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 2.2

DataStax Enterprise 2.2 introduces these features:

- **Updates Cassandra 1.0 to Cassandra 1.1.5** - In Cassandra 1.1, key improvements have been made in the areas of CQL, performance, and management ease of use.

- **Support for Installation on the HP Cloud** - In addition to Amazon Elastic Compute Cloud, DataStax now supports installation of DataStax Enterprise in the HP Cloud environment. You can install DataStax on Ubuntu 11.04 Natty Narwhal and Ubuntu 11.10 Oneiric Ocelot.

- **Support for SUSE Enterprise Linux** - DataStax Enterprise adds SUSE Enterprise Linux 11.2 and 11.4 to its list of supported platforms.

- **Improved Solr Shard Selection algorithm** - Previously, for each queried token range, Cassandra selected the first closest node to the node issuing the query within that range. Equally distant nodes were always tried in the same order, so that resulted in one or more nodes being hotspotted and often selecting more shards than actually needed. The 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.

- **Capability to Set Solr Column Expiration** - You can update a DSE Search column to set a column expiration date using CQL, which eventually causes removal of the column from the database.

Key Features of DataStax Enterprise

The key features of DataStax Enterprise include:

- **Production Certified Cassandra** – DataStax Enterprise contains a fully tested, benchmarked, and certified version of Apache Cassandra that is suitable for mission-critical production deployments.

- **No Single Point of Failure** - In the Hadoop Distributed File System (HDFS) master/slave architecture, the NameNode entry point into the cluster stores configuration metadata about the cluster. If the NameNode fails, the Hadoop system goes down. DataStax Enterprise improves upon this architecture by making nodes peers. Being peers, any node in the cluster can load data files, and any analytics node can assume the responsibilities of job tracker for MapReduce jobs.

- **Reserve Job Tracker** - DataStax Enterprise keeps a job tracker in reserve to take over in the event of a problem that would affect availability.

- **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 datasets 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

Ring View of a 2-Node Cluster

Graph of Cluster Latency Metric

Storage Metrics

Status of MapReduce Jobs

<table>
<thead>
<tr>
<th>Job</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>select count(*) as c, ds, col from inv1...c(Stage-2)</td>
<td>50%</td>
</tr>
<tr>
<td>select count(*) as c, ds, col from inv2...c(Stage-1)</td>
<td>100%</td>
</tr>
<tr>
<td>PEstimator</td>
<td>100%</td>
</tr>
</tbody>
</table>
Installation

Initializing a DataStax Enterprise Cluster on Amazon EC2
For instructions on installing the DataStax AMI (Amazon Machine Image), see the latest AMI documentation.

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

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

- The Yum Package Management application installed.
- Root or sudo access to the install machine.
- The latest version of Oracle Java SE Runtime Environment (JRE) 6 is installed. Java 7 is not recommended.
- Java Native Access (JNA) is required for production installations. See Installing JNA.
- Your DataStax username and password (provided in your DataStax registration confirmation email). If you do not have a DataStax username and password, register before attempting to download the software.
- If you are installing DataStax Enterprise on a 64-bit Oracle Linux, first install 32-bit versions of glibc libraries. Also, see Recommended Settings for Production Installations.

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

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

   ```
   # java -version
   ```

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

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

   ```
   # rpm -Uvh http://dl.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
   ```
3. Add a yum repository file for DataStax in /etc/yum.repos.d. For example:

```
# /etc/yum.repos.d/datastax.repo
```

4. In this file add the following lines:

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

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

5. Install the DataStax Enterprise packages:

```
# yum install dse-full-2.2.3-1
```

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

You can install any 2.2.x version using the following format:

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

**Next Steps**

- **Configuring OpsCenter** - Configuring the browser-based user interface for monitoring, administering, and configuring multiple clusters from a single centralized management console.
- Taking a look at **Packaged Install Locations** - Information about where DataStax Enterprise files are installed.
- **Deployment** - Setting the configuration properties on each node in the cluster before starting the cluster.
- **Starting DataStax Enterprise as a Service** - Starting DataStax Enterprise as a Service.

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

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

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

**Prerequisites**

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

- The Aptitude Package Manager is installed.
- Root or sudo access to the install machine.
- The latest version of Oracle Java SE Runtime Environment (JRE) 6 is installed. Java 7 is not recommended.
- Java Native Access (JNA) is required for production installations. See Installing JNA.
Note
If you are using Ubuntu 10.04 LTS, you need to update to JNA 3.4, as described in Install JNA on Ubuntu 10.04.

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

DataStax Enterprise Installation Steps
DataStax provides Debian package repositories for Debian and Ubuntu systems.

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

   ```
   # java -version
   ```

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

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

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

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

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

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

4. Install the DataStax Enterprise packages:

   ```
   $ sudo apt-get update
   $ sudo apt-get install dse-full=2.2.3-1 dse=2.2.3-1 dse-hive=2.2.3-1 dse-pig=2.2.3-1 dse-demos=2.2.3-1 ... dse-libhadoop-native=2.2.3-1 dse-libcassandra=2.2.3-1 dse-libhive=2.2.3-1 dse-libpig=2.2.3-1 dse-libhadoop=2.2.3-1
   ```

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

   You can install any 2.2.x version using the following format:

   ```
   $ sudo apt-get install dse-full=<version>-1 dse=<version>-1 dse-hive=<version>-1 dse-pig=<version>-1 ... dse-libcassandra=<version>-1 dse-libhive=<version>-1 dse-libpig=<version>-1 dse-libhadoop=<version>-1
   ```

Next Steps

- Configuring OpsCenter - Configuring the browser-based user interface for monitoring, administering, and configuring multiple clusters from a single centralized management console.
- Taking a look at Packaged Install Locations - Information about where DataStax Enterprise files are installed.
- Deployment - Setting the configuration properties on each node in the cluster before starting the cluster.
- Starting DataStax Enterprise as a Service - Starting DataStax Enterprise as a Service.

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.
Installing DataStax Enterprise on HP Cloud

**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 --ip_proto tcp --from_port 7000 --to_port 7000
```

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

A Cassandra/DataStax Enterprise cluster requires the following ports:

<table>
<thead>
<tr>
<th>Port</th>
<th>IP Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>icmp</td>
<td>Use for ping</td>
</tr>
<tr>
<td>22</td>
<td>tcp</td>
<td>Default SSH port</td>
</tr>
<tr>
<td>8012</td>
<td>tcp</td>
<td>Hadoop Job Tracker client port</td>
</tr>
<tr>
<td>8983</td>
<td>tcp</td>
<td>Solr port and Demo applications website port (Portfolio, Search, Search log)</td>
</tr>
<tr>
<td>50030</td>
<td>tcp</td>
<td>Hadoop Job Tracker website port</td>
</tr>
<tr>
<td>50060</td>
<td>tcp</td>
<td>Hadoop Task Tracker website port</td>
</tr>
<tr>
<td>8888</td>
<td>tcp</td>
<td>OpsCenter website port</td>
</tr>
<tr>
<td>1024+</td>
<td>tcp (use security group)</td>
<td>JMX reconnection/loopback ports</td>
</tr>
<tr>
<td>7000</td>
<td>tcp (use security group)</td>
<td>Cassandra intra-node port</td>
</tr>
<tr>
<td>7199</td>
<td>tcp (use security group)</td>
<td>Cassandra JMX monitoring port</td>
</tr>
<tr>
<td>9160</td>
<td>tcp (use security group)</td>
<td>Cassandra client port</td>
</tr>
<tr>
<td>9290</td>
<td>tcp (use security group)</td>
<td>Hadoop Job Tracker Thrift port</td>
</tr>
<tr>
<td>50031</td>
<td>tcp (use security group)</td>
<td>OpsCenter HTTP proxy for Job Tracker</td>
</tr>
<tr>
<td>61620</td>
<td>tcp (use security group)</td>
<td>OpsCenter intra-node monitoring port</td>
</tr>
<tr>
<td>61621</td>
<td>tcp (use security group)</td>
<td>OpsCenter agent ports</td>
</tr>
</tbody>
</table>

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

### Rules for DSESecurity

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

*Add Rule’ to create new rule

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

### Creating the Server

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

### 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>Running</td>
<td>483481</td>
<td>standard.large</td>
<td>8645 - Ubuntu Oneric 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 (<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 dialog box](image)

   **Example**

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

**Install the JRE and JNA**

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

1. The easiest way to put the Oracle JRE on an HP Cloud instance is to download it to your local machine from Oracle Java SE Downloads and then use the secure copy command to copy it onto the node:

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

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

3. Install the JNA as described in Installing JNA.

**Install DataStax Enterprise**

Install DataStax Enterprise as described in Installing the DataStax Enterprise Package on Debian and Ubuntu.
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.

    ```yaml
    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:

    ```yaml
    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*:

      ```yaml
      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:

      ```yaml
      seed_provider:
        - class_name: org.apache.cassandra.locator.SimpleSeedProvider
          parameters:
            - seeds: "<public_ip_of_seed1>,<public_ip_of_seed2>"
      listen_address: <private_ip_of_the_node>
      broadcast_address: <public_ip_of_the_node>
      ```

Configuring OpsCenter and Agents

DataStax Enterprise OpsCenter is installed when you install DataStax Enterprise using the `sudo apt-get install dse-full opscenter` command. If you have not already installed OpsCenter, install it as described in [Installing OpsCenter from Debian or Ubuntu Packages](#).

*Note*

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

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

   [webserver]
   port = 8888
   interface = <private_ip_of_the_opscenter_node>

2. Connect to the OpsCenter using the following URL:

   http://<private_ip_of_the_opscenter_node>:8888

3. Install the agents as described in Automatically Deploying Agents - Packaged Installations.
   - In the Welcome to DataStax OpsCenter! dialog box, use the private IP address for each node.
   - In the Node SSH Credentials dialog box, use ubuntu for the user name and the private key from the key pair you use to connect to the HP Cloud.

Installing the DataStax Enterprise Tarball on Mac OSX or any Linux OS

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

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

Prerequisites

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

- The latest version of Oracle Java SE Runtime Environment (JRE) 6 is installed. Java 7 is not recommended.
- Java Native Access (JNA) is required for production installations. See Installing JNA.

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

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

DataStax Enterprise Installation Steps

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

   # java -version

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

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

3. Unpack the distributions:

   $ tar -xzvf dse.tar.gz
   $ rm *.tar.gz
4. By default, DataStax Enterprise is configured to use /var/lib/cassandra and /var/log/cassandra directories.

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

```bash
$ 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:

      ```bash
      $ mkdir <install_location>dse-data
      $ cd dse-data
      $ mkdir commitlog
      $ mkdir saved_caches
      
      b. Go the directory containing the cassandra.yaml file. For example:

      ```bash
      $ cd <install_location>/resources/cassandra/conf
      
      c. Edit the following lines in the cassandra.yaml file. For example:

      ```bash
      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

- Configuring OpsCenter - Configuring the browser-based user interface for monitoring, administering, and configuring multiple clusters from a single centralized management console.
- Taking a look at Binary Tarball Install Locations - Information about where DataStax Enterprise files are installed.
- Deployment - Setting the configuration properties on each node in the cluster before starting the cluster.
- Starting DataStax Enterprise as a Stand-Alone Process - Starting DataStax Enterprise as a stand-alone process.

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

- The latest version of Oracle Java SE Runtime Environment (JRE) 6 is installed. Java 7 is not recommended. See Installing the JRE on SUSE Systems.
- Java Native Access (JNA) is required for production installations. See Installing JNA.
- Your DataStax username and password (provided in your DataStax registration confirmation email). If you do not have a DataStax username and password, register before attempting to download the software.

Also see Recommended Settings for Production Installations.

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*.
Preparing to Upgrade

You can upgrade from these releases to DataStax Enterprise 2.2.x:

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

Critical Pre-upgrade Steps

1. Make a backup of the data by taking a snapshot of the node to be upgraded.
2. Save old installation keyspaces as described on the next page.
3. Run `nodetool drain` to flush the commit log of the old installation:
   ```
   nodetool drain -h <hostname>
   ```
   This step is mandatory. Failing to perform this step could result in data loss if a power failure occurs during the upgrade.
4. On Debian/Ubuntu, save the cassandra.yaml (and cassandra.topologies file if you use the PropertyFileSnitch) from the old installation in a safe location.
   On RHEL-based platforms, RPM saves the file automatically during the upgrade process instead of overwriting it.
   RPM output looks something like this:
   ```
   warning: /etc/cassandra/default.conf/cassandra.yaml
   saved as /etc/cassandra/default.conf/cassandra.yaml.rpmsave
   ```
   On tarball platforms, you install the new release in a different location, so the old files are not overwritten.
   Regardless of the platform, if you customized any other files, copy the files from the old installation to a safe location before performing an in-place upgrade that overwrites customized files.
5. Upgrade your installation according to these instructions:
   - Upgrading from DataStax Community 1.0.x to DataStax Enterprise 2.2.x:
     - Tarball
     - Debian/Ubuntu
     - RHEL-based platforms
   - Upgrading from DataStax Enterprise 1.x.x - 2.1.x to 2.2.x
   - Upgrading from Other Versions of Cassandra or DataStax Community to DataStax Enterprise 2.2.x

Order of Upgrading Nodes

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

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

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

The following component version changes and other major changes are included in DSE upgrades:

<table>
<thead>
<tr>
<th>Upgrade</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSE 1.0, 1.0.x, or 2.0.x</td>
<td>Cassandra updated to 1.1.5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>DSE 2.1</td>
<td>Cassandra updated to 1.1.6</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>DSE 2.2</td>
<td>Cassandra updated to 1.1.9</td>
</tr>
</tbody>
</table>

For a complete list of Cassandra changes and new features, see https://github.com/apache/cassandra/blob/cassandra-1.1.9/CHANGES.txt.

Upgrading and Counter Columns

If you use counter columns, upgrading SSTables is highly recommended.

Tarball:

```
<install_location>/bin/nodetool -h upgradesstables
```

Package or AMI:

```
nodetool -h upgradesstables
```

Tapping NEWS.txt for Upgrading Information

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

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

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

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

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

```
cd /usr/share/doc/dse-libcassandra-2.2.  
sudo gunzip NEWS.txt.gz
```

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

DataStax recommends saving your keyspaces before upgrading. Follow this procedure, which is recommended as a best practice.

To save old installation keyspaces:

1. Create a file in a directory for which you have permission that contains the SHOW KEYSPACES command. For example:

   ```
   $ echo "SHOW KEYSPACES;" > /tmp/do_show.txt
   ```

2. Start CLI, using the -f option to execute the command in the saved file. Use the redirect function to save the output to another file in a directory for which you have permission. For example:

   ```
   cd <old installation>/bin
   $ cassandra-cli -host localhost -port 9160 -f
   /tmp/do_show.txt > /tmp/my_saved_ks.txt
   ```

3. Check that the file, my_saved_ks.txt, contains the schema definition output of SHOW KEYSPACES.

   ```
   less /tmp/my_saved_ks.txt
   ```

4. Move the file containing the schema definition output to a safe location.

5. Upgrade to DataStax Enterprise 2.2.x. Do not run the nodetool scrub command. Follow one of these sets of instructions:

   - Upgrading a Tarball Installation of DataStax Community 1.0.x/1.1 to DataStax Enterprise 2.2.x
   - Upgrading from DataStax Community 1.0.x/1.1 to DataStax Enterprise 2.2.x on Debian/Ubuntu
   - Upgrading from DataStax Community 1.0.x/1.1 to DataStax Enterprise 2.2.x on RHEL-based Distributions
   - Upgrading from DataStax Enterprise 1.x.x - 2.1.x to 2.2.x

6. After the upgrade is complete, rerun the SHOW KEYSPACES command.

7. Check for any drastic changes in the keyspace list.

If any keyspaces disappeared during the upgrade:

1. Open the file containing the schema definitions that you saved in step 2 of the previous procedure.

2. Recreate keyspaces and table definitions:

   a. Shutdown the cluster.

   b. Run the following command on the command line:

      ```
      $ rm data/system/system_schema*
      ```

   c. Restart the cluster.

   d. Using CQL or CLI, recreate the keyspaces and tables to replace the missing data: For example:

      ```
      cd <old installation>/bin
      ./cqlsh --cql3
      CREATE KEYSPACE <missing keyspace definition>;
      CREATE TABLE <missing table definition>>;
      ```

Upgrading a Tarball Installation of DataStax Community 1.0.x/1.1 to DataStax Enterprise 2.2.x

This procedure shows how to upgrade a cluster of nodes from DataStax Community 1.0.x/1.1 to DataStax Enterprise 2.2.x.
Upgrading to DataStax Enterprise 2.2.x

Upgrading a Node and Migrating the Data

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

To upgrade a node and migrate the data

1. Create a directory for the new installation, download the tarball, and move it to that directory.
2. Unpack the DataStax Enterprise 2.2.x tarball in the new install location.
   ```bash
tar -xzvf <dse-2.2.x tarball name>
   ```
3. If you customized the location of the data in the old installation, create a hard link to the old data directory:
   ```bash
cd <new install location>
ln -s <old data directory> <new install location>/<new data directory>
   ```

To configure the upgraded node

1. In the new installation, open the cassandra.yaml for writing. The file is located in:
   ```bash
<install location>/resources/cassandra/conf
   ```
2. In the old installation of Cassandra, open the cassandra.yaml. The file is located in:
   ```bash
<install location>/conf
   ```
3. Diff the new and old cassandra.yaml files.
4. Merge the diffs by hand from the old file into the new one, except do not merge snitch settings.
5. Configure the snitch setting and complete the configuration as described in Completing the Configuration and Starting Up the Upgraded Node.

Upgrading from DataStax Community 1.0.x/1.1 to DataStax Enterprise 2.2.x on Debian/Ubuntu

This procedure shows how to upgrade a cluster of Ubuntu 11.10 nodes from DataStax Community 1.0.x/1.1 to DataStax Enterprise 2.2.x. The installer migrates the data from the old to the new version of Cassandra automatically.

Upgrading a Node

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

To upgrade a node

1. Remove the old packages and dependencies on them.
   ```bash
   sudo apt-get remove dsc cassandra
   sudo apt-get autoremove
   ```
   This action shuts down Cassandra on the node.
2. Add the DataStax repository to the /etc/apt/sources.list.
   ```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.
3. Upgrade the node.

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

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

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

To configure the upgraded node

1. Open the old cassandra.yaml located in /etc/cassandra.

2. Open the new cassandra.yaml located in /etc/dse/cassandra.

3. Diff the new and old cassandra.yaml files, merging the diffs by hand from the old file into the new one, except do not merge snitch settings.

4. Configure the snitch setting and complete the upgrade as described in Completing the Configuration and Starting Up the Upgraded Node.

Upgrading from DataStax Community 1.0.x/1.1 to DataStax Enterprise 2.2.x on RHEL-based Distributions

This procedure shows how to upgrade a cluster of RHEL-based nodes from DataStax Community 1.0.x/1.1 to DataStax Enterprise 2.2.x on the following systems:

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

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

Upgrading a Node

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

To upgrade a node

1. Remove dependencies on the old version of the package:

   rpm -e apache-cassandra -noscripts

   The old Cassandra configuration file is renamed to cassandra.yaml.rpmsave as shown in the output of this command.

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

2. Open the yum repository file for DataStax in /etc/yum.repos.d for editing.

   sudo vi /etc/yum.repos.d/datastax.repo

3. Replace the contents of the file with the following lines:

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

   sudo yum clean all
   sudo yum install dse-full

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

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

5. Type y.

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

To configure the upgraded node

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

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

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

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

5. Configure the snitch setting and complete the upgrade as described in Completing the Configuration and Starting Up the Upgraded Node.

Configuring the Snitch Setting

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

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

The default delegated_snitch setting is specified in the new dse.yaml file:

delegated_snitch: com.datastax.bdp.snitch.DseSimpleSnitch

The dse.yaml is located in <install_location>/resources/dse/conf (binary installs) or in /etc/dse (packaged installs).

Completing the Configuration and Starting Up the Upgraded Node

A few configuration settings need to be made before completing the upgrade.

To complete the upgrade
1. Configure the snitch setting as described in *Configuring the Snitch Setting*.

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

   It is ok to overwrite the old with the new cassandra-topology.properties file as instructed in *Configuring the Snitch Setting*.

3. *Start the node.*

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

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

5. Restart client applications.

6. You can use a *rolling restart* to upgrade a cluster: Repeat the upgrade procedures on the next node in the cluster, following instructions in *Performing a Rolling Upgrade* exactly. Monitor the log files for any issues.

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

8. *Validate the upgrade.*

   If you use counter columns, *upgrading SSTables* is highly recommended.

### Scrubbing SSTables

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

First, upgrade all nodes to the latest version of DataStax Enterprise, according to the platform-specific instructions presented earlier in this document. Next, complete steps 1-7 of *Completing the Configuration and Starting Up the Upgraded Node*. At this point, all nodes are upgraded and started.

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

#### Tarball Installations

Download the sstablescrub and dse.in.sh utilities.

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

#### Packaged Installations (deb/rpm)

Download the sstablescrub and dse.in.sh

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

*Note*

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

### To scrub SSTables:

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

```
usage: sstablescrub [options] <keyspace> <column_family>
--
Scrub the sstable for the provided column family.
--
Options are:
--debug display stack traces
-h,--help display this help message
-m,--manifest-check only check and repair the leveled manifest, without
actually scrubbing the sstables
-v,--verbose verbose output
```

For example, on a tarball installation:

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

3. Restart each node and client applications, one node at-a-time.

4. Validate the upgrade.

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

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

Performing a Rolling Upgrade

Using a rolling restart, you upgrade and start one node at a time, instead of bringing down the entire cluster and starting all nodes at once. Between the time the first node begins the upgrade process until the last node completes the process, a schema disagreement condition exists. This is expected behavior.

When the schema disagreement condition 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 upgrade, these are the CQL commands that are and are not supported:

<table>
<thead>
<tr>
<th>OK to Run</th>
<th>Do Not Run</th>
<th>Do Not Run (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>ALTER TABLE [1]</td>
<td>DROP TABLE [1]</td>
</tr>
<tr>
<td>INSERT</td>
<td>CREATE TABLE [1]</td>
<td>DROP INDEX</td>
</tr>
<tr>
<td>SELECT</td>
<td>CREATE INDEX</td>
<td>DROP KEYSPACE</td>
</tr>
<tr>
<td>UPDATE</td>
<td>CREATE KEYSATE</td>
<td>TRUNCATE</td>
</tr>
</tbody>
</table>

Cassandra throws a SchemaDisagreementException when a schema disagreement occurs. Continue upgrading until you complete all upgrade steps on all nodes, then using the Command Line Interface (CLI), run the DESCRIBE CLUSTER command:

---

1 TABLE and COLUMNFAMILY are interchangeable.
$ cd <new DSE installation>

$ cassandra-cli -host localhost -port 9160

[default@demo]DESCRIBE CLUSTER;

Ensure that the output shows a single schema version for all nodes. If the output indicates a schema disagreement, or if a node is UNREACHABLE, perform these steps:

1. Restart the node.
2. Run the DESCRIBE CLUSTER command again.
3. Repeat this process until the output shows a single schema version for all nodes.

Ensure that the schema agrees before running DDL workloads.

Performing a Rolling Restart on Analytics or Solr Nodes

A rolling restart is not fully supported on Analytics and Solr nodes in that exceptions, which you can ignore, flood the log file.

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

ERROR [GossipStage:1] 2012-09-21 01:09:21,510 AbstractCassandraDaemon.java (line 139) Fatal exception in thread ...

INFO [JOB-TRACKER-INIT] 2012-09-20 07:06:38,064 JobTracker.java (line 2427) problem cleaning system directory: cfs:/tmp/hadoop-automaton/mapred/system java.io.IOException: java.lang.RuntimeException: TimedOutException() ...

The runtime exceptions you might see when starting Solr nodes look something like these snippet.

javax.management.InstanceAlreadyExistsException: solr/Logging.log_entries: type= ...

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

Validating the Upgrade

After all nodes are upgraded, validate the upgrade by checking that these conditions do not exist:

- A schema disagreement
- Missing keyspaces

To check for a schema disagreement
1. Using the Command Line Interface (CLI), run the DESCRIBE CLUSTER command:

   $ cassandra-cli -host localhost -port 9160

   [default@unknown] DESCRIBE cluster;

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

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

2. Restart unreachable nodes.
3. Repeat steps 1 and 2 until all nodes show the same schema number.

To check for missing keyspaces

1. Use the CLI to run a SHOW KEYSPACES command:

   cd <new installation>/bin
   $ cassandra-cli -host localhost -port 9160
   [default@unknown] SHOW KEYSPACES

   The output consists of the schema definitions in the upgraded installation.
2. Compare the output with the schema definitions you saved.

Post-Upgrade Problems?

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

Upgrading from DataStax Enterprise 1.x.x - 2.1.x to 2.2.x

Under either of these conditions, you need to restart the nodes as real-time Cassandra nodes before upgrading as described in the following procedures:

   • If you are upgrading from DataStax Enterprise 2.0.x to 2.2.x.
   • If the cluster contains any Solr or Hadoop nodes.

If you are not upgrading under these conditions, restarting the nodes as real-time Cassandra nodes is an extra step that you can skip. Restarting the nodes as real-time Cassandra nodes prevents unwanted schema changes from occurring when you start the upgraded node.

To upgrade a tarball release

1. Stop the first node to be upgraded and restart it as a real-time Cassandra node:

   dse cassandra

2. Follow the instructions in Upgrading a Tarball Installation of DataStax Community 1.0.x/1.1 to DataStax Enterprise 2.2.x.

3. Start each node as a real-time Cassandra node during the rolling upgrade.
4. Validate the upgrade of each node.
5. After all nodes are upgraded and the schemas agree, use another rolling restart to put the nodes back to their original roles as Hadoop or Solr nodes:

For Hadoop:

dse cassandra -t

or

For Solr:

dse cassandra -s

To upgrade a packaged release

1. Stop the dse service, and then disable Hadoop or Solr by setting options in /etc/default/dse:

   - To disable Hadoop: HADOOP_ENABLED=0
   - To disable Solr: SOLR_ENABLED=0

2. Restart the dse service.

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

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

5. Start the first node.

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

   Debian/Ubuntu: /etc/dse/cassandra

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

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

7. Configure the snitch setting and complete the upgrade as described in Completing the Configuration and Starting Up the Upgraded Node.

8. Start up each node as a real-time Cassandra node during the rolling upgrade (leave HADOOP or SOLR disabled).

9. Validate the upgrade.

After all nodes are upgraded and the schemas agree, you can use a rolling restart to set the nodes back to their original roles as Hadoop or Solr nodes.

To restart nodes

1. Stop the dse service, and then enable Hadoop or Solr by setting options in /etc/default/dse:

   - To enable Hadoop: HADOOP_ENABLED=1
   - To enable Solr: SOLR_ENABLED=1

2. Start DSE as a Solr node.

3. Repeat the previous steps for each node.

Upgrading from Other Versions of Cassandra or DataStax Community to DataStax Enterprise 2.2.x

The procedures listed in the previous section are based on upgrading from DataStax Community 1.0.x to DataStax Enterprise 2.2.x. Generally, these procedures work for upgrading from other releases to DataStax Enterprise 2.2.x. Exceptions are:
Upgrading the DataStax AMI

- Upgrading directly from versions 0.8.8 and 1.0.3-1.0.5 to 2.2.x does not work. These older versions generate cross-dc forwarding and are incompatible with Cassandra 1.1, a component of DataStax Enterprise 2.2.x. First upgrade to 0.8.11 or 1.0.11, then upgrade to 2.2.x.
- To upgrade from a Brisk release, contact Support.
- To upgrade from Cassandra 0.8 or earlier, refer to NEWS.txt.

Upgrading the DataStax AMI

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

**Note**

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

1. On each node ensure that the DataStax repository is listed in the /etc/apt/sources.list:
   ```
   deb http://<username>:@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 2.2 **cassandra.yaml**
   b. Merge the versions by hand from the old **cassandra.yaml** into the new DSE 2.2 **cassandra.yaml**:
      - **Don't** add snitch settings from the old file to the new file. The new default snitch in the **cassandra.yaml** is com.datastax.bdp.snitch.DseDelegateSnitch. In previous versions, the default snitch was com.datastax.bdp.snitch.DseSimpleSnitch.
      - **Don't** copy property files from the prior release and overwrite files in the new release. Users who have attempted this have reported problems.
5. Configure the snitch setting as described in Configuring the Snitch Setting.
6. If necessary, upgrade any CQL drivers and client libraries, such as python-cql, Hector, or Pycassa that are incompatible with the new DSE version. You can download CQL drivers and client libraries from the DataStax download page.
7. Run nodetool drain to flush the commit log.
8. Restart the node:
   
   ```bash
   sudo service dse restart
   ```

9. Restart client applications.
Deployment

Production Deployment Planning

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

Planning includes the following activities:

- Prerequisites
- Selecting Hardware for Enterprise Implementations
- Planning an Amazon EC2 Cluster
- Calculating Usable Disk Capacity
- Calculating User Data Size

Prerequisites

Before starting to plan a production cluster, you need:

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

Selecting Hardware for Enterprise Implementations

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

Note

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

Memory

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

DataStax recommends the following memory requirements:

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

CPU

Insert-heavy workloads are CPU-bound in Cassandra before becoming memory-bound. Cassandra is highly concurrent and uses as many CPU cores as available.
For dedicated hardware, 8-core processors are the current price-performance sweet spot.

For virtual environments, consider using a provider that allows CPU bursting, such as Rackspace Cloud Servers.

**Disk**

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

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

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

**Recommendations:**

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

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

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

**Network**

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

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

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

**Ports**

If using a firewall, make sure that nodes within a cluster can reach each other. See [Configuring Firewall Port Access](#).

**Planning an Amazon EC2 Cluster**

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

Use the following guidelines when setting up your cluster:

- For most production clusters, use or Extra Large instances with local storage.
- For low to medium data throughput production clusters, use Large instances with local storage (which are generally adequate for about a year).
- RAID0 the ephemeral disks, and put both the data directory and the commit log on that volume. This has proved to be better in practice than putting the commit log on the root volume (which is also a shared resource).
- For data redundancy, consider deploying your cluster across multiple availability zones or using EBS volumes to store your backup files.

*Note*

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

Deployment
• EBS volumes are **not** recommended for Cassandra data volumes. Their network performance and disk I/O are not good fits for Cassandra for the following reasons:

  • EBS volumes contend directly for network throughput with standard packets. This means that EBS throughput is likely to fail if you saturate a network link.

  • EBS volumes have unreliable performance. I/O performance can be exceptionally slow, causing the system to backload reads and writes until the entire cluster becomes unresponsive.

  • Adding capacity by increasing the number of EBS volumes per host does not scale. You can easily surpass the ability of the system to keep effective buffer caches and concurrently serve requests for all of the data it is responsible for managing.

For more information and graphs related to ephemeral versus EBS performance, see the blog article at http://blog.scalyr.com/2012/10/16/a-systematic-look-at-ec2-io/.

### Calculating Usable Disk Capacity

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

Start with the raw capacity of the physical disks:

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

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

\[
\frac{\text{raw\_capacity} \times 0.9}{2} = \text{formatted\_disk\_space}
\]

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

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

### Calculating User Data Size

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

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

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

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

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

\[
\text{primary\_key\_index} = \text{number\_of\_rows} \times (32 + \text{average\_key\_size})
\]
Configuring Replication

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

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

**Partitioner Settings**

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

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

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

**Snitch Settings**

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

In cassandra.yaml, the snitch is set to the DSE Delegated Snitch (endpoint_snitch: com.datastax.bdp.snitch.DseDelegateSnitch). The following sections describe a few commonly-used snitches. All snitches are described in the Apache Cassandra documentation.

**DSE Delegated Snitch**

In DataStax Enterprise, the default delegated snitch is the DseSimpleSnitch (org.apache.cassandra.locator.DseSimpleSnitch), located in:

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

**DseSimpleSnitch**

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

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

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

**Configuring the PropertyFileSnitch**

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

Every node in the cluster should be described in this file, and specified exactly the same on every node in the cluster. For example, supposing you had non-uniform IPs and two physical data centers with two racks in each, and a third logical data center for replicating analytics data:

```yaml
# Data Center One
175.56.12.105=DC1:RAC1
175.50.13.200=DC1:RAC1
175.54.35.197=DC1:RAC1
120.53.24.101=DC1:RAC2
120.55.16.200=DC1:RAC2
120.57.102.103=DC1:RAC2

# Data Center Two
110.56.12.120=DC2:RAC1
110.50.13.201=DC2:RAC1
110.54.35.184=DC2:RAC1
50.33.23.120=DC2:RAC2
50.45.14.220=DC2:RAC2
50.17.10.203=DC2:RAC2

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

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

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

**Choosing Keypspace Replication Options**

When you create a keyspace, you must define the replica placement strategy and the number of replicas you want. DataStax recommends choosing NetworkTopologyStrategy for single and multiple data center clusters. It is as easy to use as SimpleStrategy and allows for expansion to multiple data centers in the future, should that become useful. It is much easier to configure the most flexible replication strategy up front, than to reconfigure replication after you have already loaded data into your cluster.
NetworkTopologyStrategy takes as options the number of replicas you want per data center. Even for single data center (or single node) clusters, you can use this replica placement strategy and just define the number of replicas for one data center. For example (using cassandra-cli):

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

Or for a multiple data center cluster:

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

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

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

### Changing Replication Settings

The default replication for system keyspaces, such as the HiveMetaStore keyspace, is 1. The default replication factor is suitable for development and testing of a single node. If you use Hive in a production environment, you should increase the cfs and HiveMetaStore keyspace replication factors to at least 2 to be resilient to single-node failures.

The procedure for changing the replication of data in the Cassandra File System (CFS) involves these tasks:

1. Update the replication factor of the default keyspaces on the node: cfs and, if you use cfs_archive, also update it.
2. Update the default HiveMetaStore keyspace if the node is an Analytics node that uses Hive.

**To change the replication of data in the CFS**

1. Check the name of the data center of the node.
   ```bash
   bin/nodetool -h localhost ring
   ```
   The output tells you that the name of the data center for your node is, for example, datacenter1.

2. Change the replication factor of the cfs and cfs_archive keyspaces:
   ```cql
   [default@unknown] UPDATE KEYSPACE cfs
       WITH strategy_options = {datacenter1:3};
   
   [default@unknown] UPDATE KEYSPACE cfs_archive
       WITH strategy_options = {datacenter1:3};
   ```

3. If you use Hive, update the HiveMetaStore keyspace accordingly:
   ```cql
   [default@unknown] UPDATE KEYSPACE HiveMetaStore
       WITH strategy_options = {datacenter1:3};
   ```

If you change the replication factor of a keyspace that contains data, run `nodetool repair` to avoid having missing data problems or data unavailable exceptions.

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

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

**Storage Settings**

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

**Gossip Settings**

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster_name</td>
<td>Name of the cluster that this node is joining. Should be the same for every node in the cluster.</td>
</tr>
<tr>
<td>listen_address</td>
<td>The IP address or hostname that other Cassandra nodes will use to connect to this node. Should be changed from <code>localhost</code> to the public address for the host.</td>
</tr>
<tr>
<td>seeds</td>
<td>A comma-delimited list of node IP addresses used to bootstrap the gossip process. Every node should have the same list of seeds. In multi data center clusters of Cassandra or analytics nodes, the seed list should include a node from each data center.</td>
</tr>
<tr>
<td>storage_port</td>
<td>The intra-node communication port (default is 7000). Should be the same for every node in the cluster.</td>
</tr>
<tr>
<td>initial_token</td>
<td>The initial token is used to determine the range of data this node is responsible for.</td>
</tr>
</tbody>
</table>

**Purging Gossip State on a Node**

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

```
-Dcassandra.load_ring_state=false
```

**Single Data Center Deployment**

In this scenario, data replication is distributed across a single data center in mixed workload clusters. For example, if the cluster has 3 Hadoop nodes, 3 Cassandra nodes, and 2 Solr nodes, the cluster has 3 data centers: one for each type of node. A multiple data center cluster has more than one data center for each type of node.
Data replicates across the data centers automatically and transparently - no ETL work is necessary to move data between different systems or servers. You can configure the number of copies of the data in each data center and Cassandra handles the rest, replicating the data for you. To configure a multiple data center cluster, see Multiple Data Center Deployment.

**Prerequisites**

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

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

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

**Configuration Example**

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

**Location of the property file:**

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

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

**Note**

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

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

1. The nodes have the following IPs, and one node per rack will serve as a seed:

   - node0 110.82.155.0 (Cassandra seed)
   - node1 110.82.155.1 (Cassandra)
   - node2 110.82.155.2 (Cassandra)
   - node3 110.82.155.3 (Analytics seed)
   - node4 110.82.155.4 (Analytics)
   - node5 110.82.155.5 (Analytics)
   - node6 110.82.155.6 (Search - seed nodes are not required for Solr.)
   - node7 110.82.155.7 (Search)
2. Calculate the token assignments using the *Token Generating Tool* for a single data center.

<table>
<thead>
<tr>
<th>Node</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>node0</td>
<td>0</td>
</tr>
<tr>
<td>node1</td>
<td>21267647932558653966460912964485513216</td>
</tr>
<tr>
<td>node2</td>
<td>42535295865117307932921825928971026432</td>
</tr>
<tr>
<td>node3</td>
<td>63802943797675961899382738893456539648</td>
</tr>
<tr>
<td>node4</td>
<td>85070591730234615865843651857942052864</td>
</tr>
<tr>
<td>node5</td>
<td>106338239662793269832304564822427566080</td>
</tr>
<tr>
<td>node6</td>
<td>12760588759535192379876547778691307929</td>
</tr>
<tr>
<td>node7</td>
<td>148873535527910577765226390751398592512</td>
</tr>
</tbody>
</table>

3. If you have a firewall running on the nodes in your Cassandra or DataStax Enterprise cluster, you must open certain ports to allow communication between the nodes. See *Configuring Firewall Port Access*.

4. Stop the nodes and clear the data.

   - For packaged installs, run the following commands:
     
     ```bash
     $ sudo service dse stop  # stops the service
     $ sudo rm -rf /var/lib/cassandra/*  # clears the data from the default directories
     ```

   - For binary installs, run the following commands from the install directory:
     
     ```bash
     $ ps auwx | grep cassandra  # finds the Cassandra and DataStax Enterprise Java process ID [PID]
     $ sudo kill <pid>  # stops the process
     $ sudo rm -rf /var/lib/cassandra/*  # clears the data from the default directories
     ```
5. Modify the following property settings in the `cassandra.yaml` file for each node:

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

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

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

<table>
<thead>
<tr>
<th>Node</th>
<th>initial_token</th>
<th>listen address</th>
</tr>
</thead>
<tbody>
<tr>
<td>node1</td>
<td>212676479325586539666460912964485513216</td>
<td>110.82.155.1</td>
</tr>
<tr>
<td>node2</td>
<td>42535295865117307932921825928971026432</td>
<td>110.82.155.2</td>
</tr>
<tr>
<td>node3</td>
<td>6380294379767596189938273893456539648</td>
<td>110.82.155.3</td>
</tr>
<tr>
<td>node4</td>
<td>85070591730234615865843651857942052864</td>
<td>110.82.155.4</td>
</tr>
<tr>
<td>node5</td>
<td>1063382396662793269832304564822427566080</td>
<td>110.82.155.5</td>
</tr>
<tr>
<td>node6</td>
<td>1276058875953519237987654778691307929</td>
<td>110.82.155.6</td>
</tr>
<tr>
<td>node7</td>
<td>148873535527910577765226390751398592512</td>
<td>110.82.155.7</td>
</tr>
</tbody>
</table>

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

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

   - Packaged installs: See *Starting DataStax Enterprise as a Service*
   - Binary installs: See *Starting DataStax Enterprise as a Stand-Alone Process*
7. Check that your ring is up and running:

   - **Packaged installs**: `nodetool ring -h localhost`
   - **Binary installs**:
     
     ```
     $ cd /<install_directory>
     $ bin/nodetool ring -h localhost
     ```

## Multiple Data Center Deployment

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

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

### Prerequisites

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

- DataStax Enterprise is installed on each node.
- The total number of nodes in the cluster.
- A name for the cluster.
- The IP addresses of each node in the cluster.
- Which nodes will serve as the seed nodes. (DataStax Enterprise nodes use this host list to find each other and learn the topology of the ring.)
- If the nodes are behind a firewall, make sure you know what ports you need to open. See **Configuring Firewall Port Access**.

Other configuration settings you may need are described in **Choosing Node Configuration Options** and **Node and Cluster Configuration**.

This information is used to configure the following properties on each node in the cluster:

- The Node and Cluster Initialization Properties in the `cassandra.yaml` file.
- Assigning the data center and rack names to the IP addresses of each node in the `cassandra-topology.properties` file.
**Configuration Example**

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

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

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

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

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

**Note**

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

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

1. Suppose you install DataStax Enterprise on these nodes:

   10.168.66.41
   10.176.43.66
   10.168.247.41
   10.176.170.59
   10.169.61.170
   10.169.30.138

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

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

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

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

node0:

initial_token: 56713727820156410577229101238628035242
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      - seeds: "10.168.66.41,10.176.170.59"
listen_address: 10.176.43.66

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

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

<table>
<thead>
<tr>
<th>Node</th>
<th>initial_token</th>
<th>listen address</th>
</tr>
</thead>
<tbody>
<tr>
<td>node1</td>
<td>56713727820156410577229101238628035242</td>
<td>10.176.43.66</td>
</tr>
<tr>
<td>node2</td>
<td>113427455640312821154458202477256070485</td>
<td>10.168.247.41</td>
</tr>
<tr>
<td>node3</td>
<td>10</td>
<td>10.176.170.59</td>
</tr>
<tr>
<td>node4</td>
<td>56713727820156410577229101238628035252</td>
<td>10.169.61.170</td>
</tr>
<tr>
<td>node5</td>
<td>113427455640312821154458202477256070495</td>
<td>10.169.30.138</td>
</tr>
</tbody>
</table>

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

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

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

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

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

**Note**

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

- Packaged installs: See *Starting DataStax Enterprise as a Service*
- Binary installs: See *Starting DataStax Enterprise as a Stand-Alone Process*

9. Check that your ring is up and running:

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

---

### More Information About Configuring Data Centers

Links to more information about configuring a data center:

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

### Generating Tokens

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

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

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

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

Token Generating Tool
DataStax provides a Python program for generating tokens. Tokens are integers ranging from 0 to $2^{127} - 1$.

To set up the Token Generating Tool:

1. Using a text editor, create a new file named tokengentool for your token generator program.
2. Go to https://raw.github.com/riptano/ComboAMI/2.2/tokentoolv2.py.
3. Copy and paste the program into the tokengentool file.
4. Save and close the file.
5. Make it executable:
   ```bash
   chmod +x tokengentool
   ```
6. Run the program:
   ```bash
   ./tokengentool <nodes_in_dc1> <nodes_in_dc2> ...
   ```
   The Token Generating Tool calculates the token values.
7. Enter the corresponding value for each node in the initial_token property of the node's cassandra.yaml file.

Calculating Tokens for a Single Data Center
For a single data center, DataStax recommends always using the NetworkTopologyStrategy and the RandomPartitioner.

The NetworkTopologyStrategy is as easy to use as SimpleStrategy and allows for expansion to multiple
data centers in the future. It is much easier to configure the most flexible replication strategy initially, than to reconfigure
replication after you have already loaded data into your cluster. Be sure to configure the strategy_options for your
replication strategy.

For a single data center, enter the number of nodes in Token Generating Tool. For example, for 6 nodes in a single data
center, you enter:

   ```bash
   ./tokengentool 6
   ```

The tool displays the token for each node:

```json
{
    "0": {
        "0": 0,
        "1": 28356863910078205288614550619314017621,
        "2": 56713727820156410577229101238628035242,
        "3": 8507059173023465843651857942052864,
    }
}
```
Calculating Tokens for Multiple Racks in a Single Data Center

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

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

./tokengentool 8

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

Calculating Tokens for a Multiple Data Center Cluster

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

There are different methods you can use when calculating multiple data center clusters. The important point is that the nodes within each data center manage an equal amount of data. The distribution of the nodes within the cluster is not as important. DataStax recommends using DataStax Enterprise OpsCenter to rebalance a cluster.

**Alternating Token Assignments**

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

```
./tokengentool 3 3
```

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

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

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

![Token Position Diagram](image)

**Avoiding Token Collisions**
Expanding a DataStax AMI Cluster

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

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

```
./tokengentool 3
{
  "0": {
    "0": 0,
    "1": 56713727820156410577229101238628035242,
    "2": 113427455640312821154458202477256070485
  }
}
./tokentool 2
{
  "0": {
    "0": 0,
    "1": 85070591730234615865843651857942052864
  }
}
```

The graphic shows the distribution of the nodes with the associated offsets.

---

Expanding a DataStax AMI Cluster

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

About Hadoop

In DataStax Enterprise, Hadoop is continuously available for analytics workloads. DataStax Enterprise is 100% compatible with Apache's Hadoop. Instead of using the Hadoop Distributed File System (HDFS), DataStax Enterprise uses Cassandra File System (CassandraFS) keyspaces for the underlying storage layer. This provides all of the benefits of HDFS such as replication and data location awareness, with the added benefits of the Cassandra peer-to-peer architecture.

DataStax Enterprise fully supports:

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

Starting DataStax Enterprise Hadoop

Assuming an analytics node is running, use the following command to start Hadoop:

```
dse hadoop fs <args>
```

where the available `<args>` are described in the HDFS File System Shell Guide on the Apache Hadoop web site.

For example:

```
dse hadoop fs -help
```

For information on starting an analytics node, see Starting and Stopping DataStax Enterprise.

For information on starting Hive, Pig, or using Hadoop, see:
- About Hive
- About Pig
- Apache Hadoop web site

Hadoop Demos

After starting Hadoop, run these demos for a good introduction to Hadoop solutions:

- Portfolio Manager Demo: Demonstrates a hybrid workflow using DataStax Enterprise.
- Hive Demo: Demonstrates using Hive to access data in Cassandra.
- Mahout Demo: Demonstrates Mahout support in DataStax Enterprise by determining which entries in the sample input data file remained statistically in control and which have not.
- Pig Demo: Create a Pig relation, perform a simple MapReduce job, and put the results back into CassandraFS or into a Cassandra column family.
- Sqoop Demo: Migrates data from a MySQL database containing information from the North American Numbering Plan.

Setting the Replication Factor
The default replication for system keyspaces is 1. This replication factor is suitable for development and testing of a single node, not for a production environment. For production increase the replication factors to at least 2. This ensures resilience to single-node failures. For example:

```
[default@unknown] UPDATE KEYSPACE cfs
  WITH placement_strategy = 'org.apache.cassandra.locator.NetworkTopologyStrategy'
  AND strategy_options={Analytics:3};
```

For more information, see Changing Replication Settings.

**Connecting to External Addresses**

This information is intended for advanced users.

**How to enable Hadoop to connect to external addresses:**

- In the core-site.xml file, change the property fs.default.name from file:/// to cfs:<listen_address>:<rpc_port>.
- This eliminates the need to specify the IP address or hostname for MapReduce jobs and all other calls to Hadoop.
- The core-site.xml file is located in the following locations:
  - Packaged installations: /etc/dse/hadoop
  - Binary installations: /<install_location>/resources/hadoop/conf
- Or run the following embedded parameter:
  
  dse hadoop fs -Dfs.default.name="cfs:<listen_address>:<rpc_port>" -ls /

**About Portfolio Manager Demo Application**

Your DataStax Enterprise (DSE) installation contains a demo application that shows a sample mixed workload on a DSE cluster. The use case is a financial application where users can actively create and manage a portfolio of stocks.

On the Cassandra OLTP (online transaction processing) side, each portfolio contains a list of stocks, the number of shares purchased, and the purchase price. The demo's pricer utility simulates real-time stock data where each portfolio updates based on its overall value and the percentage of gain or loss compared to the purchase price. This utility also generates 100 days of historical market data (the end-of-day price) for each stock.

On the DSE OLAP (online analytical processing) side, a Hive MapReduce job calculates the greatest historical 10 day loss period for each portfolio, which is an indicator of the risk associated with a portfolio. This information is then fed back into the real-time application to allow customers to better gage their potential losses.

**Prerequisites**

- A single-node or multiple node instance of DataStax Enterprise is installed.
- Your cluster is configured and running.

**Running the Demo**
1. Go to the portfolio manager demo directory.
   - Binary install: cd <install_location>/demos/portfolio_manager
   - Packaged install: cd /usr/share/dse-demos/portfolio_manager

   **Note**
   You must run the pricer utility from a directory where you have write permissions (such as your home directory), or else run it as root or using sudo.

2. Run the bin/pricer utility to generate stock data for the application:
   - To see all of the available options for this utility:
     ```
     bin/pricer --help
     ```
   - If running on a single node cluster on localhost:
     ```
     bin/pricer -o INSERT_PRICES
     bin/pricer -o UPDATE_PORTFOLIOS
     bin/pricer -o INSERT_HISTORICAL_PRICES -n 100
     ```

3. Start the web service.
   ```
   cd website
   java -jar start.jar &
   ```
4. Open a browser and go to http://localhost:8983/portfolio to see the real-time Portfolio Manager demo application.

![Portfolio Manager Demo](image)

5. Open another terminal.
6. Start Hive and run the MapReduce job for the demo in Hive.
   - Binary
     ```
     <install_location>/bin/dse hive -f <install_location>/demos/portfolio_manager/10_day_loss.q
     ```
   - Packaged install:
     ```
     dse hive -f /usr/share/dse-demos/portfolio_manager/10_day_loss.q
     ```

   The MapReduce job take several minutes to run.
7. Open the URL http://localhost:50030/jobtracker.jsp in a browser to watch the progress in the job tracker.
8. After the job completes, refresh the Portfolio Manager web page to see the results of the Largest Historical 10 day Loss for each portfolio.

![Portfolio charts showing Largest Historical 10 day Loss](image)

**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 `-j` option.
  
  ```
  dse cassandra -t -j
  ```

  or in a binary distribution:

  ```
  <install_location>/bin/dse cassandra -t -j
  ```

- Use the dsetool movejt command.

If you list any IP addresses in the seeds list of the `cassandra.yaml` file, DataStax Enterprise nominates a node from the list in each data center to be the job tracker.
About the Reserve Job Tracker

DataStax Enterprise 2.1 and later nominates a node in the cluster as a reserve job tracker for a data center. The reserve job tracker becomes the job tracker when, for some reason, there is no local node in the data center that can function as job tracker.

When you upgrade from DataStax Enterprise 2.0 and earlier to DataStax Enterprise 2.2, the job tracker node from the old release is automatically designated as the temporary, reserve job tracker. After migration, the local job tracker election process runs in each data center to determine permanent, reserve job trackers.

Managing the Job Tracker Using dsetool Commands

Several dsetool commands are useful for managing job tracker nodes:

- `dsetool jobtracker`
  Returns the job tracker hostname and port to your location in the data center where you issued the command.
- `dsetool movejt <data center>-<workload> <node IP>`
  Moves the job tracker and notifies the task tracker nodes.
- `dsetool movejt <node IP>`
  In DataStax Enterprise 2.1 and later, if you do not specify the data center name, the command moves the reserve job tracker.
- `dsetool listjt`
  Lists all job tracker nodes grouped by their local data center.
- `dsetool ring`
  Lists the nodes and types of the nodes in the ring.

Listing Job Trackers Example

If you are not sure which nodes in your DSE cluster are job tracker, run the following command:

```
dsetool jobtracker
```

or in a binary distribution:

```
<install_location>/bin/dsetool jobtracker
```

Moving the Job Tracker Node Example

If your primary job tracker node fails, you can use `dsetool movejt` to move the job tracker to another analytics node in the cluster.

1. Log in to a DataStax Enterprise analytics node.
2. Run the `dsetool movejt` command and specify the data center name, hyphen, Analytics (for the workload), and the IP address of the new job tracker node in your DataStax Enterprise cluster. For example, to move the job tracker to node 110.82.155.4 in the DC1 data center:

   ```
dsetool movejt DC1-Analytics 110.82.155.4
   ```

   or in a binary distribution:

   ```
<install_location>/bin/dsetool movejt DC1-Analytics 110.82.155.4
```

3. Allow 20 seconds for all of the analytics nodes to detect the change and restart their task tracker processes.
4. In a browser, connect to the new job tracker and confirm that it is up and running. For example (change the IP to reflect your job tracker node IP):

http://110.82.155.4:50030

5. If you are running Hive or Pig MapReduce clients, you must restart them to pick up the new job tracker node information.

**Changing the Job Tracker Client Port**

By default, the job tracker listens on port 8012 for client messages. You can use another port by configuring the `mapred.job.tracker` property.

**To change the job tracker client port:**

1. Open the `mapred-site.xml` file for editing. The location of this file is:
   - Packaged installations: `/etc/dse/hadoop`
   - Binary installations: `<install_location>/resources/hadoop/conf`

2. Locate the `mapred.job.tracker` property.

   <!-- Auto detect the dse job tracker -->
   <property>
     <name>mapred.job.tracker</name>
     <value>${dse.job.tracker}</value>
     <description>
       The address of the job tracker
     </description>
   </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>

**Using Multiple Cassandra File Systems**

The **Cassandra File System (CassandraFS)** replaces the Hadoop Distributed File System (HDFS). It is designed to simplify the operational overhead of Hadoop by removing the single points of failure in the Hadoop NameNode and to offer easy Hadoop integration for Cassandra users. When an analytics node starts up, DSE creates a default CassandraFS rooted at `cfs:/` and an archive file system named `cfs-archive`.

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:

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

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

### About Hive

DataStax Enterprise (DSE) includes a Cassandra-enabled Hive MapReduce client. **Hive** is a data warehouse system for Hadoop that allows you to project a relational structure onto data stored in Hadoop-compatible file systems, and to query the data using a SQL-like language called HiveQL. HiveQL is extensible and lets you upload custom user-defined functions to manipulate the data in your queries. DSE removes the need to run a stand-alone Hive MetaStore and automaps any existing column families into Hive tables. This means you can start the Hive client on any analytics node and run MapReduce queries directly on data already stored in Cassandra. Additionally, you can use Hive with CassandraFS just as you would in a regular Hadoop implementation and define Hive tables and load them with data using the regular HiveQL SQL-like syntax.

**Note**

DataStax Enterprise includes a JDBC compliant user interface to connect to and work with Hive from inside the server. It also includes support for binary data and support for wide rows (up to 2 billion columns).

### About the Hive metastore

Metadata about the objects you define in Hive is stored in a database called the metastore. In regular HDFS-based Hive, when you run Hive on your local machine, your DDL commands create objects in a local metastore that is not available to other Hive clients. In DataStax enterprise, the Hive metastore is implemented as a keyspace within Cassandra. This automatically makes it a shared metastore without requiring any additional configuration except for increasing the default replication factor.

**Note**

The default replication for system keyspaces is 1. This replication factor is suitable for development and testing, not for a production environment. For production increase the replication factors for the HiveMetaStore and cfs keyspaces to at least 2; see **Changing Replication Settings.**
Using Hive

This section provides information about:

- Setting the Job Tracker node for Hive
- Starting a Hive client
- Starting the Hive server
- Creating Hive CassandraFS tables
- Changing Hive storage properties on the fly
- ODBC driver for Hive

Setting the Job Tracker node for Hive

Hive generates MapReduce jobs for most of its queries. Hive MapReduce jobs are submitted to the job tracker node for the DataStax Enterprise cluster. Hive clients will automatically select the correct job tracker node upon startup. You set the job tracker node for Hive as you would for any analytics node. Use the dsetool commands to manage the job tracker.

Starting a Hive client

When you install DataStax Enterprise using the packaged or AMI distributions, you can start Hive as follows:

dse hive

or in a binary distribution:

<install_location>/bin/dse hive

Starting the Hive server

To connect to Hive via the JDBC driver, start Hive on one of the Hadoop nodes as follows:

dse hive --service hiveserver

or in a binary distribution:

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

Creating Hive CassandraFS tables

DataStax Enterprise allows you to use Hive with CassandraFS just as you would in a regular Hadoop implementation. You can define Hive tables and load them with data using the regular HiveQL SQL-like syntax. In this type of usage, you create your Hive tables using the CREATE TABLE command.

For example:

```
hive> CREATE TABLE invites (foo INT, bar STRING)
       PARTITIONED BY (ds STRING);
```

You can then load a table using the LOAD DATA command. The HiveQL Manual provides more information about the HiveQL syntax. In this usage, your loaded data resides in the cfs keyspace. Your Hive metadata store also resides in Cassandra in its own keyspace.

For example:

```
hive> LOAD DATA LOCAL INPATH '<install_location>/resources/hive/examples/files/kv2.txt'
       OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');
```
hive> LOAD DATA LOCAL INPATH '<install_location>/resources/hive/examples/files/kv3.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-08');

hive> SELECT count(*), ds FROM invites GROUP BY ds;

**Note**
The paths to the Hive example files shown in the example *LOAD* commands above are for the binary distribution.

**Changing Hive storage properties on the fly**
You can change Hive storage properties listed in the *serdeproperties* and *tblproperties* on the fly using DataStax Enterprise 2.1 and later. Using the Hive *SET* command, set properties in the hive session. The settings become effective for the next query. Using DataStax Enterprise 2.0 and earlier, you had to use ALTER TABLE to change the storage properties.

**Running the Hive demo**
The Hive demo shows you how to access data in Cassandra. DataStax Enterprise uses a custom storage handler to allow direct access to data stored in Cassandra through Hive.

**Mapping a Hive database to a Cassandra keyspace**
To access data stored in Cassandra, first define a database in Hive that maps to a keyspace in Cassandra. One way you can map them is by making sure that the name is the same in both Hive and Cassandra. For example:

hive> CREATE DATABASE PortfolioDemo;

Optionally, if your Hive database and Cassandra keyspace use different names (or the Cassandra keyspace does not exist), you can declare keyspace properties in your external table definition using the *TBLPROPERTIES* clause. If the keyspace does not yet exist in Cassandra, Hive will create it.

For example, in the case where the keyspace exists in Cassandra but under a different name:

hive> CREATE DATABASE MyHiveDB;

hive> CREATE EXTERNAL TABLE MyHiveTable(row_key string, col1 string, col2 string)
    STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
    TBLPROPERTIES ( "cassandra.ks.name" = "MyCassandraKS" )

Or if the keyspace does not exist in Cassandra yet and you want to create it:

hive> CREATE EXTERNAL TABLE MyHiveTable(row_key string, col1 string, col2 string)
    STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
    TBLPROPERTIES ( "cassandra.ks.name" = "MyCassandraKS",
                    "cassandra.ks.repfactor" = "2",
                    "cassandra.ks.strategy" = "org.apache.cassandra.locator.NetworkTopologyStrategy" );

**Note**
The default host is localhost.

**Hive to Cassandra table mapping**
An external table in Hive maps to a column family in Cassandra. In DataStax Enterprise 2.0 and earlier, all automatically created hive tables relied on the SERDE property to map typed data in the Cassandra column family to strings. Hive did not store Cassandra data in a typed manner.

In DataStax Enterprise 2.1 and later, automatically created hive tables use the following logic for mapping Cassandra Types to Hive:

<table>
<thead>
<tr>
<th>Cassandra Type</th>
<th>Hive Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTF8Type</td>
<td>string</td>
</tr>
<tr>
<td>AsciiType</td>
<td>string</td>
</tr>
<tr>
<td>DateType</td>
<td>timestamp</td>
</tr>
<tr>
<td>LongType</td>
<td>bigint</td>
</tr>
<tr>
<td>Int32Type</td>
<td>int</td>
</tr>
<tr>
<td>DoubleType</td>
<td>double</td>
</tr>
<tr>
<td>FloatType</td>
<td>float</td>
</tr>
<tr>
<td>BooleanType</td>
<td>boolean</td>
</tr>
<tr>
<td>UUIDType</td>
<td>binary</td>
</tr>
<tr>
<td>TimeUUIDType</td>
<td>binary</td>
</tr>
<tr>
<td>all other types</td>
<td>binary</td>
</tr>
</tbody>
</table>

Validating types

The `STORED BY` clause specifies the storage handler to use, which for Cassandra is `org.apache.hadoop.hive.cassandra.CassandraStorageHandler`. The `WITH SERDEPROPERTIES` clause specifies the properties used when serializing/deserializing data passed between the Hive table and Cassandra.

Validate types using the `cassandra.cf.validatorType`. Set the value of the validatorType to the Cassandra types that map to Hive types.

```sql
hive> DROP TABLE IF EXISTS StockHist;
create external table StockHist(row_key string, column_name string, value double) 
STORRED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
WITH SERDEPROPERTIES (
"cassandra.ks.name" = "PortfolioDemo",
"cassandra.cf.validatorType" = "UTF8Type,UTF8Type,DoubleType"
);
```

This forces the columns to be deserialized from CassandraTypes into Strings.

Specifying CassandraFS and MapReduce properties

The `TBLPROPERTIES` clause specifies CassandraFS and MapReduce properties for the table. For example:

```sql
hive> CREATE EXTERNAL TABLE Users(userid string, name string, 
   email string, phone string) 
STORRED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
WITH SERDEPROPERTIES ( "cassandra.columns.mapping" = ":key,user_name,primary_email,home_phone")
TBLPROPERTIES ( "cassandra.range.size" = "100", 
"cassandra.slice.predicate.size" = "100" );
```

For static Cassandra column families that model objects (such as users), mapping them to a relational structure is straightforward. In the example above, the first column of the Hive table (userid) maps to the row key in Cassandra. The row key in Cassandra is similar to a PRIMARY KEY in a relational table and should be the first column in your Hive table. If you know what the column names are in Cassandra, you can map the Hive column names to the Cassandra column names as shown above.
However, for dynamic column families (such as time series data), all rows likely have a different set of columns, and in most cases you do not know what the column names are. To convert this type of column family to a Hive table, you would convert a wide row in Cassandra to a collection of short rows in Hive using a special set of column names (row_key, column_name, value). For example:

```sql
hive> CREATE EXTERNAL TABLE PortfolioDemo.Stocks
    (row_key string, column_name string, value double)
    STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler';
```

### Mapping column names to Cassandra row and column names

Optionally, you can add a WITH SERDEPROPERTIES clause to map meaningful column names in Hive to the Cassandra row key, column names and column values. For example:

```sql
hive> CREATE EXTERNAL TABLE PortfolioDemo.PortfolioStocks
    (portfolio string, ticker string, number_shares int)
    STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
    WITH SERDEPROPERTIES ("cassandra.columns.mapping" = ":key,:column,:value");
```

Using `cassandra.columns.mapping`, you can use a mapping of meaningful column names you assign in the Hive table to Cassandra row key, column/subcolumn names and column/subcolumn values. In the mapping, :key is a special name reserved for the column family row key, :column for column names, :subcolumn for subcolumn names (in super column families), and :value for column (or subcolumn) values. If you do not provide a mapping, then the first column of the Hive table is assumed to be the row key of the corresponding Cassandra column family.

Once you have defined your external tables in Hive, you can query the database to select from the Hive table. For example:

```sql
hive> SELECT * FROM PortfolioDemo.Stocks;
```

Any other query besides a SELECT * in Hive runs as a MapReduce job.

### Inserting data into Cassandra via Hive

Once you have defined an external table object in Hive that maps to a Cassandra column family, you can move the results of MapReduce queries back into Cassandra using the INSERT OVERWRITE TABLE command. For example:

```sql
hive> CREATE EXTERNAL TABLE PortfolioDemo.HistLoss
    (row_key string, worst_date string, loss string)
    STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler';
```

```sql
hive> INSERT OVERWRITE TABLE PortfolioDemo.HistLoss
    SELECT a.portfolio, rdate, cast(minp as string)
    FROM (SELECT portfolio, MIN(preturn) as minp
    FROM portfolio_returns
    GROUP BY portfolio )
a JOIN portfolio_returns b ON
    (a.portfolio = b.portfolio and a.minp = b.preturn);
```

### SERDEPROPERTIES reference

The SERDEPROPERTIES clause specifies the properties used when serializing/deserializing data passed between the Hive table and Cassandra. You can add a WITH SERDEPROPERTIES clause to map meaningful column names in Hive to the Cassandra row key, column names and column values.

The following properties can be declared in a WITH SERDEPROPERTIES clause:

- `cassandra.columns.mapping` - Mapping of Hive to Cassandra columns
- `cassandra.cf.name` - Column family name in Cassandra
- `cassandra.host` - IP of a Cassandra node to connect to
- `cassandra.port` - Cassandra RPC port - default 9160
- `cassandra.partitioner` - Partitioner - default RandomPartitioner

**TBLPROPERTIES reference**

The `TBLPROPERTIES` clause specifies CassandraFS and MapReduce properties for the table. The following properties can be declared in a `TBLPROPERTIES` clause:

- `cassandra.ks.name` - Cassandra keyspace name.
- `cassandra.ks.repfactor` - Cassandra replication factor - default 1.
- `cassandra.ks.strategy` - Replication strategy - default SimpleStrategy.
- `cassandra.input.split.size` - MapReduce split size - default 64 * 1024. This property dictates how many rows are processed per mapper (that is, 64k rows per split).
- `cassandra.range.size` - MapReduce key range size - default 1000. This property specifies the number of rows fetched at a time over the split. For example, if a mapper is processing a total of 64k rows, it pulls 1000 rows at a time 64 times.
- `cassandra.slice.predicate.size` - MapReduce slice predicate size - default 1000. This property describes which columns to fetch from each row and how many columns per row are fetched. For example, for a wide row in Hive, this is the paging size for columns across a row. This means that a row with 10,000 columns is fetched 1000 columns at a time.

**MapReduce performance tuning**

You can change performance settings in the following ways:

- In your external table definitions, using the `TBLPROPERTIES` or `SERDEPROPERTIES` clauses.
- Using the `set` Hive command. For example: `set mapred.reduce.tasks=32;`
- In the `mapred-site.xml` file.
  - Packaged installations: `/etc/dse/hadoop/mapred-site.xml`
  - Binary installations: `<install_location>/resources/hadoop/conf/mapred-site.xml`

**Note**

This is a system setting so if you change it you must restart the analytics nodes.

**Speeding up map reduce jobs:**

Increase your mappers to one per CPU core by setting `mapred.tasktracker.map.tasks.maximum` in `mapred-site.xml`.

**Accessing rows with 100,000 columns or more:**

In the `TBLPROPERTIES` clause, set the `cassandra.range.size` and `cassandra.slice.predicate.size` to fetch one row with 100,000 columns at once. Although this requires more disk usage and scan runs, it is better to fetch one row with 100,000 columns at once than fetching 1000 rows with 100,000 columns at a time.

**Increasing the number of map tasks to maximize performance:**

- Turn off map output compression, in `mapred-site.xml`, to lower memory usage.
- The `cassandra.input.split.size` property (in `TBLPROPERTIES`) sets how many rows are processed per mapper. The default size is 64k rows per split. You can decrease the split size to create more mappers.

**Improving Counter Performance:**
For example, when performing `select count(1) from <column family>;`, you can improve the speed of the counter by setting `cassandra.enable.widerow.iterator=false`. This setting causes all columns after the 1000th column to be ignored for each row, thus improving the speed of the counter.

**Out of Memory Errors:**

When your mapper or reduce tasks fail with OOMs, turn the `mapred.map.child.java.opts` setting in Hive to:

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

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

**ODBC driver for Hive**

DataStax provides an [ODBC driver for Hive on Windows](https://datastax.com/products/bigdata-driver-for-hive/).

**Using the DataStax ODBC Driver for Hive on Windows**

The DataStax ODBC Driver for Hive provides Windows users access to the information stored in the Hadoop distribution bundled into DataStax Enterprise. This driver allows you to access the data stored on your DataStax Enterprise Hadoop nodes using business intelligence (BI) tools, such as Tableau and Microsoft Excel. The driver is compliant with the latest ODBC 3.52 specification and automatically translates any SQL-92 query into HiveQL.

**Prerequisites**

- Windows® 7 Professional or Windows® 2008 R2. Both 32- and 64-bit editions are supported.
- Microsoft Visual C++ 2010 runtime.
- A cluster with a Hadoop node running the Hive server. See [Starting the Hive server](https://datastax.com/products/bigdata-driver-for-hive/).

**Installing the Driver**

To install the DataStax ODBC driver on a Windows platform:

1. Download the driver from [Client Libraries and CQL Drivers](https://datastax.com/products/bigdata-driver-for-hive/).
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.
3. Create either a User or System DSN (data source name) for your BI tool connection.
   a. Click the User DSN or System DSN tab.
   b. Click Add, select DataStax Hive ODBC Connector, and then click Finish.
   c. In DataStax Hive ODBC Connector Setup, enter the following:

<table>
<thead>
<tr>
<th>Data Source Name</th>
<th>The name for your DSN.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Optional.</td>
</tr>
<tr>
<td>Host</td>
<td>IP or hostname of your Hive server.</td>
</tr>
<tr>
<td>Port</td>
<td>Listening port for the Hive service.</td>
</tr>
<tr>
<td>Database</td>
<td>By default, all tables reside within the default database. To check for the appropriate database, use the <code>show databases</code> Hive command.</td>
</tr>
</tbody>
</table>

   d. Click Test.

   The test results are displayed.

   **Note**

   If your DataStax Enterprise cluster is on Amazon EC2, you must open the listing port for the Hive Server.
   For more information, refer to the latest AMI documentation.

4. To configure the advanced options, see Appendix C in the DataStax Hive ODBC Connector User Guide for Windows:

   Start > Program Files > DataStax Hive ODBC Connector > User’s Guide

**Using the DataStax ODBC Driver for Hive**

After configuring the ODBC data source for Hive, you can connect and pull data from Hive using any compliant BI tool. For example, to retrieve data using Microsoft Excel:
1. Use the data connection wizard to select your new ODBC data source:

2. In **Connect to OBDC Data Source**, select **DSE2 Hive**, and then click **Next**.
3. Select one or more data objects (or construct a query) to retrieve the data, and then click **Finish**.

After the ODBC query is executed and the data is retrieved, a Hive MapReduce job runs on the server:

```
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there is 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
```

About Mahout .. include:: ../latest.txt

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-2.2
   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-2.2
   bin/dse mahout arff.vector --help

   The output is help on the arff.vector command.

   Add Mahout classes to the class path, execute Hadoop command
You can use Hadoop commands to work with Mahout. Using this syntax first adds Mahout classes to the class path, and then executes the Hadoop command.

```
dse mahout hadoop <hadoop command> <options>
```

For example, a Mahout file as input to this command, converts the file to text, so you can read it:

```
cd ~/dse-2.2
bin/dse mahout hadoop fs -text <mahout file> | more
```

The Apache web site offers an in-depth tutorial.

**About Pig**

DataStax Enterprise (DSE) includes a CassandraFS-enabled Apache Pig Client. Pig is a platform for analyzing large data sets that uses a high-level language (called Pig Latin) for expressing data analysis programs. Pig Latin lets developers specify a sequence of data transformations such as merging data sets, filtering them, and applying functions to records or groups of records. Pig comes with many built-in functions, but developers can also create their own user-defined functions for special-purpose processing. Documentation for Pig Latin is available from Apache.

Pig Latin programs run in a distributed fashion on a DSE cluster (programs are compiled into MapReduce jobs and executed using Hadoop). When using Pig with DSE, all jobs can be run in MapReduce mode (even on a single-node cluster). Since all Hadoop nodes are peers in DSE (no Name Node), there is no concept of local mode for Pig. DSE Pig includes a custom storage handler for Cassandra that allows you to run Pig programs directly on data stored in Cassandra. The native Pig storage handler stores data in CassandraFS (the Cassandra-enabled Hadoop distributed file system).

**Setting the Job Tracker Node for Pig**

Pig Latin programs are compiled into sequences of MapReduce jobs that are run in parallel. Jobs are submitted to the job tracker node for the DataStax Enterprise cluster. Pig clients will automatically select the correct job tracker node upon startup. You set the job tracker node for pig as you would for any analytics node and use the dsetool commands to manage the job tracker.

**Starting Pig**

When you install DSE using the packaged distributions, you can start the Pig shell (grunt) as follows:

```
dse pig
```

or in a binary distribution:

```
<install_location>/bin/dse pig
```

**Working in DSE Pig**

DSE allows you to use Pig with data stored in CassandraFS just as you would in a regular Hadoop implementation (using the default Pig storage handler). Pig Latin statements work with relations. A relation can be defined as follows:

- A relation is a bag (more specifically, an outer bag).
- A bag is a collection of tuples.
- A tuple is an ordered set of fields.
- A field is a piece of data.

A Pig relation is a bag of tuples. A Pig relation is similar to a table in a relational database, where the tuples in the bag correspond to the rows in a table. Unlike a relational table, however, Pig relations do not require that every tuple contain
the same number of fields or that the fields in the same position (column) be of the same type. So in a way, Pig relations are more similar to Cassandra column families than they are to a relational table.

The Apache Pig documentation contains more information on defining and working with Pig relations.

**Using Pig to Access Data in Cassandra**

DSE uses a custom storage handler, CassandraStorage() to allow direct access to data stored in Cassandra through Pig. In order to access data in Cassandra, the target keyspace and column family must already exist (Pig can read and write data from/to a column family in Cassandra, but it will not create the column family if it does not already exist).

Using the Pig LOAD command, you pull data into a Pig relation from Cassandra via the CassandraStorage handler. When pulling data from Cassandra, you do not need to specify type information as it is automatically inferred from the column family comparators and validators.

The format of the Pig LOAD command is as follows for a regular column family:

```
<pig_relation_name> = LOAD 'cassandra://<keyspace>/<column_family>'
USING CassandraStorage();
```

Using the Pig STORE command, you push data from a Pig relation to Cassandra via the CassandraStorage handler. You can then push a Pig relation from Pig to Cassandra as follows:

```
STORE <relation_name> INTO 'cassandra://<keyspace>/<column_family>'
USING CassandraStorage();
```

**Running the Pig Demo**

Pig operates on data stored in the Hadoop distributed file system (or CassandraFS in DSE). Your DSE installation contains sample data that you can use to run the Pig examples documented in this section. The sample data file contains tuples of two fields each (name and score). Using Pig, the examples in this section show how to create a Pig relation and perform a simple MapReduce job to calculate the total score for each user. Result output can then be stored back into CFS or into a Cassandra column family.

**Loading Pig Sample Data Into CFS**

The Pig sample data file is located in /usr/share/dse-demos/pig/files/example.txt for packaged installations or <install_location>/demos/pig/files/example.txt for binary installations.

To load the Pig sample data file into CFS:

```
dse hadoop fs -put /usr/share/dse-demos/pig/files/example.txt /
```

or in a binary distribution:

```
dse hadoop fs -put <install_location>/demos/pig/files/example.txt /
```

**Creating a Pig Relation from a Data File**

Here we are creating a relation called score_data that defines a schema of two fields (or columns) - named name and score. Using the LOAD command, we are loading the relation with data in the example.txt file stored in CFS. The USING PigStorage() clause is optional, since this is already the default storage handler for Pig.

```
grunt> score_data = LOAD 'cfs:///example.txt' USING PigStorage()
   AS (name:chararray, score:long);
```

To see the tuples stored in the relation:

```
grunt> DUMP score_data;
```
Running a MapReduce Job in Pig

In this example, we take the raw data we loaded into the `score_data` relation, and perform a number of calculations on the data using the Pig built-in relational operators. Intermediate results are also stored in Pig relations.

First we `GROUP` the tuples in the `score_data` relation by the `name` field, and store the results in a relation called `name_group`. The `PARALLEL` keyword controls how many reducers are used.

```pig
grunt> name_group = GROUP score_data BY name PARALLEL 3;
```

Next we use the `FOREACH` operator to calculate the total score for each user grouping in the `name_group` relation, and store the results in a relation called `name_total`.

```pig
grunt> name_total = FOREACH name_group GENERATE group, COUNT(score_data.name),
             LongSum(score_data.score) AS total_score;
```

Finally we order the results in descending order by total score and store the results in a relation called `ordered_scores`.

```pig
grunt> ordered_scores = ORDER name_total BY total_score DESC PARALLEL 3;
```

Then if we wanted to output the final results, we could use the `DUMP` command to send the results to standard output. Or we could use the `STORE` command to output the results to a file in CFS. The `USING` clause is optional in this case, since `PigStorage()` is already the default storage handler.

```pig
grunt> DUMP ordered_scores;
grunt> STORE ordered_scores INTO 'cfs:///final_scores.txt' USING PigStorage();
```

Creating the PigDemo Keyspace in Cassandra

In order for Pig to access data in Cassandra, the target keyspace and column family must already exist (Pig can read and write data from/to a column family in Cassandra, but it will not create the column family if it does not already exist).

To create the `PigDemo` keyspace and `Scores` column family used in the following examples, run the following commands in the `cassandra-cli` utility.

1. Start the `cassandra-cli` utility:
   ```bash
cassandra-cli
   
or in a binary distribution:
   <install_location>/resources/cassandra/bin/cassandra-cli
   
2. Connect to a node in your DSE cluster on port 9160. For example:
   ```bash
   [default@unknown] connect 110.82.155.4/9160
   
or if running on a single-node cluster as localhost:
   ```bash
   [default@unknown] connect localhost/9160
   ```
3. Create the PigDemo keyspace.

```
[default@unknown] CREATE KEYSPACE PigDemo
  WITH placement_strategy = 'org.apache.cassandra.locator.SimpleStrategy'
  AND strategy_options = [{replication_factor:1}];
```

**Note**
The default replication for system keyspaces is 1. This replication factor is suitable for development and testing, not for a production environment. For more information, see Changing Replication Settings.

4. Connect to the PigDemo keyspace you just created.

```
[default@unknown] use PigDemo;
```

5. Create the Scores column family.

```
[default@unknown] create column family Scores with comparator = 'LongType';
```

6. Exit cassandra-cli:

```
[default@unknown] exit;
```

**Writing Data to a Cassandra Column Family**

In this example, we are using the scores example data loaded into CFS (see Loading Pig Sample Data Into CFS). This data has tuples containing 2 fields (name and score). For a Cassandra column family, however, we need to store 3 fields: the row key (name), the column name (score), and the column value (an empty value in this case).

We want to calculate the total score for each user in the same manner as we did in the Running a MapReduce Job in Pig example, however in this example our relations contain an extra empty field for the column value.

To run these commands, start the Pig shell if you do not have it running (see Starting Pig).

1. If you have not already, create the score_data relation from the example.txt file stored in CFS.

   `grunt> score_data = LOAD 'cfs:///example.txt' AS (name:chararray, score:long);`

2. Create a relation called cassandra_tuple to define a tuple of three fields for Cassandra (row key, column name, column value). In this case, the column value is an empty string (using null would be equivalent to a delete).

   `grunt> cassandra_tuple = FOREACH score_data GENERATE name, score, '' AS value;`

3. Group by name and store the results into a relation called group_by_name. The PARALLEL keyword controls how many reducers are used.

   `grunt> group_by_name = GROUP cassandra_tuple BY name PARALLEL 3;`
4. Create an aggregated row for each user containing tuples of their scores and store the results in a relation called `aggregate_scores`.

```plaintext
grust> aggregate_scores = FOREACH group_by_name GENERATE group,
cassandra_tuple.(score, value);
grust> DUMP aggregate_scores;
```

Notice how the data was aggregated for input into Cassandra. A *tuple* was constructed for each Cassandra row. In Pig notation, a tuple is enclosed in parentheses ( ). Within each row tuple, is a *bag* of column tuples - each column tuple representing an individual score. A bag is a collection of tuples in Pig. In Pig notation an inner bag is enclosed in curly brackets {}. So a Pig tuple that represents a row in a column family is structured as:

```plaintext
(<row_key>, [((<column_name1>,<value1>),(<column_name2>,<value2>))])
```

Note that in this example, the *value* is empty (creating a value-less column in Cassandra):

```plaintext
(brandon, [((36,), (128,))])
```

5. Now that the data is in a format that can map to the Cassandra column family, we can store the Pig results into Cassandra using the `CassandraStorage` handler. The INTO clause specifies where to store the data in Cassandra in the format of: `cassandra://<keyspace>/<column_family>`

```plaintext
grust> STORE aggregate_scores INTO 'cassandra://PigDemo/Scores' USING CassandraStorage();
```

**Reading Data From a Cassandra Column Family**

The examples in this section assume you have completed [*Writing Data to a Cassandra Column Family*](#) to group the raw score data into rows by user and load it into Cassandra. In this example, we calculate the total scores for each user.

1. First create a Pig relation called `cassandra_data` by loading rows from the Cassandra column family:

```plaintext
grust> cassandra_data = LOAD 'cassandra://PigDemo/Scores' USING CassandraStorage()
   AS (name, columns: bag [I: tuple(score, value)]);
```

2. Use the `FOREACH` operator to calculate the total score for each user, and store the results in a relation called `total_scores`.

```plaintext
grust> total_scores = FOREACH cassandra_data GENERATE name, COUNT(columns.score),
   SUM(columns.score) as total PARALLEL 3;
```

3. Order the results in descending order by total score and store the results in a relation called `ordered_scores`.

```plaintext
grust> ordered_scores = ORDER total_scores BY total DESC PARALLEL 3;
grust> DUMP ordered_scores;
```

**About Sqoop**

Sqoop is an [Apache Software Foundation tool](https://sqoop.apache.org/) for transferring data between an RDBMS data source and Hadoop or between other data sources, such as NoSQL.

DataStax Enterprise support for Sqoop empowers you to import data from an external data source to Hadoop, Hive, or Cassandra column families. A DSE node runs the Hadoop/Analytics workload, and the Hadoop job imports data from a data source using Sqoop.

**Running the Sqoop Demo**

To get started using Sqoop, first run the *Sqoop demo* to import data from a MySQL table to text files in the Cassandra File System (CFS).
**Importing Data**

You can import data from any JDBC-compliant data source. For example:

- DB2
- MySQL
- Oracle
- SQL Server
- Sybase

You need a JDBC driver for the RDBMS or other type of data source.

**Migrating Data to a Cassandra Table**

After importing data into text files in Cassandra, take a look at how to expand the basic dse sqoop import command used by the demo to *migrate data to a Cassandra column family*.

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

**Running the Sqoop Demo**

The Sqoop demo migrates the data from a MySQL table to text files in CFS. The Sqoop data migration demo uses the MySQL database and data from the North American Numbering Plan. This data consists of the area-code (NPA) and telephone number (Nxx) for the USA and Canada.

**Demo Requirements**

To run the demo, you need:

- An installation of MySQL
- Sufficient MySQL database privileges to create database objects
- A JDBC driver in a directory that Sqoop can access
- The connection string that is appropriate for the JDBC driver
- One or more DSE nodes running the Analytics workload to run the Hadoop job that actually imports data from the external data source
- A PATH environment variable that includes the bin directory of the DSE installation

**Step-by-Step Procedure**

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

1. 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. Put the connector in a directory included in the Sqoop classpath, such as the resources/sqoop subdirectory of your DataStax Enterprise (DSE) installation.
3. On the MySQL command line, start the MySQL daemon. For example:
   ```
   sudo ./mysqld_safe --user=mysql
   ```
4. Start MySQL and create the demo database:
   ```
   sudo ./mysql
   CREATE DATABASE npa_nxx_demo;
   ```
5. Then connect to the database and create the table:

```sql
USE npa_nxx_demo;

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**
   ```
   cd /usr/share/dse-demos/sqoop
   ```

   **Tar distribution, such as Mac**
   ```
   cd <install_location>/demos/sqoop
   ```

7. Populate the table by loading the CSV file in the demos/sqoop directory.

   ```sql
   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 as an Analytic node.
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.

**Note**
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.

```bash
sudo ./dse sqoop import --connect jdbc:mysql://127.0.0.1/npa_nxx_demo
  --username root
  --password <password>
  --table npa_nxx
  --target-dir /npa_nxx
```

DSE returns this message: INFO mapreduce.ImportJobBase: Retrieved 105291 records.

To run this demo and import data to nodes in a cluster, the database permissions must be granted to the nodes. For example, use the `grant ALL` command to grant MySQL access to the hosts.

## Cassandra Log4j Appender Solutions

DataStax Enterprise allows you to stream your web and application log information into a database cluster via Apache log4j.

### About the log4j Utility

Apache log4j is a Java-based logging framework that provides runtime application feedback. It provides the ability to control the granularity of log statements using an external configuration file (`log4j.properties`).

With the Cassandra Appender, you can store the log4j messages in a column family where they're available for in-depth analysis using the Hadoop and Solr capabilities provided by DataStax Enterprise. For information about Cassandra logging, see [Logging Configuration](#). Additionally, DataStax provides a [Log4j Search Demo](#).

The log4j utility has three main components: loggers, appenders, and layouts. Loggers are logical log file names. They are the names known to the Java application. Each logger is independently configurable for the level of logging. Outputs are controlled by Appenders. Numerous Appenders are available and multiple Appenders can be attached to any Logger. This makes it possible to log the same information to multiple outputs. Appenders use Layouts to format log entries. In the example below, messages show the level, the thread name, the message timestamp, the source code file, the line number, and the log message.

### Log Levels

The available levels are:

- **All** - turn on all logging
- **OFF** - no logging
- **FATAL** - severe errors causing premature termination
- **ERROR** - other runtime errors or unexpected conditions
- **WARN** - use of deprecated APIs, poor use of API, near errors, and other undesirable or unexpected runtime situations
- **DEBUG** - detailed information on the flow through the system
- **TRACE** - more detailed than DEBUG
- **INFO** - highlight the progress of the application at a coarse-grained level
Datastax does not recommend using TRACE or DEBUG in production due to verbosity and performance.

**Log Messages**

As mentioned above, the messages that appear in the log are controlled via the `conf/log4j.properties` file. Using this properties file, you can control the granularity to the Java package and class levels. For example, DEBUG messages from a particular class can be included in the log while messages from others remain at a higher level. This is helpful to reduce clutter and to identify messages. The log is most commonly a file and/or stdout. The format, behavior (such as file rolling), and so on is also configurable at runtime.

Below are sample log messages from a Cassandra node startup:

```
INFO [main] 2012-02-10 09:15:33,112 DatabaseDescriptor.java (line 495)
  Found table data in data directories. Consider using the CLI to define your schema.
INFO [main] 2012-02-10 09:15:33,135 CommitLog.java (line 166)
  No commitlog files found; skipping replay
INFO [main] 2012-02-10 09:15:33,150 StorageService.java (line 400)
  Cassandra version: 1.0.7
INFO [main] 2012-02-10 09:15:33,150 StorageService.java (line 401)
  Thrift API version: 19.20.0
INFO [main] 2012-02-10 09:15:33,150 StorageService.java (line 414)
  Loading persisted ring state
...
```

**Storing log4j Messages in a Column Family**

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

**To enable the Cassandra Appender:**

1. Add `resources/log4j-appender/lib/` to your application classpath.
2. Modify the `conf/log4j.properties` file, as shown in the example below:

```properties
# Cassandra Appender
log4j.appender.CASS=com.datastax.logging.appender.CassandraAppender
log4j.appender.CASS.hosts=127.0.0.1
log4j.appender.CASS.port=9160
log4j.appender.CASS.appName="myApp"
log4j.appender.CASS.keyspaceName="Logging"
log4j.appender.CASS.columnFamily="log_entries"
log4j.appender.CASS.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.

The `log4j.appender.CASS=com.datastax.logging.appender.CassandraAppender` specifies the CassandraAppender class and assigns it the `CASS` alias. This alias is referenced in the last line.

`log4j.appender.CASS.hosts=127.0.0.1` allows using a comma delimited list of Cassandra nodes (in case a node goes down).

Specify replication options in lines:

```
log4j.appender.CASS.strategyOptions="{"DC1":"1", "DC2":"3"}
```

`log4j.logger.com.foo.bar=INFO, CASS` specifies that all log messages of level INFO and higher, which are generated from the classes and sub-packages within the `com.foo.bar` package, are sent to the Cassandra server by the Appender.

By default, the CassandraAppender records log messages in the Column Family `log_entries` in the `Logging` keyspace. The definition of this Column Family is as follows:

```
cqlsh:Logging> describe columnfamily log_entries;
```

```
CREATE COLUMNFAMILY log_entries (  
  KEY uuid PRIMARY KEY,  
  app_start_time bigint,  
  app_name text,  
  class_name text,  
  file_name text,  
  level text,  
  line_number text,  
  log_timestamp bigint,  
  logger_class_name text,  
  host_ip text,  
  host_name text,  
  message text,  
  method_name text,  
  ndc text,  
  thread_name text,  
  throwable_str_rep text  
) WITH  
  comment='' AND  
  comparator=text AND
```

Cassandra Log4j Appender Solutions
row_cache_provider='ConcurrentLinkedHashCacheProvider' AND 
key_cache_size=200000.000000 AND 
row_cache_size=0.000000 AND 
read_repair_chance=1.000000 AND 
gc_grace_seconds=864000 AND 
default_validation=text AND 
min_compaction_threshold=4 AND 
max_compaction_threshold=32 AND 
row_cache_save_period_in_seconds=0 AND 
key_cache_save_period_in_seconds=144000 AND 
replication_on_write=True;

Example

Consider the following log snippet:

09:20:55,470  WARN SchemaTest:68 - This is warn message #163
09:20:55,470  INFO SchemaTest:71 - This is info message #489
    java.io.IOException: Danger Will Robinson, Danger!
   at com.datastax.logging.SchemaTest.testSavedEntries(SchemaTest.java:58)
   at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      ...

Note that the ERROR entry above includes the stack trace associated with an Exception. The associated rows in the log_entries Column Family appear as follows (queried using cqlsh):

KEY,eea1256e-db24-4cef-800b-843b3b2fb72c | app_start_time,1328894454774 | level,WARN | 
log_timestamp,1328894455391 | logger_class_name,org.apache.log4j.Category | message, 
This is warn message #163 | thread_name,main |

KEY,f7283a71-32a2-43cf-888a-0c1d3328548d | app_start_time,1328894454774 | level,INFO | 
log_timestamp,1328894455064 | logger_class_name,org.apache.log4j.Category | message, 
This is info message #489 | thread_name,main |

KEY,37ba6b9c-9fd5-4dba-8fbc-51c1696bd235 | app_start_time,1328894454774 | level,ERROR | 
log_timestamp,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 column family.

Storing logging information in Cassandra provides the capability to do in-depth analysis via the DataStax Enterprise platform using Hadoop and Solr.

Log4j Search Demo

The Log4j Search Demo shows an example of searching and filtering log4j messages generated by a standard Java application. In the demo, a Hadoop pi calculation is run with a log4j.properties file set to use the CassandraAppender that comes with DataStax Enterprise. As the logs are generated, they are indexed in real time by Solr and made available for searching in the demo user interface.

For information on configuring log4j, see Cassandra Log4j Appender Solutions.
Before starting this demo, be sure that you have started DataStax Enterprise and Solr on a single node. See Starting DSE and DSE Search.

**Running the Demo**

1. Open a shell window or tab and make the `log_search` directory your current directory. The location of the demo directory depends on your platform:
   - **RHEL or Debian installations**
     ```
     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**
About DSE Search/Solr

The major new enhancement made to DataStax Enterprise is enterprise search support using Lucene and Apache Solr. Coming from the Apache Lucene project, Solr is the most popular open source enterprise search platform in use today.

Solr’s primary features include robust free-text search, hit highlighting, and rich document (PDF, Microsoft Word, and so on) handling. Solr also provides more advanced features like aggregation, grouping, and geo spatial search. Today, Solr powers the search and navigation features of many of the world's largest Internet sites. With the inclusion of Solr 4.0, near real-time indexing can be performed.

The unique combination of Cassandra, Solr, and Hadoop in DSE bridges the gap between online transaction processing (OLTP) and online analytical processing (OLAP). DSE Search in Cassandra offers a way to aggregate and look at data in many different ways in real-time. Cassandra speed compensates for typical MapReduce performance problems. By integrating Solr into the DataStax Enterprise big data platform, DataStax extends Solr’s capabilities and overcomes the shortcomings of native Solr mentioned in the next section.

DSE Search is easily scalable. You add search capacity to your cluster in the same way as you add Hadoop or Cassandra capacity to your cluster. You can have a hybrid cluster of nodes, some running Cassandra, some running search, and some running Hadoop. If you don’t need Cassandra or Hadoop, migrate to DSE strictly for Solr and create an exclusively Solr cluster. The DSE cluster configuration improves upon the master-slave configuration supported by native Solr.
DSE supports native Solr tools and APIs, simplifying migration from Solr to DSE Search for Solr users.

**Integration of Solr into DataStax Enterprise**

DSE Enterprise supports cluster partitioning by workload as described in About Replication in Cassandra.

Using this approach, you can make some of your DSE nodes handle search while others handle MapReduce, or just act as ordinary Cassandra nodes. In production environments, do not run Solr and Hadoop on the same node. In development environments, running both is feasible.

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.

**Benefits of Using Solr in DataStax Enterprise**

DataStax Enterprise Search is built on top of Solr 4.0, which 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 native Solr, including:

- A fully fault-tolerant, no-single-point-of-failure search architecture
- Linear performance scalability that comes from adding new search nodes online
- Automatic indexing of data stored in 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 instead of having to update an entire document
• Search indexes that can span multiple data centers (native Solr cannot)
• CQL supports Solr/search queries

DSE Search takes secondary indexes to a new level: data added to Cassandra is locally indexed in Solr and data added to Solr is locally indexed in Cassandra.

**Unsupported Features**

DSE Search does not support:
• Supercolumns
• Counter columns
• Timeseries type rows
• Composite columns, Solr fields must be strings.

**DataStax Enterprise to Solr Objects Map**

Solr calls an index of documents a core. Each document in a core 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 links Solr cores to Cassandra column families, Solr documents to Cassandra rows, and document fields to columns. This table shows the relationship between Cassandra and Solr concepts:

<table>
<thead>
<tr>
<th>Cassandra</th>
<th>Solr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Family</td>
<td>Core</td>
</tr>
<tr>
<td>Row</td>
<td>Document</td>
</tr>
<tr>
<td>Row Key</td>
<td>Unique</td>
</tr>
<tr>
<td>Column</td>
<td>Field</td>
</tr>
<tr>
<td>Node</td>
<td>Shard</td>
</tr>
</tbody>
</table>

**Schema and Configuration Files**

Solr has a number of required and optional configuration files. 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.

For more information about creating the schema, see Creating a Schema.

**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.

After generating valid schema.xml and solrconfig.xml files, you can create a new Solr index by posting the files through a specific HTTP endpoint. Use this format:
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.

**Example of Creating an Index**

For example, to create a Solr index on a column family, make two HTTP POST requests using the cURL utility as follows:

**Configuration file POST request:**

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

**Schema file POST request:**

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

DSE Search stores the files on all the Cassandra nodes and creates a new Solr core. If you HTTP post the files to a pre-existing column family, DSE Search starts indexing the data. If you HTTP post the files to a non-existing column keyspace or column family, DSE Search creates the keyspace and column family, and then starts indexing the data. For example, you can change the stopwords.txt file, repost the schema, and the index updates.

Changing the Solr schema makes reindexing necessary and reindexing can be disruptive. Users can be affected by performance hits caused by reindexing. Changing the schema is recommended only when absolutely necessary. Also, changing the schema during scheduled down time is recommended.

**See also**

- Solr Tutorial on the Solr site
- Solr Tutorial on Apache Lucene site
- Solr data import handler
- Comma-Separated-Values (CSV) file importer
- JSON importer
- Solr cell project, which includes a tool for importing data from PDFs

**Running the Demo**

You can run Solr on one or more nodes, assuming you installed DataStax Enterprise 2.0 or later. DataStax does not support running Solr and Hadoop on the same node, although it's possible to do so in a development environment. In production environments, run Solr and Hadoop on separate nodes.

**Starting a Solr node**

Follow these steps to start DSE Search/Solr on a single node.

1. **Start DSE as a Solr node.**
2. In another shell, check that your Cassandra ring is up and running. For example, on a Mac:

**RHEL or Debian installations**

dsetool ring -h localhost

**Tar distribution, such as Mac**

cd <install_location>/bin

./dsetool ring -h localhost

A table of information appears showing the state of the node and identifying it as a Solr node. Now, set up and run the *DSE search demo*.

**Running the Wikipedia Demo**

After starting DSE as a Solr node, open a shell window or tab, and follow these steps to run the demo.

1. Make the wikipedia demo directory your current directory. The location of the demo directory depends on your platform:

**RHEL or Debian installations**

cd /usr/share/dse-demos/wikipedia

**Tar distribution**

cd <install_location>/demos/wikipedia

2. Add the schema:

./1-add-schema.sh

The script posts solrconfig.xml and schema.xml to these locations:


wiki.solr in the URL creates the keyspace (wiki) and the column family (solr) in Cassandra.

3. Index the articles contained in the wikipedia-sample.bz2 file in the demo directory:

./2-index.sh --wikifile wikipedia-sample.bz2

Three thousand articles load.
4. To see a sample Wikipedia search UI, open your web browser and go to:

http://localhost:8983/demos/wikipedia
5. Inspect the index keyspace, wiki, using the Solr Admin tool:
   

   Be sure to enter the trailing "/".

6. Inspect the column family, solr. In the Solr Admin tool, click SCHEMA to inspect the schema.

To load all Wikipedia articles from the internet into Solr, repeat steps 1 and 2. In step 3, use the name of the file on the internet instead of the name of wikipedia-sample.bz2. The name of the file on the internet is:

enwiki-20111007-pages-articles25.xml-p023725001p026625000.bz2

Loading all the articles takes a long time, so be patient. To limit the number of articles, use the limit option. For example, to limit the number of articles to 10,000, use this command in step 3:

   ./2-index.sh --wikifile enwiki-20111007-pages-articles25.xml-p023725001p026625000.bz2 --limit 10000

Using DataStax Enterprise and DSE Search, you can now:

- Run Hadoop MapReduce on the data on DataStax Enterprise analytics nodes.
- Update an individual column under a row in Cassandra and find the updated data in search results.
- Take advantage of Solr searching to query Cassandra using CQL.

Creating a Schema

A Solr schema defines the relationship between data in a column family and a Solr core. The schema identifies the columns to index in Solr and maps column names to Solr types. This document describes the Solr schema at a high level. For details about all the options and Solr schema settings, see the Solr wiki.

Wikipedia Sample Schema Elements
The sample schema.xml for the Wikipedia demo represents a typical schema. It specifies a tokenizer that determines the parsing of the wiki text. The set of fields specifies what Solr indexes and stores. In this example, these name, body, title, and date fields are indexed.

```xml
<schema name="wikipedia" version="1.1">
  <types>
    <fieldType name="string" class="solr.StrField"/>
    <fieldType name="text" class="solr.TextField">
      <analyzer><tokenizer class="solr.WikipediaTokenizerFactory"/></analyzer>
    </fieldType>
  </types>
  <fields>
    <field name="id" type="string" indexed="true" stored="true"/>
    <field name="body" type="text" indexed="true" stored="true"/>
    <field name="date" type="string" 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>
```

The example schema.xml meets the requirement to have a unique key and no duplicate rows. The unique key maps to the row key and is necessary for DSE to route documents to cluster nodes. This unique key is like a primary key in SQL. The last element in the schema.xml example designates that the unique key is `id`.

**Checking a Schema**

After creating a schema and indexing documents, you can check that the Solr index is working by using the Solr Admin tool in this location:

```
http://hostname/solr/{keyspace}.{columnfamily}/admin/
```

If the tool appears, the index is working. The tool looks something like this:
About Column Family Metadata

After indexing the Wikipedia articles, Cassandra columns in the column family contain metadata corresponding to the fields listed in the demo schema. The output of the CLI command, DESCRIBE wiki, shows this metadata:

Column Name: body
  Validation Class: org.apache.cassandra.db.marshal.UTF8Type
  Index Name: wiki_solr_body_index
  Index Type: CUSTOM
  Index Options: {class_name=com.datastax.bdp.cassandra.index.solr.SolrSecondaryIndex}

Column Name: date
  Validation Class: org.apache.cassandra.db.marshal.UTF8Type
  Index Name: wiki_solr_date_index
  Index Type: CUSTOM
  Index Options: {class_name=com.datastax.bdp.cassandra.index.solr.SolrSecondaryIndex}

Column Name: name
  Validation Class: org.apache.cassandra.db.marshal.UTF8Type
  Index Name: wiki_solr_name_index
  Index Type: CUSTOM
  Index Options: {class_name=com.datastax.bdp.cassandra.index.solr.SolrSecondaryIndex}

Column Name: solr_query
  Validation Class: org.apache.cassandra.db.marshal.UTF8Type
  Index Name: wiki_solr_solr_query_index
  Index Type: CUSTOM
  Index Options: {class_name=com.datastax.bdp.cassandra.index.solr.SolrSecondaryIndex}

Column Name: title
  Validation Class: org.apache.cassandra.db.marshal.UTF8Type
  Index Name: wiki_solr_title_index
  Index Type: CUSTOM
Column metadata matches each field in the schema except the id field because id is the unique key.

The column metadata example shows some of the Cassandra Validator types in the Validation Class attribute. The Solr types map to Cassandra validator types as shown in this table:

<table>
<thead>
<tr>
<th>Solr Type</th>
<th>Cassandra Validator</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>StringField</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>All Others</td>
<td>UTF8Type</td>
</tr>
</tbody>
</table>

Using Dynamic Fields instead of Composite Columns

You can use Solr dynamic fields for pattern matching on a wildcard instead of using composite columns, which are not supported. The number of dynamic fields allowed for a particular row is 1024. Adding the following element to the schema will index anything with the column name that ends with -tag.

```xml
<dynamicField name="*-tag" type="string" indexed="true"/>
```

When you use the dynamicField element, DSE Search adds a special solr field, _dynFld, to the index, so you can search for rows that have columns X, Y and Z.

To learn more about the Solr schema, see the well-documented sample Solr schema file.

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 Existing Solr Clients

All existing Solr clients work with DSE 2.0 and later. If you have an existing Solr application, and you want to use DSE, it is straightforward. Create a schema, then import your data and query using your existing Solr tools. The Wikipedia demo is built and queried using Solrj. The query is done using pure Ajax. No Cassandra API is used for the demo.

Integration of Solr Queries into Cassandra API

Assuming you have set up DSE Search, and have data indexed in Solr from a column family, you can include a solr_query expression to CQL that takes advantage of the DSE Search hooks into the Solr API. This capability offers extensive query options, such as fuzzy matching.
The solr_query value supports any Lucene syntax. You can also use any Thrift API, such as Pycassa or Hector. Pycassa supports secondary indexes. You can use secondary indexes in Pycassa just as you use the Solr_query expression in DSE Search.

**Querying Search Results Using CQL**

You can use the CQL select statement to retrieve Solr data.

**Synopsis**

```cql
SELECT [FIRST <n>] [REVERSED] <select expression>
FROM <column family>
[USING <consistency>]
[WHERE solr_query = '<search expression>' [LIMIT <n>]]
```

The `select expression` syntax is:

```cql
{ <start_of_range> .. <end_of_range> | * } | COUNT(* | 1)
```

A `SELECT` expression reads one or more records from a Cassandra column family and returns a result-set of rows. Each row consists of a row key and a collection of columns corresponding to the query.

Unlike the projection in a SQL SELECT, there is no guarantee that the results will contain all of the columns specified because Cassandra is schema-optional. An error does not occur if you request non-existent columns. In a production environment that uses a mixed workload cluster, you must search using the LOCAL_QUORUM consistency, as described in the Data Consistency in DSE Search article.

**Example**

To query the Wikipedia demo search results:

1. Connect to the Cassandra Query Language (CQL) shell program. On the Mac, for example:
   ```sql
cd <install_location>/bin
cqlsh localhost
   ```

2. Use the wiki keyspace and include the solr_query expression in a select statement to find the titles in the solr column family that begin with the letters `natio`:
   ```sql
   use wiki;
   SELECT title FROM solr USING CONSISTENCY local_quorum
   WHERE solr_query='title:natio*';
   ```

The query output appears:

```
title
Bolivia national football team 2002
List of French born footballers who have played for other national teams
Lithuania national basketball team at Eurobasket 2009
Bolivia national football team 2000
Kenya national under-20 football team
Bolivia national football team 1999
Israel men's national inline hockey team
Bolivia national football team 2001
```

**Querying Multiple Column Families**
To map multiple Cassandra column families to a single Solr core, use the Solr API. Specify multiple column families using the shards parameter. For example:

```
http://<host>:<port>/solr/<keyspace1>.<cf1>/select?q=*:*&shards=<host>:+<port>/solr/<keyspace1>.<cf1>,<host>:+<port>/solr/<keyspace2>.<cf2>
```

Using the Solr API, you can query multiple column families simultaneously if they have same schema.

**Expanding a DSE Search Column**

You can update a DSE Search column to set a column expiration date using CQL. Eventually, this action causes removal of the column from the database.

To set a DSE Search column to expire, add a field named `ttl_expire` to the schema. Next, update the column using CQL to set the time-to-live (TTL) option. The following section shows you the step-by-step procedure.

**To expire a DSE Search column**

This procedure builds upon the Wikipedia demo to expire a DSE Search column.

1. Make the `wikipedia demo` directory your current directory. Modify the sample schema.xml file of the Wikipedia demo to add the `ttl_expire` field:

   ```xml
   <field name="ttl_expire" type="string" indexed="true" stored="true"/>
   ```

2. Post the schema and Solr configuration file for the Wikipedia demo by rerunning the demo script. On Linux, for example:

   ```
sudo ./1-add-schema.sh
   ```

3. Index the articles contained in the `wikipedia-sample.bz2` file in the demo directory. For example:

   ```
sudo ./2-index.sh --wikifile wikipedia-sample.bz2
   ```

   Three thousand articles load.

4. Start cqlsh.

**To test expiration of a DSE Search column**

1. On the cqlsh command line, use the wiki keyspace, and then alter the Solr column to set `gc_grace_seconds` to 0.

   ```
   USE wiki;
   ALTER TABLE solr WITH gc_grace_seconds = 0;
   ```

   By setting `gc_grace_seconds` to 0, the column will be removed as soon as the TTL seconds expire.

2. Use the CQL UPDATE command to update, or create if the column doesn't exist, the Solr column. For example, set TTL values on two, non-existent rows.

   ```
   UPDATE solr USING TTL 10
   SET title='testtitle', body='solr body',
   WHERE KEY='key1';
   ```

   ```
   UPDATE solr USING TTL 3600
   SET title='testtitle2', body='solr body',
   WHERE KEY='key2';
   ```
3. After 10 seconds, query the database to check that the column entitled testtitle was removed from the database, but the column entitled testtitle2 has not yet been removed.

```sql
SELECT * FROM solr WHERE solr_query='title:testtitle';
SELECT * FROM solr WHERE solr_query='title:testtitle2';
```

The first query returns no results after 10 seconds. The second query returns the key2 if an hour (3600 seconds) has not elapsed.

**Using Solr and Hadoop in a cluster**

A common question is how to use real-time (Cassandra), analytics (Hadoop), or search (Solr) nodes in the same cluster. To mix workloads in a cluster, you need to segregate the real-time, analytics, or search nodes into separate data centers.

*Note*

Within the same data center, attempting to run Solr on some nodes and real-time queries or analytics on other nodes does not work.

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

A DSE data center (DC) can be physical or virtual. In this diagram, nodes in data centers 1 and 2 (DC 1 and DC 2) run a mix of:

- Real-time queries (Cassandra and no other services)
- Analytics (Cassandra and Hadoop)

Data centers 3 and 4 (DC 3 and DC 4) are dedicated to search.

Using separate data centers for different types of nodes, you can make some of your DSE nodes handle search while others handle MapReduce, or just act as ordinary Cassandra nodes. Cassandra ingests the data, Solr indexes the data, and you run MapReduce against that data, all in one cluster without having to do any manual extract, transform, and load (ETL) operations. Cassandra handles the replication and isolation of resources.

The Solr nodes run HTTP and hold the indexes for the column family data. If a Solr node goes down, the commit log replays the Cassandra inserts, which correspond to Solr inserts, and the node is restored automatically.

**Deploying multiple data centers**

To set up a mixed workload cluster, which is a cluster that has more than one data center to accommodate different types of nodes, see *Multiple Data Center Deployment*.

**Replicating data across data centers**
You set up replication for Solr nodes exactly as you do for other nodes in a Cassandra cluster, by creating or altering a keyspace to define the replication strategy.

You can use the pre-release CQL 3 CREATE KEYSPACE and ALTER KEYSPACE statements to set up replication.

**Data center operations**

Common DSE Search/Solr operations are:

- Adding a new Solr node
- Modifying Solr data
- Updating individual fields in a Solr document
- Decommissioning and repairing a node
- Rebuilding an index
- Managing the Location of Solr Data
- Changing the Solr Connector Port

**Adding a new Solr node**

To increase the number of nodes in a Solr cluster, you can add or bootstrap a DSE node to the cluster. If you want to increase capacity of your search, you need to bootstrap the node, then optionally, rebalance the cluster. To bootstrap a Solr node, use the same method you use to bootstrap a Cassandra node. Using the default DSESimpleSnitch automatically puts all the Solr nodes in the same data center. Use OpsCenter Enterprise to rebalance the cluster.

**Modifying Solr data**

When you insert data into Cassandra, it shows up in Solr. When you add data to Solr, it shows up in Cassandra. You can use any Solr API to write data to Solr, however, the native Solr HTTP REST API is recommended. Writes are durable. A Solr API client writes data to Cassandra first, and then Cassandra updates secondary indexes.

To modify or remove data from a Solr node use the Cassandra Query Language (CQL), the Command Line Interface (CLI), or Solr APIs. By virtue of updating a field in Cassandra, the data in Solr is updated. When you update the column family, the Solr document is updated.

**Updating individual fields in a Solr document**

You can use the Solr API to insert into, modify, or delete data from a Solr node. When using the Solr API to change a document, the entire document is updated. Using DSE Search, you can update an individual field. After writing the modifications to the Solr document, use a URL in the following format to update the document:

```bash
curl http://<host>[:<port>]/{solr/<keyspace>.<column family>/update?replacefields=false
```

The Solr convention is to use curl for issuing update commands instead of using a browser.

When you use CQL or CLI to update a field, DSE Search implicitly sets replacefields to false and updates individual fields in the Solr document.

**Warning about using optimize**

Do not use the optimize command. Using the optimize command in a URL can cause nodes to fail.

**Decommissioning and repairing a node**

You can decommission and repair a Solr node in the same manner as you would a Cassandra node.
**Rebuilding an index**

The dsetool is equipped to rebuild a Solr index from existing Cassandra data. To rebuild a corrupted index:

1. Run `nodetool drain`.
2. Shut down the node.
3. Delete the Solr index directory for the bad column family. The Solr index directory path is `<Cassandra data directory>/solr.data/<keyspace_name>.<column-family-name>`.
4. Restart the node.
5. Use this command to rebuild the index:
   ```
   ./dsetool rebuild_indexes <keyspace> <columnfamily>
   ```

**Managing the Location of Solr Data**

Solr has its own set of data files. Like Cassandra data files, you can control where the Solr data files are saved on the server. By default, the data is saved in `<Cassandra data directory>/solr.data`. You can change the location from the `<Cassandra data directory>` to another directory, from the command line. For example:

```
cassandra -s -Ddse.solr.data.dir=/opt
```

In this example, the data in solr.data is saved in the /opt directory.

**Accessing the Validation Log**

DSE Search stores validation errors that arise from non-indexable data sent from non-Solr nodes in this log:

```
/var/log/cassandra/solrvalidation.log
```

For example, if a Cassandra node that is not running Solr puts a string in a date field, an exception is logged for that column when the data is replicated to the Solr node.

**Changing the Solr Connector Port**

To change the Solr port from the default, 8983, change the `http.port` setting in the `catalina.properties` file installed with DSE in `<dse-version>/resources/tomcat/conf`.

**Tuning DSE Search Performance**

DataStax Enterprise server is able to support real-time, analytic, and search workloads in the same cluster of machines with smart workload isolation. This ensures that workloads do not compete with the other for data or computing resources and helps deliver consistently high performance. In the event of a performance degradation, high memory consumption, or other problem with DataStax Enterprise Search nodes, try:

- Using column family compression
- Configuring the `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

**Using Column Family Compression**
Search nodes typically engage in read-dominated tasks, so maximizing storage capacity of nodes, reducing the volume of data on disk, and limiting disk I/O can improve performance. In Cassandra 1.0 and later, you can configure data compression on a per-column family basis to optimize performance of read-dominated tasks.

Configuration affects the compression algorithm for compressing SSTable files. For read-heavy workloads, such as those carried by Enterprise Search, Snappy compression is recommended. Compression using the Snappy compressor is enabled by default when you create a column family in Cassandra 1.1 and later. You can change compression options using CQL. Developers can also implement custom compression classes using the org.apache.cassandra.io.compress.ICompressor interface. You can configure the compression chunk size for read/write access patterns and the average size of rows in the column family.

**Setting the High-Performance Update Handler**

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:

```xml
<!-- The default high-performance update handler -->
<updateHandler class="solr.DirectUpdateHandler2">
  <autoSoftCommit>
    <maxTime>1000</maxTime>
  </autoSoftCommit>
</updateHandler>
```

This example uses the maxTime update handler option. The update handler options enable near real-time performance and trigger a soft commit of data automatically, so checking synchronization of data to disk is not necessary. Data durability is maintained by letting cassandra do hard commits along with Cassandra memtable flushes. This table describes both update handler options.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxDocs</td>
<td>No default</td>
<td>Maximum number of documents to add since the last soft commit before automatically triggering a new soft commit.</td>
</tr>
<tr>
<td>maxTime</td>
<td>1000</td>
<td>Maximum expired time in milliseconds between the addition of a document and a new, automatically triggered soft commit.</td>
</tr>
</tbody>
</table>

For more information about the update handler and modifying SolrConfig.xml, see the Solr documentation.

**Changing the Stack Size and Memtable Space**

Some Solr users have reported that increasing the stack size improves performance under Tomcat. To increase the stack size, uncomment and modify the default -Xss128k setting in the cassandra-env.sh file. Also, decreasing the memtable space to make room for Solr caches might improve performance. Modify the memtable space using the memtable_total_space_in_mb property in the cassandra.yaml file.

**Managing Caching**

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>
</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.

- `maxcachemb`
  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.

   
   ```sh
   ./nodetool -h localhost ring
   
   Note
   The data center names, DC1 and DC2 in this example, must match the data center name configured for your snitch.
   
   2. Start CQL on the DSE command line and create a keyspace that specifies the number of replicas you want.

   ```cql
   CREATE KEYSPACE test
   WITH strategy_class = 'NetworkTopologyStrategy'
   AND strategy_options:DC1 = 1
   AND strategy_options:DC2 = 3;
   
   The strategy options set the number of replicas in data centers, one replica in data center 1 and three in data center 2. For more information about adding replicas, see Choosing Keyspace Replication Options.

**Changing the Replication Factor for a Solr Keyspace**

When you post the solrconfig.xml and schema.xml, DSE Search creates a keyspace and column family in Cassandra. The default replication factor for this keyspace is 1. If you need more than one replica of the keyspace in your cluster, you need to update the replication factor of the keyspace.

The following procedure builds on the wikipedia demo example. Assume the solrconfig.xml and schema.xml files have already been posted using wiki.solr in the URL, which creates a keyspace named wiki that has a default replication factor of 1. You want three replicas of the keyspace in the cluster, so you need to update the Solr keyspace replication factor.

**To change the Solr keyspace replication factor**

1. Check the name of the data center of the Solr/Search nodes.

   ```sh
   ./nodetool -h localhost ring
   
   The output tells you that the name of the data center for your node is, for example, datacenter1.
2. Use the Cassandra CLI to change the replication factor of the keyspace. Set a replication factor of 3.

cassandra-cli -host localhost -port 9160

[default@unknown] UPDATE KEYSPACE wiki
   WITH strategy_options = {Solr:3};

If you have data in a keyspace and then change the replication factor, run `nodetool repair` to avoid having missing data problems or data unavailable exceptions.

**Managing the Consistency Level**

Consistency refers to how up-to-date and synchronized a row of data is on all of its replicas. Like Cassandra, DSE-Search extends Solr by adding an HTTP parameter, `cl`, that you can send with Solr data to tune consistency. The format of the URL is:

http://<host>:<port>/solr/<keyspace>.<column family>/update?cl=ONE

The `cl` parameter specifies the consistency level of the write in Cassandra. The default consistency level is QUORUM, but you can change the default using the “search.consistencylevel.write” system property.

**Frequently Asked Questions**

1. Can you run Solr and Hadoop on the same node?

   Only in development environments. In production environments, running Solr and Hadoop on the same node will cause a failure.

2. How do you add a file to the Solr index?

   HTTP post the file using this URL format:

   http://<host>:<port>/solr/resource/<keyspace>.<columnfamily>/<filename>.<ext>

3. Why does the Solr schema need to have a unique key?

   To route cluster documents.

4. Does DSE Search support composite columns?

   No.

5. How can you query data in Solr?

   Cassandra Command Line Interface (CLI), Cassandra Query Language (CQL) library, the CQLsh tool, existing Solr APIs, and Thrift APIs.

6. As an existing Solr user, how much trouble is it to transition to DSE Search?

   Making the change is straightforward. If you have an existing Solr application, and you want to use DSE 2.0 or later, create a schema, then import your data and query using your existing Solr tools.

7. Is it possible to search data using fuzzy matching?

   Assuming the data you want to search is indexed in Solr from a column family, you can include a solr_query expression to CQL that offers extensive query options, such as fuzzy matching.

8. When you search for non-existent Solr data, why doesn’t an error occur?

   Unlike the projection in a SQL SELECT, there is no guarantee that the results will contain all of the columns specified because Cassandra is schema-optional. An error does not occur if you request non-existent columns. In a production environment, you must search using the LOCAL_QUORUM consistency.

9. Can you run Hadoop on some nodes and Solr on others in a virtual data center?

   All nodes in that dedicated virtual data center must be running Solr. Attempting to run Solr on some nodes and Cassandra or Hadoop on others within the same virtual data center does not work.
10. How do you increase Solr capacity in a cluster?
Bootstrap one or more additional nodes, using the bootstrap method.

11. How do you modify or remove data from a Solr node?
Use CQL, CLI, or Solr APIs as described in *Inserting into, Modifying, and Deleting Data from a Solr Node*.

12. How do you rebuild an index?
Use the dsetool and the procedure in *Rebuilding an Index*.

13. How do you change the Solr port?
Change http.port in the catalina.properties file in `<dse-version>/resources/tomcat/conf`.

14. How do you tweak performance of DSE Search?
You need to configure the solrconfig.xml to use near real-time capabilities in Solr by setting the default high-performance update handler flag in the solrconfig.xml.

15. How do you verify that your Solr index is working?
Use the Solr Admin tool.
DSE Command Reference

You can issue dse commands from the bin directory of the DataStax Enterprise installation.

**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 background.</td>
<td>link to example</td>
</tr>
<tr>
<td>dse cassandra</td>
<td>-s</td>
<td>Start up a DSE Search/Solr node in the background.</td>
<td>link to example</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 foreground.</td>
<td>none</td>
</tr>
<tr>
<td>dse cassandra</td>
<td>-f -t</td>
<td>Start up an analytics node in the 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 foreground.</td>
<td>none</td>
</tr>
<tr>
<td>dse cassandra-stop</td>
<td>-p &lt;pid&gt;</td>
<td>Stop the DataStax Enterprise process number pid.</td>
<td>link to example</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 standard output.</td>
<td>link to example</td>
</tr>
<tr>
<td>dse hadoop</td>
<td>fs -put &lt;URI&gt;</td>
<td>Load the Pig sample data file into the Cassandra File System.</td>
<td>link to example</td>
</tr>
<tr>
<td>dse hive</td>
<td></td>
<td>Start the Hive client.</td>
<td>link to example</td>
</tr>
<tr>
<td>dse hive</td>
<td>--service &lt;name&gt;</td>
<td>Start Hive by connecting through the JDBC driver.</td>
<td>link to example</td>
</tr>
<tr>
<td>dse mahout</td>
<td></td>
<td>Describe Mahout commands.</td>
<td>link to example</td>
</tr>
<tr>
<td>dse mahout</td>
<td>&lt;mahout command&gt; &lt;options&gt;</td>
<td>Run the Mahout command.</td>
<td>link to example</td>
</tr>
</tbody>
</table>
### Configuring Firewall Port Access

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

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Public Facing Ports</strong></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>SSH (default)</td>
</tr>
<tr>
<td><strong>DataStax Enterprise Specific</strong></td>
<td></td>
</tr>
<tr>
<td>8012</td>
<td>Hadoop Job Tracker client port. The Job Tracker listens on this port for job submissions and communications from task trackers; allows traffic from each Analytics node in a cluster.</td>
</tr>
<tr>
<td>8983</td>
<td>Solr port and Demo applications website port (Portfolio, Search, Search log)</td>
</tr>
<tr>
<td>50030</td>
<td>Hadoop Job Tracker website port. The Job Tracker listens on this port for HTTP requests. If initiated from the OpsCenter UI, these requests are proxied through the opscenterd daemon; otherwise, they come directly from the browser.</td>
</tr>
<tr>
<td>50060</td>
<td>Hadoop Task Tracker website port. Each Task Tracker listens on this port for HTTP requests coming directly from the browser and not proxied by the opscenterd daemon.</td>
</tr>
<tr>
<td><strong>OpsCenter Specific</strong></td>
<td></td>
</tr>
<tr>
<td>8888</td>
<td>OpsCenter website. The opscenterd daemon listens on this port for HTTP requests coming directly from the browser.</td>
</tr>
<tr>
<td><strong>Inter-node Ports</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Cassandra Specific</strong></td>
<td></td>
</tr>
<tr>
<td>1024+</td>
<td>JMX reconnection/loopback ports. See description for port 7199.</td>
</tr>
<tr>
<td>7000</td>
<td>Cassandra inter-node cluster communication.</td>
</tr>
<tr>
<td>7199</td>
<td>Cassandra JMX monitoring port. After the initial handshake, the JMX protocol requires that the client reconnects on a randomly chosen port (1024+).</td>
</tr>
<tr>
<td>9160</td>
<td>Cassandra client port (Thrift). OpsCenter agents makes Thrift requests to their local node on this port. Additionally, the port can be used by the opscenterd daemon to make Thrift requests to each node in the cluster.</td>
</tr>
<tr>
<td><strong>DataStax Enterprise Specific</strong></td>
<td></td>
</tr>
<tr>
<td>9290</td>
<td>Hadoop Job Tracker Thrift port. The Job Tracker listens on this port for Thrift requests coming from the opscenterd daemon.</td>
</tr>
<tr>
<td><strong>OpsCenter Specific</strong></td>
<td></td>
</tr>
</tbody>
</table>
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.
   ```bash
   cd /etc/yum.repos.d
   ```
2. Download the public-yum-ol6.repo package from the repository.
   ```bash
   ```
3. Check that glibc.i686 is ready for installation and install it.
   ```bash
   yum list
   yum install glibc.i686
   ```

Locations of Files

This section lists the locations of configuration files, packaged install directories, and tarball install directories.

Locations of Configuration Files

The configuration files, such as `cassandra.yaml`, are located in the following directories:

- DataStax Enterprise packaged installs: `/etc/dse/cassandra`
- DataStax Enterprise binary installs: `<install_location>/resources/cassandra/conf`

Packaged Install Locations

The DataStax Enterprise CentOS/RHEL/Oracle Linux and Debian/Ubuntu packages are installed into the following directories:

Cassandra Directories

- `/var/lib/cassandra` (Cassandra and CassandraFS data directories)
- `/var/log/cassandra`
- `/var/run/cassandra`
- `/usr/share/dse/cassandra` (Cassandra environment settings)
- `/usr/share/dse/cassandra/lib`
- `/usr/share/dse-demos` (Portfolio, Solr, Sqoop)
- `/usr/bin`
- `/usr/sbin`
Installing glibc on Oracle Linux 6.x and Later

- /etc/dse/cassandra  (Cassandra configuration files)
- /etc/init.d
- /etc/security/limits.d
- /etc/default/

**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)

**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)
- /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)
Starting and Stopping DataStax Enterprise

After you have installed and configured DSE on one or more nodes, you are ready to start your cluster starting with the seed nodes. In a mixed-workload DSE cluster, you must start the analytics seed node first.

Packaged installations include startup and stop scripts for running DSE as a service. Binary packages do not.

- **Starting DataStax Enterprise as a Stand-Alone Process**
- **Starting DataStax Enterprise as a Service**
- **Stopping a DataStax Enterprise Node**

**Starting DataStax Enterprise as a Stand-Alone Process**

If running a mixed-workload cluster, determine which nodes to start as analytics, Cassandra, and search nodes. Begin with the seed nodes first - analytics seed node, followed by the Cassandra seed node - then start the remaining nodes in the cluster one at a time. For additional information, see *Multiple Data Center Deployment*.

To start DataStax Enterprise as a stand-alone process:

From the install directory:

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

**Note**

DataStax does not support using the `-t` search tracker option in combination with the `-s` option to mark the node for Hadoop analytics and search.

- To check that your ring is up and running (from the install directory):
  
  `$ bin/nodetool ring -h localhost`

**Starting DataStax Enterprise as a Service**
Packaged installations provide startup scripts in `/etc/init.d` for starting DSE as a service.

For mixed-workload clusters, nodes that are Cassandra-only can simply start the DSE service (skip step 1).

**To start DataStax Enterprise as a service:**

1. Edit the `/etc/default/dse` file, and then add the appropriate line to this file, depending on the type of node you want:
   - `HADOOP_ENABLED=1` - Designates the node as DataStax Enterprise analytics and starts the Hadoop Job Tracker and Task Tracker services.
   - `SOLR_ENABLED=1` - Starts the node as DSE Enterprise Search. See *Running the Demo*.

   **Note**
   DataStax does not support using the SOLR_ENABLED and HADOOP_ENABLED options to mark the same node for both search and Hadoop analytics.

2. Start the DSE service:
   ```bash
   sudo service dse start
   ```

3. To check if your cluster is up and running:
   ```bash
   nodetool ring -h localhost
   ```

   On RHEL and CentOS, the DSE service runs as a `java` process. On Debian and Ubuntu systems, the DSE service runs as a `jsvc` process.

**Stopping a DataStax Enterprise Node**

To speed up the restart process, before stopping the dse service or the Cassandra or DataStax Enterprise process, run `nodetool drain`. This step writes the current memtables to disk. When you restart the node, Cassandra does not need to read through the commit log. If you have durable writes set to false, which is unlikely, there is no commit log and you must drain the node to prevent losing data.

**To stop the service on a node:**

```bash
nodetool drain -h <host name>
```

**To stop the stand-alone process on a node:**

To stop a node, find the Cassandra or DataStax Enterprise Java process ID (PID) and kill the process using its PID number (using sudo if necessary). For example:

```bash
$ nodetool drain -h <host name>
$ ps auwx | grep dse
$ kill <pid>
```

**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:
Troubleshooting Tips


**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.

**Confusing Solr Error Messages**

- When uploading the schema.xml, Solr issues an error message if the solrconfig.xml doesn't exist in the Solr core. You can ignore this message.

  For example:

  Posted solrconfig.xml to http://localhost:8983/solr/resource/
  demo.solr/solrconfig.xml
  SUCCESS
  Posted schema.xml to http://localhost:8983/solr/resource/
  demo.solr/schema.xml

- When uploading the solrconfig.xml, Solr issues an error message if the schema.xml hasn't been uploaded. You can ignore this message. After uploading the schema and the configuration file, you should see a SUCCESS message.

- If you upload a malformed xml files, Solr issues an error message. Do not ignore these messages. Fix the xml.
Release Notes

- **DataStax Enterprise 2.2.3**
- **DataStax Enterprise 2.2.2**
- **DataStax Enterprise 2.2.1**
- **DataStax Enterprise 2.2**

**DataStax Enterprise 2.2.3**

This release includes features that improve DSE Search/Solr operations.

- Two features for performing an anti-entropy node repair on a subrange of data instead of all the data in a keyspace.
  - A new dsetool command, list_subranges, estimates subranges of data in a keyspace based on a specified number of rows.
  - New nodetool repair options, start token (-st) and end token (et), designate subranges of data for distribution within those ranges.

  Using these commands, DSE Search now performs a partial re-index instead of a full re-index of Solr data after an anti-entropy repair.

- A new nodetool error code. When a replica node is dead and repair cannot proceed, nodetool sends an error status code to standard output.

**New dsetool list_subranges command**

The new dsetool command syntax for listing subranges of data in a keyspace is:

```
dsetool list_subranges <keyspace> <table> <rows per subrange> <start token> <end token>
```

- `<rows per subrange>` is the approximate number of rows per subrange.
- `<start token>` is the start range of the node.
- `<end token>` 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 tokens used on that node.

**Example**

```
dsetool list_subranges Keyspace1 Standard1 10000 113427455640312821154458202477256070485 0
```

The dsetool is located in `<install_location>/bin` on Linux platforms.

**Output**

The output lists the subranges to use as input to the nodetool repair command. For example:

<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>
**New nodetool repair command options**

The start token (-st) and end token (-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 113427455640312821154852047725670485 -et 132425442279562452127151664615147681247
nodetool repair Keyspace1 Standard1 -st 1324254422795624521227151664615147681247 -et 151409576048389227347757997936583470460
nodetool repair Keyspace1 Standard1 -st 151409576048389227347757997936583470460 -et 0
```

These commands begins an anti-entropy node repair from the start token to the end token.

**Resolved issue**

The following issue has been resolved:

- The SliceFromReadCommand assertion has been removed: assert maxLiveColumns <= count; (CASSANDRA-5284).

**Unresolved issues**

Issues listed in *DataStax Enterprise 2.2* to *DataStax Enterprise 2.2.2* that have not been listed as resolved have not yet been fixed.

---

**DataStax Enterprise 2.2.2**

This release contains the following changes:

- This release updates Cassandra to Cassandra 1.1.9. See 1.1.9 CHANGES.txt.
- Fix for CASSANDRA-5168: Word count example fails with InvalidRequestException.

**Issues**

The Cassandra log4j appender doesn't support multiple hosts. (DSP-1601)

**DataStax Enterprise 2.2.1**

This release contains the following changes:

- Updates Cassandra to Cassandra 1.1.6. See 1.1.6 CHANGES.txt.
- Fixes issues, including the following noteworthy ones:
  - **cassandra.yaml and cassandra-env.sh** - Corrects Issue 1 in 2.2 Issues. DSP-1053
  - **Oracle JDK/JRE 6 update 34-35 can now be used.** - Corrects Issue 4 in 2.2 Issues.
  - **Cassandra CLI error** - Cannot update keyspace without first issuing `use <keyspace>;`. DSP-833
  - **Hive error** - Hive returns incorrect results (missing columns). DSP-1076
  - **Hinted Handoff fails** - Hinted handoff fails to deliver hints. Cassandra-4772
Includes two patches to Cassandra 1.1.6. These patches are:

- Patch to fix method get_paged_sliced: Cassandra-4816
- Patch to fix "Keys must not be empty" assertion error: Cassandra-4832

**Issues**

This release has the following issues:

- Issue 1: In this release, an exception occurs under either of these conditions:
  - Dropping a Solr keyspace and then recreating it. DSP-1126
  - Updating the schema.xml or solrconfig.xml. (The schema.xml exception is a known issue from all previous releases.) DSP-655 and DOC-62

  The workaround is to perform a rolling restart (restart each node one-at-a-time) before adding any data to the database.

  **Note**
  
  Adding data before performing the workaround can cause unpredictable problems.

After you drop a Solr keyspace or column family, Solr-specific residual data remains in memory until you perform the workaround. For example, if you drop a Solr keyspace on node 1 and search for the data on node 2, Solr returns residual data. To completely remove the residual data, you need to perform the workaround to restart all Solr nodes.

- Issue 2: DataStax Enterprise is designed for a multiple data center environment and not intended for use with the SimpleStrategy replication placement strategy. SimpleStrategy is not data center-aware. DataStax Enterprise does not work correctly using SimpleStrategy. Use NetworkTopologyStrategy. DSP-1195

- Issue 3: The nodetool repair does not completely repair a keyspace unless it is in every datacenter. CASSANDRA-5424

**DataStax Enterprise 2.2**

- Apache Cassandra 1.1.5
- Apache Hadoop 1.0.2
- Apache Hive 0.9.0
- Apache Pig 0.9.2
- Apache Solr 4.0
- Apache Thrift 0.6.1
- Apache log4j 1.2.16
- Apache Sqoop 1.4.2
- Apache Mahout 0.6

**What’s New**

DataStax Enterprise 2.2 has been enhanced in the following ways:

- **Production certified Cassandra** – DataStax Enterprise contains a fully tested, benchmarked, and certified version of Apache Cassandra that is suitable for mission-critical production deployments.
Updates Cassandra 1.0 to Cassandra 1.1.5 - In Cassandra 1.1, key improvements have been made in the areas of CQL, performance, and management ease of use.

Support for Installation on the HP Cloud - In addition to Amazon Elastic Compute Cloud, DataStax now supports installation of DataStax Enterprise in the HP Cloud environment. You can install DataStax on Ubuntu 11.04 Natty Narwhal and Ubuntu 11.10 Oneiric Ocelot.

Support for SUSE Enterprise Linux - DataStax Enterprise adds SUSE Enterprise Linux 11.4 and 11.2 to its list of supported platforms.

Improved Solr shard selection algorithm - Previously, for each queried token range, Cassandra selected the first closest node to the node issuing the query within that range. Equally distant nodes were always tried in the same order, so that resulted in one or more nodes being hotspotted and often selecting more shards than actually needed. The 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.

Capability to Set Solr Column Expiration - You can update a DSE Search column to set a column expiration date using CQL, which eventually causes removal of the column from the database.

Issues

This release has the following issues:

- Issue 1: The cassandra.yaml file in DataStax 2.2 is incomplete. Download the correct cassandra.yaml file for DataStax Enterprise 2.2 from:


Use this file to overwrite the existing cassandra.yaml file in the following location:

  - Binary Tarball Install
    <install location>/resources/cassandra/conf
  - Packaged Install
    <install location>/etc/dse/cassandra

The next release will correct this issue. DSP-1053

- Issue 2: You might experience a problem upgrading to DataStax Enterprise 2.2. You definitely will not lose data if you experience the problem. The workaround is to save keyspaces in your old installation before upgrading and validate the upgrade to ensure that keyspaces were migrated. A patch will be issued to resolve this issue. CASSANDRA-4698

- Issue 3: This release has no native support for Cassandra composite columns when using Hive, Pig, Solr, Mahout, or Sqoop components. When using these components, the columns are transposed in CQL 3 query results. It is the user's responsibility to create a user-defined function (UDF) to display the tables correctly.

- Issue 4: DataStax recommends that you use the latest version of Oracle JDK/JRE 6, but not Oracle JDK/JRE 6 updates 34-35, updates prior to 30, or JDK/JRE 7.

- Issue 5: Sometimes, under a heavy write load, Cassandra fails with an assertion error that looks something like this:

  ```java
  java.lang.AssertionError: DecoratedKey(xxx, yyy)
  != DecoratedKey(zzz, kkk) . . .
  ```

  The workaround is to disable caching using CQL.

- Issue 6: If a node has hints for a few nodes, that node delivers hints only for the first one of them. Cassandra-4772

- Issue 7: MapReduce jobs hang before completing or finishing cleanup with older versions of Hadoop (MAPREDUCE-4560, MAPREDUCE-4299). The workaround is remove the mapred.reduce.slowstart.completed.maps parameter and restart. DSP-1154
• Issue 8: The nodetool repair -pr command does not completely repair a keyspace unless the keyspace is in every datacenter. CASSANDRA-5424