the updated components
- DataStax Enterprise 3.1.3 release notes
- DataStax Enterprise 3.1.2 release notes
- DataStax Enterprise 3.1.1 release notes
- DataStax Enterprise 3.1 release notes and the list of components

DataStax Enterprise 3.1.6 release notes

DataStax Enterprise has been enhanced to include Cassandra 1.2.13. Apache documentation covers release notes for Cassandra 1.2.13 and earlier. Cassandra 1.2.13 supports CQL 3 and CQL2; CQL 2 is deprecated and removal is planned for Cassandra 3.0.

Components

- Apache Cassandra 1.2.13
- Apache Hadoop 1.0.4.8
- Apache Hive 0.9.0.3
- Apache Pig 0.9.2
- Apache Solr 4.3.0.1.4
- Apache log4j 1.2.16
- Apache Sqoop 1.4.2.11.1
- Apache Mahout 0.6
- Apache Tomcat 6.0.32
- Apache Thrift 0.7.0
- Apache Commons

Issue resolved

This release fixes the issue causing snapshot repairs to block operations when another node fails to respond to the repair message. (CASSANDRA-6415)

Issues

- In this release, the flush_largest_memtables_at setting is 0.75, which is typically too small causing excessive flushing of the memtable to disk. The workaround is to change the setting to 0.80 in the cassandra.yaml. (DSP-2989)
A DSE 3.1 node can fail to start if you comment out or remove one of the following sections but leave the other. (DSP-3078)

```java
# Replication strategy to use for the auth keyspace.
# Following an upgrade from DSE 3.0 to 3.1, this should be removed
auth_replication_strategy: org.apache.cassandra.locator.SimpleStrategy

# Replication options to use for the auth keyspace.
# Following an upgrade from DSE 3.0 to 3.1, this should be removed
auth_replication_options:
    replication_factor: 1
```

This can occur on either a fresh installation of 3.1 or an upgrade from 3.0.

After the node is restarted the first time, the following error appears in system.log and when running nodetool ring:

```
ERROR 16:12:35,327 Exception in thread Thread[OptionalTasks:1,5,main] java.lang.RuntimeException
    createReplicationStrategy(AbstractReplicationStrategy.java:274)
    at org.apache.cassandra.db.Table.createReplicationStrategy(Table.java:278)
```

**Warning:** To avoid this issue make sure that you comment out or remove both sections at once. If you restart the node a second time, it will fail to start, so do not restart any nodes once this error has occurred until you have performed one of the following workarounds.

If you have already encountered the issue, use one of the following workarounds.

**Workaround 1: If the error is occurring but all nodes are still running**

Use this workaround on a fresh installation or upgrade from DSE 3.0 to 3.1.

1. Comment out both sections in the `cassandra.yaml` file:

   ```java
   # Replication strategy to use for the auth keyspace.
   # Following an upgrade from DSE 3.0 to 3.1, this should be removed
   #auth_replication_strategy: org.apache.cassandra.locator.SimpleStrategy

   # Replication options to use for the auth keyspace.
   # Following an upgrade from DSE 3.0 to 3.1, this should be removed
   #auth_replication_options:
   #    replication_factor: 1
   ```

2. Fix the replication strategy on the `dse_auth` keyspace using `cqlsh`:

   ```sql
   ALTER KEYSPACE dse_auth WITH replication = {
    'class': 'SimpleStrategy',
    'replication_factor': 1
   };
   ```

3. Follow instructions for enabling internal security in the documentation.

**Workaround 2: If some nodes are already down**

1. Perform steps 1-2 of the first workaround.
2. On any nodes that do not start, move these directories from the system keyspace directory to a backup location:
   - Schema
   - Migrations
     - schema_columnfamilies
     - schema_columns
     - schema_keyspaces
3. Restart the nodes. They should start and get the schema from the other nodes that are still running.
4. Follow instructions for enabling internal security in the documentation.

**Workaround 3: If all of the nodes in the cluster are already down**

1. Perform steps 1-2 of the second workaround.
2. Start the nodes. The schema will now be empty.
3. If you have a backup schema creation script written in cqlsh or cassandra-cli, replay the script to restore the schema after you start the nodes. If you do not have a backup, recreate the schema from memory to avoid losing the data.
4. Follow instructions for enabling internal security in the documentation.

**DataStax Enterprise 3.1.5 release notes**

These release notes cover components, enhancements and changes, and issues resolved and unresolved:

**Components**

- Apache Cassandra 1.2.12.2
- Apache Hadoop 1.0.4.8
- Apache Hive 0.9.0.3
- Apache Pig 0.9.2
- Apache Solr 4.3.0.1.4
- Apache log4j 1.2.16
- Apache Sqoop 1.4.2.11.1
- Apache Mahout 0.6
- Apache Tomcat 6.0.32
- Apache Thrift 0.7.0
- Apache Commons

**Enhancements and changes**

This release includes the following changes and enhancements:

- Support for using an external file system, such as s3ql in Hive while using the CassandraFS as the DSE Analytics/Hadoop file system.

  *Follow instructions* in this document to set up and use this feature.
• Improved Hadoop performance when running vnodes within the same, or a different, data center. (DSP-2572/CASSANDRA-6288)

• For DSE Analytics/Hadoop nodes, the default consistency level has changed from ONE to LOCAL_ONE for MapReduce jobs.

Issues resolved
This release fixes the following issues:

• Fixed the issue preventing Hadoop from accessing libraries in Hive. (DSP-1495)
• Fixed the DESCRIBE SCHEMA command on an Analytics node that resulted in an error message on the cqlsh command line. (DSP-2268)
• Fixed issue that caused an error when setting max_solr_concurrency_per_core parameter to 1. (DSP-2321)
• Fixed the issue preventing access to external file systems by Hive. (DSP-2377)
• Fixed the problem causing repeated Hive queries to impact CassandraFS performance. (DSP-2441)
• Two files are now available on packaged as well as tarball installations to copy/paste code and run a pig script instead of stepping through the Explore library data demo manually. (DSP-2461)
• Fixed issue of Job Tracker moving from one node to another. (DSP-2520)
• Fixed the problem that returned incorrect values when applications set widerows=true or set the cassandra.range.batch.size=1. (DSP-2537)
• Fixed the problem causing the Nodetool utility to fail to recognize the --port option. (DSP-2545)
• Fixed the problem that caused secondary indexes to work unreliably with row caches. (DSP-2551)
• Fixed an issue that caused duplicate DSE Search/Solr results to be returned. (DSP-2563)
• Include and set up the old snappy for older distributions. (DSP-2567)
• Fixed the issue involving long-running repairs on Solr admin keyspace that prevented timely core create/reload operations on large clusters. (DSP-2570)
• Fixed the issue involving multiple dse cassandra -Dcassandra.replace_address commands. (DSP-2574)
• Fixed the issue that caused a Java-level deadlock. (DSP-2579)
• Fixed the issue causing the Solr core recovery to deadlock while waiting for the core to be recreated. (DSP-2585)
• Fixed issue where dsetool checkcfs throws NPE when file listed in the directory does not exist. (DSP-2594)
• Fixed file handle leaks in the CassandraFS. (DSP-2660)
• Fixed the issue that affected distribution of data by the bulk loader. The sstableloader utility now works correctly. (DSP-2612)
• Fixed error when upgrading packages on Debian appears to corrupt the limits configuration. (DSP-2696)
• Fixed a race condition when loading solr cores during an upgrade. (DSP-2702)
• Fixed the issue that caused CQL Native connections to be refused when you enable internal authentication. (DSP-4097)

Issues
This release has the following unresolved issues:
• Running Hive queries on a node in an analytics data center that has no replica causes a TimedoutException. The error might look something like this:

...
Caused by: java.io.IOException: Server returned HTTP response code: 400 for URL: http://ip-10-182-188-95.ec2
...

The job tracker error message contains more details information than shown here.

Perform one of these workarounds to increase the replication factor:

• If data is not local, in the TBLPROPERTIES clause of the Hive query, configure the \texttt{cassandra.consistency.level} property to increase the replication from the new default \texttt{LOCAL\_ONE} to at least \texttt{ONE}. (DSP-2718)

• Configure the read and write properties to change the Cassandra consistency level for MapReduce jobs in the mapred-site.xml. Change the consistency level from \texttt{LOCAL\_ONE} to at least \texttt{ONE}:

\begin{verbatim}
<property>
  <name>cassandra.consistencylevel.read</name>
  <value>ONE</value>
</property>
<property>
  <name>cassandra.consistencylevel.write</name>
  <value>ONE</value>
</property>
\end{verbatim}

• Alter the keyspace replication factor to guarantee at least one replication of the analytics node in the data center.

DataStax Enterprise 3.1.4 release notes

These release notes cover components, \textit{enhancements and changes}, \textit{issues resolved}, and \textit{outstanding issues} in the release.

\textit{Components}

• Apache Cassandra 1.2.10
• Apache Hadoop 1.0.4.8
• Apache Hive 0.9.0.1
• Apache Pig 0.9.2
• Apache Solr 4.3.0.1.2
• Apache log4j 1.2.16
• Apache Sqoop 1.4.2.3
• Apache Mahout 0.6
• Apache Tomcat 6.0.32
• Apache Thrift 0.7.0
• Apache Commons
Enhancements and changes

This release includes the following changes and enhancements:

- Pig CQL 3 push down filter (DSP-2214)
- Support for CQL collections in Pig (DSP 2373, DSP-2360)
- Improved debugging of CassandraFS corruption using the dsetool utility (DSP-2416)
- Enforced syntax for flags, such as -h, prefaced by a hyphen. These flags must come before other dsetool command arguments. (DSP-2430) For example:

  dsetool -h 127.0.0.1 ring

- Error logging of Solr request errors that includes parameters sent. (DSP-2450)
- Counting now performed when the commit log stores and replays Solr entries. A new CommitLog-core.name mbean publishes the counters, named entries and replayed. (DSP-2454)
- Finalized finalized Pig formatting syntax, which differs from that of DSE 3.1.2-3.1.3 and conforms to Cassandra 1.2.10. (DSP-2464)
- Includes tuning knobs for dealing with large blobs and many CFs (CASSANDRA-5982, DSP-2470)
- Fixes the issue that deletes snapshots in use during snapshot repair. (CASSANDRA-6011, DSP-2489)
- Updated version of Cassandra to 1.2.10 (DSP-2504)

Issues resolved

This release fixes the following issues:

- After deleting Solr data by dropping table using the CQL DROP TABLE command, or by manually deleting the Solr index directory, you needed to shut down and restart the server before attempting to recreate the Solr core. In this release, after dropping the table, you can upload the Solr schema and configuration and create the Solr core. (DSP-2024)

- The issue causing DataStax Enterprise to crash when you added client_encryption_options to the dse.yaml file has been resolved. In this release, when you attempt to add these options to the dse.yaml file, an error message results. (DSP-2279)

  Configure the client_encryption_options only in the cassandra.yaml file, as described in the SSL documentation.

- The incorrect default gc_grace storage parameter for Cassandra tables has been corrected from 60 seconds to 10 days. (DSP-2342)

- Added missing information in User resource limits. (DSP-2344)

- The issue that prevented MapReduce Jobs from running longer than 24 hours on kerberized clusters is resolved. (DSP-2402)

- Under certain circumstances the DataStax Enterprise service reported a failed exit status when the service actually started. This issue has been resolved. (DSP-2422)

- The issue causing Pig to return an extraneous empty tuple has been resolved. (DSP-2424)

- Fixed race condition between CFS compaction and nodetool scrub. (DSP-2425)

- The issue causing complex Hive and Pig queries to cause a deadlock has been resolved. (DSP-2434)
The back-pressure implementation has been enhanced to improve indexing performance in the following ways: (DSP-2447)

- Throttling new index requests rather than completely blocking them
- Making the default back-pressure threshold based on the total number of queued requests, rather than the average
- Adding support for configurable back-pressure using a `back_pressure_threshold_per_core` option in `dse.yaml` file

Scrubbing SSTables in the CassandraFS erroneously removed information about CassandraFS files. This caused SSTables containing deleted CassandraFS files to accumulate despite compaction or other cleanup operations. This issue has been resolved. (DSP-2472)

The issue causing Solr multivalued date fields to malfunction has been resolved. Multivalued date fields now work. (DSP-2480)

The issue caused by pending compactions causing nodes to die as if disk space was low has been resolved. Cassandra-5605 has been backported to this release. (DSP-2485)

Liveness issues calling CassandraSolrConfig#getColumnLimit under high concurrency have been resolved. (DSP-2494)

The issue causing the ShardRouter to write many messages to the system.log has been resolved. (DSP-2495)

Solr core resource management has been made more robust. Core loading/creation/reload no longer fails at the first attempt when resource loading/writing fails. (DSP-2505)

The issue with inserting a NULL value using Hive, for example when import data into Cassandra, has been resolved. (DSP-2521)

Making a mistake when typing a Pig LOAD command, for example, a mistake in the keyspace name, would cause not only the mistaken command to fail, but also the next, corrected command to fail with the same message, for example:

```
Unexpected internal error. InvalidRequestException(why: Keyspace 'ks1' does not exist.)
```

This issue has been resolved. (DSP-2527)

## Issues

The following issues are present in this release:

- Commit log file handles can be left open during heavy Solr indexing. Do not truncate a table that Solr has indexed during the indexing operation. You can check indexing status using the Solr Admin. (DSP-2540)

- To prevent Hive from throwing the error, “Unrecognized option: -javaagent:/usr/share/dse/cassandra/lib/jamm-0.2.5.jar” comment out the following line in the cassandra-env.sh: (DSP-2549)

  ```
  #echo "$xss = $JVM_OPTS"
  ```

- The sstableloader data distribution is broken. The workaround is to use 3.1.3 as the bulk loading client; it will work with a 3.1.4 cluster. (Cassandra-6272)

---

### DataStax Enterprise 3.1.3 release notes

This release fixes the following issues:

- The exception (array index out-of-bounds) that occurred when writing to a wide-row Thrift table using CqlRecordWriter has been resolved. (DSP-2334)
The problem causing a slow replay of the commit log has been resolved by optimizing flushing of SSTables, making inserts multi-threaded, and other improvements. (DSP-2405)

The problem causing an index to fail when an empty string is inserted into an indexed column has been resolved. (DSP-2429, CASSANDRA-5965)

DataStax Enterprise 3.1.2 release notes
This release includes the changes, enhancements, and resolved issues.

Changes in the upgrade procedure

- Upgrading to DataStax Enterprise 3.1.0 - 3.1.2 directly from some versions of DataStax Enterprise, DataStax Community, and Cassandra are not supported. See version restrictions.
- The client_encryption_options for enabling client-to-node SSL have been removed from dse.yaml in 3.1.2. To enable client-to-node SSL, set the option in the cassandra.yaml file. If you are upgrading from an earlier version of 3.1, manually remove these settings from your dse.yaml.

Other changes and enhancements

- DataStax Enterprise 3.1.2 exposes the com.datastax.bdp:type=ShardRouter Mbean.
- The cassandra-stop command provides a convenient way to stop a stand-alone process on a node:

  From the install location:

  bin/dse cassandra-stop

- In this release, you can determine the status of the Solr indexing process from log files.

Issues

A Cassandra issue can cause a problem when decommissioning a node that interferes with streaming data from the node to SSTables. The workaround is to decommission the node and then repair the cluster. The efficient and recommended way to repair a node, or cluster, is to use the subrange repair method.

Running the DESCRIBE SCHEMA command on an Analytics node results in an error message on the cqlsh command line. (DSP-2268)

Issues resolved

This release resolves these issues:

- Fixes the issue of sstableloader failing with Kerberos and SSL by adding support for Kerberos and SSL to the sstableloader command. (DSP-1168)
- Fixed the issue that caused the sstableloader command to return 0 on success or failure. Now, the command returns 1 on failure and 0 on success. (DSP-2325)
- Fixed the nodetool enablethrift and nodetool disablethrift commands that failed to enable and disable the Thrift transport. (DSP-2343)
- Fixed the issue that prevented the Hive hwi service from starting. (DSP-2364)
- Backported CASSANDRA-5855 that fixed the nodetool scrub command to handle the CQL compound primary key. (DSP-2367)
- Fixed broken hive Views. (DSP-2369)
• Fixed the issue associated with using the EC2MultiRegionSnitch that caused Solr to report unavailable shards for ranges. (DSP-2371)

• Fixed the issue that caused Hive to return an error when you query a table having null values in any columns. (DSP-2372)

• Fixed the issue with the pre-flight check tool that falsely reported a cassandra.yaml error. The pre-flight check tool is located in /usr/share/dse/tools of packaged installs and is a collection of tests that can be run on a node to ensure that it uses the best settings. The tool can fix most of the settings it finds if invalid, or not ideal. The tool is not available in tarball installations. (DSP-2273)

• 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)

• Removed redundant versions of Jetty from the distribution files. (DSP-2378)

• Fixes a problem using the Solr DataImportHandler to import into a Solr core running in DataStax Enterprise 3.1.1, which caused an exception when the SolrWriter submitted the document to Cassandra. (DSP-2381)

• Fixed the issue that selected and re-indexed all data having a TTL property, even data set to expire in the future. (DSP-2385)

• Fixed the shard availability problem after nodes go down and come up. (DSP-2400)

• Backported CASSANDRA-5234 to fix a problem with counters. (DSP-2408)

• Fixed the issue that caused Solr queries to fail when bootstrapping nodes are selected as shards. (DSP-2411)

DataStax Enterprise 3.1.1 release notes

Release 3.1.1 resolves this issue:

The issue that causes a Solr startup problem when a PropertyFileSnitch or GossipingPropertyFileSnitch is used is resolved. (DSP-2283)

Issues

Thread stack size. To avoid StackoverflowErrors, you may need to set the JVM option -Xss to 190k or higher in the cassandra-env.sh file:

`JVM_OPTS="$JVM_OPTS -Xss190k"`

DataStax Enterprise 3.1 release notes

DataStax Enterprise 3.1 includes updated components, enhancements, and changes. These release notes list issues and resolved issues.

Components

• Apache Cassandra 1.2.6.1
• Apache Hadoop 1.0.4.8
• Apache Hive 0.9.0.1
• Apache Pig 0.9.2
• Apache Solr 4.3.0.1
• Apache log4j 1.2.16
• Apache Sqoop 1.4.2.3
DataStax Enterprise 3.1.1 release notes

- Apache Mahout 0.6
- Apache Tomcat 6.0.32
- Apache Thrift 0.7.0
- Apache Commons

Enhancements and changes

General

- Support for virtual nodes in Cassandra. Currently, DataStax recommends using virtual nodes only on data centers running purely Cassandra workloads. You should disable virtual nodes on data centers running either Hadoop or Solr workloads.
- Support for the Murmur3 partitioner.
- Tested on Oracle Java 1.7; no issues found.
- Support for audit logging of queries and prepared statements submitted to the DataStax Java Driver, which uses the CQL binary protocol.
- Support for the Cassandra sstableupgrade tool.

Solr

- Capability to perform a relatively fast repair of a subrange instead of repairing the entire range, which can incur a time-consuming, full re-index of Solr data.

Although announced in a minor release of DataStax Enterprise 3.0, this enhancement is listed here to raise awareness of its availability and use. Using the faster process to repair subranges is recommended for handling inconsistencies in Solr query results and for handling other problems.
- Support for docValues, introduced in Solr 4.2, in the schema field definition.
- Change to the DSE Search/Solr ttl rebuild timeout properties, which ensures purging of expired data from Solr indexes. New options are:
  - ttl_index_rebuild_options.initial_delay
  - ttl_index_rebuild_options.fixed_rate_period
  - ttl_index_rebuild_options.max_docs_per_batch
  The ttl (time-to-live) field format and query on the Lucene side has not changed, so upgrading to DataStax Enterprise 3.1 is not affected by this change.
- Configurable TTL for a field or document using the Solr HTTP API.
- Configurable column limit prevents out of memory errors by controlling the maximum number of indexed columns overall, not just dynamic field columns, as well as columns returned during queries. Effective only when using dynamic fields.
- Per-segment caching for filters and docsets, which improves real-time search performance.
- The dseTypeMapping version includes a force option <changing-solr-type> (for use by experts only).
- Solr shard routing has been changed resulting in a slightly improved throughput and query times on Solr clusters, but not when using Vnodes.
- DSE Search Solr distributed delete and search performance has been improved using a technique that loads only the unique key and explicitly requested fields.

Major Issue: Solr integration problem using PropertyFileSnitch or GossipingPropertyFileSnitch
Solr integration will not work with PropertyFileSnitch or the GossipingPropertyFileSnitch. This limitation will be removed as soon as possible. (DSP-2283)

**Other Issues**

- GLIBCXX_3.4.9 not found. This error may appear in older Linux distributions when installing DSE from the binary tarball. The workaround is to replace snappy-java-1.0.5.jar with snappy-java-1.0.4.1.jar. (DSP-2189)
- Do not run MapReduce jobs while the cluster is in a partially upgraded state or fail to observe any other limitations during upgrading.
- Before upgrading a cluster in which you have decommissioned a node, follow the relevant steps in Recommissioning a node.
- Issuing a DESCRIBE SCHEMA command on an Analytics node results in an error message on the cqlsh command line that you can ignore: (DSP-2268)

  Don't know how to parse type string
  u'org.apache.cassandra.db.marshal.DynamicCompositeType
  (t=>org.apache.cassandra.db.marshal.TimeUUIDType,
  ...
  - The cqlsh DESCRIBE command can produce ddl that has the wrong parameters in it. If compression is not set for a table, cqlsh omits the compression attribute. If you rename the table and issue the DESCRIBE command again, the Snappy compression setting appears. This may occur with other parameters.(CASSANDRA-5766)

**Issues resolved**

- Distributed search with spellcheck problems have been resolved. (DSP-2132) For usage information, see Querying using spellcheck.
- Distributed search with Solr groupby and trie fields now works. (DSP-2130)
- Performance problems executing CQL over Thrift in DataStax Enterprise have been resolved by refactoring CQL handing in DataStax Enterprise. (DSP-2054)
- 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. The fairscheduler jar has been updated. (DSP-1964)
- Support for running MapReduce jobs on a remote cluster. (DSP-2113) See Configuration for running jobs on a remote cluster.
- Datetime parsing in cqlsh 3 has been fixed. (DSP-2170)
- The problem causing Hive queries to fail when connecting to local host when the rpc_address is not 0.0.0.0 has been resolved. (DSP-1996)
- The dse shell script no longer spawns an extra parent process for java. (DSP-1779)