# Reading Data From a Cassandra Column Family

*Enterprise Search with Solr* .................................................. 46

**About DataStax Enterprise Search** ......................................... 47

- Benefits of Using Solr in DataStax Enterprise .......................... 48
- Integration of Solr into DataStax Enterprise ............................. 48
- DataStax Enterprise to Solr Objects Map .................................. 49
- Schema and Configuration Files ............................................. 49
- About the DSE Search Resource REST API ................................. 49
- Unsupported Features ......................................................... 50
- Links to Solr and Lucene Documentation .................................. 50

**Getting Starting with DSE Search** ....................................... 50

- Starting DSE and DSE Search ............................................... 50

**Creating a Schema** ............................................................ 53

- Wikipedia Sample Schema Elements ...................................... 53
- Checking a Schema ............................................................. 54
- Wikipedia Sample Column Family Metadata ............................... 54
- Using Dynamic Fields instead of Composite Columns .................. 55

**Querying Search Results** .................................................... 56

- Using Existing Solr Clients ............................................... 56
- The Integration of Solr Queries with Cassandra APIs ................. 56
- Querying Search Results Using CQL ..................................... 56
- Querying Multiple Column Families ................................. 57

**DSE Search Management Operations** .................................... 57

- Adding a New Solr Node ..................................................... 57
- Inserting into, Modifying, and Deleting Data from a Solr Node .... 58
- Updating Individual Fields in a Solr Document ........................ 58
  
  Warning about using *optimize* ............................................. 58

- Increasing Read Performance by Adding Replicas ......................... 58
- Decommissioning and Repairing a Node .................................. 58
- Rebuilding an Index ............................................................ 59
- Managing the Location of Solr Data ....................................... 59
- About the Validation Log ..................................................... 59
- Changing the Solr Connector Port .......................................... 59
- Tuning Performance ............................................................ 59

**Changing the Stack Size and Memtable Space** .......................... 60

**Frequently Asked Questions** ............................................. 60

**Moving Data to DataStax Enterprise Using Sqoop** ....................... 61

**About Sqoop** ................................................................. 61
Getting Started .......................................................... 62
Running the Sqoop Demo ............................................... 62
Demo Requirements .................................................. 62
Step-by-Step Procedure .............................................. 62
Migrating Data to a Cassandra Column Family .................. 64
  Cassandra Options to the Import Command ....................... 64
  About the Generated Sqoop JAR File .............................. 65
  Sqoop Output ...................................................... 65
Checking Imported Data ............................................. 66
  Using CQL to Check Imported Data ................................. 66
  Validating Import Results in a Cluster ........................... 66
Getting Information about the Sqoop Command .................. 67
  Cassandra arguments .............................................. 67
  Other Arguments .................................................. 68
Using the Log4j Appender ............................................ 71
Configuring Logging Using Cassandra Log4j Appender ............ 71
  Log Levels ......................................................... 71
  Log Messages ..................................................... 71
  Storing log4j Messages in a Column Family ....................... 72
  Example .......................................................... 73
Log4j Search Demo .................................................. 74
  Running the Demo ............................................... 74
Configuring Multiple Data Centers Quick Start .................... 75
  Configuration Example (Tarball Installation) .................... 75
Balancing the Data Center Nodes ................................... 77
Frequently Asked Questions ......................................... 77
More Information About Configuring Data Centers ................. 77
Elastic Workload Re-provisioning ................................... 78
  Setting up a Node for Re-provisioning ........................... 78
  Delegating the Snitch ............................................ 79
  Re-provisioning Packaged RHEL and Debian Installations ....... 79
  Re-provisioning Tarball Installations ............................ 79
DataStax Enterprise Release Notes .................................. 80
  DataStax Enterprise 2.0 ........................................... 80
    What's New ..................................................... 80
    Other issues .................................................. 80
  DataStax Enterprise 1.0 ........................................... 80
    DataStax Enterprise 1.0.2 ...................................... 81
    DataStax Enterprise 1.0.1 ...................................... 81
Getting Started with DataStax Enterprise

DataStax Enterprise is a big data platform that integrates real-time, analytic, and enterprise search to help you respond quickly to user and business demands. It leverages Apache Cassandra, Apache Hadoop, and Apache Solr to shift your focus from your data infrastructure to using your data strategically. Additionally, with the Apace Sqoop integration, you can transfer data from relational and non-relational data source into Cassandra and Hadoop.

About DataStax Enterprise

DataStax Enterprise is a big data platform that integrates real-time, analytic, and enterprise search features to help you respond quickly to user and business demands. It leverages Apache Cassandra, Apache Hadoop, and Apache Solr to shift your focus from grappling with your own infrastructure to using your data strategically.

Key Features of DataStax Enterprise

The key features of DataStax Enterprise include:
No Single Point of Failure - The Hadoop Distributed File System (HDFS) utilizes a master/slave architecture. The NameNode is the entry point into the cluster and it stores all of the configuration metadata about the cluster. If the NameNode fails, the Hadoop system is down. With DataStax Enterprise, all nodes are peers: data files can be loaded through any node in the cluster, and any Analytic node can perform the role of JobTracker for MapReduce jobs. Additionally in DataStax Enterprise, Solr is fully fault-tolerant and has no single point of failure.

Streamlined Setup and Operations - In Hadoop, you have to set up different configurations depending on the mode you want to run in: stand-alone mode or pseudo-distributed mode for a single node setup, or cluster mode for a multi-node configuration. Moving from one mode to another requires multiple configuration steps. In DataStax Enterprise, there is only one mode (cluster mode). It does not matter if it is a cluster has one or one hundred nodes, the configuration is the same.

Hadoop MapReduce capabilities using CassandraFS - CassandraFS is an HDFS-compatible storage layer inside of Cassandra. By replacing HDFS with CassandraFS, you can leverage your current MapReduce jobs on Cassandra's peer-to-peer, fault-tolerant, and scalable architecture.

Analytics Without ETL - When using DataStax Enterprise, it is possible to run MapReduce jobs directly against your data in Cassandra. You can even perform real-time and analytic workloads at the same time without one workload affecting the performance of the other. In a DataStax Enterprise cluster, you can start some nodes as Hadoop analytics nodes and some nodes as pure Cassandra real-time nodes. With this split-workload configuration, data is automatically replicated between the Cassandra real-time nodes and the Hadoop analytics nodes.

Enterprise Search Capabilities - DataStax Enterprise is fully integrated with Apache Solr to provide ad-hoc querying on the data; full-text search; hit highlighting; multiple search attributes; search rich documents, such as PDF and Microsoft Word; geo-spatial search, and more. Additionally, near real-time indexing can be performed to manage real-time, analytic, and enterprise search features within a single integrated platform.

Hive Support - Hive is a data warehouse system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop compatible file systems. DataStax Enterprise allows you to use any JDBC compliant user interface to connect to and work with Hive from within the server. In regular Hive, its metastore is a stand-alone database that requires multiple configuration steps to make it a database instance that can be shared by multiple Hive clients. In DataStax Enterprise, the Hive metastore is automatically a shared metastore that is available through any Cassandra node in the cluster without any additional configuration.

Pig Support - Pig is a high-level platform for creating MapReduce programs used with Hadoop. You use Pig for analyzing large data sets. When using Pig with DataStax Enterprise, all jobs can be run in MapReduce mode and you can run Pig programs directly on data stored in Cassandra.

Elastic Workload Re-provisioning - DataStax Enterprise provides the ability to re-provision existing nodes to assume a different workload, such as changing a real-time node to an analytic node thereby changing the overall usage and capacity of a cluster. For example, you could change two real-time/transactional nodes to Analytic nodes during off-peak hours and then return them to the original configuration once the analytic tasks have completed.

Migration of RDBMS data - DataStax Enterprise provides easy migration of RDBMS data into the DataStax Enterprise server using Apache Sqoop. For example, you can import data from Oracle, Microsoft SQL Server, MySQL, Sybase, and DB2 RDBMS, and non-relational data sources, such as NoSQL.

Runtime Logging - DataStax Enterprise integrates Apache log4j. This Java-based logging framework provides runtime application feedback and the ability to control the granularity of log statements using an external configuration file. With 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. Also included in DataStax Enterprise is a log4j search demo that shows an example of searching and filtering log4j messages.

Full Integration with DataStax OpsCenter - DataStax OpsCenter allows you to monitor, administer, and configure one or more DataStax Enterprise clusters in one easy-to-use graphical interface. Additionally, you can perform and schedule automatic backups. Using OpsCenter, you can see detailed health and status information about multiple clusters, such as the status of Hadoop MapReduce jobs running on a cluster.
About the DataStax Enterprise Architecture

DataStax Enterprise combines Apache Cassandra with Hadoop and Solr. A DataStax Enterprise cluster can be run as a pure Hadoop MapReduce cluster (using Hadoop with Cassandra as its underlying storage) or as a combination of Hadoop analytics nodes, Cassandra real-time/transactional nodes, and Solr enterprise search nodes. The DataStax Enterprise distribution also includes the Hive and Pig MapReduce clients.

About Hadoop and MapReduce in DataStax Enterprise

Like Apache Cassandra, Apache Hadoop is an open-source project administered by the Apache Software Foundation. Hadoop consists of two key services, the Hadoop Distributed File System (HDFS) and a parallel data processing framework using a technique called MapReduce.

In DataStax Enterprise, CassandraFS replaces the Hadoop Distributed File System (HDFS). CassandraFS is compatible with Hadoop MapReduce clients, but uses a cfs keyspace in Cassandra for the underlying storage layer. CassandraFS 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.

On top of the distributed file system is the MapReduce engine, which consists of a centralized Job Tracker service. Client applications submit their MapReduce jobs to the Job Tracker. For each job submitted to the Job Tracker, a series of tasks are scheduled on the compute nodes. There is one Task Tracker service per node to handle the map and reduce tasks scheduled for that node. The Job Tracker monitors the execution and status of all of the distributed tasks that comprise a MapReduce job. In DataStax Enterprise, you must choose one node to be the Job Tracker for your MapReduce jobs, which is set by configuring the Cassandra seed node for your DataStax Enterprise cluster.

About Hive in DataStax Enterprise

DataStax Enterprise 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. The HiveQL language also allows traditional MapReduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL. In DataStax Enterprise, you can start the Hive client on any DataStax Enterprise Analytics node, define Hive data structures, and issue MapReduce queries. DataStax Enterprise Hive includes a custom storage handler for Cassandra that allows you to run Hive queries directly on data stored in Cassandra. Hive includes support for binary data and support for wide rows (up to 2 billion columns).

About Pig in DataStax Enterprise

DataStax Enterprise includes a Cassandra-enabled Pig MapReduce client. Pig is a platform for analyzing large data sets. It 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.

Pig Latin programs run in a distributed fashion on a DataStax Enterprise cluster (programs are compiled into MapReduce jobs and executed using Hadoop). When using Pig with DataStax Enterprise, all jobs can be run in MapReduce mode (even on a single-node cluster). Since all Hadoop nodes are peers in DataStax Enterprise (no Name Node), there is no concept of local mode for Pig. Pig in DataStax Enterprise 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.

About Solr in DataStax Enterprise

DataStax Enterprise Search provides powerful free-text search capabilities based on the Apache Solr project. Solr is an open source, widely-used search engine technology. In addition to free-text search, Solr provides more advanced features like aggregation, grouping, and geo-spatial search.
The unique combination of Cassandra, Hadoop, and Solr in DataStax Enterprise overcomes Map Reduce performance problems when querying real-time data. DataStax Enterprise Search adds capabilities to Cassandra for performing complex queries and searches. It offers unique search capacity scaling that improves native Solr search capabilities. You can add search capacity in the same way as you add Hadoop or Cassandra capacity to your cluster. Additionally, the Cassandra Query Language (CQL) has been extended to support Solr enterprise search queries.

If you don't need Hadoop/Cassandra, you can use DataStax Enterprise strictly for Solr and create an exclusively Solr cluster. This cluster configuration improves on the master-slave configuration supported by native Solr and, because DataStax Enterprise supports the native Solr tools and APIs, migration from Solr to DataStax Enterprise is painless.

DataStax Enterprise includes a demo that downloads and indexes all or part of Wikipedia. This demo shows how simple it is to input data and perform enterprise search operations.

About Sqoop

Apache Sqoop is a tool for transferring data between an external data source and Hadoop. You can migrate data from any JDBC-compliant data source, including non-relational data sources, such as NoSQL, and relational data sources, such as Oracle, MySQL, and SQL Server.

About log4j

DataStax Enterprise provides the ability to transfer log-based data directly into the server using log4j. Apache log4j is a Java-based logging framework that provides runtime application feedback. It provides allows you 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.

About DataStax OpsCenter

DataStax OpsCenter is a browser-based user interface for monitoring and administering Cassandra and/or DataStax Enterprise clusters in a single centralized management console. The key features of OpsCenter include:

- A Dashboard that displays an overview of commonly watched performance metrics.
- An Overview page that shows a condensed view of each cluster’s Dashboard (only displayed when multiple clusters are monitored).
- Basic cluster configuration.
- Built-in external notification capabilities.
- Administration tasks using simple point-and-click actions.
- Re-balancing data across a cluster when new nodes are added.
- Alert warnings of impending issues.
- Automatic backup operations, including scheduling and removing of old backups.
- Multiple cluster management from a single OpsCenter instance.

Getting Started with DataStax Enterprise

The fastest way to get started with DataStax Enterprise is to install it on a single node and run the Portfolio Manager demo application. For quick instructions on getting up and running on a single node, see:

- Quick Start with DataStax Enterprise
- Running the Portfolio Manager Demo Application

For cluster installations of DataStax Enterprise, see:

- Installing DataStax Enterprise Packaged Releases or Tarball Distribution
Quick Start with DataStax Enterprise

- **Configuring and Initializing a DataStax Enterprise Cluster**
To get started with the Hive and Pig MapReduce clients bundled with DataStax Enterprise, see:
  - *Getting Started with Hive in DataStax Enterprise*
  - *Getting Started with Pig in DataStax Enterprise*

**Using DataStax Enterprise**
These documents contain more in-depth information about using DataStax Enterprise:
  - Enterprise Search with Solr
  - Moving Data to DataStax Enterprise Using Sqoop
  - *Planning a DataStax Enterprise Deployment*
  - *Using the Log4j Appender*

**Using DataStax OpsCenter**
Information on installing and using OpsCenter:
  - OpsCenter Installation Guide
  - Using OpsCenter (OpsCenter Help Documentation)
  - Upgrading OpsCenter and OpsCenter Agents

**Other Documentation References**
Usefull resources for using DataStax Enterprise:
  - Cassandra 1.0 Documentation
  - MapReduce Getting Started Guide
  - Hive Getting Started Guide
  - Pig Latin Reference Manuals

**DataStax Enterprise Demos**
  - *Portfolio Manager Demo* - demonstrates a hybrid workflow using DataStax Enterprise.
  - *Search Demo* - demonstrates the Solr search capabilities using Wikipedia.
  - *Sqoop Demo* - migrates data from a MySQL database containing information from the North American Numbering Plan.
  - *Log4j Search Demo* - shows an example of searching and filtering log4j messages generated by a standard Java application.
  - *Hive Demo* - shows how to use Hive to access data in Cassandra.
  - *Pig Demo* - includes a sample data file containing tuples of two fields each (name and score). Using this file you create a Pig relation, perform a simple MapReduce job to calculate the total score for each user, and then put the result back into CassandraFS or into a Cassandra column family.

**Quick Start with DataStax Enterprise**
Quick Start with DataStax Enterprise

The best way to get up and running quickly with DataStax Enterprise (DSE) is to install the DataStax Enterprise tarball distributions and start a single-node analytics instance. DataStax Enterprise is intended to be run on multiple nodes. However, installing a single-node cluster is a great way to get started.

**Note**
The instructions in this section are intended for an introduction to DataStax Enterprise, not for production installations. See [Installing DataStax Enterprise Packaged Releases or Tarball Distribution](#) and [Configuring and Initializing a DataStax Enterprise Cluster](#) for DataStax Enterprise cluster installation.

Getting up and running is simple:

1. **Register with DataStax.** DataStax Enterprise is available to DataStax registered users and support customers. To download DataStax Enterprise, you will need the username and password provided in your DataStax registration confirmation email. If you are a DataStax support customer and are not sure of your login credentials, contact DataStax Customer Support.

   **Note**
   By downloading DataStax Enterprise software from DataStax you agree to the terms of the [DataStax Enterprise EULA](#) (End User License Agreement) posted on the DataStax web site.

2. Make sure **Java** is installed.

3. Install DataStax Enterprise in a way that suits your needs:

   - **Installing and Starting the DataStax Enterprise Binaries With sudo** - This method requires the fewest steps, but is not fully contained within your home directory. If you use this method, you will need to start DataStax Enterprise and run all the demos using `sudo`.
   
   - **Installing and Starting the DataStax Enterprise Binaries Without sudo** - This method requires more steps and is fully contained within your home directory. Use it if you do not have or want to use root privileges. If you use this method, you can start DataStax Enterprise and run the demos without using `sudo`.

**Checking for a Java Installation**

DataStax Enterprise is a Java program and requires that a Java Virtual Machine (JVM) is installed before starting the server. Production deployments require that the Java Runtime Environment (JRE) 1.6.0_19 or later from Oracle is installed. However, if you are just trying the DataStax Enterprise, any JVM is fine.

To check for Java, run the following command in a terminal window:

```
# java -version
```

If you do not have Oracle Java installed on your Linux system, see [Installing the JRE on RHEL or CentOS Systems](#) or [Installing the JRE on Debian or Ubuntu Systems](#) for instructions. (Mac OS X includes Java JRE.)

**Installing and Starting the DataStax Enterprise Binaries With sudo**

The quickest way to get going on a single node with DataStax Enterprise is to install the binary tarball packages using root permissions (or sudo). This installation also creates files and directories outside of the install location. If you need to install everything in a single location, such as your home directory, and without root permissions, see [Installing and Starting the DataStax Enterprise Binaries Without sudo](#).

To install and start DataStax Enterprise:

1. If you haven't already, register with DataStax to download the DataStax Enterprise software.
2. Download the DataStax Enterprise package (required) and the OpsCenter package (optional) to your home directory. Substitute `<username>;<password>` with your login credentials.

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

3. Unpack the distributions:

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

4. Go to the install directory. For example:

   ```
   cd dse-2.0-1
   ```

5. Start DataStax Enterprise (as an Analytics node).

   ```
   $ sudo bin/dse cassandra -t
   ```

   Where `cassandra` starts the Cassandra process plus CassandraFS and the `-t` option starts the Hadoop JobTracker and TaskTracker processes.

   Use this type of node for the *Portfolio Manager Demo*.

   **Note**

   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.

6. Check that your DataStax Enterprise ring is up and running (from the install directory):

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

   When running on a single node, the Cassandra seed node and DataStax Enterprise job tracker node are automatically set to `localhost`. Because there is no Hadoop NameNode with CassandraFS, no additional configuration is needed for running MapReduce jobs in single mode. However, additional configuration is needed for distributed mode. When running on a single node, the Cassandra seed node and DataStax Enterprise job tracker node are automatically set to `localhost`.

Next Steps

- Set up *DataStax OpsCenter for a Single Node*.
- Run the *Portfolio Demo* example application.
- Run the *DataStax Enterprise Search Demo*.
- Run the *Sqoop Demo*.
- For installing OpsCenter on multiple nodes, see *Installing Opscenter* in the DataStax OpsCenter Documentation.
- To stop a node, see *Stopping a Node* below.

### Installing and Starting the DataStax Enterprise Binaries Without sudo

This section provides instructions for installing and setting up a self-contained, single-node instance of DataStax Enterprise in your home directory that does not require root permissions. It consists of two parts:
Downloading and Installing the Binary Tarball

In this section, you download and install the DataStax Enterprise binary tarball.

To download and install the tarball:

1. If you haven’t already, register with DataStax to download the DataStax Enterprise software.
2. In your home directory, download the DataStax Enterprise package (required) and the OpsCenter package (optional). Substitute `<username>:<password>` with correct DataStax login credentials.
   
   ```
   $ cd ~/
   $ curl -OL http://<username>:<password>@downloads.datastax.com/enterprise/dse.tar.gz
   $ curl -OL http://<username>:<password>@downloads.datastax.com/enterprise/opscenter.tar.gz
   ```
3. Unpack the distributions:
   
   ```
   $ tar -xzvf dse.tar.gz
   $ tar -xzvf opscenter.tar.gz
   $ rm *.tar.gz
   ```
4. Go to the install directory. For example:
   
   ```
   $ cd dse-2.0-1
   ```
5. Create the data and logging directories:
   
   ```
   $ mkdir dse-data
   ```
6. In `dse-data`, create the following directories:
   
   ```
   $ cd dse-data
   $ mkdir data
   $ mkdir saved_caches
   $ mkdir commitlog
   ```

Configuring and Starting the Cluster

In this section, you set the configuration properties needed to run DataStax Enterprise as a single-node cluster on the localhost from your home directory. These properties are specified in the `cassandra.yaml` and `log4j-server.properties` files.

To configure and start the cluster:

1. Go the directory containing the `cassandra.yaml` file. For example:
   
   ```
   $ cd ~/dse-2.0-1/resources/cassandra/conf
   ```
2. Edit the following lines in the `cassandra.yaml` file. For example:
   
   ```
   initial_token: 0
data_file_directories: - ~/dse-2.0-1/dse-data/data
commitlog_directory: ~/dse-2.0-1/dse-data/commitlog
saved_caches_directory: ~/dse-2.0-1/dse-data/saved_caches
listen_address: 127.0.0.1
   ```
3. In the `conf` directory, change the `log4j-server.properties` file:
   
   ```
   ```
4. Start DataStax Enterprise (as an Analytics node):
   
   ```
   $ cd ~/dse-2.0-
   $ bin/dse cassandra -t
   ```

   where `cassandra` starts the Cassandra process plus CassandraFS and the `-t` option starts the Hadoop JobTracker and TaskTracker processes.

   Use this type of node for the Portfolio Manager Demo.

   **Note**
   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.

5. Check that your cluster is up and running:

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

   When running on a single node, the Cassandra seed node and DataStax Enterprise job tracker node are automatically set to `localhost`. Because there is no Hadoop NameNode with CassandraFS, no additional configuration is needed for running MapReduce jobs in single mode; however, additional configuration is needed for distributed mode. When running on a single node, the Cassandra seed node and DataStax Enterprise job tracker node are automatically set to `localhost`.

**Next Steps**

- Set up DataStax OpsCenter for a Single Node.
- Run the Portfolio Demo example application.
- Run the DataStax Enterprise Search Demo.
- Run the Sqoop Demo.
- For installing OpsCenter on multiple nodes, see Installing Opscenter in the DataStax OpsCenter Documentation.
- To stop a node, see Stopping a Node below.

**Stopping a Node**

To stop a node, find the Cassandra Java process ID (PID) and `kill` the process using its PID number (using `sudo` if necessary). For example:

```
$ ps -ef | grep cassandra
$ kill 1539
```

**Setting up the OpsCenter for a Single Node**

The DataStax OpsCenter binary tarball installs OpsCenter in a single directory and does not require root permissions. When installing from the tarball, the OpsCenter agents must be manually installed. When installing from a package, you can use OpsCenter to install the agents on each node.

In this introduction, you install the agent on the same machine as the DataStax Enterprise server and OpsCenter and set it to use `localhost`. For information about production installations, see Installing OpsCenter.
**Prerequisites**

You have previously downloaded and extracted the `opscenter.tar.gz` file and DataStax Enterprise is running. In addition to the DataStax Enterprise installation requirements, the OpsCenter requires that the following software is installed:

- Python 2.6+
- OpenSSL: 0.9.8 for Mac OSX, RHEL 5, CentOS 5, Ubuntu, Debian, and Windows; 1.0.0 for RHEL 6 and CentOS 6.

**Note**

If your operating system has OpenSSL 1.0.0 installed and you see an error containing `exceptions.ImportError: libssl.so.0.9.8:`, you must install OpenSSL 0.9.8. For example on Ubuntu, use `sudo apt-get install libssl0.9.8`.

To check for the software versions:

```
$ python --version
$ openssl version
```

**Configuring the OpsCenter and OpsCenter Agent**

1. Make sure the prerequisite software is installed.

2. Start the OpsCenter:

   ```
   $ cd ~/opscenter-2.0
   $ bin/opscenter (in the background - default)
   $ bin/opscenter -f (in the foreground)
   ```

3. Set the agent to `localhost`:

   ```
   $ cd ~/opscenter-2.0/agent
   $ bin/setup localhost
   ```

4. Start the agent:

   ```
   $ bin/opscenter-agent (in the background - default)
   $ bin/opscenter-agent -f (in the foreground)
   ```

5. Launch OpsCenter from a browser using the following URL:

   ```
   http://<opscenter_host>:8888
   ```

6. In the Welcome to DataStax OpsCenter dialog box, enter localhost in the **Cluster Hosts/IP field**, and then click **Add Cluster**.

   The OpsCenter Dashboard is displayed.
Planning a DataStax Enterprise Deployment

This section provides guidelines for determining the size of your Cassandra cluster based on the data you plan to store. To estimate capacity, you should have a good understanding of the sizing of the raw data you plan to store, a good estimate of your typical application workload, and how you 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 filesystem 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 to a maximum of 8GB or half of your total RAM, whichever is lower. (A greater heap size has more intense garbage collection periods.)
- 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:

- 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 http://www.datastax.com/docs/1.0/install/recommended_settings#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 Enterprise clusters can be deployed on cloud infrastructures such as Amazon EC2.

For production DataStax Enterprise clusters on EC2, use Large or Extra Large instances with local storage. RAID0 the ephemeral disks, and put both the data directory and the commit log on that volume. This has proved to be better in practice than putting the commit log on the root volume (which is also a shared resource). For data redundancy, consider deploying your cluster across multiple availability zones or using EBS volumes to store your backup files.

EBS volumes are *not* recommended for Cassandra data volumes - their network performance and disk I/O are not good fits for Cassandra for the following reasons:

- EBS volumes contend directly for network throughput with standard packets. This means that EBS throughput is likely to fail if you saturate a network link.
- EBS volumes have unreliable performance. I/O performance can be exceptionally slow, causing the system to backload reads and writes until the entire cluster becomes unresponsive.
- Adding capacity by increasing the number of EBS volumes per host does not scale. You can easily surpass the ability of the system to keep effective buffer caches and concurrently serve requests for all of the data it is responsible for managing.

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 multi data center cluster.

**Calculating Usable Disk Capacity**

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

Start with the raw capacity of the physical disks:

\[
\text{raw_capacity} = \text{disk_size} \times \text{number_of_disks}
\]

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

\[
\left(\text{raw_capacity} \times 0.9\right) / 2 = \text{formatted_disk_space}
\]

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

\[
\text{formatted_disk_space} \times 0.5 = \text{usable_disk_space}
\]

**Calculating User Data Size**

As with all data storage systems, the size of your raw data will be larger once it is loaded into Cassandra due to storage overhead. On average, raw data will be about 2 times larger on disk after it is loaded into the database, but could be much smaller or larger depending on the characteristics of your data and column families. The calculations in this section account for data persisted to disk, not for data stored in memory.
• **Column Overhead** - Every column in Cassandra incurs 15 bytes of overhead. Since each row in a column family can have different column names as well as differing numbers of columns, metadata is stored for each column. For counter columns and expiring columns, add an additional 8 bytes (23 bytes column overhead). So the total size of a regular column is:

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

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

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

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

• **Replication Overhead** - The replication factor obviously plays a role in how much disk capacity is used. For a replication factor of 1, there is no overhead for replicas (as only one copy of your data is stored in the cluster). If replication factor is greater than 1, then your total data storage requirement will include replication overhead.

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

### 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 multi-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 localhost 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, the seed list should include a node from each data center.</td>
</tr>
</tbody>
</table>
The intra-node communication port (default is 7000). Should be the same for every node in the cluster.

The initial token is used to determine the range of data this node is responsible for.

**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

**Partitioner Settings**

When you deploy a Cassandra cluster, you need to make sure that each node is responsible for roughly an equal amount of data. This is also known as load balancing. This is done by configuring the partitioner for each node, and correctly assigning 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.

For clusters where all nodes are in a single data center, you can calculate tokens by dividing the range by the total number of nodes in the cluster. In multi-data center deployments, tokens should be calculated such that each data center is individually load balanced as well. See *Generating Tokens* for the different approaches to generating tokens for nodes in single and multi-data center clusters.

**Snitch Settings**

The snitch is a configurable component that is responsible for knowing the location of nodes within your network topology. This affects where replicas are placed as well as how requests are routed between replicas. All nodes must have the exact 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 Delegated Snitch is used to implement Elastic Workload Re-provisioning.

**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

You use the delegated snitch for re-provisioning the workload of a cluster.

**DseSimpleSnitch**

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 analytics or real-time Cassandra data center names, respectively.
**SimpleSnitch**

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

**Configuring the PropertyFileSnitch**

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

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

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

```
# Data Center One
175.56.12.105=DC1:RAC1
175.50.13.200=DC1:RAC1
175.54.35.197=DC1:RAC1

120.53.24.101=DC1:RAC2
120.55.16.200=DC1:RAC2
120.57.102.103=DC1:RAC2

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

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

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

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

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

**Choosing Keyspace Replication Options**

When you create a keyspace, you must define the replica placement strategy and the number of replicas you want. DataStax recommends always choosing NetworkTopologyStrategy for both single and multi-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):

```
[default@unknown] CREATE KEYSPACE test
WITH placement_strategy = 'NetworkTopologyStrategy'
AND strategy_options=[{us-east:6}];
```

Or for a multi-data center cluster:

```
[default@unknown] CREATE KEYSPACE test
WITH placement_strategy = 'NetworkTopologyStrategy'
AND strategy_options=[{DC1:6,DC2:6,DC3:3}];
```

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.

### Installing DataStax Enterprise Packaged Releases or Tarball Distribution

DataStax provides Debian and Red Hat Enterprise Linux (RHEL) packaged releases for DataStax Enterprise (DSE) and a binary tarball distribution. RHEL and Debian packages are currently supported through the Yum and Apt package management tools.

The following instructions are available:

- [Installing DataStax Enterprise on RHEL and CentOS Systems](#)
- [Installing DataStax Enterprise on Debian and Ubuntu Systems](#)
- [Installing the DataStax Enterprise Binary Distribution](#)

### Installing DataStax Enterprise on RHEL and CentOS Systems

DataStax provides Yum repositories for RHEL and CentOS systems. 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:

- Yum Package Management application installed.
- Root or `sudo` access to the install machine.
- Oracle Java SE Runtime Environment (JRE) 6. 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.

#### Steps for Installing on RHEL and CentOS Systems
1. Check which version of Java is installed by running the following command in a terminal window:

   $ java -version

DataStax recommends using the most recently released version of Oracle Java SE Runtime Environment (JRE) 6 on all DSE nodes. Versions earlier than 1.6.0_19 should not be used. 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 DataStax Enterprise. 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:

   # vi /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 DataStax Enterprise and OpsCenter using Yum:

   # yum install dse-full opscener

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

**Next Steps**

- **DataStax OpsCenter** - A browser-based user interface for monitoring, administering, and configuring multiple clusters from a single centralized management console.
- **Packaged Installation Directories** - Information about install directories, user, and scripts.
- **Configuring and Initializing a DataStax Enterprise Cluster** - Setting the configuration properties on each node in the cluster before starting the cluster.
- **Starting DataStax Enterprise as a Service** - Start DataStax Enterprise as a Service.

**Installing DataStax Enterprise on Debian and Ubuntu Systems**

DataStax provides Debian package repositories for Debian and Ubuntu systems. 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:

- Aptitude Package Manager installed.
Installing DataStax Enterprise Packaged Releases or Tarball Distribution

- Root or `sudo` access to the install machine.
- Oracle Java SE Runtime Environment (JRE) 6. 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.

**Steps for Installing on Debian or Ubuntu Systems**

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

   ```bash
   $ java -version
   ```

   DataStax recommends using the most recently released version of Oracle Java SE Runtime Environment (JRE) 6 on all DSE nodes. Versions earlier than 1.6.0_19 should not be used. 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` file:

   ```bash
   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:

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

4. Install the DataStax Enterprise packages and OpsCenter:

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

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

**Next Steps**

- **DataStax OpsCenter** - A browser-based user interface for monitoring, administering, and configuring multiple clusters from a single centralized management console.
- **Packaged Installation Directories** - Information about install directories, users, and scripts.
- **Configuring and Initializing a DataStax Enterprise Cluster** - Setting the configuration properties on each node in the cluster before starting the cluster.
- **Starting DataStax Enterprise as a Service** - Start DataStax Enterprise as a Service.

**Installing the DataStax Enterprise Binary Distribution**

DataStax provides a binary tarball distribution of DataStax Enterprise for platforms that do not have package support, such as Mac, or if you do not have or want to do a root installation.
Prerequisites

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

- Oracle Java SE Runtime Environment (JRE) 6. 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.

Steps for Binary Installations

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

   ```
   $ java -version
   ```

   DataStax recommends using the most recently released version of Oracle Java SE Runtime Environment (JRE) 6 on all DSE nodes. Versions earlier than 1.6.0_19 should not be used. Java 7 is not recommended. If you need help installing Java, see Installing the JRE on RHEL or CentOS Systems or Installing the JRE on Debian or Ubuntu Systems.

2. Download the distribution to a location on your machine:

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

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

3. Unpack the distributions:

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

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

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

   ```
   $ sudo mkdir -p /var/lib/cassandra; sudo chown -R $USER:$GROUP /var/lib/cassandra
   $ sudo mkdir -p /var/log/cassandra; sudo chown -R $USER:$GROUP /var/log/cassandra
   ```
5. (Optional) If you do not want to use the default data and logging directories, you can define your own directory locations:

   a. Make the directories for data and logging directories. For example:
      
      ```
      $ mkdir <install_directory>dse-data
      $ cd dse-data
      $ mkdir commitlog
      $ mkdir saved_caches
      ```

   b. Go the directory containing the `cassandra.yaml` file. For example:
      
      ```
      $ cd <install_directory>/resources/cassandra/conf
      ```

   c. Edit the following lines in the `cassandra.yaml` file. For example:
      
      ```
      data_file_directories: <install_directory>/dse-data
      commitlog_directory: <install_directory>/dse-data/commitlog
      saved_caches_directory: <install_directory>/dse-data/saved_caches
      ```

**Next Steps**

- **DataStax OpsCenter** - A browser-based user interface for monitoring, administering, and configuring multiple clusters from a single centralized management console.
- **Binary Install Directories** - Information about install directories, users, and scripts.
- **Configuring and Initializing a DataStax Enterprise Cluster** - Setting the configuration properties on each node in the cluster before starting the cluster.
- **Starting DataStax Enterprise as a Stand-Alone Process** - Start DataStax Enterprise as a stand-alone process.

**Install Locations**

**Packaged Installation Directories**

The packages installs into the following directories:

<table>
<thead>
<tr>
<th>Cassandra Directories</th>
</tr>
</thead>
<tbody>
<tr>
<td>· /var/lib/cassandra  (Cassandra and CassandraFS data directories)</td>
</tr>
<tr>
<td>· /var/log/cassandra</td>
</tr>
<tr>
<td>· /var/run/cassandra</td>
</tr>
<tr>
<td>· /usr/share/dse/cassandra  (Cassandra environment settings)</td>
</tr>
<tr>
<td>· /usr/share/dse/cassandra/lib</td>
</tr>
<tr>
<td>· /usr/share/dse-demos  (Portfolio, Solr, Sqoop demo applications)</td>
</tr>
<tr>
<td>· /usr/bin</td>
</tr>
<tr>
<td>· /usr/sbin</td>
</tr>
<tr>
<td>· /etc/dse/cassandra  (Cassandra configuration files)</td>
</tr>
<tr>
<td>· /etc/init.d</td>
</tr>
<tr>
<td>· /etc/security/limits.d</td>
</tr>
<tr>
<td>· /etc/default/</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hadoop Directories</th>
</tr>
</thead>
</table>

21
Installing DataStax Enterprise Packaged Releases or Tarball Distribution

- /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)

**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 Install Directories**

**DataStax Enterprise Directories**
- bin (DataStax Enterprise start scripts)
- demos (Portfolio Manager Demo)
- interface
- javadoc
- lib
- resources/cassandra/bin (Cassandra utilities)
- resources/cassandra/conf (Cassandra configuration files)
- resources/hadoop (Hadoop installation)
- resources/hive (Hive installation)
- resources/log4j-appender (log4j logging)
Using a DataStax Enterprise on Amazon EC2

DataStax Enterprise provides an AMI (Amazon Machine Image) to set up and expand a DataStax Enterprise (DSE) cluster using the Amazon Web Services EC2 Management Console.

**Initializing a DataStax Enterprise Cluster on Amazon EC2**

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

**Expanding a DataStax Enterprise AMI Cluster**

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

**Configuring and Initialing a DataStax Enterprise Cluster**

Before you can start DataStax Enterprise (DSE) on either a single or multi-node cluster, there are a few Cassandra configuration properties you must set on each node in the cluster. You set these properties in the cassandra.yaml file (located in /etc/dse/cassandra in packaged installations or <install_location>/resources/cassandra/conf in binary distributions).

*Note*

These instructions apply only to single data center clusters. For information about configuring clusters with multiple data centers, see Configuring Multiple Data Centers Quick Start.

In DataStax Enterprise, the term **data center** is a grouping of nodes. You should configure these data centers by type of node: Cassandra, Analytics, and Search.

**Initializing a Multi-Node DataStax Enterprise Cluster**

Before starting a multi-node DSE cluster, you must determine the following:

- A name for your cluster.
- How many total nodes your DSE cluster will have.
- The internal IP addresses of each node.
- The token for each node (see Generating Tokens). If you are deploying a mixed-workload DSE Cluster, make sure to alternate token assignments between Cassandra nodes and Analytics nodes so that replicas are evenly balanced.
- Which nodes will serve as the seed nodes. You need at least one seed node per data center for Cassandra and Hadoop nodes. Solr nodes don’t require a seed node.
- If you intend to run a mixed-workload cluster determine which nodes will serve which purpose.
- If you have a firewall enabled on the machines that you plan to use for your cluster, make sure that nodes within a cluster can reach each other. See Configuring Firewall Port Access.
- If you want to use Solr, you must create a data center for Solr nodes and all nodes in that data center must also be running with Solr enabled. The default DseSimpleSnitch does this for you automatically.
To determine tokens assignments:

For example, suppose you are starting a 8 node mixed-workload cluster with 3 Analytics nodes, 3 Cassandra nodes, and 2 Search nodes. The nodes have the following IPs:

- node0 (Cassandra seed) 110.82.155.0
- node1 (Cassandra) 110.82.155.1
- node2 (Cassandra) 110.82.155.2
- node3 (Analytics seed) 110.82.155.3
- node4 (Analytics) 110.82.155.4
- node5 (Analytics) 110.82.155.5
- node6 (Search) 110.82.155.6
- node7 (Search) 110.82.155.7

To assign tokens in a multi data-center cluster, you generate tokens for the nodes in one data center, and then offset those token numbers by 1 for all nodes in the next data center, by 2 for the nodes in the next data center, and so on (larger increments are allowed, such as 10 or 50).

Because the number of nodes are not the same in each data center, you need to run the Token Generating Tool twice. The first run generates the tokens for the Cassandra data center. The second run generates tokens for the Search data center. For the Analytics data center, you offset the tokens generated by the first run. In this example, the tokens are incremented by 10. For the Solr data center, you use the tokens generated by the tool and then increment the first Solr node by 20.

<table>
<thead>
<tr>
<th>Node</th>
<th>Token</th>
<th>Offset</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>node 0</td>
<td>0</td>
<td>Na</td>
<td>Cassandra seed</td>
</tr>
<tr>
<td>node 1</td>
<td>56713727820156410577229101238628035242</td>
<td>NA</td>
<td>Cassandra</td>
</tr>
<tr>
<td>node 2</td>
<td>113427455640312821154458202477256070485</td>
<td>NA</td>
<td>Cassandra</td>
</tr>
<tr>
<td>node 3</td>
<td>10</td>
<td>10</td>
<td>Analytics seed</td>
</tr>
<tr>
<td>node 4</td>
<td>56713727820156410577229101238628035252</td>
<td>10</td>
<td>Analytics</td>
</tr>
<tr>
<td>node 5</td>
<td>113427455640312821154458202477256070495</td>
<td>10</td>
<td>Analytics</td>
</tr>
<tr>
<td>node 6</td>
<td>20 (offset twice)</td>
<td>20</td>
<td>Search</td>
</tr>
<tr>
<td>node 7</td>
<td>85070591730234615865843651857942052864</td>
<td>10</td>
<td>Search</td>
</tr>
</tbody>
</table>

Since this is a mixed-workload cluster, the token placement alternates between Cassandra, Analytics, and Search nodes. This ensures even distribution of replicas on both sides of the cluster. The cassandra.yaml file for each node has the following modified property settings.

- node 0: 0 (Cassandra seed)
- node 3: 10 (Analytics seed)
- node 6: 20 (Search)
- node 1: 56713727820156410577229101238628035242 (Cassandra)
- node 4: 56713727820156410577229101238628035252 (Analytics)
- node 7: 85070591730234615865843651857942052864 (Search)
- node 2: 113427455640312821154458202477256070485 (Cassandra)
- node 5: 113427455640312821154458202477256070495 (Analytics)
Node0

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

Node1

cluster_name: 'DSECluster'
initial_token: 56713727820156410577229101238628035242
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      - seeds: "110.82.155.3,110.82.155.0"
listen_address: 110.82.155.1
rpc_address: 0.0.0.0

Node2

cluster_name: 'DSECluster'
initial_token: 113427455640312821154458202477256070485
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      - seeds: "110.82.155.3,110.82.155.0"
listen_address: 110.82.155.2
rpc_address: 0.0.0.0

Node3

cluster_name: 'DSECluster'
initial_token: 10
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      - seeds: "110.82.155.3,110.82.155.0"
listen_address: 110.82.155.3
rpc_address: 0.0.0.0

Node4

cluster_name: 'DSECluster'
initial_token: 56713727820156410577229101238628035252
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      - seeds: "110.82.155.3,110.82.155.0"
listen_address: 110.82.155.4
rpc_address: 0.0.0.0

Node5

cluster_name: 'DSECluster'
initial_token: 113427455640312821154458202477256070495
Generating Tokens

Tokens are used to assign a range of data to a particular node within a data center. Assuming you are using the `RandomPartitioner`, this approach ensures even data distribution. For a multi data-center cluster, generate the tokens for the nodes in one data center, and then offset those token numbers by 1 for all nodes in the next data center, by 2 for the nodes in the next data center, and so on. (Instead of using single digits, you might want to offset the token number by a larger value, such as 10 or 50.)

**Note**

The following steps illustrate token generation for the above example.

To create tokens:

1. Create a new file for your token generator program:

   ```bash
   vi tokengentool
   ```

2. Paste the following Python program into this file:

   ```python
   #!/usr/bin/python
   import sys
   if (len(sys.argv) > 1):
       num=int(sys.argv[1])
   else:
       num=int(raw_input("How many nodes are in your cluster? "))
   for i in range(0, num):
       print 'node %d: %d' % (i, (i*(2**127))/num)
   ```
3. Save and close the file and make it executable:

   chmod +x tokengentool

4. Run the script:

   ./tokengentool

5. When prompted, enter the total number of nodes in your Cassandra data center:

   How many nodes are in your cluster? 3

   node 0: 0
   node 1: 56713727820156410577229101238628035242
   node 2: 113427455640312821154458202477256070485

6. Run the tool again for two nodes (Solr data center):

   How many nodes are in your cluster? 2

   node 0: 0
   node 1: 85070591730234615865843651857942052864

6. On each node, edit the cassandra.yaml file and enter its corresponding token value in the initial_token property.

**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</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</td>
</tr>
<tr>
<td>50060</td>
<td>Hadoop Task Tracker website port</td>
</tr>
<tr>
<td><strong>OpsCenter Specific</strong></td>
<td></td>
</tr>
<tr>
<td>8888</td>
<td>OpsCenter website port</td>
</tr>
<tr>
<td><strong>Intranode 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</td>
</tr>
<tr>
<td>7000</td>
<td>Cassandra intra-node port</td>
</tr>
<tr>
<td>7199</td>
<td>Cassandra JMX monitoring port</td>
</tr>
<tr>
<td>9160</td>
<td>Cassandra client port</td>
</tr>
<tr>
<td><strong>DataStax Enterprise Specific</strong></td>
<td></td>
</tr>
</tbody>
</table>
### Starting a DataStax Enterprise Cluster

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 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**

**Note**

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.

### 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 [Configuring Multiple Data Centers Quick Start](#).

To start DataStax Enterprise as a stand-alone process:

- **Analytics node**: `dse cassandra -t`
- **Cassandra node**: `dse cassandra`
- **Solr node**: `dse cassandra -s`
- 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. Create 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 DSE Analytic and starts the Hadoop Job Tracker and Task Tracker services.
   - SOLR_ENABLED=1 - Starts the node as DSE Enterprise Search. See *Getting Starting with DSE Search*.

   **Note**
   Using the SOLR_ENABLED and HADOOP_ENABLED options together to enable both search and Hadoop analytics on the same node is only recommended for development. In production environments each node should be used only for one or the other.

2. Start the DSE service:

   `sudo service dse start`

3. To check if your cluster is up and running:

   `nodetool ring -h localhost`

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

## Upgrading DataStax Enterprise

You can upgrade these releases to DataStax Enterprise 2.0:

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

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

To upgrade DataStax OpsCenter, see **Upgrading OpsCenter and OpsCenter Agents**.

This section lists component version changes and other major changes included in DSE upgrades:

<table>
<thead>
<tr>
<th>Upgrade</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSE 1.0 or 1.0.x to 2.0.x</td>
<td>Cassandra updated to 1.0.8, Hadoop updated to 1.0, Hive updated to 0.8.1, Pig updated to 0.8.3 (effective in DSE 1.0.2), Sqoop 1.4.1 added, Solr 4.0 added.</td>
</tr>
</tbody>
</table>

**Best Practices for Upgrading**

The following best practices are recommended when upgrading:

- Always take a **snapshot** before upgrading to a new release. This allows you to rollback to the previous version if necessary. Cassandra is able to read data files created by the previous version, but the inverse is not always true.

  **Note**
  Taking a snapshot is fast, especially if you have JNA installed, and consumes effectively zero disk space until you start compacting the live data files again.
Always read NEWS.txt before starting an upgrade. News.txt contains critical information about upgrading from early releases, especially Cassandra 0.6.x and 0.7, to DSE. NEWS.txt is in the following directory:

**Binary tarball**: `<install_location>/resources/cassandra/NEWS.txt`

**Debian or RPM**: `/usr/share/doc/dse-libcassandra*/NEWS.txt`

If your cluster includes a job tracker (Hadoop-enabled) node, upgrade that node first, then upgrade other nodes.

### Upgrading to DataStax Enterprise 2.0

1. In general, follow the instructions for a new installation with a few modifications for upgrading:

   - **Binary tarball installation**: *Installing the DataStax Enterprise Binary Distribution*. To upgrade a binary tarball installation, unpack the tarball into a different folder than your old installation.

   - **Debian or RPM package**: see *Installing DataStax Enterprise Packaged Releases or Tarball Distribution*.

     - To upgrade from the community project version, Apache Cassandra, to DataStax Enterprise (DSE) 2.0, first read NEWS.txt, next uninstall Cassandra, then install DSE.

     - To upgrade from DataStax Community to DataStax Enterprise 2.0, first uninstall DataStax Community, then install DSE.

     - To upgrade from a previous version of DataStax Enterprise to DataStax Enterprise 2.0, first run the Yum (RPM) or Aptitude (Debian) update commands, then run the install commands shown in *Installing DataStax Enterprise RedHat RPM Packages or Installing DataStax Enterprise Debian Packages*.

2. Diff the following configuration files:

   - The `cassandra.yaml` from the old installation

   - The new DSE 2.0 `cassandra.yaml`

   The new DSE 2.0 `cassandra.yaml` is in:

     - **Binary tarball installation**: `<install_location>/resources/cassandra/conf`

     - **Debian or RPM package**: `/etc/dse/cassandra`

3. Merge the diffs, except those related to snitches, from your old file into the new DSE 2.0 version of `cassandra.yaml`. Observe these Do's and Don'ts:

   **Do** perform merging by hand. For example, set the seed location and local host name in the new `cassandra.yaml` to the same values as the old `cassandra.yaml`.

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

   **Don't** add snitch settings from the old `cassandra.yaml` to the new `cassandra.yaml`. 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`. 
4. Perform one of the following tasks, depending on the snitch setting (endpoint_snitch URL) of your old cassandra.yaml file:


- org.apache.cassandra.locator.PropertyFileSnitch - Copy cassandra-topology.properties from the old installation. Paste it to <install_location>/resources/cassandra/conf (binary installs) or /etc/dse/cassandra (packaged installs), overwriting the new properties file.

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

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

5. 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.

6. Flush the commit log on the upgraded node by running nodetool drain.

7. Stop the old Cassandra process. Start the new Cassandra process as described in the next section.

**About Starting the Upgraded Node**
DataStax supports rolling restarts of nodes other than Analytic nodes. 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.

**Note**
You can actually start Analytic nodes using a rolling restart if you can accept your log files being flooded with exceptions.

The Hadoop job tracker repeatedly logs exceptions until all Analytic nodes are upgraded. The runtime exception you see when starting Analytic nodes looks something like this snippet:

```
INFO [pool-3-thread-1] 2012-03-22 02:09:08,868 Server.java (line 542) IPC Server listener on 8012: readAndProcess threw exception
java.lang.RuntimeException: readObject can't find class. Count of bytes read: 0
java.lang.RuntimeException: readObject can't find class
at org.apache.hadoop.io.ObjectWritable.readObject(ObjectWritable.java:185)
```

You can ignore these exceptions. When the last node upgrades, restarts, and joins the cluster, the exceptions cease.

**Note**
If you have Analytics nodes in the cluster, upgrade and start the new job tracker node first.

**Completing the Upgrade**

1. Flush the commit log on the upgraded node by running nodetool drain.
Upgrading DataStax Enterprise

2. You might need to run the following command against each node before running repair, moving nodes, or adding new ones.

- **Binary tarball:** `<install_location>/bin/nodetool -h upgradesstables`
- **Debian or RPM Package:** `nodetool -h upgradesstables`

To determine whether or not you need to run this command, see `News.txt` in the location mentioned earlier.

3. Monitor the log files for any issues.

4. Upgrade the next node.

**Upgrading a DataStax Enterprise AMI**

Before upgrading, be sure to read *Best Practices for Upgrading* above and the `/usr/share/doc/dse-libcassandra/NEWS.txt` in the newer packages.

**Note**

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

1. On each node ensure that the the DataStax repository is listed in the `/etc/apt/sources.list`:

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

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

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

```
$ wget -O - 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/conf/cassandra/conf` 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 2.0 `cassandra.yaml`

b. Merge the versions by hand from the old `cassandra.yaml` into the new DSE 2.0 `cassandra.yaml`:

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

  Don't copy property files from the prior release and overwrite files in the new release. Users who have attempted this have reported problems.
5. Perform one of the following tasks, depending on the snitch setting of 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 default delegated_snitch in the new <code>dse.yaml</code> unchanged.</td>
</tr>
<tr>
<td>org.apache.cassandra.locator.PropertyFileSnitch</td>
<td>Copy <code>cassandra-topology.properties</code> from the old installation. Paste it to <code>&lt;install_location&gt;/resources/cassandra/conf</code>, overwriting the new properties file.</td>
</tr>
<tr>
<td>Any other snitch URL</td>
<td>Change the default delegated_snitch in the new <code>dse.yaml</code> file to your current snitch setting.</td>
</tr>
</tbody>
</table>

**Note**
The default delegated_snitch setting in the new `dse.yaml` file in `<install_location>/resources/dse/conf` is `delegated_snitch: com.datastax.bdp.snitch.DseSimpleSnitch`.

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. Flush the commit log on the upgraded node by running `nodetool drain`.

8. Restart the node:
   ```
   sudo service dse restart
   ```

9. Restart client applications.

**Analytics with Hadoop**

In DataStax Enterprise, Hadoop is continuously available for analytic workloads. DataStax Enterprise is 100% compatible with Hadoop. Instead of using the Hadoop Distributed File System (HDFS), it uses uses a cfs keyspace in Cassandra 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 MadReduce, Hive, and Pig.
Getting Started with Hadoop in DataStax Enterprise

Built into DataStax Enterprise is an enhanced Hadoop distribution that is fully compatible with existing HDFS, Hadoop, and Hive tools and utilities. This topic contains information on:

- Starting DataStax Enterprise Hadoop
- Running the Portfolio Manager Demo Application
- Divergence from Apache Hadoop for Advanced Users

Starting DataStax Enterprise Hadoop

Use the following command to start Hadoop:

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

where the available `<args>` are described the HDFS File System Shell Guide on the Apache Hadoop web site. For information on starting Hive, Pig, or using Hadoop, see:

- Getting Started with Hive in DataStax Enterprise
- Getting Started with Pig in DataStax Enterprise
- Apache Hadoop landing page

Running the Portfolio Manager Demo Application
Your DataStax Enterprise (DSE) installation contains a Portfolio Manager sample application that shows a sample mixed workload on a DSE cluster. The demo is located in /usr/share/dse-demos/portfolio_manager for packaged installations or <install_location>/demos/portfolio_manager for binary installations.

The Portfolio Manager Use Case

The portfolio manager application demonstrates a hybrid workflow using DataStax Enterprise. 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 price at which the shares were purchased. A live stream of data simulates an active stock market, and updates each portfolio based on its overall value and the percentage of gain or loss compared to the purchase price. Historical market data is tracked for each stock (the end-of-day price) going back in time.

In the demo, simulated real-time stock data is generated by the pricer utility. This utility generates portfolios, live stock prices, and historical market data.

On the DSE OLAP 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.

Running the Portfolio Manager Demo

Before you begin, make sure you have installed, configured, and started DSE on either a single node (as an Analytics node) or a cluster. If running the demo on a cluster, install and run the demo from the DSE Job Tracker (analytics seed) node.

1. Go to the portfolio manager demo directory.
   - Tarball 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 100 days worth of historical stock data:
   - To see all of the available options for this utility:
     
     bin/pricer --help
   - If running on a single node cluster on localhost (as described in Quick Start with DataStax Enterprise):
     
     bin/pricer -o INSERT_PRICES
     bin/pricer -o UPDATE_PORTFOLIOS
     bin/pricer -o INSERT_HISTORICAL_PRICES -n 100
   - If running the demo on a cluster:
     
     LOCAL_IP=<demo_node_ip_address>
     bin/pricer -o INSERT_PRICES -d $LOCAL_IP
     --replication-strategy="org.apache.cassandra.locator.NetworkTopologyStrategy"
     --strategy-properties="Analytics:1,Cassandra:1"
     
     bin/pricer -d $LOCAL_IP -o UPDATE_PORTFOLIOS
     bin/pricer -d $LOCAL_IP -o INSERT_HISTORICAL_PRICES -n 100

   NetworkTopologyStrategy is the preferred replication placement strategy. For more information, see NetworkTopologyStrategy.
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.

5. Open another shell window.

6. Start Hive and run the MapReduce job for the demo in Hive.
   ```
   dse hive -f /usr/share/dse-demos/portfolio_manager/10_day_loss.q
   ```
   or for binary installations:
   ```
   <install_location>/bin/dse hive -f <install_location>/demos/portfolio_manager/10_day_loss.q
   ```

7. The MapReduce job will take several minutes to run. 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.

---

**Divergence from Apache Hadoop 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 /

Getting Started with Hive in DataStax Enterprise

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. The HiveQL language also allows traditional MapReduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL. In DataStax Enterprise, you can start the Hive client on any analytics node, define Hive data structures, and issue MapReduce queries. DSE Hive includes a custom storage handler for Cassandra that allows you to run Hive queries directly on data stored in Cassandra.

Note
DataStax Enterprise 2.0 supports Hive 0.8.1**. This version 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 any additional configuration required.

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. In DataStax Enterprise, the job tracker node information is stored in a column family in CassandraFS, and is initially populated on cluster startup by selecting the first Analytics node from the Cassandra seeds list. Assuming you have properly configured the Cassandra seeds list for DataStax Enterprise in cassandra.yaml, there is no additional configuration required. Hive clients will automatically select the correct job tracker node upon startup.

The default job tracker client port is 8012. If you are not sure which node in your cluster is the job tracker, run the following command:

dsetool jobtracker

or in a binary distribution:

<install_location>/bin/dsetool jobtracker

Moving the Job Tracker Node for DataStax Enterprise

If your primary Job Tracker node fails, DataStax Enterprise provides a utility (dsetool movejt) that allows you 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 IP address of the new job tracker node in your DataStax Enterprise cluster. For example:

   ```
dsetool movejt 110.82.155.4
   
   or in a binary distribution:
   
   <install_location>/bin/dsetool movejt 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.

**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:

```
./bin/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:

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

You can then load a table using the `LOAD DATA` command. See the `HiveQL Manual` for 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:

```hivedebug
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.

Demo - Using Hive 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.

Mapping Hive External Tables to Cassandra Column Families
An external table in Hive maps to a column family in Cassandra. 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. The TBLPROPERTIES clause specifies CassandraFS and MapReduce properties for the table. For example:

hive> CREATE EXTERNAL TABLE Users(userid string, name string,
email string, phone string)
STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
WITH SERDEPROPERTIES ( "cassandra.columns.mapping" = ":key, user_name, primary_email, home_phone" )

39
TBLPROPERTIES ( "cassandra.range.size" = "100", 
"cassandra.slice.predicate.size" = "100" );

For static Cassandra column families that model objects (such as users), mapping them to a relational structure is straightforward. In the example above, the first column of the Hive table (userid) maps to the row key in Cassandra. The row key in Cassandra is similar to a PRIMARY KEY in a relational table and should be the first column in your Hive table. If you know what the column names are in Cassandra, you can map the Hive column names to the Cassandra column names as shown above.

However, for dynamic column families (such as time series data), all rows likely have a different set of columns, and in most cases you do not know what the column names are. To convert this type of column family to a Hive table, you would convert a wide row in Cassandra to a collection of short rows in Hive using a special set of column names (row_key, column_name, value). For example:

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

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

```
hive> CREATE EXTERNAL TABLE PortfolioDemo.PortfolioStocks
    (portfolio string, ticker string, number_shares string)
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 should be able to do a SELECT to see the data stored in them. For example:

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

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

**Inserting Data into Cassandra via Hive**

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

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

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

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

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

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

**TBLPROPERTIES Reference**

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

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

**Performance Tuning and Best Practices**

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:

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

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

---

**Getting Started with Pig in DataStax Enterprise**

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.

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 DSE cluster. In DSE, the job tracker node information is stored in a column family in CassandraFS, and is initially populated on cluster startup by selecting the first Analytics node from the Cassandra seeds list. Assuming you have properly configured the Cassandra seeds list for DSE in `cassandra.yaml`, there is no additional configuration required. Pig clients will automatically select the correct job tracker node upon startup.

The default job tracker client port is 8012. If you are not sure which node in your DSE cluster is the job tracker, run the following command:

```plaintext
dsetool jobtracker
```

or in a binary distribution:

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

**Moving the Job Tracker Node for DSE**

If your primary Job Tracker node fails, DSE provides a utility to allow you to fail-over to an alternate job tracker node. The `dsetool movejt` utility can be used to move the job tracker to another Analytics node in the cluster.

1. Log in to a DSE Analytics node.
Analytics with Hadoop

2. Run the `dsetool movejt` command and specify the IP address of the new job tracker node for your DSE cluster. For example:

   ```
   dsetool movejt 110.82.155.4
   ```

   or in a binary distribution:

   ```
   <install_location>/bin/dsetool movejt 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, see if you can 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 will need to restart them so that they pick up the new job tracker node information.

**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.

See the [Pig Latin Manual](#) for 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()
AS (<rowkey_name>, columns: bag [T: tuple(<column_name>, <column_value>)]);

and for a super column family:

<pig_relation_name> = LOAD 'cassandra://<keyspace>/<column_family>'
USING CassandraStorage()
AS (<rowkey_name>, columns: bag [(<super_column_name>, subcolumns: bag [T: tuple(<column_name>, <column_value>)])));

Using the Pig STORE command, you push data from a Pig relation to Cassandra via the CassandraStorage handler. When pushing data to Cassandra, the Pig tuples you push must be in a format that maps to the column family (or super column family) structure.

The format of your tuples in Pig must be as follows for a regular column family:

(<row_key>,{(column_name),<value>},<column_name>,<value>))

and for a super column family:

(<row_key>,<supercolumn>:{(<column_name>,<value>),<column_name>,<value>}),<supercolumn>:{(<column_name>,<value>)})

Assuming that the tuples are in the correct format in your Pig relation, 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.

`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`.

`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`.

`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.

`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:

   `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:

   `[default@unknown] connect 110.82.155.4/9160`

   or if running on a single-node cluster as localhost:

   `[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}];`

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`.

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

   Notice how the data was aggregated for input into Cassandra. A *tuple* was constructed for each Cassandra row. In Pig notation, a tuple is enclosed in parentheses ( ). Within each row tuple, is a *bag* of column tuples - 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:

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

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

   ```
   (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>`

   ```
   grunt> 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:

   ```
   grunt> cassandra_data = LOAD 'cassandra://PigDemo/Scores' USING CassandraStorage() AS (name, columns: bag [T: tuple(score, value)]);
   ```

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

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

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

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

---

**Enterprise Search with Solr**

### About DataStax Enterprise Search

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.

**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. Data added to Solr is locally indexed in Cassandra.

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

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.

**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*.

**About the DSE Search Resource REST API**

DSE Search includes a REST API for adding and retrieving 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:

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

Generally, you can post any resource required by Solr to this URL. For example, stopwords.txt and elevate.xml are optional, frequently-used Solr configuration files that you post using this URL.

**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:

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

  --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.

Unsupported Features

DSE Search does not support:

- Supercolumns
- Counter columns
- Timeseries type rows
- Compound columns

Solr fields must be strings.

Links to Solr and Lucene Documentation

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

Getting Starting with DSE Search

DSE Search/Solr is supported on Linux and Mac. You can run Solr on one or more nodes, assuming you installed DataStax Enterprise 2.0 or higher. 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 DSE and DSE Search

Follow these steps to start DSE Search on a single node.
1. Start DSE as a Solr node. The method you use depends on your platform:

**RPM-Redhat or Debian installations**

Edit `/etc/default/dse`, set `SOLR_ENABLED=1`, and run this command:

```
/etc/init.d/dse service start
```

**Note**

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

**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 `-s` option.

```
cd <install_location>/bin
sudo ./dse cassandra -s
```

The `-s` option starts the Solr container inside DSE and marks the server as a search node.

**Note**

DataStax does not support using the `-s` and `-t` search and trackers options to mark the node for search and Hadoop analytics.

2. In another shell, check that your Cassandra ring is up and running. For example, on a Mac:

**RPM-Redhat or Debian installations**

```
dsetool ring -h localhost
```

**Tar distribution**

```
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 DSE Search 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:

   **RPM-Redhat or Debian installations**

   ```
cd /usr/share/dse-demos/wikipedia
```

   **Tar distribution**

   ```
cd <install_location>/demos/wikipedia
```

Enterprise Search with Solr
Enter the schema:

`./1-add-schema.sh`

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


`wiki.solr` in the URL represents the keyspace (wiki) and the column family (solr).

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.

If you want to download all the Wikipedia articles from the internet, start indexing articles using the following `wikifile` option instead of `wikipedia-sample.bz2`:

`./2-index.sh --wikifile enwiki-20110107-pages-articles25.xml-p023725001p026625000.bz2 --limit 10000`

The first 10k articles load. To load all the articles, use the `--limit` option.

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.

Using DataStax Enterprise and DSE Search, you can now:

- Run Hadoop MapReduce on the data through DSE Analytics.
- 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">
```

53
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:

![Solr Admin tool](image)

**Wikipedia Sample 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:
Enterprise Search with Solr

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
  Index Options: {class_name=com.datastax.bdp.cassandra.index.solr.SolrSecondaryIndex}

Compaction Strategy: org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy

Column metadata matches each field in the schema except the id field because id is the unique key.

The column metadata example shows some of the Cassandra Validator types in the Validation Class attribute. 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>StrField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>LongField</td>
<td>LongType</td>
</tr>
<tr>
<td>IntField</td>
<td>Int32Type</td>
</tr>
<tr>
<td>FloatField</td>
<td>FloatType</td>
</tr>
<tr>
<td>DoubleField</td>
<td>DoubleType</td>
</tr>
<tr>
<td>DateField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>ByteField</td>
<td>BytesType</td>
</tr>
<tr>
<td>BinaryField</td>
<td>BytesType</td>
</tr>
<tr>
<td>BoolField</td>
<td>UTF8Type</td>
</tr>
<tr>
<td>UUIDField</td>
<td>UUIDType</td>
</tr>
<tr>
<td>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.
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. 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](https://en.wikipedia.org/wiki/Solr) is built and queried using Solrj. The query is done using pure Ajax. No Cassandra API is used for the demo.

**The Integration of Solr Queries with Cassandra APIs**

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>]]
```

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, you must search using the LOCAL_QUORUM consistency.

**Example**

To query the Wikipedia demo search results:

1. Connect to the Cassandra Query Language (CQL) shell program. On the Mac, for example:

   ```bash
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:

```
use wiki;

SELECT title FROM solr 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>,<host>:<port>/solr/<keyspace2>.<cf2>
```

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

**DSE Search Management Operations**

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.

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

The 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.

**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 DSESimplenSnitch automatically puts all the Solr nodes in the same data center. Use OpsCenter Enterprise to rebalance the cluster.

**Inserting into, Modifying, and Deleting Data from a Solr Node**

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, by using a URL in the following format to update the document:

```
http://<host>:<port>/solr/<keyspace>.<column family>/update?
replacefields=false
```

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.

**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 if you are using a PropertyFileSnitch, you can use CQL.

1. Check the data center names of your nodes using the nodetool command.

   ```
   ./nodetool -h localhost ring
   ```

   **Note**

   The data center names, DC1 and DC2 in this example, must match the data center name configured for your snitch.

2. Start CQL on the DSE command line and create a keyspace that specifies the number of replicas you want.

   ```
   CREATE KEYSPACE test
   WITH strategy_class = 'NetworkTopologyStrategy'
   AND strategy_options:DC1 = 1
   AND strategy_options:DC2 = 3;
   ```

   The strategy options set the number of replicas in data centers, one replica in data center 1 and three in data center 2. For more information about adding replicas, see Choosing Keyspace Replication Options.

**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.

**About 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 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 neither workload competes with the other for data or computing resources.

**Setting the High-Performance Update Handler and Caching**

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:

```
<!-- 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>

You can also manage a number of caching operations that affect performance by setting the options in solrconfig.xml. For more information about the update handler and modifying SolrConfig.xml, see the [Solr documentation](http://solr.apache.org/).

### 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 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>.<column family>/<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, 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.

Moving Data to DataStax Enterprise Using Sqoop

DataStax Enterprise 2.0 includes support for Sqoop, an Apache Software Foundation tool for transferring data between an external data source and Hadoop.

About Sqoop
Sqoop is an Apache Software Foundation tool for transferring data between an RDBMS data source and Hadoop or between other data sources, such as NoSQL.

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

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.
**Getting Started**

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

Next, 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:
   ```bash
   sudo ./mysqld_safe --user=mysql
   ```
4. Start MySQL and create the demo database:
   ```bash
   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:

   **RPM-Redhat 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.

   ```
   LOAD DATA LOCAL INFILE 'npa_nxx.csv'
   INTO TABLE npa_nxx_demo.npa_nxx
   FIELDS TERMINATED BY ','
   ENCLOSED BY '"'
   LINES TERMINATED BY '
';
   ```

   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:

   **RPM-Redhat or Debian installations**

   Edit `/etc/default/dse`, set `HADOOP_ENABLED=1`, and run this command:

   ```
   /etc/init.d/dse service start
   ```

   **Tar distribution, such as Mac**

   Make the bin directory in the DSE installation directory, the current directory and run the `dse cassandra -t` command.

   ```
   cd <install_location>/bin
   sudo ./dse cassandra -t
   ```

   The -t option starts Hadoop and marks the node for Analytics.
9. Use the dse command in the bin directory to migrate the data from the MySQL table to text files in the CFS directory, npa_nxx.

*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.

```
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.

### Migrating Data to a Cassandra Column Family

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

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

### Cassandra Options to the Import Command

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

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

Next, the dse command passes the Cassandra parameters:

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

DSE interprets these command options as follows:
--cassandra-row-key <key>

The values of the key column of the source table become the row keys for each corresponding row in Cassandra.

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

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

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

--cassandra-create-schema

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

**Note**
The column family is created with no column metadata and all data is imported as strings.

**Usage Notes**

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

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

**About the Generated Sqoop JAR File**

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

**Sqoop Output**

Snippets of the output and end result of the import operation look like this:

```
INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
INFO tool.CodeGenTool: Beginning code generation
INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `npa_nxx` AS t LIMIT 1
INFO orm.CompilationManager: HADOOP_HOME is /Users/robin/dev/dse-2.0/resources/hadoop/bin/.. 
INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-robin/compile/2e2b8b85fba83ccf1f52a8ee77c3b12f/npa_nxx.jar
. . .
INFO mapreduce.ImportJobBase: Beginning import of npa_nxx
. . .
INFO config.DseConfig: Load of settings is done.
```
Moving Data to DataStax Enterprise Using Sqoop

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

Checking Imported Data

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

Using CQL to Check Imported Data

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

```
./cqlsh
use newKS;
select count(*) from npa_nxx_cf limit 200000;
```

The number of records appears.

```
count
-------
105291
```

```
select * from npa_nxx_cf where key IN (626794, 212524, 512538);
```

Records appear for Pasadena, New York, and Austin.

```
KEY | city     | lat | linetype | lon  | npa | nxx | state
----+----------+-----+----------+------+-----+-----+-------
626794| Pasadena | 34.17| L        | 118.13| 626 | 794 | CA
212524| New York | 40.71| L        | 074.01| 212 | 524 | NY
512538| Austin   | 30.27| L        | 097.74| 512 | 538 | TX
```

Validating Import Results in a Cluster
Moving Data to DataStax Enterprise Using Sqoop

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

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

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

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

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

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

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

**Getting Information about the Sqoop Command**

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

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

The help output for usage is:

```
usage: sqoop import [GENERIC-ARGS] [TOOL-ARGS]
```

**Cassandra arguments**

The help output for Cassandra is:

```bash
--cassandra-column-family <cf>
   Sets the target Cassandra column family for the import

--cassandra-create-schema
   If specified, Cassandra keyspace and column family are created.
   The column family must not exist if this flag is specified.

--cassandra-keyspace <keyspace>
   Import to <keyspace> in Cassandra

--cassandra-partitioner <partitioner>
   The partitioner class to use for writing to the Column Family.
   The default is RandomPartitioner.

--cassandra-password <passwd>
   Cassandra user password, if necessary

--cassandra-replication-factor <repFactor>
   The replication factor to use for the Keyspace.
   Requirements:
```

67
1) use `--cassandra-create-schema`

2) the keyspace does not already exist. Implies a Simple replication strategy. Defaults to 1 if neither `cassandra-rep-factor` nor `cassandra-strategy-options` are specified.

`--cassandra-row-key <keyCol>`
Specifies which input column to use as the row key

`--cassandra-strategy-options <stratOptions>`
Strategy options apply to the keyspace if `cassandra-create-schema` is specified and the keyspace does not already exist. Implies a Network topology replication strategy. This option and `cassandra-rep-factor` are mutually exclusive.

`--cassandra-thrift-host <thriftHost>`
Comma separated list of Cassandra thrift host(s)

`--cassandra-thrift-port <thriftPort>`
Cassandra thrift port - default 9160

`--cassandra-username <user>`
Cassandra user name, if necessary

**Other Arguments**
The help output for other arguments is:

**Common arguments**

`--connect <jdbc-uri>` Specify JDBC connect string

`--connection-manager <class-name>` Specify connection manager class name

`--connection-param-file <properties-file>` Specify connection parameters file

`--driver <class-name>` Manually specify JDBC driver class to use

`--hadoop-home <dir>` Override $HADOOP_HOME

`--help` Print usage instructions

`-P` Read password from console

`--password <password>` Set authentication password

`--username <username>` Set authentication username

`--verbose` Print more information while working

**Import control arguments**

`--append` Imports data in append mode

`--as-avrodatafile` Imports data to Avro data files
--as-sequencefile  Imports data to SequenceFiles
--as-textfile     Imports data as plain text (default)
--boundary-query <statement>  Set boundary query for retrieving max and min value of the primary key
--columns <col,col,col...>  Columns to import from table
--compression-codec <codec>  Compression codec to use for import
--direct           Use direct import fast path
--direct-split-size <n>  Split the input stream every 'n' bytes when importing in direct mode
-e,-query <statement>  Import results of SQL 'statement'
--fetch-size <n>  Set number 'n' of rows to fetch from the database when more rows are needed
--inline-lob-limit <n>  Set the maximum size for an inline LOB
-m,-num-mappers <n>  Use 'n' map tasks to import in parallel
--split-by <column-name>  Column of the table used to split work units
--table <table-name>  Table to read
--target-dir <dir>  HDFS plain table destination
--warehouse-dir <dir>  HDFS parent for table destination
--where <where clause>  WHERE clause to use during import
-z,-compress  Enable compression

Incremental import arguments
--check-column <column>  Source column to check for incremental change
--incremental <import-type>  Define an incremental import of type 'append' or 'lastmodified'
--last-value <value>  Last imported value in the incremental check column

Output line formatting arguments
--enclosed-by <char>  Sets a required field enclosing character
--escaped-by <char>  Sets the escape character
--fields-terminated-by <char>  Sets the field separator character
--lines-terminated-by <char>  Sets the end-of-line character
--mysql-delimiters  Uses MySQL's default delimiter set:
fields: ,lines: \n escaped-by: \n optionally-enclosed-by: '

--optionally-enclosed-by <char> Sets a field enclosing character

Input parsing arguments

--input-enclosed-by <char> Sets a required field encloser
--input-escaped-by <char> Sets the input escape character
--input-fields-terminated-by <char> Sets the input field separator
--input-lines-terminated-by <char> Sets the input end-of-line char
--input-optionally-enclosed-by <char> Sets a field enclosing character

Hive arguments

--create-hive-table Fail if the target hive table exists
--hive-delims-replacement <arg> Replace Hive record \0x01 and row delimiters (\n\r) from imported string fields with user-defined string
--hive-drop-import-delims Drop Hive record \0x01 and row delimiters (\n\r) from imported string fields
--hive-home <dir> Override $HIVE_HOME
--hive-import Import tables into Hive (Uses Hive's default delimiters if none are set.)
--hive-overwrite Overwrite existing data in the Hive table
--hive-partition-key <partition-key> Sets the partition key to use when importing to hive
--hive-partition-value <partition-value> Sets the partition value to use when importing to hive
--hive-table <table-name> Sets the table name to use when importing to hive
--map-column-hive <arg> Override mapping for specific column to hive types.

HBase arguments

--column-family <family> Sets the target column family for the import
--hbase-create-table If specified, create missing HBase tables
--hbase-row-key <col> Specifies which input column to use as the row key
--hbase-table <table> Import to <table> in HBase
Using the Log4j Appender

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

Configuring Logging Using Cassandra Log4j Appender

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

71
Using the Log4j Appender

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:

   ```
   # Cassandra Appender
   log4j.appender.CASS=com.datastax.logging.appender.CassandraAppender
   log4j.appender.CASS.hosts = 127.0.0.1
   log4j.appender.CASS.port = 9160
   #log4j.appender.CASS.appName = "myApp"
   #log4j.appender.CASS.keyspaceName = "Logging"
   #log4j.appender.CASS.columnFamily = "log_entries"
   #log4j.appender.CASS.placementStrategy =
   "org.apache.cassandra.locator.NetworkTopologyStrategy"
   #log4j.appender.CASS.strategyOptions = {"DC1" : "1", "DC2" : "3" }
   #log4j.appender.CASS.replicationFactor = 1
   #log4j.appender.CASS.consistencyLevelWrite = ONE
   #log4j.appender.CASS.maxBufferedRows = 256

   log4j.logger.com.foo.bar= INFO, CASS
   ```

   Commented lines are included for reference and to show the default values.

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

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

   Specify replication options in lines:

   ```
   log4j.appender.CASS.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,
   ```

72
Using the Log4j Appender

log_timestamp bigint,
logger_class_name text,
host_ip text,
host_name text,
message text,
method_name text,
ndc text,
thread_name text,
throwable_str_rep text
) WITH
  comment='',
  comparator=text,
  row_cache_provider='ConcurrentLinkedHashCacheProvider',
  key_cache_size=200000.000000,
  row_cache_size=0.000000,
  read_repair_chance=1.000000,
  gc_grace_seconds=864000,
  default_validation=text,
  min_compaction_threshold=4,
  max_compaction_threshold=32,
  row_cache_save_period_in_seconds=0,
  key_cache_save_period_in_seconds=14400,
  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,13288944555391 | logger_class_name,org.apache.log4j.Category | message,This is warn message #163 | thread_name,main |

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

KEY,37ba6b9c-9fd5-4dba-8fbc-51c1696bd235 | app_start_time,1328894454774 | level,ERROR | log_timestamp,1328894455392 | logger_class_name,org.apache.log4j.Category | message,Test exception. | thread_name,main | throwable_str_rep,java.io.IOException: Danger Will Robinson, Danger! | thread_name,main | throwable_str_rep,java.io.IOException: Danger Will Robinson, Danger! | thread_name,main |

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.
Using the Log4j Appender

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 *Configuring Logging Using Cassandra Log4j Appender*.

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:

   RPM-Redhat 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.

**Configuring Multiple Data Centers Quick Start**

Cassandra supports multiple data centers and replicates data across them automatically and transparently--no ETL work is necessary to move data between different systems or servers. In Cassandra, the term data center is synonymous with replication group - it is a grouping of nodes configured together for replication purposes. The data replication protects against hardware failure and other problems that cause data loss in a single cluster. This data replication can be distributed across multiple, geographically dispersed data centers, between different physical racks in a data center, and between public cloud providers and on-premise managed data centers. You can configure the number of *copies of the data* in each data center, and Cassandra handles the rest, replicating your data for you.

**Configuration Example (Tarball Installation)**

The procedures for configuring multiple data centers on Tarball and Debian/RPM installations are the same except the configuration files are located in different places.

To configure multiple data centers:

1. Install DSE on the nodes. Suppose, for example, you install DSE 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. Decide on names for each data center, for example DC1, DC2 or 100, 200.
3. Calculate tokens for nodes using one of the procedures that ensure even distribution of data. For example:

<table>
<thead>
<tr>
<th>Token</th>
<th>Token Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>56713727820156410577229101238628035242</td>
</tr>
<tr>
<td>2</td>
<td>113427455640312821154458202477256070485</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>56713727820156410577229101238628035252</td>
</tr>
<tr>
<td>5</td>
<td>113427455640312821154458202477256070495</td>
</tr>
</tbody>
</table>

4. For each node, change the cassandra.yaml file in `<install_location>/resources/cassandra/conf/`:

- Assign an `endpoint-snitch`.
- Specify an initial `token`.
- Specify a listen IP address.

For example:

```yaml
endpoint_snitch: com.datastax.bdp.snitch.DseDelegateSnitch
initial_token: <one of the tokens from step 3>
seeds: 10.168.66.41, 10.176.170.59
listen_address: <localhost IP address>
```

**Note**
It is a best practice to have at least one seed node per data center.

If you are configuring a mixed workload cluster, you need at least one seed node for each type of Cassandra and Hadoop workload. Solr nodes don’t require a seed node. All nodes need to have the same snitch configuration and the same list of seeds. The seed list needs to include a node from each data center.

5. For each node, change the dse.yaml file in `<install_location>/resources/dse/conf` to specify the snitch to be delegated by the DseDelegateSnitch. You can use the default, DseSimpleSnitch, or another snitch. For more information about snitches, see the `snitches` section.

6. In the cassandra-topology.properties file, assign a data center name and rack name to the IP addresses of each node. For example, use DC1 and DC2 for data center names and RAC1 for the rack name.

```properties
# Cassandra Node IP=Data Center:Rack

token 0: 0

10.168.66.41=DC1:RAC1
10.176.43.66=DC2:RAC1
10.168.247.41=DC1:RAC1
10.176.170.59=DC2:RAC1
10.169.61.170=DC1:RAC1
10.169.30.138=DC2:RAC1

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

7. Also, in the cassandra-topologies.properties file, assign a default data center name and rack name for unknown nodes.

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

8. **Start the nodes.**

9. Run the nodetool utility on one of the nodes to check that your multiple data centers are up and running:

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

<table>
<thead>
<tr>
<th>Address</th>
<th>DC</th>
<th>Rack</th>
<th>Status</th>
<th>State</th>
<th>Load</th>
<th>Owns</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Balancing the Data Center Nodes

When you deploy a Cassandra cluster, you need to use the partitioner to distribute roughly an equal amount of data to nodes. You also use identifiers for each data center (see step 6) in a formula to calculate tokens that balance nodes within a data center (DC). For example, assign each DC a numerical name that is a multiple of 100. Then for each DC, determine the tokens as follows: token = \( \left( \frac{2^{127}}{\text{num}_\text{nodes}_\text{in}_\text{dc}} \times n + \text{DC}_\text{ID} \right) \) where n is the node for which the token is being calculated and \( \text{DC}_\text{ID} \) is the numerical name.

Frequently Asked Questions

1. Can all the application data be 100% owned by Cassandra nodes?

There is no ownership. You set a replication factor (RF) for each data center, including the virtual analytics one. So, you might have one copy of the data in each of C1, C2, C3, and Analytics (AN), for example. Regardless of what data center or nodes you write to, the data is replicated to all four data centers. Replication is configured per keyspace.

For example, one keyspace with RF = \{C1:1, C2:1, C3:1, AN:0\}, and a different keyspace with RF = \{C1:0, C2:0, C3:0, AN:1\}. In such a configuration, if you write into the first keyspace, the analytics nodes do not have any copies of the data. If you write into the second keyspace only the analytics nodes have copies. If you write data (e.g. flat files) directly into CFS, by default only the AN nodes have copies of the data. The assumption is only the AN nodes need access to the flat files because their only use is for analytics. This is actually accomplished by having a Cassandra File System (CFS) keyspace, where AN has a RF > 0 and the others have RF=0.

2. Through replication, can you give the analytics nodes all of the application data?

Yes, as exemplified in the previous answer.

3. If all the analytics data is written to column families (CF), how can application nodes get a copy of the data?

The destination CFs used for the output of the analytics jobs are in a keyspace where only the Cassandra data centers have a RF > 0 (that is, the output of the analytics jobs do not need to be stored on the analytics nodes. There are common exceptions to this. If the output does not really belong in Cassandra, for some reason, such as the output is for analysis and not part of the operational data set, you can write the output into a keyspace where only the analytics DC had RF > 0. If you want both Cassandra and analytic nodes to have copies of the data, it is just a matter of setting the RF correctly on the keyspace you write to.

4. If you add or remove a node from a ring, do all nodes in the data centers need to be rebalanced?

You need to rebalance a data center after adding or removing a node. Nodes in other data centers do not have to be rebalanced. You need to stagger tokens between data centers for maximum data availability.

More Information About Configuring Data Centers

Links to more information about configuring a data center:

- Configuring nodes
- Choosing keyspace replication options
Elastic Workload Re-provisioning

When you install a node within a cluster, you mark it as either real-time (Cassandra), analytic (Hadoop), or search (Solr). To meet the requirements of changing workloads, DataStax Enterprise allows you to re-provision your existing Cassandra and Hadoop nodes at will and change the overall dynamic and capacity of your clusters.

For example, suppose an online Web application's daily operations dictate that a cluster’s is allocated as follows:

- 4 Cassandra nodes (for real-time/transactional processing)
- 2 Hadoop nodes (for analytics)
- 2 Solr nodes (for search)

To meet the analytic requirements of various marketing programs, more than two nodes are needed to perform the required analysis. To meet this need, you can re-provision two of the Cassandra nodes to Hadoop nodes during low traffic volume hours so the cluster looks like:

- 2 Cassandra nodes
- 4 Hadoop nodes
- 2 Solr nodes

Then after the Hadoop tasks are completed, you can return the cluster to its daily configuration.

Setting up a Node for Re-provisioning

The first step for enabling workload re-provisioning is to change the delegated snitch, which is designated by the DseDelegateSnitch (located in the dse.yaml file). A snitch is a configurable component of a Cassandra cluster that defines how the nodes are grouped together within the overall network topology.

By default the delegated snitch is the DseSimpleSnitch (org.apache.cassandra.locator.DseSimpleSnitch). In addition to the DseDelegateSnitch, DataStax Enterprise delegates to the standard Cassandra snitches. For more information see About Snitches in the DataStax Cassandra documentation.

The second step for RHEL and Debian Installations is to set the node's role in a configuration file and then restart the node.
The second step for tarball installations is stop the node and restart it in the desired role by setting an option. For more information about installing, see *Installing DataStax Enterprise Packaged Releases or Tarball Distribution*.

**Note**
Solr nodes cannot be re-provisioned.

**Delegating the Snitch**

The `DseDelegateSnitch` sets which snitch is used for re-provisioning. You need to only set the snitch one time. All nodes must use the same snitch in a cluster.

This section provides an example of delegating the `RackInferringSnitch` to enable workload re-provisioning. The `RackInferringSnitch` infers the topology of the network by analyzing the node IP addresses.

**To delegate a snitch:**

1. Open the `dse.yaml` file.
   - Packaged installations - `/etc/dse/dse.yaml`
   - Tarball installations - `<install_location>/resources/dse/conf/dse.yaml`
2. Set the delegated snitch and save the file:
   ```yaml```
   ```
   delegated_snitch: org.apache.cassandra.locator.RackInferringSnitch
   ```

**Re-provisioning Packaged RHEL and Debian Installations**

Packaged installations provide startup scripts in `/etc/init.d`.

1. Edit the `/etc/default/dse` file to set the node's role:
   - To make the node analytic: `HADOOP_ENABLE=1`
   - To make the node transaction, comment out `HADOOP_ENABLED=1`
2. Restart the node:
   ```
   $ sudo service dse restart
   ```

**Re-provisioning Tarball Installations**

Use these instructions for Mac and other tarball installations:

1. To stop a node, find the Cassandra Java process ID (PID) and *kill* the process using its PID number. For example:
   ```
   $ ps -ef | grep cassandra
   $ kill 1539
   ```
2. Start the node:
   - Analytics node: `dse cassandra -t`
   - Cassandra node: `dse cassandra`

**Note**
DataStax does not recommend running Hadoop and Solr on the same node in production environments.
DataStax Enterprise Release Notes

- DataStax Enterprise 2.0
- DataStax Enterprise 1.0
- DataStax Enterprise 1.0.2
- DataStax Enterprise 1.0.1

DataStax Enterprise 2.0

- Apache Cassandra 1.0.8
- Apache Hadoop 1.0.0
- Apache Hive 0.8.1
- Apache Pig 0.8.3
- Apache Sqoop 1.4.1
- Apache Solr 4.0
- DataStax OpsCenter 2.0

What’s New

With version 2.0 of DataStax Enterprise, DataStax has extended its big data platform to include:

- **Integration with Apache Solr™** - Enables DataStax Enterprise to use full-text search; hit highlighting; multiple search attributes; search rich documents, such as PDF and Microsoft Word; and use geo-spatial search. By including Solr 4.0, near real-time indexing can be performed to manage real-time, analytic, and enterprise search features within a single integrated platform.

- **Hadoop Update** - DataStax Enterprise 2.0 includes Hadoop 1.0.0.

- **Improved Hive Support** - DataStax Enterprise is now enabled for any JDBC compliant user interface to connect to and work with Hive from within the server. This version of Hive includes support for binary data and support for wide rows (up to 2 billion columns).

- **Sqoop** - ApacheSqoop supports migration of RDBMS data into the DataStax Enterprise server. You can now easily import data from RDBMS’s such as Oracle, Microsoft SQL Server, MySQL, Sybase, DB2, and others.

- **Elastic Workload Re-provisioning** - Provides the ability to re-provision existing nodes to assume a different workload, such as changing a real-time node to an analytic node, and the ability to change the overall usage and capacity of a cluster.

- **Log4j** - 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. Additionally, 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 more detailed information, see About DataStax Enterprise and What’s New in DataStax Enterprise 2.0? white paper.

Other issues

Do not use SolrJ (Java Interface for Solr) to add cores in DataStax Enterprise.

DataStax Enterprise 1.0
DataStax Enterprise 1.0.x is the first release of the DataStax commercial database platform. It is built on Apache Cassandra and designed for managing both real-time and analytic data workloads. Real-time data is managed with Cassandra and analytic operations are carried out via Apache Hadoop. DataStax Enterprise server is able to support both real-time and analytic workloads in the same cluster of machines with smart workload isolation transparently. This ensures that neither workload competes with the other for data or computing resources.

For component-specific information, refer to the components release notes and documentation.

### DataStax Enterprise 1.0.2

- Apache Cassandra 1.0.7 (updated from 1.0.5)
- Apache Hadoop 0.20.204.1
- Apache Hive 0.7.1
- Apache Pig 0.8.3
- DataStax OpsCenter 1.4

### Changes in 1.0.2

Pig driver now support integer data types. This is in addition to current support for ASCII, UTF8, and long types. Exceptions are no longer thrown for bytes, UUID, and counters, but the data isn't returned correctly.

### DataStax Enterprise 1.0.1

- Apache Cassandra 1.0.5
- Apache Hadoop 0.20.204.1
- Apache Hive 0.7.1
- Apache Pig 0.8.3
- DataStax OpsCenter 1.4

### Resolved Issues in 1.0.1

<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>363</td>
<td>Create the CassandraFS (cfs) keyspace with replication strategy options that respect the currently configured snitch.</td>
</tr>
<tr>
<td>368</td>
<td>Fix Debian packages so they do not start the DSE service by default.</td>
</tr>
</tbody>
</table>