DataStax Enterprise Documentation

February 16, 2012

Contents

DataStax 1.0 Documentation

Getting Started with DataStax Enterprise

DataStax Enterprise (DSE) Edition is a packaged distribution of *Apache Cassandra 1.0* and Apache Hadoop made available by DataStax for its enterprise support customers. It comes bundled with CQL drivers and a sample application to quickly get you up and running with Cassandra and Hadoop.

About DataStax Enterprise

DataStax Enterprise (DSE) is a commercial distribution of Apache Cassandra and Apache Hadoop developed by DataStax. DSE provides Hadoop MapReduce capabilities using CassandraFS, an HDFS-compatible storage layer inside of Cassandra. By replacing HDFS with CassandraFS, users are able to leverage their current MapReduce jobs on Cassandra's peer-to-peer, fault-tolerant, and scalable architecture. DataStax Enterprise is also able to support dual workloads, allowing you to use the same cluster of machines for both real-time applications and data analytics without having to move the data around between systems.

Key Features of DataStax Enterprise

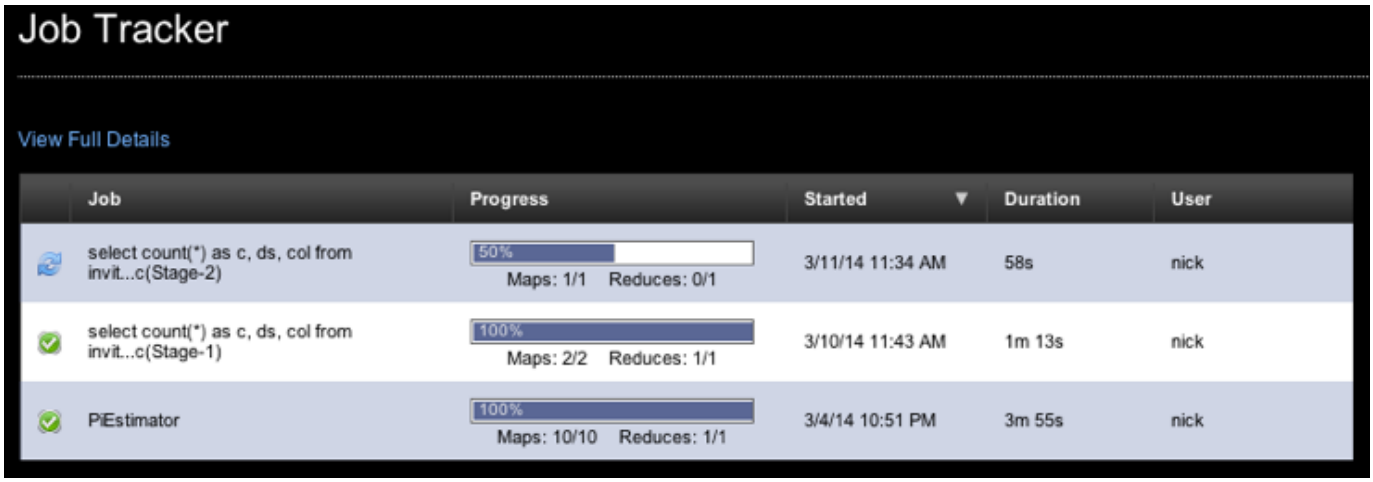
Some of the key features of DataStax Enterprise include:

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 metadata about how the cluster is configured. If the NameNode fails, the Hadoop system is down. With CassandraFS, all nodes are peers. Data files can be loaded through any node in the cluster, and any node can serve as the JobTracker for MapReduce jobs.

Streamlined Setup and Operations - In Hadoop, there is the notion of having 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 of one or one hundred, the configuration is the same. Since there is no NameNode, all nodes in a CassandraFS are the same. Likewise with running Hive against DataStax Enterprise. Hive has a metastore which is where it stores its schema. In regular Hive, this is a stand-alone database which 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 (Cassandra keyspace available through any node in the cluster without any additional configuration.

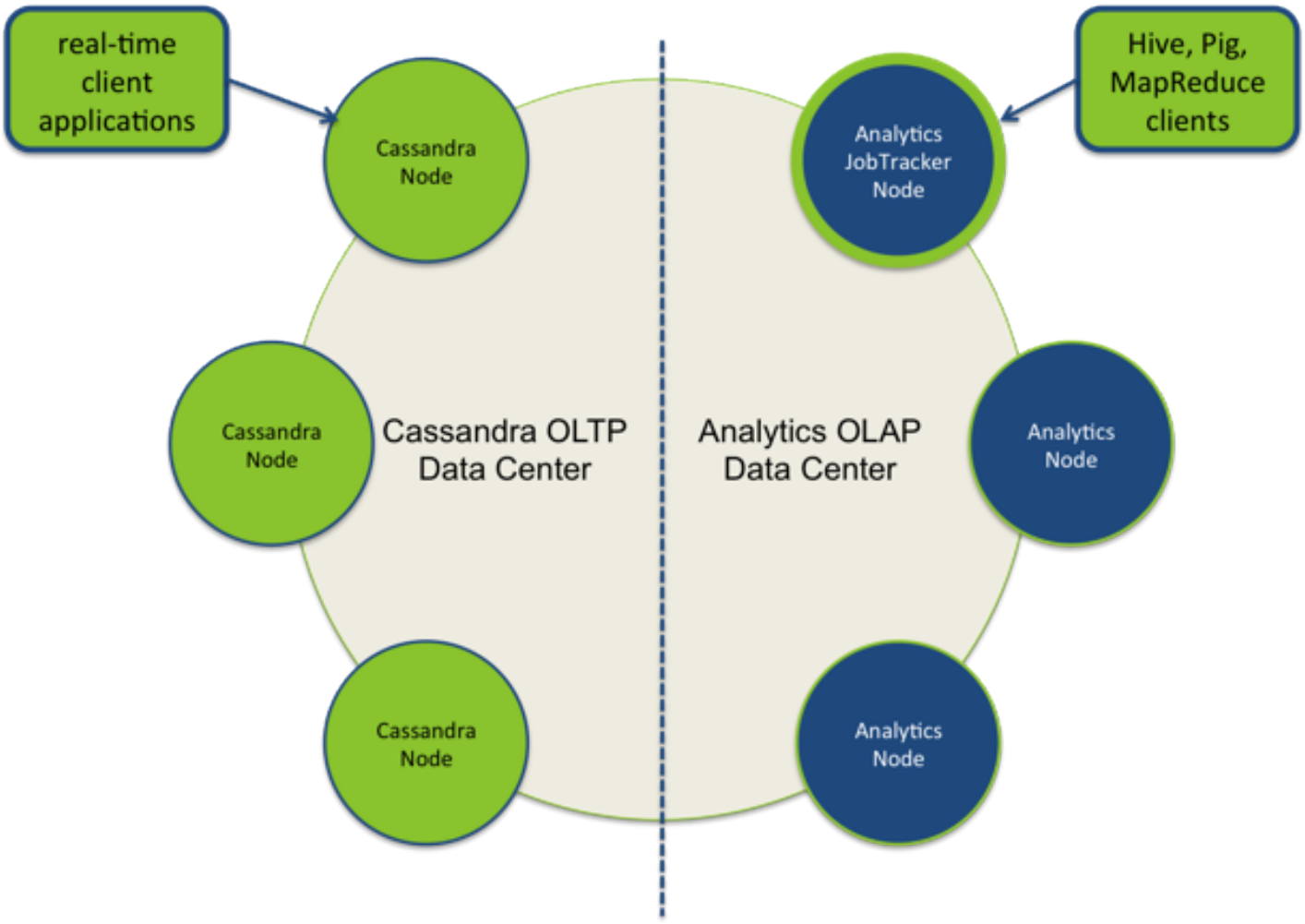
Analytics Without ETL - When using DataStax Enterprise, it is possible to run MapReduce jobs directly against your data in Cassandra. You can even perform real-time and analytic workloads at the same time without one workload affecting the performance of the other. Using Cassandra's multi-datacenter support, 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.

Full Integration with DataStax OpsCenter 1.4 - DataStax OpsCenter allows you to monitor and administer your DataStax Enterprise cluster in one easy-to-use graphical interface. Using OpsCenter, you can see detailed health and status information about your DataStax Enterprise cluster, including the status of Hadoop MapReduce jobs running on the cluster. To install DataStax OpsCenter, see the DataStax OpsCenter Install Guide.



About the DataStax Enterprise Architecture

DataStax Enterprise combines Apache Cassandra with Hadoop. 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 and Cassandra real-time 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, the Hadoop Distributed File System (HDFS) is replaced by CassandraFS. 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 (set by configuring the Cassandra seed node for your DSE 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 DSE 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.

About Pig in DataStax Enterprise

DataStax Enterprise includes a Cassandra-enabled Pig MapReduce 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 complied 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 DataStax Enterprise (no Name Node), there is no concept of local mode for Pig. Pig in DSE 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.

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 DSE, see:

- *Installing DataStax Enterprise Packaged Releases or Tarball Distribution*
- *Initializing a Multi-Node DataStax Enterprise Cluster*

To get started with the Hive and Pig MapReduce clients bundled with DSE, see:

- *Getting Started with Hive in DataStax Enterprise*
- *Getting Started with Pig in DataStax Enterprise*

Additional Documentation References

For more information about Apache Cassandra 1.0, see the *Cassandra 1.0 Documentation*.

For more information about Hadoop MapReduce, Hive, and Pig, see the MapReduce Getting Started Guide, the Hive Getting Started Guide, and the Pig Latin Reference Manuals on the Apache Hadoop project web site.

Quick Start with DataStax Enterprise

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

Getting up and running takes just four simple steps:

1. Register with DataStax. DataStax Enterprise is available to DataStax registered users and support customers. To download DSE, 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.
2. *Make sure you have Java installed*
3. *Install DataStax Enterprise*
4. *Set a couple of configuration properties and start the DSE server*

Checking for a Java Installation

DSE is a Java program and requires a Java Virtual Machine (JVM) to be installed before you can start the server. For production deployments, you will need the Sun Java Runtime Environment 1.6.0_19 or later, but if you are just kicking the tires, any JVM should do.

To check for Java, run the following command:

```
# java -version
```

If you do not have Java installed, see *install-jre-rh* or *install-jre-deb* for instructions.

Installing the DataStax Enterprise Binaries

The quickest way to get going on a single node with DataStax Enterprise (DSE) is to install the DSE binary tarball packages. This allows you to install everything in a single location (such as your home directory), and does not require root permissions.

DataStax Enterprise is comprised of two components - The DSE server (Cassandra and CassandraFS-enabled Hadoop plus a sample demo application) and DataStax OpsCenter (a web-based monitoring application for Cassandra and DSE Analytics). Note that DataStax OpsCenter is not currently supported on MacOS.

These instructions will walk you through setting up a self-contained, single-node instance of DataStax Enterprise in your home directory (does not require root permissions).

Note

The instructions in this section are not intended for production installations, just for a quick start. See *Installing DataStax Enterprise Packaged Releases or Tarball Distribution* and `init-dse` for DSE cluster installation best practices.

Note

By downloading DSE software from DataStax you agree to the terms of the DataStax Enterprise EULA (end user license agreement) posted on the DataStax web site.

1. In your home directory, create a directory called `datastax`. In that directory download the DSE package (required), plus the OpsCenter package (optional). Substitute `<username>:<password>` with your correct DataStax login credentials.

```
$ cd $HOME

$ mkdir datastax

$ cd datastax

$ wget http://<username>:<password>@downloads.datastax.com/enterprise/dse-1.0.1-bin.tar.gz

$ wget http://<username>:<password>@downloads.datastax.com/enterprise/opscenter.tar.gz
```

2. Unpack the distributions:

```
$ tar -xzvf dse-1.0.1-bin.tar.gz

$ tar -xzvf opscenter-1.4.tar.gz

$ rm *.tar.gz
```

3. For convenience, set the following environment variables in your user environment. For example, to configure your environment in your `$HOME/.bashrc` file:

   a. Open your .bashrc file in a text editor (such as `vi`):

```
vi $HOME/.bashrc
```

   b. Add the following lines to bottom of the file:

```
export DSE_HOME=$HOME/datastax/dse-1.0.1
export CASSANDRA_HOME=$DSE_HOME/resources/cassandra
export OPSC_HOME=$HOME/datastax/opscenter-1.4
export PATH="$PATH:$DSE_HOME/bin:$CASSANDRA_HOME/bin:$OPSC_HOME/bin"
```

   c. Save and close the file.

   d. Source the file.

```
source $HOME/.bashrc
```

4. Create the data and logging directories needed by DSE.

```
$ mkdir $HOME/datastax/dse-data
```

Configuring and Starting a Single-Node DataStax Enterprise Cluster

DSE is intended to be run on multiple nodes, however a single node cluster is great for evaluation purposes. To configure and start a single DSE analytics node:

1. Set the configuration properties needed to start your cluster in the `$DSE_HOME/resources/cassandra/conf/cassandra.yaml` file. This will configure DSE to run a single-node cluster on the localhost and store all of its data files in your home directory.

```
$ sed -i -e "s,initial_token:,initial_token: 0," \
  $DSE_HOME/resources/cassandra/conf/cassandra.yaml

$ sed -i -e "s,listen_address:,listen_address: 127.0.0.1," \
  $DSE_HOME/resources/cassandra/conf/cassandra.yaml

$ sed -i -e "s,- /var/lib/cassandra/data,- $HOME/datastax/dse-data," \
  $DSE_HOME/resources/cassandra/conf/cassandra.yaml

$ sed -i -e "s,saved_caches_directory: /var/lib/cassandra/saved_caches, \
  saved_caches_directory: $HOME/datastax/dse-data/saved_caches," \
  $DSE_HOME/resources/cassandra/conf/cassandra.yaml

$ sed -i -e "s,commitlog_directory: /var/lib/cassandra/commitlog,commitlog_directory: \
  $HOME/datastax/dse-data/commitlog," $DSE_HOME/resources/cassandra/conf/cassandra.yaml
```

2. Set the DSE server log location in the `$DSE_HOME/resources/cassandra/conf/log4j-server.properties` file to the log directory you created earlier:

```
$ sed -i -e "s,log4j.appender.R.File=/var/log/cassandra/system.log, \
  log4j.appender.R.File=$HOME/datastax/dse-data/system.log," \
  $DSE_HOME/resources/cassandra/conf/log4j-server.properties
```

3. Configure the DataStax demo application to point to the correct Cassandra installation location:

```
$ sed -i -e "s,/usr/share/cassandra,$HOME/datastax/dse-1.0.1/resources/cassandra," \
  $DSE_HOME/demos/portfolio_manager/bin/pricer
```

4. Start DSE (as an analytics node). The `-t` option starts Cassandra plus CassandraFS and the Hadoop JobTracker and TaskTracker processes. Because there is no Hadoop NameNode with CassandraFS, there is no additional configuration to run MapReduce jobs in single mode versus distributed mode when using DataStax Enterprise.

```
$ dse cassandra -t
```

   When running on a single node, the Cassandra seed node and DSE job tracker node are automatically set to `localhost`.

5. Check that your DSE ring is up and running:

```
$ nodetool ring -h localhost
```

6. For the next hands-on steps, run the *Portfolio Demo* example application.

Configuring and Starting DataStax OpsCenter

DataStax OpsCenter is a graphical web application that can be used to manage and monitor a DataStax Enterprise cluster. These instructions explain how to configure and start OpsCenter to monitor the single-node DSE Analytics cluster you just set up in your home directory.

Note

OpsCenter is currently not supported on MacOS, only on Linux platforms.

1. OpsCenter requires that python 2.5 or higher and `openssl` be installed. Check for these and install them if they are not found before you continue.

   ```
   $ python -V
   $ which openssl
   ```

2. Set the configuration properties needed to start OpsCenter in the `$OPSC_HOME/conf/opscenterd.conf` file.

   ```
   $ sed -i -e "s,interface = 127.0.0.1,interface = 0.0.0.0," \
     $OPSC_HOME/conf/opscenterd.conf
   ```

3. From the OpsCenter home directory, run the `create-agent-tarball` utility. This sets up SSL for OpsCenter and creates a `agent.tar.gz` file for installing the agents (may take a couple of minutes to return).

   ```
   $ cd $OPSC_HOME

   $ create-agent-tarball
   ```

4. Start OpsCenter in the background.

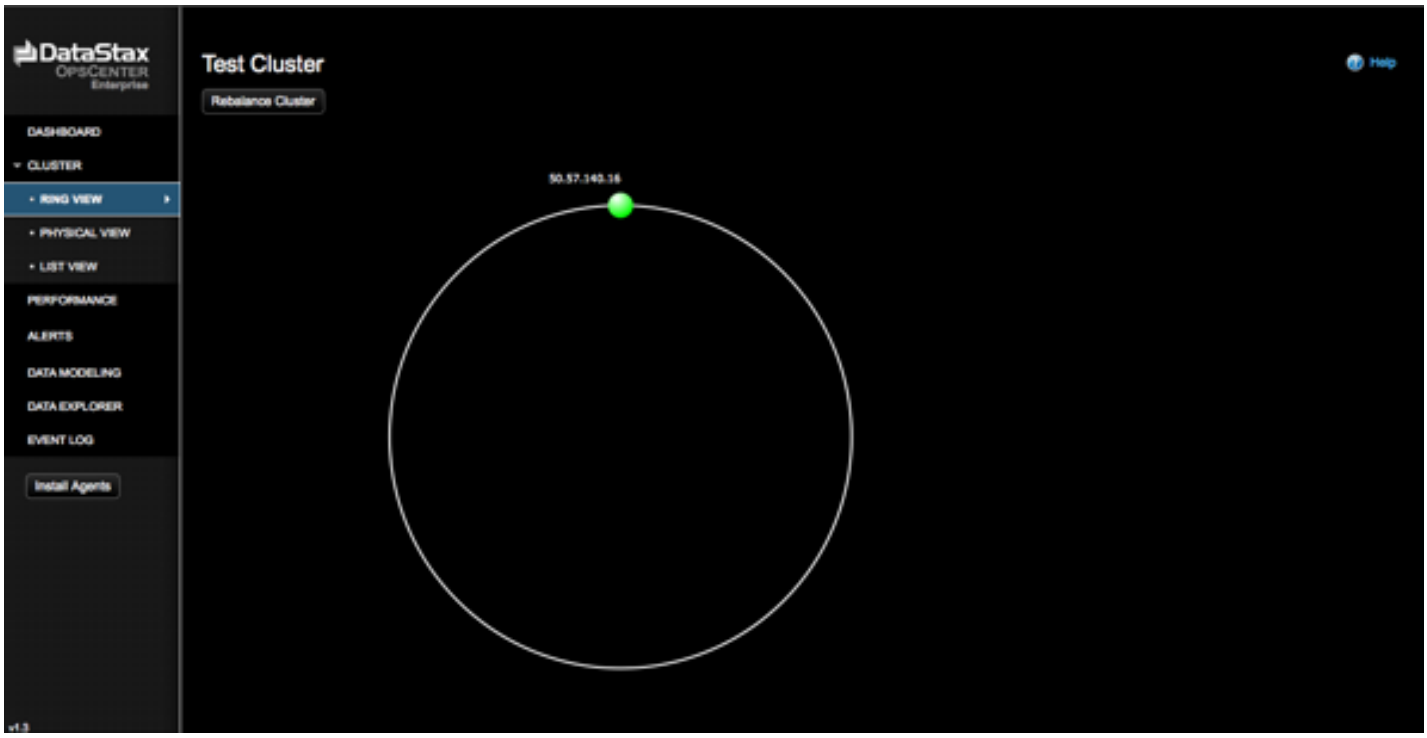
   ```
   $ opscenter &
   ```

5. The next step is to install, configure, and start the OpsCenter agent on the local node.

   ```
   $ mkdir $HOME/datastax/opsc-agent

   $ mv $OPSC_HOME/agent.tar.gz $HOME/datastax/opsc-agent

   $ cd $HOME/datastax/opsc-agent

   $ tar -xvzf agent.tar.gz

   $ rm agent.tar.gz

   $ ./agent/bin/setup 127.0.0.1 127.0.0.1

   $ ./agent/bin/opscenter-agent
   ```

6. Open a browser and go to `http://localhost:8888` (if running on the local machine) or `http://<webhost_ip>:8888` (if running remotely - specify the correct IP address of the remote web server).

This will open the OpsCenter Dashboard. If you select **Cluster > Ring View**, you should see your single DSE Analytics node running on the localhost.



Running the Portfolio Manager Demo Application

Your DataStax Enterprise (DSE) installation contains a demo portfolio manager application that showcases how you can run a mixed workload on a DSE cluster. The demo is located in `/usr/share/dse-demos/portfolio_manager` for packaged installations or `$DSE_HOME/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 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 is used to calculate 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 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.

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

   or

   ```
   cd $DSE_HOME/demos/portfolio_manager
   ```

   Note

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

2. Run the `./bin/pricer` utility to generate stock data for the OLTP application. To see all of the available options for this utility:

   ```
   ./bin/pricer --help
   ```

   The following examples will generate 100 days worth of historical data.

   If running on a single node cluster on localhost:

   ```
   ./bin/pricer -o INSERT_PRICES
   ./bin/pricer -o UPDATE_PORTFOLIOS
   ./bin/pricer -o INSERT_HISTORICAL_PRICES -n 100
   ```

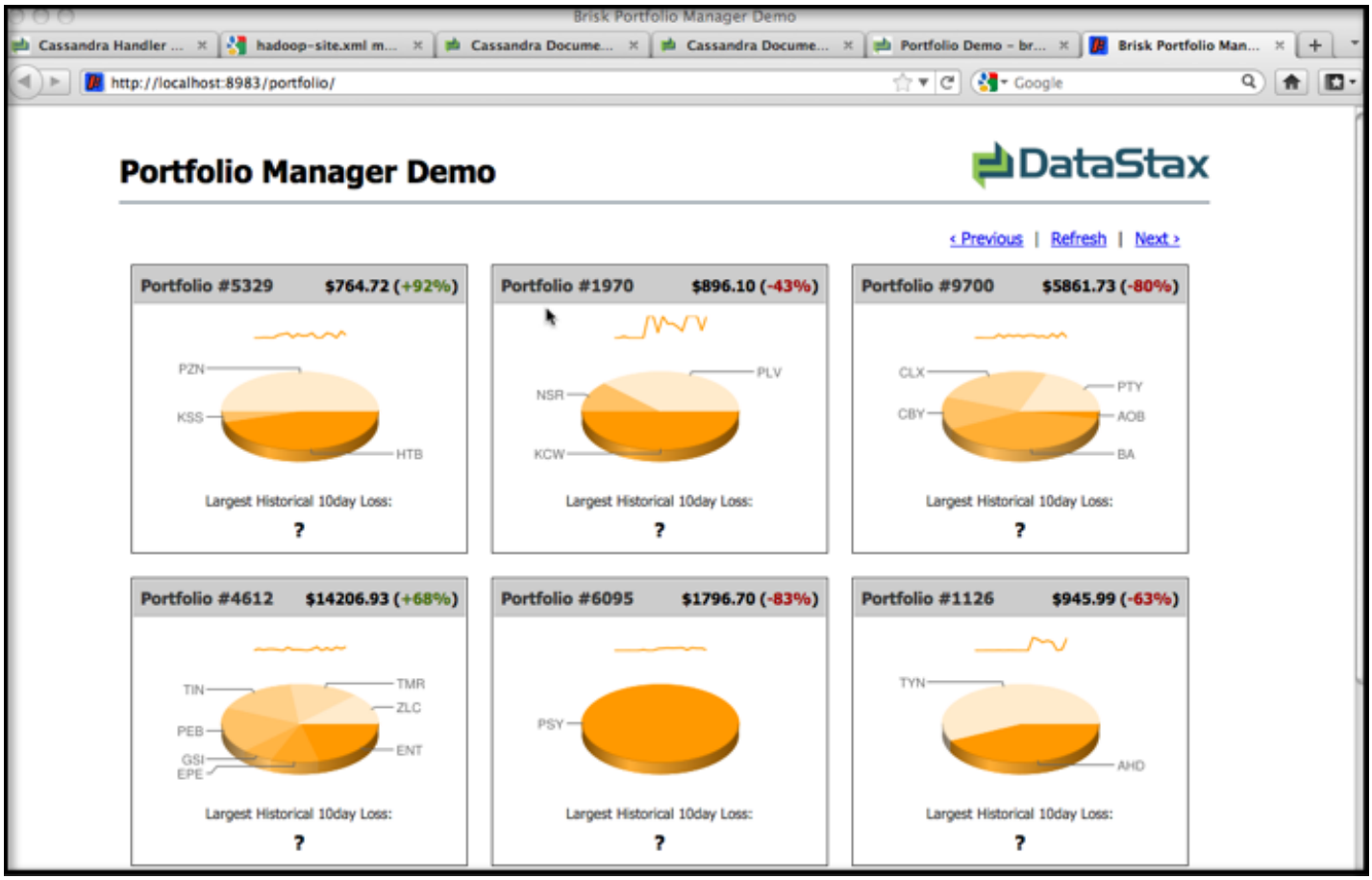
   If running the demo on a cluster, you also need to supply the IP address of the DSE Analytics node from where you are running the demo. For the first command, also supply the replication strategy for the keyspace that gets created. For example:

   ```
   ./bin/pricer -o INSERT_PRICES -d 110.82.155.0 \
   --replication-strategy="org.apache.cassandra.locator.NetworkTopologyStrategy" \
   --strategy-properties="Analytics:1,Cassandra:1"

   ./bin/pricer -d 110.82.155.0 -o UPDATE_PORTFOLIOS

   ./bin/pricer -d 110.82.155.0 -o INSERT_HISTORICAL_PRICES -n 100
   ```

4. Start the web service.

   ```
   cd website
   java -jar start.jar &
   ```

5. Open a browser and go to `http://localhost:8983/portfolio` to see the real-time Portfolio Manager demo application.



6. Open another shell window.
7. 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:

   ```
   $DSE_HOME/bin/dse hive -f $DSE_HOME/demos/portfolio_manager/10_day_loss.q
   ```

8. 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.
9. After the job completes, refresh the Portfolio Manager web page to see the results of *Largest Historical 10day Loss* for each portfolio.

Installing DataStax Enterprise Packaged Releases or Tarball Distribution

DataStax provides Debian and RedHat packaged releases for DataStax Enterprise (DSE) and a binary tarball distribution. RPM and Debian packages are currently supported through the `yum` and `apt` package management tools.

To access the DSE package repositories, you will need your DataStax registration username and password. If you are not sure of your username and password, contact DataStax Support.

Note

DataStax Enterprise requires the Sun Java Runtime Environment version 1.6.0_19 or later be installed on all DSE nodes. RPM packaged releases do not install the Sun JRE, but most Debian packaged releases do. See the Sun Java installation instructions in *install-jre-rh* or *install-jre-deb* if you need help installing Java.

Installing DataStax Enterprise RPM Packages

DataStax provides yum repositories for RedHat Enterprise Linux 5 and 6 and Fedora 12, 13 and 14. These instructions assume that you have the yum package management application installed, and that you have root (or sudo) access on the machine where you are installing.

Note

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

1. Before you begin, make sure you have EPEL (Extra Packages for Enterprise Linux) installed. EPEL contains dependent packages required by DSE, such as `jna` and `jpackage-utils`:

   ```
   rpm -Uvh http://download.fedora.redhat.com/pub/epel/5/i386/epel-release-5-4.noarch.rpm
   ```

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

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

3. In this file add the following lines where `<username>` and `<password>` are the DataStax account credentials from your registration confirmation email:

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

4. Install DSE using yum:

   ```
   yum install dse-full opscenter
   ```

Installing DataStax Enterprise Debian Packages

DataStax provides debian package repositories for Debian 6.0 (Squeeze), Debian 5.0 (Lenny), Ubuntu Lucid (10.04), and Ubuntu Maverick (10.10). These instructions assume that you have the aptitude package management application installed, and that you have root (or sudo) access on the machine where you are installing.

Note

By downloading DSE software from DataStax you agree to the terms of the DataStax Enterprise EULA (end user license agreement) posted on the DataStax web site.

1. Edit the `/etc/apt/sources.list` file, and add the DataStax repository for your operating system where `<username>` and `<password>` are the DataStax account credentials from your registration confirmation email.

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

2. Add the DataStax repository key to your aptitude trusted keys.

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

3. Install the DSE packages.

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

4. By default, the Debian packages start the `dse` service automatically. To stop the service and clear the initial gossip history that gets populated by this initial start:

```
$ sudo service dse stop
$ sudo bash -c 'rm /var/lib/cassandra/data/system/*'
```

About DataStax Enterprise Packaged Installations

The packaged releases create a user `cassandra`. When starting DSE 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.

The package installs into the following directories:

Cassandra Directories

- `/var/lib/cassandra` (Cassandra and CassandraFS data directories)
- `/var/log/cassandra`
- `/var/run/cassandra`
- `/usr/share/dse/cassandra` (Cassandra environment settings)
- `/usr/share/dse/cassandra/lib`
- `/usr/share/dse-demos` (Portfolio Manager demo application)
- `/usr/bin`
- `/usr/sbin`
- `/etc/dse/cassandra` (Cassandra configuration files)
- `/etc/init.d`
- `/etc/security/limits.d`
- `/etc/default/`

Hadoop Directories

- `/usr/share/dse/hadoop` (Hadoop environment settings)
- `/etc/dse/hadoop` (Hadoop configuration files)

Hive Directories

- `/usr/share/dse/hive` (Hive environment settings)
- `/etc/dse/hive` (Hive configuration files)

Pig Directories

- `/usr/share/dse/pig` (Pig environment settings)
- `/etc/dse/pig` (Pig configuration files)

DataStax OpsCenter Directories

See the OpsCenter Installation Guide for instructions on setting up OpsCenter to monitor your DataStax Enterprise cluster.

- /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)

Installing the DataStax Enterprise Tarball Distribution

For platforms that do not have package support of if you want to do a not-root installation, DataStax provides a binary tarball distribution of DataStax Enterprise. Before installing, make sure that you have the Sun Java Runtime Environment version 1.6.0_19 or later installed on all DSE nodes.

Note

By downloading DSE software from DataStax you agree to the terms of the DataStax Enterprise EULA (end user license agreement) posted on the DataStax web site.

1. Download the distribution to a location on your machine and unpack it, where `<username>:<password>` is your DataStax login credentials:

   ```
   wget http://<username>:<password>@downloads.datastax.com/enterprise/dse.tar.gz

   tar -xzvf dse.tar.gz
   ```

2. For convenience, you may want to set the following environment variables in your user environment (for example, add the following to `~/.bashrc`):

   ```
   export DSE_HOME=<install_location>/dse-1.0.2
   export PATH=$PATH:$DSE_HOME/bin
   ```

3. Create the data and logging directories needed for Cassandra.

   By default, Cassandra is configured to use `/var/lib/cassandra` and `/var/log/cassandra`. You can either create those directories (requires root or sudo), for example where *$USER* is the user that will run DSE:

   ```
   # mkdir /var/lib/cassandra
   # mkdir /var/log/cassandra
   # chown -R $USER:$GROUP /var/lib/cassandra
   # chown -R $USER:$GROUP /var/log/cassandra
   ```

Or you can define your own data directory locations and edit the following properties in the `$DSE_HOME/resources/cassandra/conf/cassandra.yaml` configuration file before starting:

```
initial_token: 0
data_file_directories: <home_directory_path>/cassandra-data
commitlog_directory: <home_directory_path>/cassandra-data/commitlog
saved_caches_directory: <home_directory_path>/cassandra-data/saved_caches
```

About DataStax Enterprise Tar Installations

DSE Directories

- `bin` (DSE 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/pig` (Pig installation)

Installing JNA

Installing JNA (Java Native Access) on Linux platforms can improve Cassandra memory usage. With JNA installed and configured as described in this section, Linux does not swap out the JVM, and thus avoids related performance issues.

To install JNA

1. Download `jna.jar` from the JNA project site.
2. Add `jna.jar` to `$DSE_HOME/lib/` and `$DSE_HOME/resources/cassandra/lib` (or otherwise place it on the CLASSPATH).
3. Edit the file `/etc/security/limits.conf`, adding the following entries for the user or group that runs DSE:

```
$USER soft memlock unlimited
$USER hard memlock unlimited
```

Next Steps

For next steps see *Configuring and Initializing a DataStax Enterprise Cluster* and then *Starting a DataStax Enterprise Cluster*.

After you have your cluster up and running, see the OpsCenter Installation Guide for instructions on setting up OpsCenter to monitor your DataStax Enterprise cluster.

Initializing a DataStax Enterprise Cluster on Amazon EC2

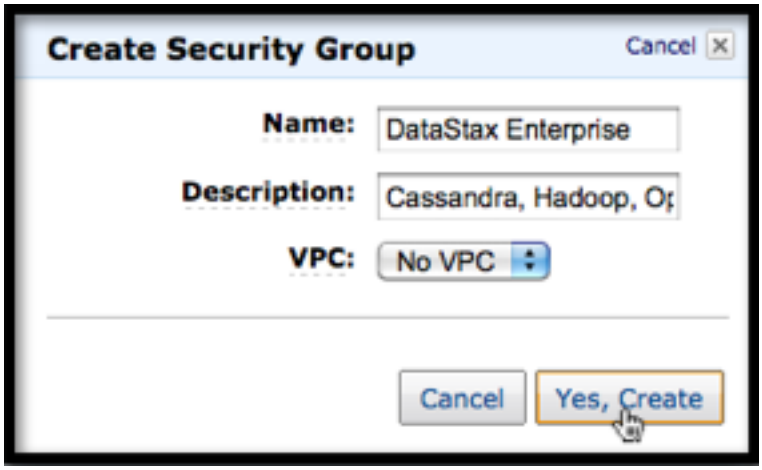
This is a step-by-step guide to using the Amazon Web Services EC2 Management Console to set up a DataStax Enterprise (DSE) cluster using the DataStax AMI (Amazon Machine Image). Installing via the AMI allows you to quickly deploy a cluster with a mixed OLTP/OLAP workload pre-configured. When you launch the AMI, you can specify the total number of nodes in your cluster and how many nodes should be Analytics (Hadoop) nodes versus pure Cassandra nodes. The AMI sets up multi-datacenter replication using a special `DseSimpleSnitch`. This allows data to be replicated between the low-latency Cassandra side of the cluster to the Hadoop analytics side of the cluster without impacting your real-time application performance.

The DataStax Enterprise AMI does the following:

- Installs the latest DataStax Enterprise on an Ubuntu 10.10 image
- Uses RAID0 ephemeral disks for data storage and commit log
- Uses the private interface for intra-cluster communication
- Starts the nodes in the specified mode (Analytics or Cassandra)
- Configures a Cassandra cluster using the RandomPartitioner
- Configures the Cassandra replication strategy using the `DseSimpleSnitch` (for a mixed-workload cluster)
- Sets the seed nodes cluster-wide
- Installs DataStax OpsCenter on the first node in the cluster (by default)

Creating an EC2 Security Group for DataStax Enterprise

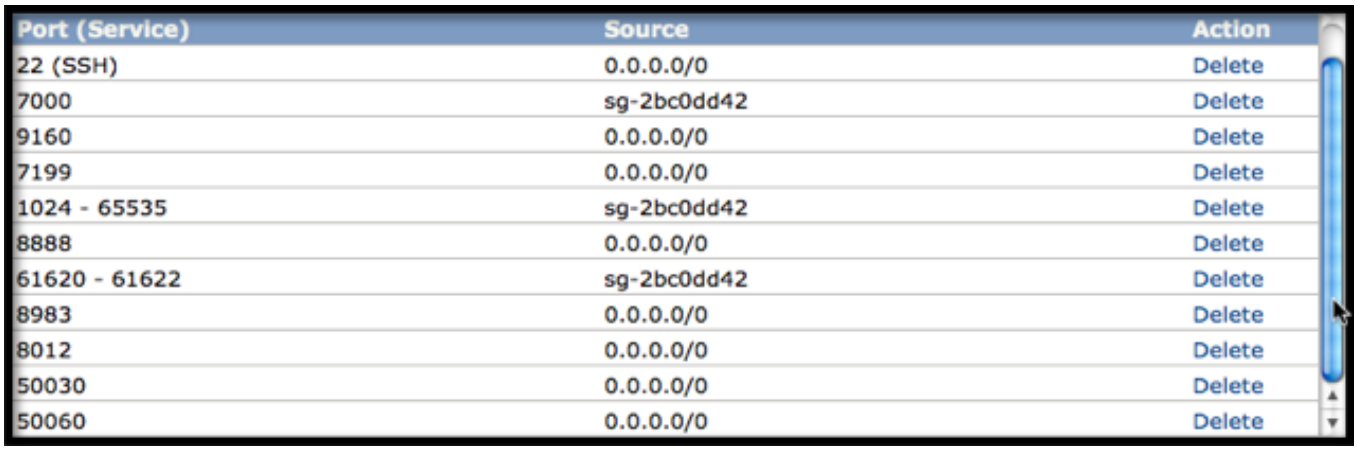
1. In your Amazon EC2 Console Dashboard, select **Security Groups** in the **Network & Security** section.
2. Click **Create Security Group**. Fill out the name and description and click **Yes, Create**.



3. Click **Inbound** and add rules for the following ports.

Port	Rule Type	Description
22	SSH	Default SSH port
7000	Custom TCP Rule	Cassandra intra-node port (source is the current security group)
9160	Custom TCP Rule	Cassandra client port
7199	Custom TCP Rule	Cassandra JMX monitoring port
1024+	Custom TCP Rule	JMX reconnection/loopback ports (source is the current security group)
8888	Custom TCP Rule	OpsCenter website port
61620	Custom TCP Rule	OpsCenter intra-node monitoring port (source is the current security group)
61621	Custom TCP Rule	OpsCenter agent ports (source is the current security group)
8983	Custom TCP Rule	DataStax demo application website port
8012	Custom TCP Rule	Hadoop Job Tracker client port
9290	Custom TCP Rule	Hadoop Job Tracker Thrift port (source is the current security group)
50030	Custom TCP Rule	Hadoop Job Tracker website port
50031	Custom TCP Rule	OpsCenter HTTP proxy for Job Tracker (source is the current security group)
50060	Custom TCP Rule	Hadoop Task Tracker website port

4. After you are done adding the above port rules, click **Apply Rule Changes**. Your completed port rules should look something like this:



Port (Service)	Source	Action
22 (SSH)	0.0.0.0/0	Delete
7000	sg-2bc0dd42	Delete
9160	0.0.0.0/0	Delete
7199	0.0.0.0/0	Delete
1024 - 65535	sg-2bc0dd42	Delete
8888	0.0.0.0/0	Delete
61620 - 61622	sg-2bc0dd42	Delete
8983	0.0.0.0/0	Delete
8012	0.0.0.0/0	Delete
50030	0.0.0.0/0	Delete
50060	0.0.0.0/0	Delete

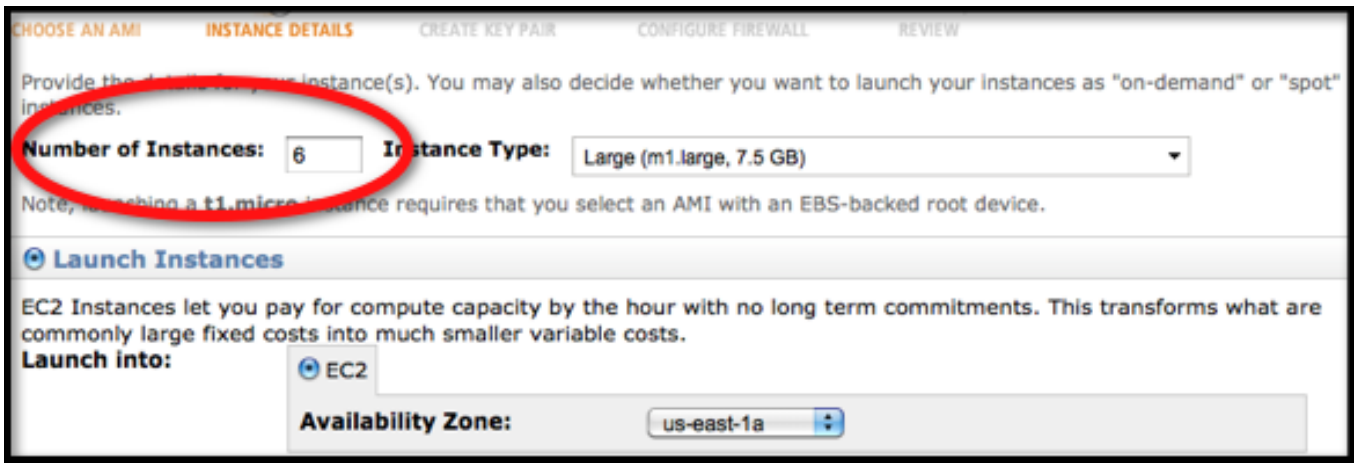
Note

This security configuration shown in this example opens up all externally accessible ports to incoming traffic from any IP address (0.0.0.0/0). If you desire a more secure configuration, see the Amazon EC2 help on Security Groups for more information on how to configure more limited access to your cluster.

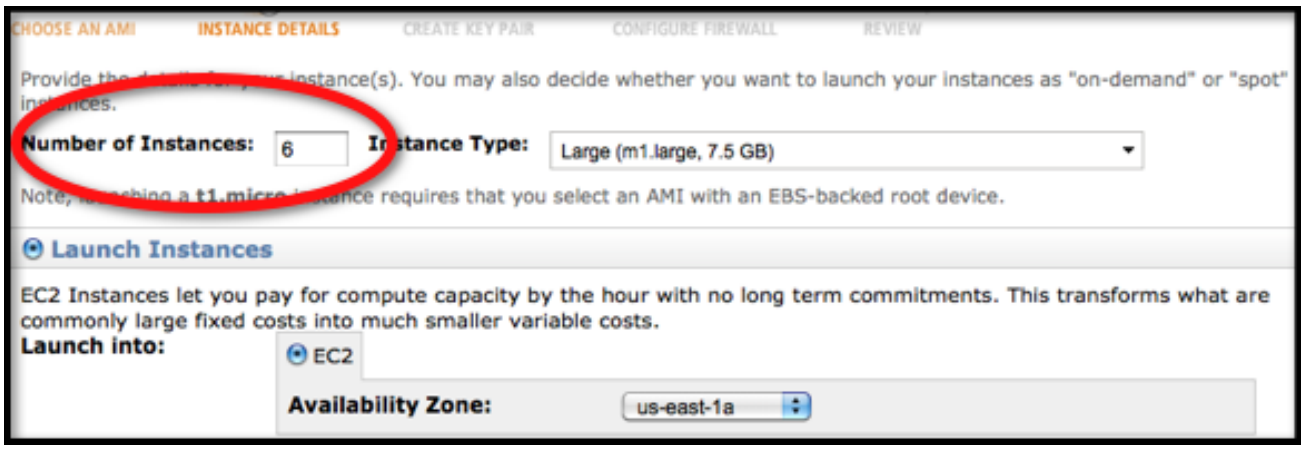
Launching the DataStax AMI

After you have created your security group, you are ready to launch an instance of DataStax Enterprise using the DataStax AMI.

1. From your Amazon EC2 Console Dashboard, click **Launch Instance**. Select **Launch Classic Wizard** and **Continue**.
2. On the **Choose an AMI** page, select the **Community AMIs** tab. Find **DataStax AMI, version 2.1** and click **Select** to launch it.
3. On the **Instance Details** page, enter the total number of nodes you want in your cluster in the **Number of Instances** field and select the **Instance Type**.



4. Click **Continue**.



5. Under **Advanced Instance Options** add the following options to the **User Data** section depending on the type of cluster you want. Option parameters cannot have spaces.

   For new DataStax Enterprise clusters the available options are:

Option	Description
`-c | --clustername <name>`	Required. The name of the cluster.
`-n | --totalnodes <num_nodes>`	Required. The total number of nodes in the cluster.
`-v | --version [enterprise | community]`	Required. The version of the cluster. Use `enterprise` to install the latest version of DataStax Enterprise (DSE).
`-u | --username <username>`	Required for DSE. DataStax registration username. Contact DataStax Customer Support if you do not know your username.
`-p | --password <password>`	Required for DSE. DataStax registration password. Contact DataStax Customer Support if you do not know your password.
`-r | --realtimenodes <num_nodes>`	Optional for DSE. For mixed-workload clusters, the number of Cassandra real-time OLTP nodes. Default is 0.
`-f | --cfsreplicationfactor <rf_num>`	Optional for DSE. Sets the replication factor for the CFS keyspace. Default is 1.
`-o | --opscenter [no]`	Optional. By default, DataStax OpsCenter will be installed on the first instance unless you specify this option with `no`.
`-e | --email <smtp>:<port>:<email>:<password>`	Optional. Sends logs from AMI install to this email address. For example: `smtp.gmail.com:587:ec2@datastax.com:pa$$word`

6. Click **Continue**.
7. On the **Tags** page, give a name to your DSE instance, such as `mixed-workload-dse`, and click **Continue**.
8. On the **Create Key Pair** page create a new key pair or select an existing key pair and click **Continue**. You will need this key (.pem file) to log in to your DataStax Enterprise instance, so save it to a location on your local machine.
9. On the **Configure Firewall** page, select the DSE security group you created earlier and click **Continue**.
10. On the **Review** page, review your cluster configuration and then click **Launch**.

11. Go to the **My Instances** page to see the status of your DSE instance. Once a node has a status of **running**, you are able to connect to it.

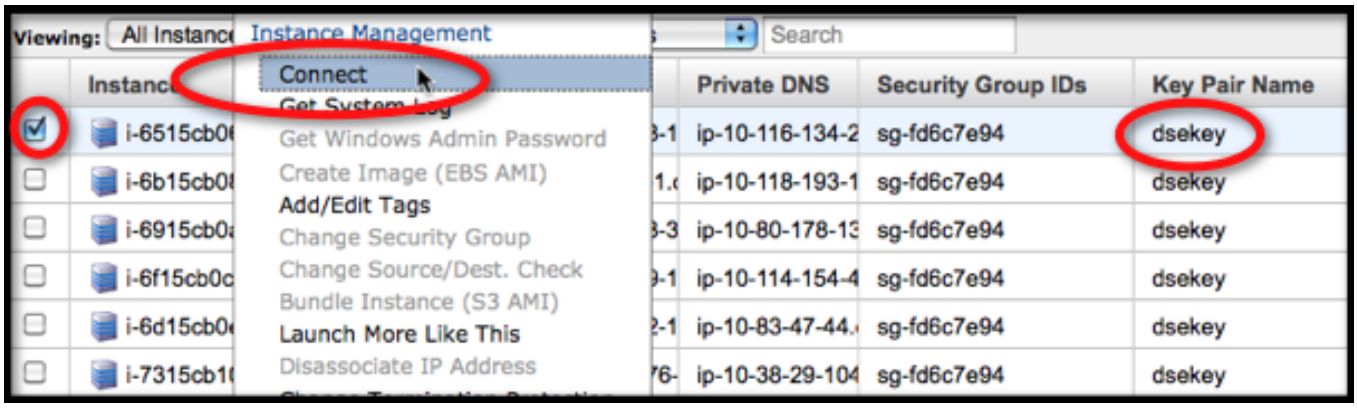
Connecting to Your DataStax Enterprise EC2 Instance

You can connect to your new DSE EC2 instance using any SSH client (PuTTY, Terminal, etc.). To connect, you will need a private key (the .pem file you created earlier) and the public DNS name of a node. Connect as user `ubuntu` rather than as `root`.

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

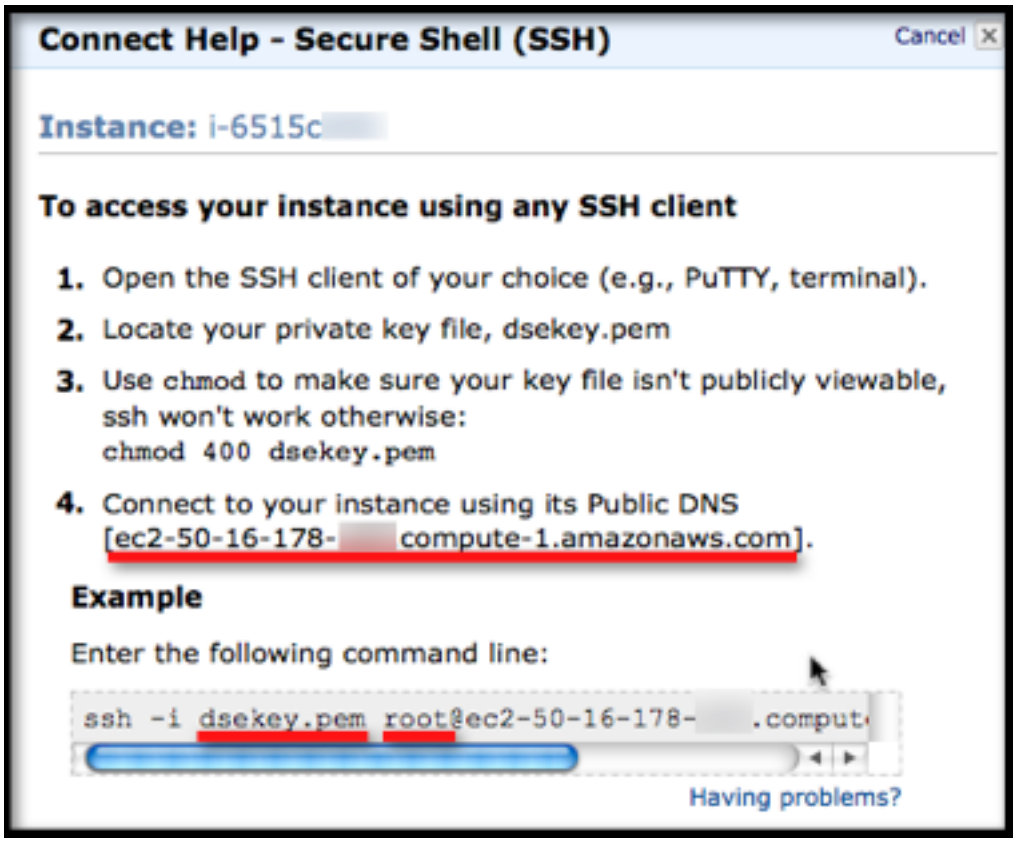
```
chmod 400 dsekey.pem
```

1. From the **My Instances** page in your AWS EC2 Dashboard, select the node you want to connect to. Since all nodes are peers in DSE, you can connect using any node in the cluster. However, the first node is typically the node running your Job Tracker and OpsCenter services (and is also the Cassandra seed node).



2. To get the public DNS name of a node, select **Instance Actions > Connect**

3. This will open a **Connect Help - Secure Shell (SSH)** page for the selected node. This page contains the information you need to connect via SSH. If you copy and paste the command line, change the connection user from `root` to `ubuntu`.



4. The AMI image configures your cluster and starts the Cassandra, Hadoop, and OpsCenter services. After you have logged in to a node, run the *nodetool ring* command to make sure your cluster is up and running. For example:

```
ubuntu@ip-10-202-xx-xxx:~$ nodetool ring -h localhost
Address         DC          Rack        Status State   Load            Owns    Token
                                                                               141784319550391026443072753096570088106
10.202.xx.xxx   Cassandra   rack1       Up     Normal  269.93 KB       16.67%  0
10.85.xxx.xx    Analytics   rack1       Up     Normal  154.14 KB       16.67%  28356863910078205288614550619314017621
10.118.xxx.xxx  Cassandra   rack1       Up     Normal  193.08 KB       16.67%  56713727820156410577229101238628035242
10.80.xxx.xx    Analytics   rack1       Up     Normal  272.32 KB       16.67%  85070591730234615865843651857942052863
10.32.xx.xx     Cassandra   rack1       Up     Normal  233.42 KB       16.67%  113427455640312821154458202477256070485
10.243.xx.xxx   Analytics   rack1       Up     Normal  270.08 KB       16.67%  141784319550391026443072753096570088106
```

5. For next steps, see *Running the Portfolio Manager Demo Application*, *Getting Started with Hive in DataStax Enterprise*, or *Getting Started with Pig in DataStax Enterprise*.

Note

If you are installing OpsCenter with your DSE cluster, allow about 60 to 90 seconds after the cluster has finished loading for OpsCenter to start. You can launch OpsCenter using the URL: `http://<public-dns-of-first-instance>:8888`. After OpsCenter loads, you must install the OpsCenter agents if you want to see cluster performance data (click **Install Agents**). When prompted for credentials for the agent nodes, use the username `ubuntu` and copy/paste the entire contents from your private key file (the `.pem` file you downloaded earlier).

Configuring and Initializing a DataStax Enterprise Cluster

Before you can start DataStax Enterprise (DSE), be it on a single or multi-node cluster, there are a few Cassandra configuration properties you must set on each node in the cluster. These are set in the `cassandra.yaml` file (located in `/etc/dse/cassandra` in packaged installations or `$DSE_HOME/resources/cassandra/conf` in binary distributions).

Initializing a Multi-Node DataStax Enterprise Cluster

Before you start 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. If you are configuring a mixed-workload cluster, you should have at least one seed node for each side (the Cassandra real-time side and the Hadoop Analytics side).
- If you intend to run a mixed-workload cluster determine which nodes will serve which purpose.
- If you have a firewalls 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*.

For example, suppose you are starting a 6 node mixed-workload cluster with 3 Analytics nodes and 3 Cassandra 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

The `cassandra.yaml` file for each node would have the following modified property settings. Note that in a mixed-workload cluster, the token placement alternates between Cassandra and Analytics nodes. This ensures even distribution of replicas on both sides of the cluster. For example:

- node 0: 0
- node 3: 28356863910078205288614550619314017621
- node 1: 56713727820156410577229101238628035242
- node 4: 85070591730234615865843651857942052864
- node 2: 113427455640312821154458202477256070485
- node 5: 141784319550391026443072753096570088106

Also note that in the **seeds** list, the seed node for the Analytics side of the cluster is listed first.

Node0

```
cluster_name: 'DSECluster'
initial_token: 0
seed_provider:
       - 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:
       - 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:
       - 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: 28356863910078205288614550619314017621
seed_provider:
       - 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: 85070591730234615865843651857942052864
seed_provider:
       - 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: 141784319550391026443072753096570088106
seed_provider:
       - seeds: "110.82.155.3,110.82.155.0"
listen_address: 110.82.155.5
rpc_address: 0.0.0.0
```

Generating Tokens

Tokens are used to assign a range of data to a particular node. Assuming you are using the RandomPartitioner, this approach will ensure even data distribution.

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

   ```
   vi tokengentool
   ```

2. Paste the following Python program into this file:

```
#! /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 cluster:

```
How many nodes are in your cluster? 6
node 0: 0
node 1: 28356863910078205288614550619314017621
node 2: 56713727820156410577229101238628035242
node 3: 85070591730234615865843651857942052864
node 4: 113427455640312821154458202477256070485
node 5: 141784319550391026443072753096570088106
```

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

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*

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.

Port	Rule Type	Description
7000	Custom TCP Rule	Cassandra intra-node port (source is the current security group)
9160	Custom TCP Rule	Cassandra client port
8012	Custom TCP Rule	Hadoop Job Tracker client port
9290	Custom TCP Rule	Hadoop Job Tracker Thrift port (source is the current security group)
10000	Custom TCP Rule	Hive Thift Server port (for JDBC Hive access)
50030	Custom TCP Rule	Hadoop Job Tracker website port

50060	Custom TCP Rule	Hadoop Task Tracker website port

Starting DataStax Enterprise as a Stand-Alone Process

If running a mixed-workload cluster, determine which nodes to start as Cassandra nodes and which nodes to start as Analytics 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.

On an Analytics node:

```
dse cassandra -t
```

On a Cassandra node:

```
dse cassandra
```

Starting DataStax Enterprise as a Service

Packaged installations provide startup scripts in `/etc/init.d` for starting DSE as a service. Before starting DSE as a service on an Analytics node, you must first configure the service to start the Hadoop Job Tracker and Task Tracker services as well.

Note

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

1. Create the file `/etc/default/dse`, and add the following line as the contents of this file:

   ```
   HADOOP_ENABLED=1
   ```

2. Start the DSE service:

   ```
   sudo service dse start
   ```

Note

On Enterprise Linux systems, the DSE service runs as a `java` process. On Debian systems, the DSE service runs as a `jsvc` process.

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 DSE, 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.

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 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. 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:

```
$DSE_HOME/bin/dsetool jobtracker
```

Moving the Job Tracker Node for DSE

If your primary Job Tracker node fails, DataStax Enterprise 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.
2. Run the `dsetool movejt` command and specify the IP address of the new job tracker node in your DSE cluster. For example:

```
dsetool movejt 110.82.155.4
```

or in a binary distribution:

```
$DSE_HOME/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 Hive

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

```
dse hive
```

or in a binary distribution:

```
$DSE_HOME/bin/dse hive
```

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 would create your Hive tables using the `CREATE TABLE` command.

For example:

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

You can then load a table using the `LOAD DATA` command. 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:

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

hive>  LOAD DATA LOCAL
       INPATH '$DSE_HOME/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. TODO - need to make sure these are in the DSE distro.

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, you would 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")
      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 pretty 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);
```

Reference: SERDEPROPERTIES and TBLPROPERTIES

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

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
- `cassandra.range.size` - MapReduce range batch size - default 1000
- `cassandra.slice.predicate.size` - MapReduce slice predicate size - default 1000

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 complied 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:

```
dsetool jobtracker
```

or in a binary distribution:

```
$DSE_HOME/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.
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:

   ```
   $DSE_HOME/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:

```
$DSE_HOME/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 {(<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 `$DSE_HOME/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 $DSE_HOME/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:

   ```
   $DSE_HOME/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;
```

Upgrading DataStax Enterprise

This section includes information on upgrading between releases:

Upgrade	Changes
DSE 1.0.1 to 1.0.2	Cassandra updated from 1.0.5 to 1.0.7. Updated Pig driver to 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.

Best Practices for Upgrading

The following best practices are recommended when upgrading:

- Always take a snapshot before any upgrade. 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*
  Snapshotting is fast, especially if you have JNA installed, and takes effectively zero disk space until you start compacting the live data files again.

- Be sure to check https://github.com/apache/cassandra/blob/trunk/NEWS.txt for any new information on upgrading.
- For a list of fixes and new features, see https://github.com/apache/cassandra/blob/trunk/CHANGES.txt

Upgrading DataStax Enterprise Between Minor Releases

Upgrading minor releases can be done with a rolling restart, one node at a time. You do not need to bring down the whole cluster at once.

To upgrade a binary tarball installation:

1. On each node, download and unpack the binary tarball package from the downloads section of the DataStax website.

   For detailed information, see *Installing the DataStax Enterprise Tarball Distribution*.
2. Merge your existing configuration file (`cassandra.yaml`) into the upgraded DSE instance and manually update it with new content.
3. Make sure any client drivers, such as Hector or Pycassa clients, are compatible with the new version.
4. Flush the commit log on the upgraded node by running `nodetool drain`.
5. Stop the old Cassandra process, then start the new binary process.
6. Monitor the log files for any issues.

To upgrade a Debian or RPM package installation:

1. On each node, download and install the package from the downloads section of the DataStax website.

   For detailed information, see *Installing DataStax Enterprise Packaged Releases or Tarball Distribution*.
2. Merge your existing configuration file (`cassandra.yaml`) into the upgraded DSE instance and manually update it with new content.
3. Make sure any client drivers, such as Hector or Pycassa clients, are compatible with the new version.
4. Flush the commit log on the upgraded node by running `nodetool drain`.
5. Restart the Cassandra process.
6. Monitor the log files for any issues.

DataStax Enterprise Release Notes

DataStax Enterprise 1.0.x

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

Issue	Description
363	Create the CassandraFS (cfs) keyspace with replication strategy options that respect the currently configured snitch.
368	Fix Debian packages so they do not start the DSE service by default.