



DataStax Enterprise 4.8

Documentation

August 16, 2016

Contents

About DataStax Enterprise.....	7
Upgrading.....	7
Installing.....	7
Installing DataStax Enterprise using GUI or Text mode.....	7
Installing on Linux without root permissions or on Mac OS X.....	12
Installer - unattended.....	17
Other install methods.....	21
Using the Yum repository.....	21
Installing DataStax Enterprise using APT repositories.....	23
Using the binary tarball.....	24
Installing on cloud providers.....	27
Installing a DataStax Enterprise cluster on Amazon EC2.....	27
Installing and deploying a DataStax Enterprise cluster in CenturyLink Cloud.....	40
Installing and deploying a DataStax Enterprise cluster using Google Compute Engine.....	41
Installing and deploying a DataStax Enterprise cluster using Microsoft Azure.....	41
Installing EPEL on RHEL OS 5.x.....	41
Installing earlier versions.....	41
Installing glibc.....	43
Uninstalling DataStax Enterprise.....	44
Starting and stopping DataStax Enterprise.....	45
Starting as a service.....	45
Starting as a stand-alone process.....	47
Stopping a node.....	48
Configuration.....	48
Configuration file (dse.yaml).....	48
Configuring and using virtual nodes (vnodes).....	64
Default file locations for Installer-Services and package installations.....	65
Default file locations for Installer-No Services and tarball installations.....	68
Configuring the Tomcat log location.....	71
Collecting node health and indexing status scores.....	71
DSE Analytics.....	73
About DSE Analytics.....	73
DSE Analytics and Search integration (experimental).....	74
About the Cassandra File System (CFS).....	75
Configuring DSE Analytics.....	76
Setting the replication factor.....	76
Job Trackers for DSE Hadoop and external Hadoop.....	77
Analyzing data using Spark.....	80
About Spark.....	80
Configuring Spark.....	81

Using Spark with DataStax Enterprise.....	88
Spark examples.....	116
Analyzing data using DSE Hadoop.....	129
About DSE Hadoop.....	129
Using common Hadoop commands.....	137
Using the cfs-archive to store huge files.....	138
Using Hive with DSE Hadoop.....	139
ODBC driver for Hive.....	164
Using Mahout.....	168
Using Pig.....	169
Analyzing data using external Hadoop systems (BYOH).....	180
About BYOH.....	180
BYOH Prerequisites and installation.....	183
Configuring an external Hadoop system.....	185
Starting up the BYOH datacenter.....	187
Using BYOH.....	187
DSE Search.....	192
About DSE Search.....	192
Starting and stopping DSE Search.....	193
DSE Search architecture.....	193
Queries.....	195
Using CQL Solr queries in DSE Search.....	195
Search queries with JSON.....	198
Using the Solr HTTP API.....	203
Using Solr pagination (cursors).....	204
Inserting/updating data using the Solr HTTP API.....	204
Querying a CQL collection set.....	205
Spatial queries.....	206
Using dynamic fields.....	207
Deleting by id.....	208
Deleting by query.....	209
Joining cores.....	209
Querying multiple tables.....	213
Using HTTP API SolrJ and other Solr clients.....	213
Working with advanced data types: tuples and UDTs.....	213
Configuring tuples and UDTs in the Solr schema.....	213
UDT query examples.....	216
Schema and data modeling.....	218
Creating a schema and data modeling.....	218
Mapping of Solr types.....	220
Legacy mapping of Solr types.....	222
Changing Solr Types.....	223
Using copy fields.....	224
Configuring DSE Search.....	227
DSE Search configuration file (solrconfig.xml).....	227
Indexing resources.....	230
Increasing indexing throughput.....	236
Configuring search components.....	237
Segregating workloads in a cluster.....	238
Configuring the Solr type mapping version.....	238
Securing a DSE Search cluster.....	238
Configuring multi-threaded queries.....	239
Shard transport options for DSE Search communications.....	240
Changing Tomcat web server settings.....	241

Contents

Configuring global filter cache for searching.....	241
Configuring the Solr library path.....	242
Limiting columns indexed and returned by a query.....	243
Configuring autocomplete/spellcheck.....	243
Changing maxBooleanClauses.....	244
Configuring the Data Import Handler (deprecated).....	244
Operations.....	244
Adding, decommissioning, repairing a DSE search node.....	244
Enabling the disk failure policy.....	244
Restricted query routing.....	245
Shuffling shards to balance the load.....	246
Shard routing for distributed queries.....	247
Managing the location of Solr data.....	247
Changing the Solr connector port.....	248
Deleting Solr data.....	248
Viewing the Solr core status.....	249
Solr log messages.....	250
Adding and viewing index resources.....	251
Checking indexing status.....	251
Fast repair.....	252
Excluding hosts from Solr-distributed queries.....	253
Expiring a DSE Search column.....	253
Changing the HTTP interface to Apache JServe Protocol.....	255
Backing up Solr indexes.....	255
Performance tuning.....	256
Using metrics MBeans.....	256
Using table compression.....	265
Configuring the update handler and autoSoftCommit.....	265
Parallelizing large Cassandra row reads.....	266
Changing the stack size and memtable space.....	266
Managing the consistency level of write in Cassandra on the client side.....	266
Configuring multi-threaded indexing threads.....	267
Configuring re-indexing.....	267
Tuning index size and range query speed.....	268
Increasing read performance by adding replicas.....	269
Changing the replication factor for a Solr keyspace.....	269
Managing caching.....	270
Capacity planning.....	270
Tuning near-real-time (NRT) indexing.....	272
Update request processor and field transformer.....	273
Custom URP example.....	273
Field input/output transformer example.....	275
FIT reference implementation.....	276
Interface for custom field types.....	278
Unsupported features.....	279
DSE Search versus Open Source Solr.....	280
DSE Search tutorials and demos.....	281
Tutorial: Basics.....	281
Tutorial: Advanced.....	288
Running Wikipedia demo using DSE Search.....	295
Troubleshooting.....	298
Handling inconsistencies in query results.....	298
Tracing Solr HTTP requests.....	299
Using Solr MBeans.....	300
Using the ShardRouter Mbean.....	301

DSE Advanced Security.....	301
About security management.....	301
Authenticating with Kerberos.....	303
Kerberos guidelines.....	303
AES-256 support.....	304
Setting up the environment for Kerberos.....	305
Adding Kerberos service principals for each node in a cluster.....	306
Configuring DataStax Enterprise for Kerberos authentication.....	309
Creating Kerberos users.....	311
Enabling and disabling Kerberos security.....	311
Enabling cqlsh to use Kerberos.....	311
Enabling dsetool to use Kerberos.....	312
Using Kerberos authentication with Sqoop.....	313
Authenticating with LDAP.....	314
Enabling LDAP authentication.....	314
Creating LDAP users.....	316
Encryption.....	317
Encrypting sensitive property values.....	317
Client-to-node encryption.....	318
Node-to-node encryption.....	319
Spark SSL encryption.....	320
Preparing server certificates.....	321
Spark security.....	322
Transparent data encryption.....	325
Configuring encryption using local encryption keys.....	326
Configuring encryption using off-server encryption keys.....	327
Configuring encryption per table (TDE).....	329
Migrating encrypted tables.....	332
Using cqlsh with Kerberos/SSL.....	332
Configuring and using data auditing.....	334
Configuring audit logging to a logback log file.....	336
Configuring audit logging to a Cassandra table.....	339
Configuring auditing for a DSE Search cluster.....	341
Configuring and using internal authentication.....	342
Configuring internal authentication and authorization.....	343
Providing credentials for authentication.....	344
Changing the default superuser.....	345
Enable internal security without downtime.....	345
Logging in with cqlsh.....	346
Managing object permissions using internal authorization.....	346
Configuring system_auth and dse_security keyspace replication.....	347
Configuring firewall port access.....	348
Making /tmp non-executable.....	350
DSE Management Services.....	351
Performance Service.....	351
About the Performance Service.....	352
Configuring Performance Service replication strategy.....	353
Collecting Cassandra data.....	353
Collecting Solr data.....	359
Monitoring Spark with Spark Performance Objects.....	364
Cassandra diagnostic table reference.....	367
Solr diagnostic table reference.....	384

Contents

Capacity Service.....	402
Repair Service.....	402
DSE In-Memory.....	402
Creating or altering tables to use DSE In-Memory.....	403
Verifying table properties.....	404
Managing memory.....	404
Backing up and restoring data.....	405
Deploying.....	405
Production deployment planning.....	405
Configuring replication.....	406
Mixing workloads in a cluster.....	409
Single datacenter deployment per workload type.....	412
Multiple datacenter deployment per workload type.....	415
Single-token architecture deployment.....	418
Calculating tokens.....	421
Expanding a DataStax AMI cluster.....	424
Migrating data.....	424
Migrating data using Sqoop.....	424
About Sqoop.....	424
Running the Sqoop demo.....	425
Importing SQL to a CQL table or CFS.....	428
Importing data into a CQL list or set.....	429
Importing data into a CQL map.....	431
Importing joined tables.....	431
Exporting CQL data to SQL.....	434
Exporting selected CQL data to SQL.....	435
Exporting data from CQL collections.....	437
Automating a Sqoop operation.....	438
Sqoop command.....	441
Migrating data using other methods.....	445
Bulk saving data from Spark RDD to Cassandra.....	446
Tools.....	447
dse commands.....	447
dsetool utility.....	450
The cfs-stress tool.....	458
Pre-flight check and yaml_diff tools.....	459
Using the Cassandra bulk loader in a secure environment.....	459
Troubleshooting.....	460
Release Notes.....	462
Cassandra changes.....	473

About DataStax Enterprise

DataStax Enterprise delivers Apache Cassandra™ in a database platform that meets the performance and availability demands of Internet-of-Things (IoT), Web, and Mobile applications. It provides enterprises a secure, fast, always-on database that remains operationally simple when scaled in a single datacenter or across multiple datacenters and clouds.

DataStax Enterprise 4.8 new features

DataStax Enterprise 4.8 introduces the following new features and enhancements:

DSE Analytics	Production certification for Spark 1.4. Built-in Spark Jobserver for easily submitting and managing Spark analytics jobs.
DSE Search	Live indexing enhancements. Support for indexing and querying of Cassandra tuples and user defined types (UDTs) .
Docker support	DataStax Enterprise is supported on Docker . See the best practices for running DataStax Enterprise with Docker . For more information, contact customer support.

For more information, see the DataStax Enterprise 4.8 [release notes](#).

Upgrading DataStax Enterprise

See [Upgrading Datastax Enterprise](#) in the *DataStax Upgrade Guide*.

Installing DataStax Enterprise

DataStax Enterprise installation methods include GUI or text mode, unattended command line or properties file, YUM and APT repository, and binary tarball.

Installing DataStax Enterprise using GUI or Text mode

For other product installations, see [Installing OpsCenter](#) and [Installing DevCenter](#).

For installing DataStax Enterprise without root permissions or on Mac OS X, click [here](#).

Important: DataStax Enterprise 4.8 uses Cassandra 2.1 and CQL3.1.

Prerequisites

- Be sure your [platform is supported](#).
- Root or sudo access when installing as a system service, or if installing missing system dependencies.

- Latest version of Oracle Java SE Runtime Environment 7 or 8 or OpenJDK 7 is recommended.
Note: If using Oracle Java 7, you must use at least 1.7.0_25. If using Oracle Java 8, you must use at least 1.8.0_40. In some cases, using JDK 1.8 causes minor performance degradation compared to JDK 1.7.
- RedHat-compatible distributions require EPEL (Extra Packages for Enterprise Linux). For RHEL 5.x, see [Installing EPEL on RHEL OS 5.x](#) on page 41.
- If installing on a 64-bit Oracle Linux distribution, first install the 32-bit versions of [glibc libraries](#).
- Python 2.6 (minimum); 2.7 (recommended).

Also see [Recommended production settings](#) and the [DataStax Enterprise Reference Architecture](#) white paper.

Table: Hardware requirements

Requirement	Minimum	Production
CPUs	2	16
Memory	8 GB	24 GB
Data directory	20 GB	200 GB
Commit log directory	20 GB	200 GB
Saved caches directory	20 GB	200 GB
Logs directory	20 GB	200 GB
Production requirements depend on the volume of data and workload.		

About the installer

The installer installs DataStax Enterprise and the DataStax Agent. It does not install OpsCenter or DevCenter. The installer sets some but not all `cassandra.yaml` parameters described in the [table](#) below. It does not set `dse.yaml` properties. You can set the remaining parameters in the following ways:

- Manually after installation.
- Use the [unattended install](#) with either [command line](#) or the [property file](#) options. These options allow you to specify pre-configured `cassandra.yaml` and `dse.yaml` files using `--cassandra_yaml_template` filename and `--dse_yaml_template` filename.

Procedure

In a terminal window:

1. Download the installer for your computer from the [DataStax download page](#).
2. From the directory where you downloaded the install file, change the permission to executable:

```
$ chmod +x DataStaxEnterprise-4.8.x-linux-x64-installer.run
```

3. To view the installer help:

```
$ ./DataStaxEnterprise-4.8.x-linux-x64-installer.run --help
```

4. Start the installation:

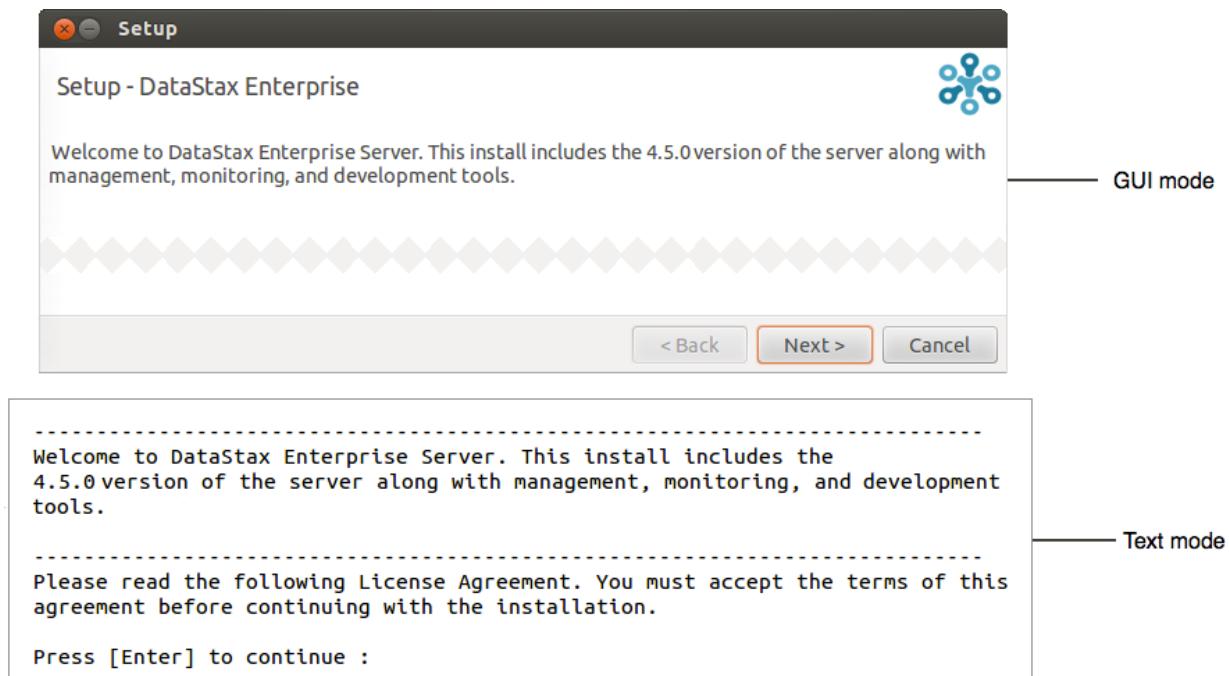
```
$ sudo ./DataStaxEnterprise-4.8.x-linux-x64-installer.run ##  
Install in GUI mode.  
$ sudo ./DataStaxEnterprise-4.8.x-linux-x64-installer.run --mode text ##  
Install in Text mode.
```

Using the `install` command to set configuration parameters:

To add configuration parameters to the installation, use the installer options described in [Installer - unattended](#). For example:

```
$ sudo ./DataStaxEnterprise-4.8.x-linux-x64-installer.run --prefix /usr/
local/dse --enable_vnodes 0 ## Command line option.
$ sudo ./DataStaxEnterprise-4.8.x-linux-x64-installer.run --optionfile ../
datastax/DC4-analytics.prop ## Property file option.
```

The installer launches.



- Follow the instructions in the setup wizard using the following table for guidance:

Screen - Panel	Recommendations and additional information
Setup	Welcome page.
License Agreement	DataStax Enterprise End User License Agreement
Install Options	
Server Installation Directory	If you use the No Services option, you can change the location of the dse directory. If you install as a service, DataStax Enterprise can only be installed in the /usr/share/dse directory.
Install Type	<p>Use Simple Install for default path names and options.</p> <p>Advanced Install allows you to configure additional parameters, including:</p> <ul style="list-style-type: none"> Enable/disable virtual nodes (vnodes). Service users and group name for non-root users. Listen and RPC addresses. Directory locations. Storage, SSL Storage, and RPC ports.

Screen - Panel	Recommendations and additional information
Update System	Updates some system packages and dependencies. Does not upgrade or install major components such as Java. Set to Yes when run as root user, otherwise set to No .
Default Interface	Network interface for the DataStax Enterprise server. Use 127.0.0.0 for single node clusters.
Service Setup	<p>No Services - installs the DataStax Enterprise server as a stand-alone process. Use these instructions for this type of installation.</p> <p>Note: For installing DataStax Enterprise without root permissions or on Mac OS X, click here.</p> <p>Services Only - installs the DataStax Enterprise server as a service running in the background.</p> <p>Services and Utilities (Linux only) - installs the DataStax Enterprise server as a service running in the background and Cassandra utilities, such as <code>cqlsh</code>, <code>sstableloader</code>, <code>sstablescrub</code>, and <code>sstableupgrade</code> to the system path.</p>
Start Services After Install	Select Yes to start all services when the installation is complete, or select No when additional configuration is needed after installation.
Node Setup	
Node Type	<p>Attention: All nodes are DataStax Enterprise nodes and run the Cassandra database.</p> <p>The following types of nodes are available:</p> <ul style="list-style-type: none"> Cassandra node - Transactional node, also used for Bring your own Hadoop (BYOH) nodes. Analytics node- Spark only and Spark + Integrated Hadoop (DSE Hadoop) nodes. DSE Search (Solr) nodes - DSE Search (DataStax Enterprise Search) simplifies using search applications for data that is stored in a Cassandra database.
Cluster Name	Name of the cluster. You must use the same cluster name for each node in the cluster.
Seeds	Cassandra nodes use the seed node list for finding each other and learning the topology of the ring. Do not make all nodes seed nodes. See the following: <ul style="list-style-type: none"> Internode communications (gossip) Initializing a multiple node cluster (single data centers) Initializing a multiple node cluster (multiple data centers)
User Setup (Advanced installation only)	
OS User ID for Service	When starting DataStax Enterprise as a service, the Cassandra and Hadoop tracker services run as this user and group. The service initialization script is located in <code>/etc/init.d/dse</code> . Run levels are not set by the package.
OS User Group for Service	
Ring Options (Advanced installation only)	
Enable Vnodes	Enable or disable Virtual nodes .

Screen - Panel	Recommendations and additional information
Listen Address	cassandra.yaml parameter: listen_address
RPC Address	cassandra.yaml parameter: rpc_address
Directory Locations (Advanced installation only)	
Data Directory	cassandra.yaml parameter: data_file_directories
Commitlog Directory	cassandra.yaml parameter: commitlog_directory
Saved Caches Directory	cassandra.yaml parameter: saved_caches_directory
Logs Directory	Log data.
Ports (Advanced installation only)	
Storage Port	cassandra.yaml parameter: storage_port
SSL Storage Port	cassandra.yaml parameter: ssl_storage_port
RPC Port	cassandra.yaml parameter: rpc_port
DataStax Agent	
OpsCenter Address	The network address of the OpsCenter. The agent provides an interface between DataStax OpsCenter and DataStax Enterprise.
Review and install	
System Configuration	Configuration overview and warnings about potential issues.
Ready to Install	The install wizard installs the software.
Setup finish	Post-installation tasks. Note: View Configuration Recommendations And Warnings opens the Pre-flight check results.

6. Review the installation logs to verify the installation.

Note: If you have closed the logs, see [Services](#) or [No-Services](#) installer locations.

Results

DataStax Enterprise is ready for [additional configuration](#).

7. Optional: Single-node cluster installations only:

- a) If you haven't started DataStax Enterprise:

- No-Services:

```
$ bin/dse cassandra #optional start options
```

- Services:

```
$ sudo service dse start ## Starts the DataStax Enterprise server
$ sudo service datastax-agent start ## Starts the DataStax Agent
```

- b) Verify that DataStax Enterprise is running:

```
$ nodetool status
```

```
Datacenter: Cassandra
=====

```

```
Status=Up/Down
| / State=Normal/Leaving/Joining/Moving
-- Address      Load      Tokens  Owns      Host ID
   Rack
UN 127.0.0.1  82.43 KB    256      ? 
40725dc8-7843-43ae-9c98-7c532b1f517e  rack1
```

What to do next

- [Configuring DataStax Enterprise.](#)
- [Configuration and log file locations.](#)
- [Starting and stopping DataStax Enterprise.](#)
- [Deploying DataStax Enterprise for production.](#)
- [Selecting hardware for enterprise implementations.](#)
- [Configuring the heap dump directory to avoid server crashes.](#)

Installing on Linux without root permissions or on Mac OS X

For other product installations, see [Installing OpsCenter](#) and [Installing DevCenter](#).

Important: DataStax Enterprise 4.8 uses Cassandra 2.1 and CQL3.1.

Prerequisites

Linux installations

- Be sure your [platform is supported](#).
- Latest version of [Oracle Java SE Runtime Environment 7 or 8](#) or [OpenJDK 7](#) is recommended.

Note: If using Oracle Java 7, you must use at least 1.7.0_25. If using Oracle Java 8, you must use at least 1.8.0_40. In some cases, using JDK 1.8 causes minor performance degradation compared to JDK 1.7.

- RedHat-compatible distributions require EPEL (Extra Packages for Enterprise Linux). For RHEL 5.x, see [Installing EPEL on RHEL OS 5.x](#) on page 41.
- If installing on a 64-bit Oracle Linux distribution, first install the 32-bit versions of [glibc libraries](#).
- Python 2.6 (minimum); 2.7 (recommended).

Also see [Recommended production settings](#) and the [DataStax Enterprise Reference Architecture](#) white paper.

Mac OS X installations

- Mac OS X is supported for development only.
- Latest version of Oracle Java 8 is recommended.
- On some versions of Mac OS X, you may need to install readline: `easy_install readline`.

Table: Hardware requirements

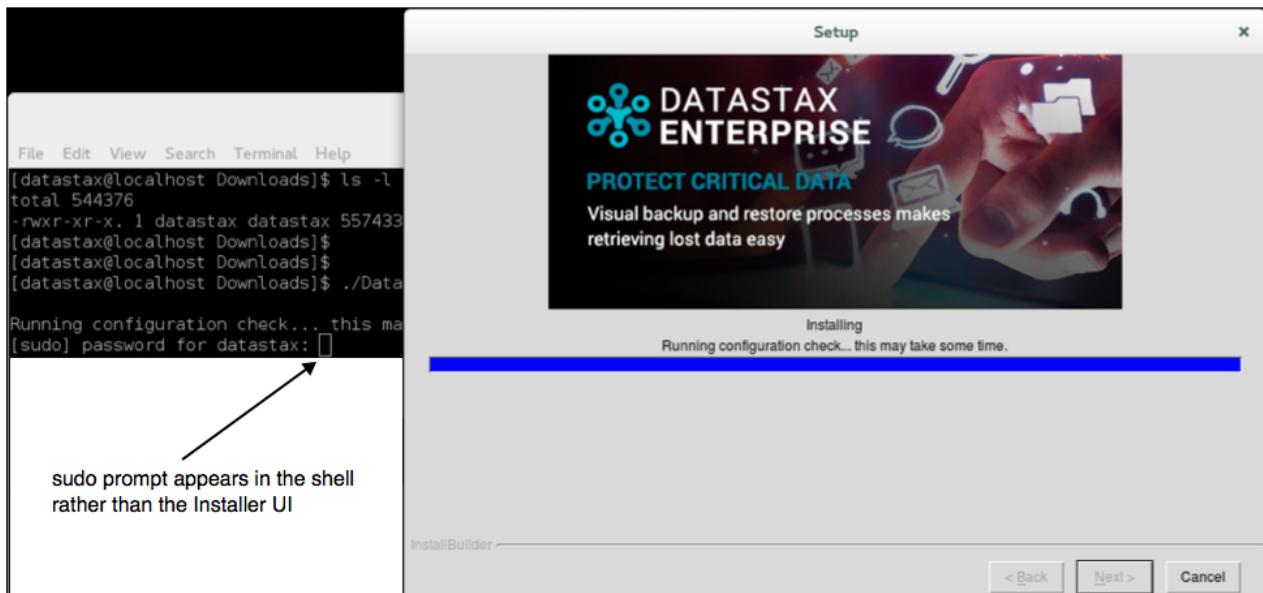
Requirement	Minimum	Production
CPUs	2	16
Memory	8 GB	24 GB
Data directory	20 GB	200 GB
Commit log directory	20 GB	200 GB

Requirement	Minimum	Production
Saved caches directory	20 GB	200 GB
Logs directory	20 GB	200 GB
Production requirements depend on the volume of data and workload.		

Installing under a user account (Linux only)

Without root or sudo access, the installer cannot set up support services because it does not have permission to create the services files. Root or sudo access allows the installer to set up support services on operating systems that support services, such as Debian-based or RHEL-based systems.

In GUI mode, if gksudo or pkexec, are not installed, the installer might not present a GUI sudo prompt. Subsequently the sudo prompt appears in the shell:



Procedure

In a terminal window:

1. Download the installer for your computer from the [DataStax download page](#).
2. From the directory where you downloaded the install file:

- **Linux:**

1. From the directory where you downloaded the install file, change the permission to executable:

```
$ chmod +x DataStaxEnterprise-4.8.x-linux-x64-installer.run ## Changes
      permission to executable
```

2. To view the installer help:

```
$ ./DataStaxEnterprise-4.8.x-linux-x64-installer.run --help
```

3. Start the installation:

```
$ ./DataStaxEnterprise-4.8.x-linux-x64-installer.run ## 
      Install in GUI mode.
$ ./DataStaxEnterprise-4.8.x-linux-x64-installer.run --mode text ## 
      Install in Text mode.
```

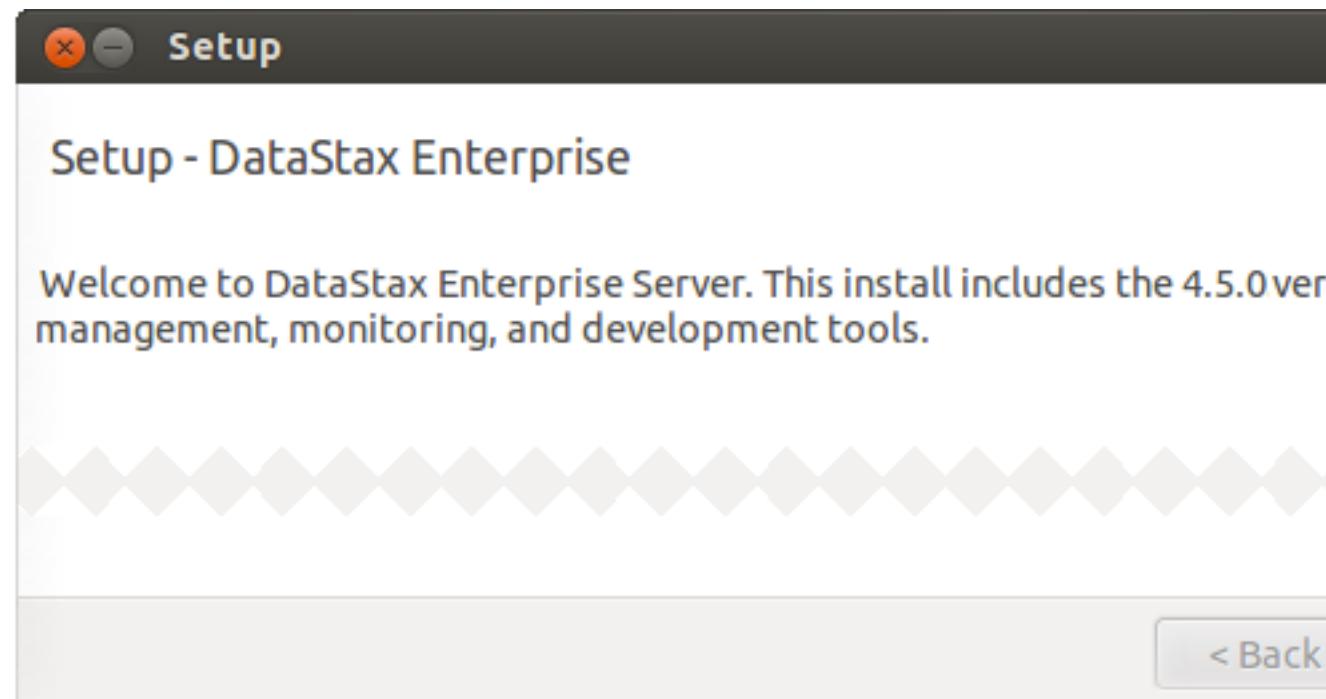
- **Mac OS X:**

Installing DataStax Enterprise

1. Double-click the DataStaxEnterprise-4.8.x.dmg file.
2. In the installer window, double-click the installer icon.

Depending on your permissions, you may need to use **Control-click**.

The installer launches.



Welcome to DataStax Enterprise Server. This install includes the 4.5.0 version of the server along with management, monitoring, and development tools.

Please read the following License Agreement. You must accept the agreement before continuing with the installation.

Press [Enter] to continue :

3. Follow the instructions in the setup wizard using the following table for guidance:

Screen - Panel	Recommendations and additional information
Setup	Welcome page.
License Agreement	DataStax Enterprise End User License Agreement
Install Options	

Screen - Panel	Recommendations and additional information
Server Installation Directory	Sets the location of the dse directory.
Install Type	<p>Use Simple Install for default path names and options.</p> <p>Advanced Install allows you to configure additional parameters, including:</p> <ul style="list-style-type: none"> Enable/disable virtual nodes (vnodes). Service users and group name for non-root users. Listen and RPC addresses. Directory locations. Storage, SSL Storage, and RPC ports.
Update System	Updates some system packages and dependencies. Does not upgrade or install major components such as Java. Set to Yes when run as root user, otherwise set to No .
Default Interface	<p>Network interface for the DataStax Enterprise server.</p> <p>Use 127.0.0.0 for single node clusters.</p>
Service Setup	<p>Linux: No ServicesNo Services - installs the DataStax Enterprise server as a stand-alone process.</p> <p>Mac:</p> <ul style="list-style-type: none"> Services onlyServices Only - installs the DataStax Enterprise server as a service running in the background. No ServicesNo Services - installs the DataStax Enterprise server as a stand-alone process.
Start Services After Install	<p>Linux: No - A non-root install doesn't use services.</p> <p>Mac: Select Yes to start all services when the installation is complete, or select No when additional configuration is needed after installation.</p>
Node Setup	
Node Type	<p>Attention: All nodes are DataStax Enterprise nodes and run the Cassandra database.</p> <p>The following types of nodes are available:</p> <ul style="list-style-type: none"> Cassandra node - Transactional node, also used for Bring your own Hadoop (BYOH) nodes. Analytics node- Spark only and Spark + Integrated Hadoop (DSE Hadoop) nodes. DSE Search (Solr) nodes - DSE Search (DataStax Enterprise Search) simplifies using search applications for data that is stored in a Cassandra database.
Cluster Name	Name of the cluster. You must use the same cluster name for each node in the cluster.
Seeds	Cassandra nodes use the seed node list for finding each other and learning the topology of the ring. Do not make all nodes seed nodes. See the following: <ul style="list-style-type: none"> Internode communications (gossip)

Screen - Panel	Recommendations and additional information
	<ul style="list-style-type: none"> • Initializing a multiple node cluster (single data centers) • Initializing a multiple node cluster (multiple data centers)
Ring Options (Advanced installation only)	
Enable Vnodes	Enable or disable Virtual nodes .
Listen Address	cassandra.yaml parameter: listen_address
RPC Address	cassandra.yaml parameter: rpc_address
Directory Locations (Advanced installation only)	
Data Directory	cassandra.yaml parameter: data_file_directories
Commitlog Directory	cassandra.yaml parameter: commitlog_directory
Saved Caches Directory	cassandra.yaml parameter: saved_caches_directory
Logs Directory	Log data.
Ports (Advanced installation only)	
SSL Storage Port	cassandra.yaml parameter: ssl_storage_port
RPC Port	cassandra.yaml parameter: rpc_port
DataStax Agent	
OpsCenter Address	The network address of the OpsCenter. The agent provides an interface between DataStax OpsCenter and DataStax Enterprise.
Review and install	
System Configuration	Configuration overview and warnings about potential issues.
Ready to Install	The install wizard installs the software.
Setup finish	Post-installation tasks. Note: View Configuration Recommendations And Warnings opens the Pre-flight check results.

4. Review the installation logs to verify the installation.

Note: If you have closed the logs, see [Services](#) or [No-Services](#) installer locations.

Results

DataStax Enterprise is ready for [additional configuration](#).

5. Optional: Single-node cluster installations only:

- a) If you haven't started DataStax Enterprise:

- No-Services:

```
$ bin/dse cassandra #optional start options
```

- Services:

```
$ sudo service dse start ## Starts the DataStax Enterprise server
$ sudo service datastax-agent start ## Starts the DataStax Agent
```

- b) Verify that DataStax Enterprise is running:

```
$ nodetool status

Datacenter: Cassandra
=====
Status=Up/Down
| / State=Normal/Leaving/Joining/Moving
-- Address      Load      Tokens     Owns      Host ID
   Rack
UN 127.0.0.1  82.43 KB  256      ?          40725dc8-7843-43ae-9c98-7c532b1f517e  rack1
```

What to do next

- [Configuring DataStax Enterprise.](#)
- [Configuration and log file locations.](#)
- [Starting and stopping DataStax Enterprise.](#)
- [Deploying DataStax Enterprise for production.](#)
- [Selecting hardware for enterprise implementations.](#)
- [Configuring the heap dump directory to avoid server crashes.](#)

Unattended DataStax Enterprise installer

For other product installations, see [Installing OpsCenter](#) and [Installing DevCenter](#).

Important: DataStax Enterprise 4.8 uses Cassandra 2.1 and CQL3.1.

Prerequisites

- Be sure your [platform is supported](#).
- Root or sudo access when installing as a system service, or if installing missing system dependencies.
- Latest version of [Oracle Java SE Runtime Environment 7 or 8 or OpenJDK 7](#) is recommended.

Note: If using Oracle Java 7, you must use at least 1.7.0_25. If using Oracle Java 8, you must use at least 1.8.0_40. In some cases, using JDK 1.8 causes minor performance degradation compared to JDK 1.7.

- RedHat-compatible distributions require EPEL (Extra Packages for Enterprise Linux). For RHEL 5.x, see [Installing EPEL on RHEL OS 5.x](#) on page 41.
- If installing on a 64-bit Oracle Linux distribution, first install the 32-bit versions of [glibc libraries](#).
- Python 2.6 (minimum); 2.7 (recommended).

Also see [Recommended production settings](#) and the [DataStax Enterprise Reference Architecture](#) white paper.

Table: Hardware requirements

Requirement	Minimum	Production
CPUs	2	16
Memory	8 GB	24 GB
Data directory	20 GB	200 GB
Commit log directory	20 GB	200 GB
Saved caches directory	20 GB	200 GB

Requirement	Minimum	Production
Logs directory	20 GB	200 GB
Production requirements depend on the volume of data and workload.		

Procedure

From a terminal window:

1. Download the installer for your computer from the [DataStax download page](#).
2. Change the permission on the file to executable:

```
$ chmod +x DataStaxEnterprise-4.8.x-linux-x64-installer.run
```

3. You can either use the command line or a properties file.

Command line installation:

```
$ sudo ./DataStaxEnterprise-4.8.x-linux-x64-installer.run --option argument  
--option argument ... --option argument --mode unattended
```

For available options, see the [table below](#). Be sure to add "--" to the option. For example:

```
$ sudo ./DataStaxEnterprise-4.8.x-linux-x64-installer.run --prefix /usr/  
share/dse --enable-components dse,datastax_agent --enable_vnodes 0 --mode  
unattended --prefix /usr/share/dse
```

The installer uses the default value for any --option that is not specified.

Properties file installation:

```
$ sudo ./DataStaxEnterprise-4.8.x-linux-x64-installer.run --  
optionfile option_file_name --mode unattended
```

where *option_file_name* is the name of the file containing the installation options. For example:

```
$ sudo ./DataStaxEnterprise-4.8.x-linux-x64-installer.run --optionfile ../  
datastax/DC4-analytics.prop --mode unattended
```

Property file format: option=argument. For example:

```
enable-components=dse,datastax_agent  
install_type=simple  
update_system=1
```

The property file options are the same as the command line options (without the --).

Note: You can download a [sample_install_4.8.prop](#) file from the DataStax Enterprise download page.

Table: Unattended install options

Option	Argument	Description
Install options		
prefix	<i>install_directory</i>	Install location. Default: /usr/share/dse
enable-components	Comma separated list of components: • dse	Components to install. Default: dse,datastax_agent

Option	Argument	Description
	<ul style="list-style-type: none"> <i>datastax_agent</i> 	
cassandra_yaml_template	<i>file_name</i>	Use this <code>cassandra.yaml</code> file as the template for the node's <code>cassandra.yaml</code> file.
dse_yaml_template	<i>file_name</i>	Use this <code>dse.yaml</code> file as the template for the node's <code>dse.yaml</code> file.
cassandra_logs_dir	<i>directory</i>	Directory for log files.
do_drain	<i>0</i> (no) or <i>1</i> (yes)	Drain the node before installing. Default: <i>1</i>
install_type	<i>simple</i> or <i>advanced</i>	Default: <i>simple</i>
system_install	Use one of the following: <ul style="list-style-type: none"> <i>no_services</i> <i>services_only</i> <i>services_and_utilities</i> 	Install system services. Default: <code>services_and_utilities</code> for root user, <code>no_services</code> for others.
update_system	<i>0</i> (no) or <i>1</i> (yes)	Upgrade any missing system files. Does not upgrade or install major components such as Oracle Java. Default: <i>1</i> for root user, <i>0</i> for others.
installdir_agent	<i>directory</i>	Directory where agent is installed.
node_type	Use one of the following: <ul style="list-style-type: none"> <i>cassandra</i> <i>analytics</i> <i>search</i> 	Type of node. Default: <i>cassandra</i> . Attention: All nodes are DataStax Enterprise nodes and run the Cassandra database. <ul style="list-style-type: none"> Cassandra node - Transactional node, also used for Bring your own Hadoop (BYOH) nodes. Analytics node- Spark only and Spark + Integrated Hadoop (DSE Hadoop) nodes. DSE Search (Solr) nodes - DSE Search (DataStax Enterprise Search) simplifies using search applications for data that is stored in a Cassandra database.
analytics_type	Use one of the following: <ul style="list-style-type: none"> <i>spark_only</i> <i>spark_integrated</i> 	Type of analytics node. <ul style="list-style-type: none"> <i>spark_only</i> - only enable Spark. <i>spark_integrated</i>- enable Spark + Integrated Hadoop (DSE Hadoop).
cassandra_user	<i>user_name</i>	User name for running service. Because the DataStax Agent relies on <code>usercassandra</code> , DataStax does not recommend changing the default. If changed, you must manually install, update, and configure the DataStax Agent .

Option	Argument	Description
cassandra_group	<i>group_name</i>	Group name for running service. Start-up scripts are provided in /etc/init.d.
start_services	0 (no) or 1 (yes)	Start services. Default: 1.
OpsCenter options		
opscenter_address	<i>IP_address</i>	Address for the OpsCenter server.
cassandra.yaml options (These values override options set in the <code>cassandra.yaml</code> template file. See the cassandra_yaml_template above.)		
ring_name	<i>name</i>	Name of ring.
enable_vnodes	0 (no) or 1 (yes)	Enable or disable virtual nodes (vnodes). Default: 1 for Cassandra nodes, 0 for others.
seeds	Comma separated list of seed <i>IP_addresses</i> Do not make all nodes seed nodes. See Internode communications (gossip).	Seed list for this node.
interface	<i>IP_address</i>	Default interface to use for listening on all services.
listen_address	<i>IP_address</i>	listen_address
rpc_address	<i>IP_address</i>	rpc_address
cassandra_data_dir	<i>directory</i>	data_file_directories
cassandra_commitlog_dir	<i>directory</i>	commitlog_dir
cassandra_saved_caches_dir	<i>directory</i>	saved_caches_directory
rpc_port	<i>port_number</i>	rpc_port
storage_port	<i>port_number</i>	storage_port
ssl_storage_port	<i>port_number</i>	ssl_storage_port

Results

DataStax Enterprise is ready for [additional configuration](#).

What to do next

- [Configuring DataStax Enterprise](#).
- [Configuration and log file locations](#).
- [Starting and stopping DataStax Enterprise](#).
- [Deploying DataStax Enterprise for production](#).
- [Selecting hardware for enterprise implementations](#).
- [Configuring the heap dump directory](#) to avoid server crashes.

Other install methods

Installing DataStax Enterprise using Yum repositories

When installed from Yum, DataStax Enterprise runs as a service.

Important: DataStax Enterprise 4.8 uses Cassandra 2.1 and CQL3.1.

To install earlier versions, see [Installing earlier versions](#).

To install on SUSE, use the [GUI installer](#) or the [binary tarball installation](#).

Prerequisites

- Be sure your [platform is supported](#).
- DataStax Academy [registration](#) email address and password.
- Yum Package Management application.
- Root or sudo access to the install machine.
- Latest version of [Oracle Java SE Runtime Environment 7 or 8](#) or [OpenJDK 7](#) is recommended.

Note: If using Oracle Java 7, you must use at least 1.7.0_25. If using Oracle Java 8, you must use at least 1.8.0_40. In some cases, using JDK 1.8 causes minor performance degradation compared to JDK 1.7.

- RedHat-compatible distributions require EPEL (Extra Packages for Enterprise Linux). For RHEL 5.x, see [Installing EPEL on RHEL OS 5.x](#) on page 41.
- If installing on a 64-bit Oracle Linux distribution, first install the 32-bit versions of [glibc libraries](#).
- Python 2.6 (minimum); 2.7 (recommended).

Table: Hardware requirements

Requirement	Minimum	Production
CPUs	2	16
Memory	8 GB	24 GB
Data directory	20 GB	200 GB
Commit log directory	20 GB	200 GB
Saved caches directory	20 GB	200 GB
Logs directory	20 GB	200 GB
Production requirements depend on the volume of data and workload.		

Also see [Recommended production settings](#) and the [DataStax Enterprise Reference Architecture](#) white paper.

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

Procedure

These steps install DataStax Enterprise, the DataStax Agent, and OpsCenter (optional). After installing, you must configure and start DataStax Enterprise.

In a terminal window:

1. Verify that a required version of Java is installed:

```
$ java -version
```

If not Oracle Java 7, Oracle Java 8, or OpenJDK 7, see [Installing Oracle JDK or the OpenJDK documentation](#).

Important: Package management tools do not install Oracle Java.

2. Make sure that the EPEL is installed. See [Installing EPEL on RHEL OS 5.x](#) on page 41.
3. Add the DataStax Yum repository to a file called `/etc/yum.repos.d/datastax.repo`.

Note: Set the `gpgcheck=1` to perform a GPG signature check.

```
[datastax]
name = DataStax Repo for DataStax Enterprise
baseurl=https://dsa_email_address:password@rpm.datastax.com/enterprise
enabled=1
gpgcheck=0
```

where `dsa_email_address` and `password` are the DataStax Academy account credentials you created on the [registration page](#).

Attention: Depending on your environment, you might need to replace @ in your email address with %40 and escape any character in your password that is used in your operating system's command line. Examples: \! and \\.

4. If you have enabled signature verification (`gpgcheck=1`), import the DataStax Enterprise repository key:

```
$ rpm --import http://rpm.datastax.com/rpm/repo_key
```

5. Install the package:

```
$ sudo yum install dse-full-4.8.x-1 (Use for all product levels.)
```

For production installations, DataStax recommends installing the OpsCenter separate from the cluster. See the [OpsCenter](#) documentation.

Note: Removing the `datastax-agent` package also removes the DataStax Enterprise package.

Results

DataStax Enterprise is ready for configuration.

What to do next

- [Configuring DataStax Enterprise](#).
- [Configuration and log file locations](#).
- [Starting and stopping DataStax Enterprise](#).
- [Deploying DataStax Enterprise for production](#).
- [Selecting hardware for enterprise implementations](#).
- [Configuring the heap dump directory](#) to avoid server crashes.

Installing DataStax Enterprise using APT repositories

When installed from APT, DataStax Enterprise runs as a service.

Important: DataStax Enterprise 4.8 uses Cassandra 2.1 and CQL3.1.

Prerequisites

- Be sure your [platform is supported](#).
- DataStax Academy [registration](#) email address and password.
- Aptitude Package Management (APT) application.
- Root or sudo access to the install machine.
- Latest version of [Oracle Java SE Runtime Environment 7 or 8 or OpenJDK 7](#) is recommended.

Note: If using Oracle Java 7, you must use at least 1.7.0_25. If using Oracle Java 8, you must use at least 1.8.0_40. In some cases, using JDK 1.8 causes minor performance degradation compared to JDK 1.7.

- Python 2.6 (minimum); 2.7 (recommended).

Table: Hardware requirements

Requirement	Minimum	Production
CPUs	2	16
Memory	8 GB	24 GB
Data directory	20 GB	200 GB
Commit log directory	20 GB	200 GB
Saved caches directory	20 GB	200 GB
Logs directory	20 GB	200 GB

Production requirements depend on the volume of data and workload.

Also see [Recommended production settings](#) and the [DataStax Enterprise Reference Architecture](#) white paper.

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

Procedure

These steps install DataStax Enterprise, the DataStax Agent, and OpsCenter (optional). After installing, you must configure and start DataStax Enterprise.

In a terminal window:

1. Verify that a required version of Java is installed:

```
$ java -version
```

If not Oracle Java 7, Oracle Java 8, or OpenJDK 7, see [Installing Oracle JDK](#) or the [OpenJDK](#) documentation.

Important: Package management tools do not install Oracle Java.

2. Add a DataStax repository file called /etc/apt/sources.list.d/datastax.sources.list:

```
$ echo "deb https://dsa_email_address:password@debian.datastax.com/enterprise stable main" | sudo tee -a /etc/apt/sources.list.d/datastax.sources.list
```

where `dsa_email_address` and `password` are the DataStax Academy account credentials you created on the [registration page](#).

Attention: Depending on your environment, you might need to replace @ in your email address with %40 and escape any character in your password that is used in your operating system's command line. Examples: \! and \\.

3. Add the DataStax repository key:

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

4. Install the package:

- a) \$ sudo apt-get update
- b) Install the package: (Use for all product levels.)

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

For production installations, DataStax recommends installing the OpsCenter separate from the cluster. See the [OpsCenter](#) documentation.

Note: Removing the datastax-agent package also removes the DataStax Enterprise package.

Results

DataStax Enterprise is ready for configuration.

What to do next

- [Configuring DataStax Enterprise](#).
- [Configuration and log file locations](#).
- [Starting and stopping DataStax Enterprise](#).
- [Deploying DataStax Enterprise for production](#).
- [Selecting hardware for enterprise implementations](#).
- [Configuring the heap dump directory](#) to avoid server crashes.

Installing DataStax Enterprise using the binary tarball

For other product installations, see [Installing OpsCenter](#) and [Installing DevCenter](#).

When installed from the binary tarball, DataStax Enterprise runs as a stand-alone process.

Important: DataStax Enterprise 4.8 uses Cassandra 2.1 and CQL3.1.

Prerequisites

- All Linux platforms:
 - Be sure your [platform is supported](#).
 - DataStax Academy [registration](#) email address and password.
 - Latest version of [Oracle Java SE Runtime Environment 7 or 8 or OpenJDK 7](#) is recommended.

Note: If using Oracle Java 7, you must use at least 1.7.0_25. If using Oracle Java 8, you must use at least 1.8.0_40. In some cases, using JDK 1.8 causes minor performance degradation compared to JDK 1.7.

- Python 2.6 (minimum); 2.7 (recommended).
- On some versions of Mac OS X, you may need to install readline: `easy_install readline`.
- RedHat-compatible distributions:
 - If installing on a 64-bit Oracle Linux distribution, first install the 32-bit versions of [glibc libraries](#).
 - If you are using an earlier RHEL-based Linux distribution, such as CentOS-5, you might need to replace the Snappy compression/decompression library; see the [DataStax Enterprise 4.5.0 Release Notes](#).
 - Before installing, make sure EPEL (Extra Packages for Enterprise Linux) is installed. See [Installing EPEL on RHEL OS 5.x](#) on page 41.

Table: Hardware requirements

Requirement	Minimum	Production
CPUs	2	16
Memory	8 GB	24 GB
Data directory	20 GB	200 GB
Commit log directory	20 GB	200 GB
Saved caches directory	20 GB	200 GB
Logs directory	20 GB	200 GB
Production requirements depend on the volume of data and workload.		

Also see [Recommended production settings](#) and the [DataStax Enterprise Reference Architecture](#) white paper.

Procedure

These steps install DataStax Enterprise, the DataStax Agent, and OpsCenter (optional). After installing, you must configure and start DataStax Enterprise.

In a terminal window:

1. Verify that a required version of Java is installed:

```
$ java -version
```

If not Oracle Java 7, Oracle Java 8, or OpenJDK 7, see [Installing Oracle JDK](#) or the [OpenJDK](#) documentation.

Important: Package management tools do not install Oracle Java.

2. Download and extract the DataStax Enterprise tarball using the DataStax Academy account credentials you created on the [registration page](#).

```
$ curl --user dsa_email_address:password -L http://downloads.datastax.com/enterprise/dse.tar.gz | tar xz
```

where `dsa_email_address` and `password` are the DataStax Academy account credentials you created on the [registration page](#).

Attention: Depending on your environment, you might need to replace @ in your email address with %40 and escape any character in your password that is used in your operating system's command line. Examples: \! and \\|.

CAUTION: If you choose to run the above command, your password will be retained in the shell history. To avoid this DataStax recommends using curl with the `--netrc` or `--netrc-file` option. Alternately, download the tarball from [DataStax downloads](#).

The files are downloaded and extracted into the `dse-4.8.x` directory.

3. Optional: Download and extract the OpsCenter tarball:

```
$ curl -L http://downloads.datastax.com/community/opscenter.tar.gz | tar xz
```

For production installations, DataStax recommends installing the OpsCenter separate from the cluster. See the [OpsCenter](#) documentation.

4. If you do not have root access to the default directories locations, you can define your own directory locations as described in the following steps or change the ownership of the directories:

- `/var/lib/cassandra`
- `/var/log/cassandra`
- `/var/lib/spark`
- `/var/log/spark`

```
$ 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  
$ sudo mkdir -p /var/lib/spark; sudo chown -R $USER:$GROUP /var/lib/spark  
$ sudo mkdir -p /var/log/spark; sudo chown -R $USER:$GROUP /var/log/spark
```

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:

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

b) Go the directory containing the `cassandra.yaml` file:

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

c) Edit the following lines in the `cassandra.yaml` file:

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

6. Optional: If you do not want to use the default Spark directories, you can define your own directory locations:

a) Make the directories for the Spark `lib` and `log` directories.

b) Edit the `spark-env.sh` file to match the locations of your Spark `lib` and `log` directories, as described in [Configuring Spark nodes](#) on page 81.

Results

DataStax Enterprise is ready for configuration.

What to do next

- [Configuring DataStax Enterprise](#).
- [Configuration and log file locations](#).
- [Starting and stopping DataStax Enterprise](#).

- Deploying DataStax Enterprise for production.
- Selecting hardware for enterprise implementations.
- Configuring the heap dump directory to avoid server crashes.

Installing on cloud providers

Installing 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 4.8 cluster using the DataStax AMI (Amazon Machine Image). Installing DataStax Enterprise with the AMI allows you to quickly deploy a cluster with a pre-configured mixed workload. When you launch the AMI, you can specify the total number of nodes in your cluster and how many nodes should be Cassandra (transactional), DSE Analytics (Hadoop and Spark), or DSE Search (Solr).

You can also use Lifecycle Manager in OpsCenter to easily provision a DataStax Enterprise cluster for versions 4.7 and later:

1. Create your instances using an AMI from a [trusted source](#) for any [supported platform](#).
2. Use the [Lifecycle Manager](#) to provision and configure your cluster.

The DataStax AMI does the following:

- Installs the latest version of DataStax Enterprise with an Ubuntu 12.04 LTS (Precise Pangolin), image (Ubuntu Cloud 20140227 release), Kernel 3.8+.
- Installs Oracle Java 7.
- Installs metrics tools such as dstat, ethtool, make, gcc, and s3cmd.
- Uses RAID0 ephemeral disks for data storage and commit logs.
- Provides a choice of virtualization types: PV (paravirtualization) or HVM (hardware-assisted virtual machine).
- Launches EBS-backed instances for faster start-up, **not** database storage.
- Uses the private interface for intra-cluster communication.
- Starts the nodes in the specified type: Cassandra (transactional), DSE Analytics, or DSE Search.
- Sets the seed nodes cluster-wide.
- Installs DataStax OpsCenter (by default).

Note: The DataStax AMI does not install DataStax Enterprise nodes with [virtual nodes enabled](#).

EC2 clusters spanning multiple regions and availability zones

The DataStax AMI is intended for a single region and availability zone. When creating an EC2 cluster that spans multiple regions and availability zones, use OpsCenter to set up your cluster. See [EC2 clusters spanning multiple regions and availability zones](#).

Production considerations

For production DataStax Enterprise clusters on EC2, see [Production deployment planning](#) on page 405. RAID0 the ephemeral disks, and put both the data directory and the commit log on that volume. This strategy is preferred to putting the commit log on the root volume (which is a shared resource). For more data redundancy, consider deploying your cluster across multiple availability zones or using OpsCenter to backup to S3.

Note: DSE Analytics and DSE Search nodes require their own nodes/disks and have specific hardware requirements. See [Capacity Planning](#) in the *DataStax Enterprise Reference Architecture* and the [Hadoop](#) and [Solr](#) documentation.

What to do next

[Launch the AMI](#)

Launching the DataStax AMI

After you select your region, you can launch your AMI instances.

If you need more help, click an informational icon or a link to the *Amazon EC2 User Guide*.

Note: Because Amazon changes the EC2 console without notice, there might be some differences in the user interface and options. For details, see the [Amazon EC2 documentation](#).

Procedure

1. Sign in to the [AWS console](#), and then click **EC2**.
2. From the Amazon EC2 console navigation bar, select the region for launching the DataStax Community AMI.



Amazon EC2 offers a number of [geographic regions](#) for launching the AMI. Factors for choosing a region include reduce latency, cost, or regulatory requirements.

3. Click **Launch Instance**.
4. In **Step 1: Choose an Amazon Machine Image (AMI)**, select **Community AMIs**, search for DataStax, and then select the appropriate instance.
5. In **Step 2: Choose an Instance Type**, select the appropriate instance type as described in [Planning an Amazon EC2 cluster](#).

When the instance is selected, its specifications are displayed near the top of the page:

Currently selected: m3.large (6.5 ECUs, 2 vCPUs, 7.5 GiB memory, 1 x 32 GiB Storage Capacity)

Because Amazon updates instance types periodically, see the following documents to determine your hardware and storage requirements:

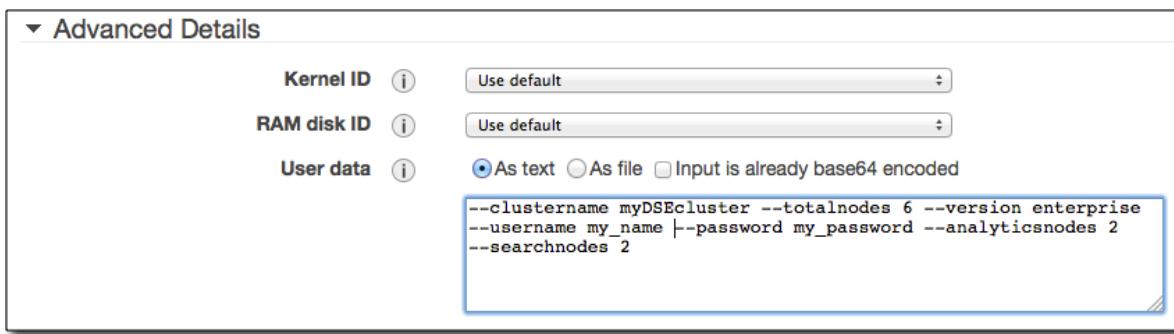
- [Planning an Amazon EC2 cluster](#)
 - [Amazon EC2 documentation](#)
 - [What is the story with AWS storage](#)
 - [Get in the Ring with Cassandra and EC2](#)
6. Click **Next: Configure Instance Details**.
 7. In **Step 3: Configure Instance Details**, configure the instances to meet your requirements:
 - a) Select the number of instances.
 - b) Open **Advanced Details**.

- c) Add the following options (as text) to the **User Data** section, according to the type of cluster.

Option	Description
Basic AMI switches	
--clustername <i>name</i>	Required. The name of the cluster.
--totalnodes <i>total_nodes</i>	Required. The total number of nodes in the cluster.
--version [enterprise community]	Required. The version of the cluster. Use enterprise to install the latest version of DataStax Enterprise.
DataStax Enterprise switches	
--username <i>username</i>	Required for DataStax Enterprise. DataStax registration username. Register at DataStax Academy .
--password <i>password</i>	Required for DataStax Enterprise. DataStax registration password. Register at DataStax Academy .
--analyticsnodes <i>num_analytics_nodes</i>	Optional for DataStax Enterprise. For mixed-workload clusters, the number of DSE Analytics with Spark nodes. Default: 0. Note: In versions earlier than DataStax Enterprise 4.5, the default is to use Hadoop.
--searchnodes <i>num_dseSearch_nodes</i>	Optional for DataStax Enterprise. For mixed-workload clusters, the number of DSE Search with Solr nodes. Default: 0.
--hadoop	Use Hadoop instead of Spark. Default: false.
Advanced Switches	
--release <i>version</i>	Optional. Allows installation of an earlier version. For example, 4.0.2-1. Default: Ignored.
--cfsreplicationfactor <i>repl_factor</i>	Optional for DataStax Enterprise. Sets the replication factor for the CFS keyspace. This number must be less than or equal to the number of analytics nodes. Default: 1.
--opscenter Yes	Optional. By default, DataStax OpsCenter is installed on the first instance. Specify no to disable.
--reflector <i>url</i>	Optional. Allows you to use your own reflector. Default: <code>http://reflector2.datastax.com/reflector2.php</code> .

For example:

```
--clustername myDSEcluster --totalnodes 6 --version enterprise --
username my_name
--password my_password --analyticsnodes 2 --searchnodes 2
```



8. Click Next: Add Storage.

9. In Step 4: Add Storage, add volumes as needed.

The number of instance store devices available to the machine depends on the instance type.

SSD-backed general purpose volumes (GP2) or provisioned IOPS volumes (PIOPS) are suitable for production workloads. These volume types are designed to deliver consistent, low latency performance:

Option	Description
GP2	<ul style="list-style-type: none"> The best choice for most workloads and has the added advantage of guaranteeing 10,000 IOPS when volumes larger than 3.5 TB are attached to instances. Designed to deliver single-digit millisecond latencies. Designed to deliver the provisioned performance 99.0% of the time.
PIOPS	<ul style="list-style-type: none"> Designed to deliver single-digit millisecond latencies. Designed to deliver the provisioned performance 99.9% of the time.

Magnetic EBS volumes are not recommended for database storage for the following reasons:

- EBS Magnetic volumes contend directly for network throughput with standard packets. This contention means that EBS throughput is likely to fail if you saturate a network link.
- EBS Magnetic volumes have unreliable performance. I/O performance can be exceptionally slow, causing the system to back load 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.

Note: Use only ephemeral instance-store or the recommended EBS volume types for Cassandra data storage. For more information and graphs related to ephemeral versus EBS Magnetic performance, see the [Systematic Look at EC2 I/O](#) blog.

10.Click Next: Tag Instance.

11.In Step 5: Tag Instance, create a meaningful tag for your DataStax Enterprise instance.

A tag consists of a case-sensitive key-value pair. Tags enable you to categorize your AWS resources in different ways, such as purpose, owner, or environment.

12.Click Next: Configure Security Group.

13.In Step 6: Configure Security Group:

- If you already have a security group, choose **Select an existing security group**. The security group uses inbound rules for the default ports for DataStax Enterprise.
- Otherwise, [Create an EC2 security group](#) to define a set of firewall rules that control the traffic for your instance.

14. Click **Review and Launch** and make any required changes.

15. Click **Launch** and then in the **Select an existing key pair or create a new key pair** dialog, do one of the following:

- Select an existing key pair from the **Select a key pair** drop list.
- If you need to create a new key pair, click **Choose an existing key pair** drop list and select **Create a new key pair**. Then download and protect the private key file as described in [Create a key pair](#) on page 34.

16. Click **Launch Instances** and review the instance in **Step 7: Review Instance Launch**.

17. Click **Launch**.

The AMI image configures your cluster and starts the Cassandra, Spark, Solr, and OpsCenter services. The **Launch Status** page is displayed.

18. Click **View Instances**.

What to do next

Depending on what you've done, take one of the following steps:

- [Create a key pair](#) on page 34
- [Connect to your DataStax Enterprise EC2 instance](#)

Creating an EC2 security group

To protect your cluster, you should define a security group. An [EC2 Security Group](#) acts as a firewall that allows you to choose which protocols and ports are open in your cluster. You must specify a security group in the same region as your instances.

You can specify the protocols and ports either by a range of IP addresses or by security group. Be aware that specifying a Source IP of 0.0.0.0/0 allows every IP address access by the specified protocol and port range.

If you need more help, click an informational icon or a link to the *Amazon EC2 User Guide*.

Procedure

1. Open the **Security Groups** page.
2. On the **Configure Security Group** page, create a security group with a name and description. DataStax recommends including the region name in the description.
3. Create rules for the security group using the information in the following table:

Table: Ports

Port number	Type	Protocol	Source	Description
For detailed information about ports, see Configuring firewall port access on page 348.				
Public facing				
22	SSH	TCP	0.0.0.0/0	SSH (default)
<i>DataStax Enterprise public ports</i>				

Port number	Type	Protocol	Source	Description
4040	Custom TCP Rule	TCP	0.0.0.0/0	Spark application web site port.
7080	Custom TCP Rule	TCP	0.0.0.0/0	Spark Master web site port.
7081	Custom TCP Rule	TCP	0.0.0.0/0	Spark Worker web site port.
8012	Custom TCP Rule	TCP	0.0.0.0/0	Hadoop Job Tracker client port. The Job Tracker listens on this port for job submissions and communications from Task Trackers; allows traffic from each analytics node in a cluster.
8983	Custom TCP Rule	TCP	0.0.0.0/0	Solr port and Demo applications web site port (Portfolio, Search, Search log, Weather Sensors)
50030	Custom TCP Rule	TCP	0.0.0.0/0	Hadoop Job Tracker web site port. The Job Tracker listens on this port for HTTP requests. If initiated from the OpsCenter, these requests are proxied through the opscenterd daemon; otherwise, they come directly from the browser.
50060	Custom TCP Rule	TCP	0.0.0.0/0	Hadoop Task Tracker web site port. Each Task Tracker listens on this port for HTTP requests coming directly from the browser and not proxied by the opscenterd daemon.
<i>OpsCenter public ports</i>				
8888	Custom TCP Rule	TCP	0.0.0.0/0	OpsCenter web site port. The opscenterd daemon listens on this port for HTTP requests coming directly from the browser.
Inter-node ports				
<i>Cassandra inter-node ports</i>				
1024 - 65355 Cassandra 1.2 or earlier only	Custom TCP Rule	TCP	My IP	JMX reconnection/loopback ports. Because JMX connects on port 7199, handshakes, and then uses any port within the 1024+ range, use SSH to execute commands remotely to connect to JMX locally or use the DataStax OpsCenter.
7000	Custom TCP Rule	TCP	My IP	Cassandra inter-node cluster communication port.
7001	Custom TCP Rule	TCP	My IP	Cassandra SSL inter-node cluster communication port.

Port number	Type	Protocol	Source	Description
7199	Custom TCP Rule	TCP	My IP	Cassandra JMX monitoring port.
9160	Custom TCP Rule	TCP	My IP	Cassandra client port (Thrift) port. OpsCenter agents makes Thrift requests to their local node on this port. Additionally, the port can be used by the opscenterd daemon to make Thrift requests to each node in the cluster.
<i>DataStax Enterprise inter-node ports</i>				
7077	Custom TCP Rule	TCP	My IP	Spark Master inter-node communication port.
8984	Custom TCP Rule	TCP	My IP	Solr inter-node communication port.
9042	Custom TCP Rule	TCP	My IP	CQL native clients port.
9290	Custom TCP Rule	TCP	My IP	Hadoop Job Tracker Thrift port. The Job Tracker listens on this port for Thrift requests coming from the opscenterd daemon.
10000	Custom TCP Rule	TCP	My IP	Hive server port.
10000	Custom TCP Rule	TCP	My IP	Shark is deprecated.
<i>OpsCenter inter-node ports</i>				
61620	Custom TCP Rule	TCP	My IP	OpsCenter monitoring port. The opscenterd daemon listens on this port for TCP traffic coming from the agent.
61621	Custom TCP Rule	TCP	My IP	OpsCenter agent port. The agents listen on this port for SSL traffic initiated by OpsCenter.

The completed port rules should look similar to this:

Edit inbound rules

Type	Protocol	Port Range	Source	
Custom TCP Rule	TCP	7000	Custom IP sg-bbc40aff	X
Custom TCP Rule	TCP	7001	Custom IP sg-bbc40aff	X
Custom TCP Rule	TCP	7199	Custom IP sg-bbc40aff	X
Custom TCP Rule	TCP	8984	Custom IP sg-bbc40aff	X
Custom TCP Rule	TCP	9042	Custom IP sg-bbc40aff	X
Custom TCP Rule	TCP	9160	Custom IP sg-bbc40aff	X
Custom TCP Rule	TCP	9290	Custom IP sg-bbc40aff	X
Custom TCP Rule	TCP	61620	Custom IP sg-bbc40aff	X
Custom TCP Rule	TCP	61621	Custom IP sg-bbc40aff	X
SSH	TCP	22	Anywhere 0.0.0.0/0	X
Custom TCP Rule	TCP	8012	Anywhere 0.0.0.0/0	X
Custom TCP Rule	TCP	8888	Anywhere 0.0.0.0/0	X
Custom TCP Rule	TCP	8983	Anywhere 0.0.0.0/0	X
Custom TCP Rule	TCP	50030	Anywhere 0.0.0.0/0	X
Custom TCP Rule	TCP	50060	Anywhere 0.0.0.0/0	X

Add Rule **Cancel** **Save**

Warning: The security configuration shown in this example opens up all externally accessible ports to incoming traffic from any IP address (0.0.0.0/0). The risk of data loss is high. For a more secure configuration, see the Amazon EC2 help on security groups.

What to do next

Depending on what you've done, take one of the following steps:

- [Launch the AMI](#)
- [Create a key pair on page 34](#)
- [Connect to your DataStax Enterprise EC2 instance](#)

Create a key pair

Amazon EC2 uses public–key cryptography to encrypt and decrypt login information. Public–key cryptography uses a public key to encrypt data and the recipient uses the private key to decrypt the data.

If you need more help, click an informational icon or a link to the *Amazon EC2 User Guide*.

Procedure

You must create a key pair for each region you use.

1. In the **Select an existing key pair or create a new key pair** dialog, enter a key pair name in the **Key pair name** field.
2. Click **Download Key Pair**.

You must download the private key file *.pem before you can continue. Store the .pem file in a secure and accessible location.

- Set the permissions of your private key file so that only you can read it.

```
$ chmod 400 my-key-pair.pem
```

What to do next

Depending on what you've done, take one of the following steps:

- [Launch the AMI](#)
- [Connect to your DataStax Enterprise EC2 instance](#)

Connecting to your DataStax Enterprise EC2 instance

After the cluster is launched, you can connect to it from a terminal or SSH client, such as PuTTY.

If you need more help, click an informational icon or a link to the *Amazon EC2 User Guide*.

Procedure

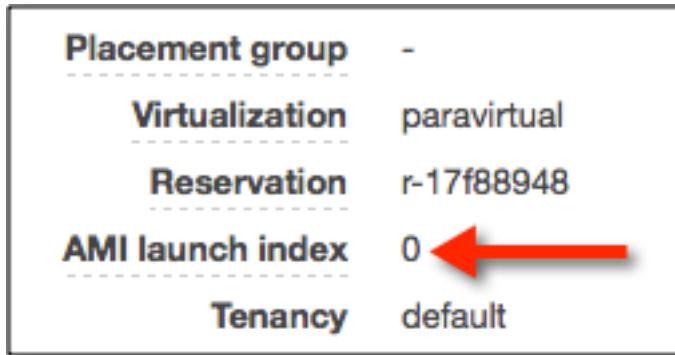
- If necessary, from the **EC2 Dashboard**, click **Running Instances**.

You can connect to any node in the cluster. However, one node (Node0) runs OpsCenter and is the [Cassandra seed](#) node.

	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
<input type="checkbox"/>	i-e0a007bd	m1.large	us-west-1a	● running	● 2/2 checks...	None		ec2-54-219-64-150.us...
<input type="checkbox"/>	i-e1a007bc	m1.large	us-west-1a	● running	● 2/2 checks...	None		ec2-54-219-4-143.us-w...
<input type="checkbox"/>	i-fca007a1	m1.large	us-west-1a	● running	● 2/2 checks...	None		ec2-184-169-235-148.u...
<input checked="" type="checkbox"/>	i-eea5b5b5	m1.large	us-west-1b	● running	● 2/2 checks...	None		ec2-54-219-14-16.us-w...
<input type="checkbox"/>	ifea007a3	m1.large	us-west-1a	● running	● 2/2 checks...	None		ec2-204-236-191-57.us...
<input type="checkbox"/>	iffa007a2	m1.large	us-west-1a	● running	● 2/2 checks...	None		ec2-54-219-53-12.us-w...

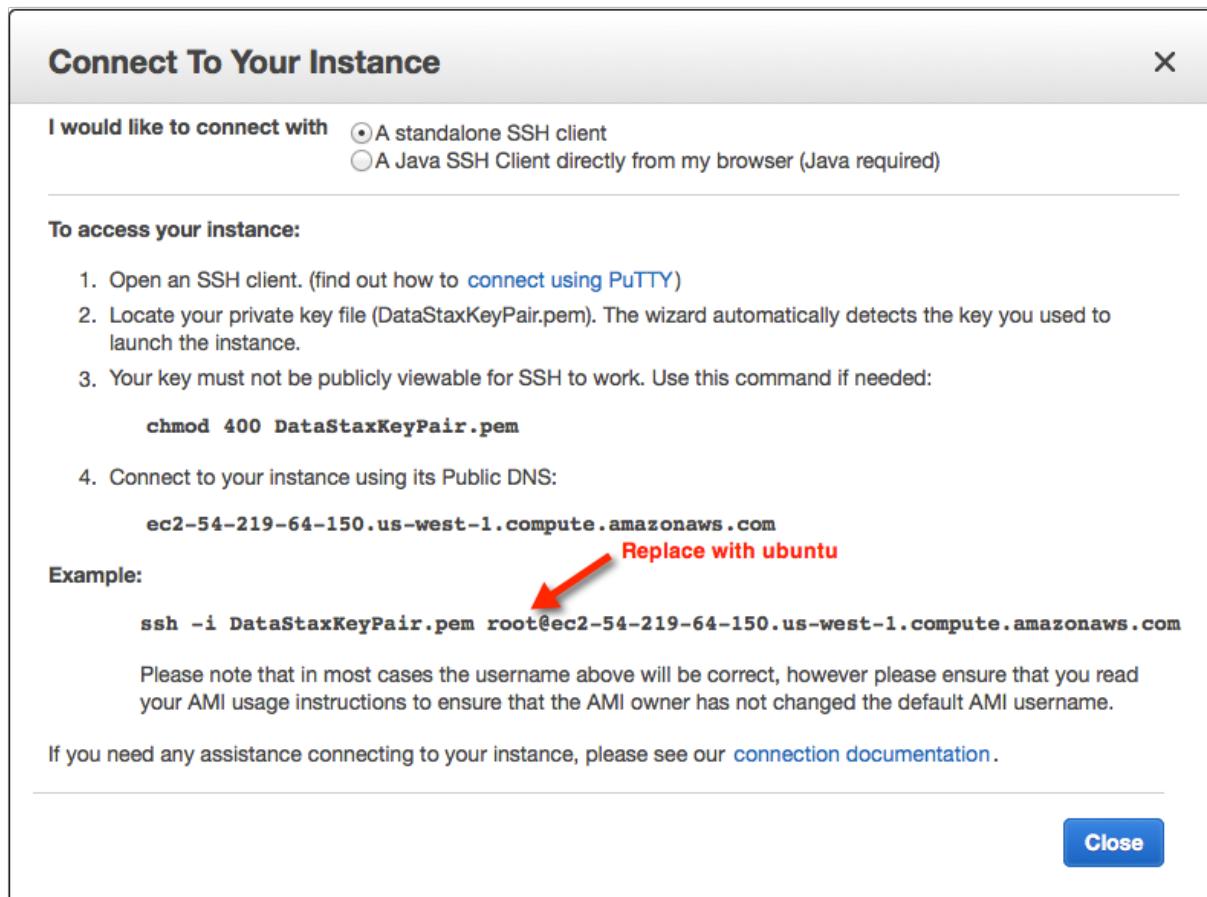
- To find which instance is Node0:

- Select an instance.
- On the **Description** tab, scroll down the description information until you see **AMI launch index**.



- Repeat until you find Node0.
- To get the public DNS name of a node, select the node to connect to, and then click **Connect**.
- In **Connect To Your Instance**, select **A standalone SSH client**.

5. Copy the example command line, change the user from root to ubuntu, and then paste it into an SSH client.



The AMI image configures your cluster and starts the DataStax Enterprise services.

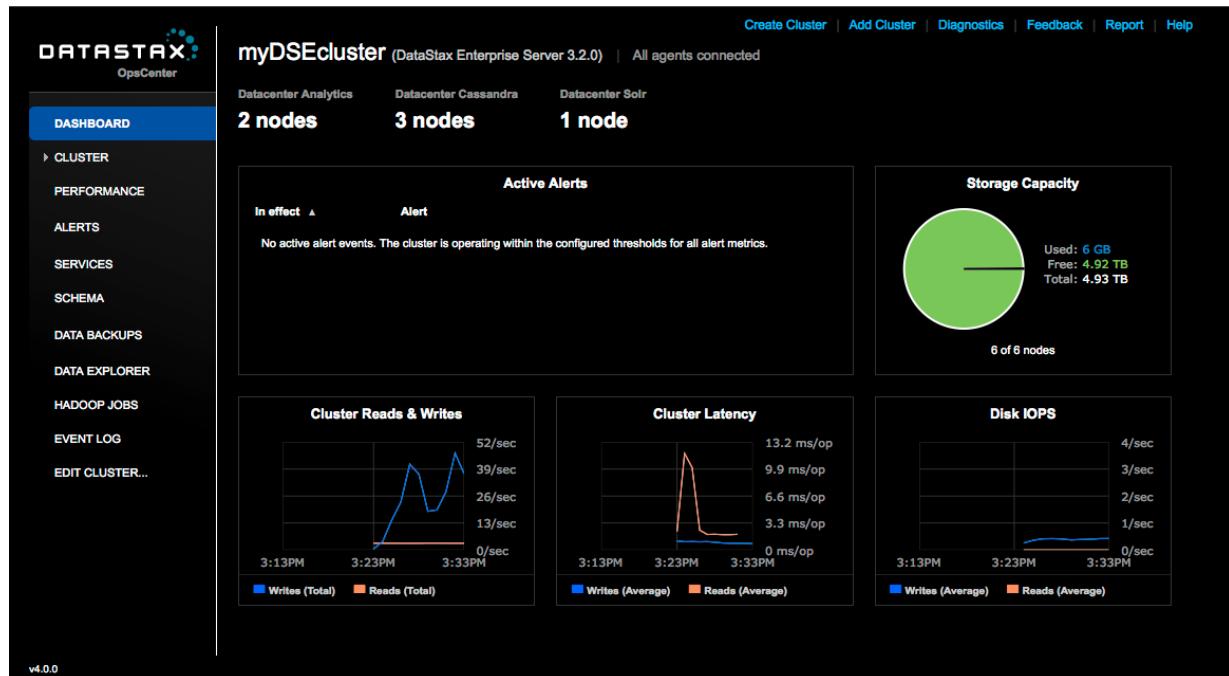
6. After you have logged into a node and the AMI has completed installing and setting up the nodes, the status is displayed:

```
Datacenter: Cassandra
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address      Load     Owns   Host ID                               Token                               Rack
UN 10.176.9.18  14.02 KB  33.3%  a2149321-6b7f-4e0c-9105-909417ea2b5c -9223372036854775808  rack1
UN 10.168.129.246 14.02 KB  33.3%  1234407f-5df1-4cf8-8078-ecefdf63677 -3074457345618258603  rack1
UN 10.176.154.48  14.02 KB  16.7%  df484f96-3b6c-4f64-8a33-02b23db651ff 3074457345618258602  rack1
Datacenter: Analytics
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address      Load     Owns   Host ID                               Token                               Rack
UN 10.168.162.240 28.44 KB  0.0%  e2451cdf-f07f-4ddb-bbc2-f93a4b983666 -9223372036854765808  rack1
UN 10.168.5.99   44.47 KB  16.7%  f322188e-a2c5-4175-827d-512b51978aed 10000   rack1
Datacenter: Solr
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address      Load     Owns   Host ID                               Token                               Rack
UN 10.176.61.212 18.85 KB  0.0%  a7cbd669-9e83-463f-824c-6785fd717329 -9223372036854755808  rack1
Opscenter: http://ec2-54-219-83-162.us-west-1.compute.amazonaws.com:8888/
Please wait 60 seconds if this is the cluster's first start...
```

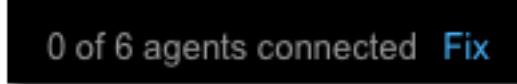
The URL for the OpsCenter is displayed after you connect to the node containing it; otherwise the OpsCenter URL is not displayed.

- If you installed OpsCenter, allow 60 to 90 seconds after the cluster has finished initializing for OpsCenter to start. You can launch OpsCenter using the URL:
`http://public_dns_of_first_instance:8888/`

The Dashboard should show that the agents are connected.



- If the agents have not automatically connected:
 - Click the **Fix** link located near the top left of the **Dashboard**.



- When prompted for credentials for the agent nodes, use the username `ubuntu` and copy and paste the entire contents from your private key file (`.pem`).

The Dashboard shows the agents are connected.

Related tasks

[Starting as a service](#) on page 45

[Stopping a node](#) on page 48

Using OpsCenter to create a cluster on Amazon EC2

You can create a cluster using OpsCenter. With this method you first launch a single node using the DataStax AMI, and then create the cluster from OpsCenter.

Procedure

- Sign in to the [AWS console](#), and then click **EC2**.
- Create rules for the security group using the ports and values in the following table:

Table: Ports

Port range	Type	Protocol	Source	Description
22	SSH	TCP	0.0.0.0/0	SSH (default)
8888	Custom TCP Rule	TCP	IP address of the machine that connects to OpsCenter or 0.0.0.0/0	OpsCenter web site port. The opscenterd daemon listens on this port for HTTP requests coming directly from the browser.
61620	Custom TCP Rule	TCP	0.0.0.0/0	OpsCenter monitoring port. The opscenterd daemon listens on this port for TCP traffic coming from the agent.
61621	Custom TCP Rule	TCP	0.0.0.0/0	OpsCenter agent port. The agents listen on this port for SSL traffic initiated by OpsCenter.

These rules open the required ports so that OpsCenter can launch the AMI instances. When launching the cluster, OpsCenter creates its own security group unless otherwise specified.

The completed port rules should look similar to this:

Type	Protocol	Port Range	Source
SSH	TCP	22	Anywhere 0.0.0.0/0
Custom TCP Rule	TCP	8888	My IP 97.178.121.80
Custom TCP Rule	TCP	61620 - 61621	Anywhere 0.0.0.0/0

Add Rule **Cancel** **Save**

3. Create the key pair.

4. Launch the AMI.

Note: You can use only PV instances for OpsCenter.

5. In Step 2: Choose an Instance Type, choose m1.large.

When the instance is selected, its specifications are displayed near the top of the page:

Currently selected: m1.large (4 ECUs, 2 vCPUs, Intel Xeon Family, 7.5 GiB memory, 2 x 420 GiB Storage Capacity)

6. Click Next: Configure Instance Details and set the following:

- **Number of instances:** 1
- **Network: Launch into EC2-Classic**
- **Advanced Details - User data:** --opscenteronly

7. Click Next several times until the Step 6: Configure Security Group page appears.

8. Choose **Select an existing security group**, and then select your security group.
9. Click **Review and Launch** and make any required changes.
- 10. Click Launch.**
11. In the **Select an existing key pair or create a new key pair** dialog, select the key pair that you created.
12. Click **Launch Instances** and review the instance in **Step 7: Review Instance Launch**.
13. Click **Launch**.

The AMI image configures and starts OpsCenter.

- 14. Click View Instances.**
15. After the instance is running and the status checks are complete, connect a web browser to OpsCenter:
 - a) If necessary, from the **EC2 Dashboard**, click **Running Instances**.
 - b) Select the instance and use the public DNS for connecting to the OpsCenter.

Instance: i-15d2bbdf **Public DNS:** ec2-54-176-58-100.us-west-1.compute.amazonaws.com

- c) Launch OpsCenter using the public DNS. For example:
<http://ec2-54-193-159-3.us-west-1.compute.amazonaws.com:8888/>

- 16. Launch your cluster.**

- 17. In Welcome to DataStax OpsCenter, click Create Brand New Cluster.**

- 18. Fill out the form as appropriate.**

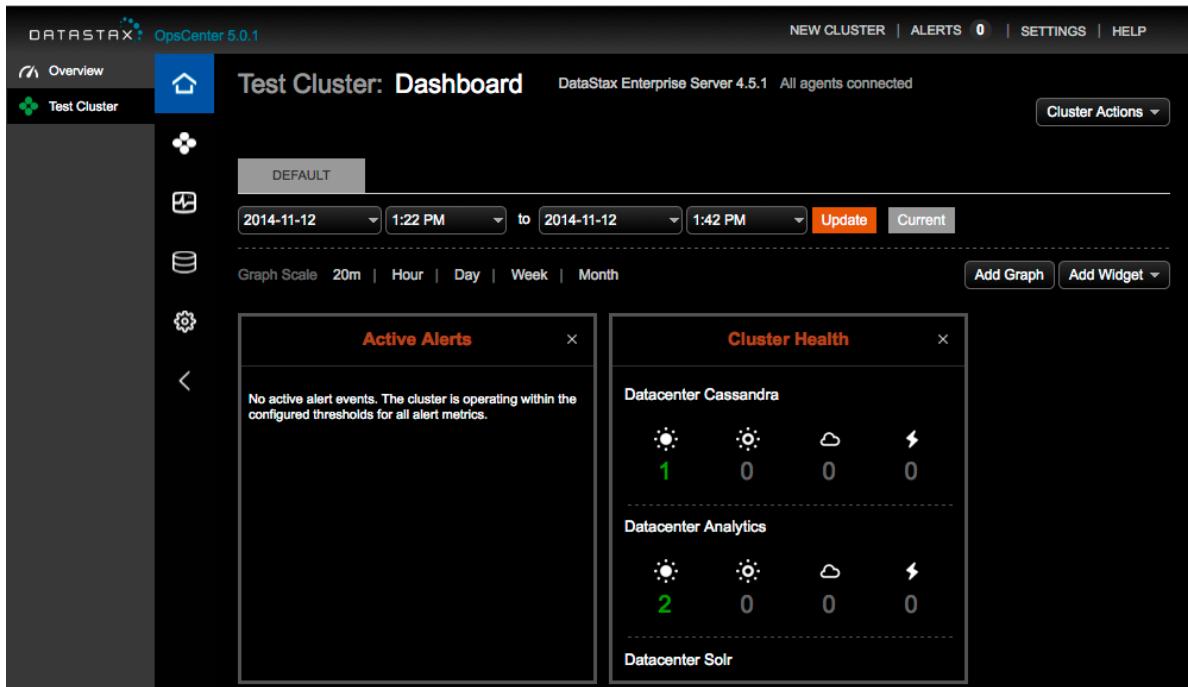
Table: New cluster fields

Field	Description
Name	My Cluster
Package	The version of DataStax Enterprise or Cassandra to install on the nodes.
DataStax Credentials	The <i>userid</i> and <i>password</i> credentials that were in the email you received from DataStax after you registered to download DataStax Enterprise.
Total Nodes	Total number of nodes for the cluster. (Only when provisioning to the cloud.)
# Solr Nodes	Total number of DSE Search (Solr) nodes for the cluster.
# Hadoop Nodes	Total number of Hadoop nodes for the cluster.
# Spark Nodes	Available for DataStax Enterprise 4.5 and later clusters, the total number of Spark nodes for the cluster.
Amazon EC2 Credentials	The <i>access-key-id</i> and <i>secret-access-key</i> to authenticate on AWS EC2.
Availability Zone	Which availability zone to use to create the cluster. (The drop-down list is populated after entering your EC2 credentials.)
Size	Which size image to use.
AMI	Which image to use.
Use OpsCenter specific security group	Determines whether OpsCenter creates its own specific security group or allows you to select one which is available using your EC2 credentials.

Field	Description
Use OpsCenter specific key pair	Determines whether OpsCenter creates its own specific key pair or allows you to select one which is available using your EC2 credentials.

19.Launch your cluster:

- Click **Build Cluster**.



20.After the cluster is running, change the inbound rules for ports 61620 and 61621 to the OpsCenter Provisioning Security Group:

- In the console, click **Security Groups**.
- Select **OpsCenter Security Group** > **Actions** > **Edit inbound rules**.
- Change the inbound rules to the **OpsCenter Security Group** and then click **Save**.

What to do next

[OpsCenter User Guide](#)

Installing and deploying a DataStax Enterprise cluster in CenturyLink Cloud

DataStax Academy provides information about installing and deploying DataStax Enterprise clusters using various cloud providers. The information provided in the [Getting Started with DataStax Enterprise in the CenturyLink Cloud](#) documentation includes:

- Detailed steps for deploying DataStax Enterprise-ready nodes using CenturyLink Cloud.
- Instructions for deploying DataStax Enterprise on those nodes using DataStax OpsCenter.

Installing and deploying a DataStax Enterprise cluster using Google Compute Engine

The DataStax Academy provides information about installing and deploying DataStax Enterprise clusters using various cloud providers. The information provided in the [DataStax Enterprise Deployment Guide for Google Compute Engine](#) (GCE) documentation includes:

- Detailed steps for deploying DataStax Enterprise-ready nodes using the Google Compute Engine.
- Instructions for deploying DataStax Enterprise on those nodes using DataStax OpsCenter.
- Deployment considerations when mapping DataStax Enterprise high-availability features to GCE high-availability mechanisms.

Installing and deploying a DataStax Enterprise cluster using Microsoft Azure

The DataStax Academy provides information about installing and deploying DataStax Enterprise clusters using various cloud providers. The information provided in the [Enterprise Deployment for Microsoft Azure Cloud](#) documentation includes:

- Description of technical conventions in Microsoft Azure.
- Detailed steps for deploying DataStax Enterprise-ready nodes.
- Instructions for deploying DataStax Enterprise on those nodes using DataStax OpsCenter.
- Configuring multi-region datacenters in Microsoft Azure.

Installing EPEL on RHEL OS 5.x

Before installing DataStax Enterprise on RHEL OS 5.x, install the Extra Packages for Enterprise Linux (EPEL). EPEL contains important dependent packages that enable installation of Python 2.6.x that is required by DataStax Enterprise.

Note: Only RHEL OS 5.x requires EPEL.

You must install EPEL as root user:

32-bit RHEL and CentOS 5.x

```
# wget http://download.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm  
# rpm -ivh epel-release-5-4.noarch.rpm
```

64-bit RHEL and CentOS 5.x

```
# wget http://download.fedoraproject.org/pub/epel/5/x86_64/epel-release-5-4.noarch.rpm  
# rpm -ivh epel-release-5-4.noarch.rpm
```

Installing earlier versions of DataStax Enterprise

DataStax provides binary tarball and packaged versions for installing earlier versions (2.2.x upwards) of DataStax Enterprise.

Note: You must use Oracle JRE/JDK 6, not 7, for versions earlier than DataStax Enterprise 3.1. These earlier versions do not support JRE/JDK 7.

Installing from the binary tarball

Download the tarball from the [Download DataStax Enterprise](#) page and follow the install instructions in the relevant documentation:

- [DataStax Enterprise 4.6.x tarball install documentation](#)
- [DataStax Enterprise 4.6.x tarball install documentation](#)
- [DataStax Enterprise 4.5.x tarball install documentation](#)
- [DataStax Enterprise 4.0.x tarball install documentation](#)
- [DataStax Enterprise 2.2.x tarball install documentation](#)

Installing the packages on RHEL-based or Debian-based platforms

Follow the install instructions in the relevant documentation and specify the specific version in the install command:

RHEL-based platforms

Format:

```
$ sudo yum -y install dse-full-version-1
```

Example:

```
$ sudo yum -y install dse-full-4.7.2-1
```

Debian-based platforms

Be sure that you have defined the DataStax repository and repository key. See the installation instructions for your version of DataStax Enterprise.

You can use Aptitude or apt-get.

Aptitude

```
$ sudo aptitude install dse-full=version
```

Example:

```
$ sudo aptitude install dse-full=4.7.2-1
```

When prompted, do not accept the first two solutions, and type Y to accept the third solution:

```
The following NEW packages will be installed:  
dse-full{b}  
0 packages upgraded, 1 newly installed, 0 to remove and 309 not upgraded.  
Need to get 10.3 kB of archives. After unpacking 53.2 kB will be used.  
The following packages have unmet dependencies:  
dse-full : Depends: dse (= 4.7.2-1) but it is not going to be installed.  
          Depends: dse-hive (= 4.7.2-1) but it is not going to be installed.  
          ...  
          Depends: dse-libspark (= 4.7.2-1) but it is not going to be  
          installed.  
The following actions will resolve these dependencies:  
  
Keep the following packages at their current version:  
1) dse-full [Not Installed]  
  
Accept this solution? [Y/n/q/?] n  
The following actions will resolve these dependencies:
```

```

Install the following packages:
1) datastax-agent [5.1.3 (stable)]
2) dse [4.7.2-1 (stable)]
...
16) dse-pig [4.7.2-1 (stable)]

Accept this solution? [Y/n/q/?] n
The following actions will resolve these dependencies:

Install the following packages:
1) datastax-agent [5.1.2 (stable)]
2) dse [4.7.2-1 (stable)]
...
16) dse-pig [4.7.2-1 (stable)]

Accept this solution? [Y/n/q/?] Y
The following NEW packages will be installed:
  datastax-agent{a} dse{a} dse-demos{a} dse-full dse-hive{a} dse-
  libcassandra{a} dse-libhadoop{a} dse-libhadoop-native{a} dse-libhive{a} dse-
  liblog4j{a}
    dse-libmahout{a} dse-libpig{a} dse-libsolr{a} dse-libspark{a} dse-
  libsqoop{a} dse-libtomcat{a} dse-pig{a}
0 packages upgraded, 17 newly installed, 0 to remove and 309 not upgraded.
Need to get 554 MB of archives. After unpacking 672 MB will be used.
Do you want to continue? [Y/n/?] Y

```

apt-get

```
$ sudo apt-get install dse-full=version-1 dse=version-1 dse-hive=version-1
  dse-pig=version-1 dse-demos=version-1 dse-libsolr=version-1 dse-
  libtomcat=version-1 dse-libsqoop=version-1 dse-liblog4j=version-1 dse-
  libmahout=version-1 dse-libhadoop-native=version-1 dse-libcassandra=version-1
  dse-libhive=version-1 dse-libpig=version-1 dse-libhadoop=version-1 dse-
  libspark=version
```

Example:

```
$ sudo apt-get install dse-full=4.7.2-1 dse=4.7.2-1 dse-hive=4.7.2-1 dse-
  pig=4.7.2-1 dse-demos=4.7.2-1 dse-libsolr=4.7.2-1 dse-libtomcat=4.7.2-1 dse-
  libsqoop=4.7.2-1 dse-liblog4j=4.7.2-1 dse-libmahout=4.7.2-1 dse-libhadoop-
  native=4.7.2-1 dse-libcassandra=4.7.2-1 dse-libhive=4.7.2-1 dse-libpig=4.7.2-1
  dse-libhadoop=4.7.2-1 dse-libspark=4.7.2-1
```

For additional instructions, see the documentation version for your installation.

Installing glibc on Oracle Linux 6.x and later

To install DataStax Enterprise on Oracle Enterprise Linux 6.x and later, you need to install the 32-bit versions of the glibc libraries. They are not installed by default.

Procedure

1. Make the `yum.repos.d` your current directory.

```
$ cd /etc/yum.repos.d
```

2. Download the `public-yum-ol6.repo` package from the repository.

```
$ wget http://public-yum.oracle.com/public-yum-ol6.repo
```

3. Check that `glibc.i686` is ready for installation and install it.

```
$ yum list  
$ yum install glibc.i686
```

Uninstalling DataStax Enterprise

Select the uninstall method for your type of installation.

Uninstalling from the DataStax Installer

Use this method when you have installed DataStax Enterprise from the [DataStax Installer](#).

1. Go to the server installation directory (default is `/usr/share/dse`).
2. Launch the uninstaller:

- **Linux:** `$./uninstall ##` Run the uninstaller as root or sudo if needed
- **Mac OS X:** Double-click **uninstaller**.

3. Select the type of uninstall and follow the instructions on the uninstaller.

Note: If you are going to reinstall DataStax Enterprise with the existing data files, be sure to [drain](#) the node and move the files somewhere else before uninstalling.

Using the Unattended Uninstaller

To use this method, you must have installed DataStax Enterprise from the [DataStax Installer](#).

1. Create a configuration file called `uninstall.property` in the same directory as the uninstaller. For example:

```
/usr/share/dse/uninstall.property
```

2. In the `uninstall.property` file, set the required properties:

- `do_drain=1|0` - drains the node before uninstalling
- `full_uninstall=1|0` - uninstalls all components

where `1=yes` and `0=no`.

3. From the directory containing the uninstaller:

```
$ sudo ./uninstall --mode unattended
```

Uninstalling Debian- and RHEL-based packages

Use this method when you have installed DataStax Enterprise using [APT](#) or [Yum](#).

1. Stop the DataStax Enterprise and DataStax Agent services:

```
$ nodetool -h drain host_name  
$ sudo service dse stop  
$ sudo service datastax-agent stop
```

2. Make sure all services are stopped:

```
$ ps auwx | grep dse  
$ ps auwx | grep datastax-agent
```

3. If services are still running, use the PID to kill the service:

```
$ bin/dse cassandra-stop -p dse_pid
$ sudo kill datastax_agent_pid
```

4. Remove the installation directories:

RHEL-based packages:

```
$ sudo yum remove "dse-*" "datastax-*"
```

Debian-based packages:

```
$ sudo apt-get purge "dse-*" "datastax-*"
```

Uninstalling the binary tarball

Use this method when you have installed DataStax Enterprise using the [binary tarball](#).

1. Stop the node:

```
$ install_location/bin/dse cassandra-stop ## Use sudo if needed
```

2. Stop the DataStax Agent:

```
$ ps auwx | grep datastax-agent
$ kill datastax_agent_pid ## Use sudo if needed
```

3. Make sure all services are stopped:

```
$ ps auwx | grep dse
$ ps auwx | grep datastax-agent
```

4. If services are still running, use the PID to kill the service:

```
$ bin/dse cassandra-stop -p dse_pid
$ sudo kill datastax_agent_pid
```

5. Remove the installation directory.

Starting and stopping DataStax Enterprise

After you install and configure DataStax Enterprise on one or more nodes, start your cluster beginning with the seed nodes. In a mixed-workload DataStax Enterprise cluster, you must start the analytics seed node first.

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

Starting DataStax Enterprise as a service

Packaged installations provide start-up scripts in `/etc/init.d` for starting DataStax Enterprise as a service.

For Cassandra-only nodes in mixed-workload clusters or BYOH nodes, skip step 1.

The following entries set the type of node:

- `HADOOP_ENABLED=1` designates the node as a DSE Hadoop node and starts the Hadoop Job Tracker and Task Tracker services.

Starting and stopping DataStax Enterprise

- SOLR_ENABLED=1 starts the node as DSE Search node.
- SPARK_ENABLED=1 starts the node as DSE Enterprise Spark node.

Note: No entry is the same as disabling it.

Procedure

1. Edit the `/etc/default/dse` file, and then edit the appropriate line in this file, depending on the type of node you want:

Cassandra nodes

Transactional nodes.

```
HADOOP_ENABLED=0  
SOLR_ENABLED=0  
SPARK_ENABLED=0
```

BYOH nodes

A Cassandra node that runs DSE Analytics with a separate Hadoop cluster from a vendor other than DataStax.

```
HADOOP_ENABLED=0  
SOLR_ENABLED=0  
SPARK_ENABLED=0
```

DSE Hadoop nodes

Spark only and Spark + Integrated Hadoop ([DSE Hadoop](#)) nodes.

```
HADOOP_ENABLED=1  
SOLR_ENABLED=0  
SPARK_ENABLED=0
```

DSE Search nodes

DSE Search (DataStax Enterprise Search) simplifies using search applications for data that is stored in a Cassandra database.

```
SOLR_ENABLED=1  
HADOOP_ENABLED=0  
SPARK_ENABLED=0
```

Spark nodes

DSE Analytics with Spark. Spark is the default mode when you start an analytics node in a packaged installation.

```
SPARK_ENABLED=1  
HADOOP_ENABLED=0  
SOLR_ENABLED=0
```

SearchAnalytics nodes (experimental)

An integrated DSE SearchAnalytics cluster allows analytics jobs to be performed using [search queries](#).

```
SPARK_ENABLED=1  
SOLR_ENABLED=1  
HADOOP_ENABLED=0
```

2. Start DataStax Enterprise and the DataStax Agent:

```
$ sudo service dse start
$ sudo service datastax-agent start
```

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

```
$ nodetool status
```

On Enterprise Linux systems, the DataStax Enterprise service runs as a Java process.

Starting DataStax Enterprise as a stand-alone process

If running a mixed-workload cluster (one or more datacenters for each type of node), determine which nodes to start as analytics, Cassandra, and DSE 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 more information, see [Multiple datacenter deployment](#).

Attention: Do not start all the nodes at the same time, because this causes contention among nodes to become the Job Tracker.

Procedure

1. From the install directory:

Option	Description
Cassandra node	\$ bin/dse cassandra
BYOH node	\$ bin/dse cassandra
DSE Hadoop node	\$ bin/dse cassandra -t
DSE Search node	\$ bin/dse cassandra -s
Spark only node	\$ bin/dse cassandra -k ## Starts Spark trackers on a cluster of analytics nodes
Spark + DSE Hadoop	\$ bin/dse cassandra -k -t ## Starts a node in Spark and in Hadoop mode
SearchAnalytics node	\$ bin/dse cassandra -s -k ## Starts a node in SearchAnalytics mode

Attention: All nodes are DataStax Enterprise nodes and run the Cassandra database.

2. Start the DataStax agent:

```
$ ./datastax-agent/bin/datastax-agent
```

3. To check that your ring is up and running:

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

If you are running an analytics node, there are [several methods for designating the Job Tracker node](#).

Stopping a DataStax Enterprise node

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

To stop the DataStax Enterprise and DataStax Agent services on a node:

```
$ nodetool -h drain host_name
$ sudo service dse stop
$ sudo service datastax-agent stop
```

To stop the stand-alone process and DataStax Agent on a node:

Running `nodetool drain` before using the `cassandra-stop` command to stop a stand-alone process is not necessary because the `cassandra-stop` command drains the node before stopping it.

From the installation location:

```
$ install_location/bin/dse cassandra-stop ## Use sudo if needed
```

In the unlikely event that the `cassandra-stop` command fails because it cannot find the process DataStax Enterprise Java process ID (PID), the output instructs you to find the DataStax Enterprise Java process ID (PID) manually, and stop the process using its PID number.

```
$ ps auwx | grep dse
$ bin/dse cassandra-stop -p PID ## Use sudo if needed
```

To stop the DataStax Agent:

```
$ ps auwx | grep datastax-agent
$ kill datastax_agent_pid ## Use sudo if needed
```

DataStax Enterprise configuration

DataStax Enterprise can start as any type of node.

- Installer-Services installations: The node type is set during installation. You can change the `node type` in the `/etc/default/dse` file.
- Package installations: Set the `node type` in the `/etc/default/dse` file.
- Installer-No Services and Tarball installations: Set the `node type` with a flag on the startup command.

DataStax Enterprise configuration file (dse.yaml)

The `dse.yaml` file is the primary configuration file for DataStax Enterprise. The location of the `dse.yaml` file depends on the type of installation:

Installer-Services	<code>/etc/dse/dse.yaml</code>
Package installations	<code>/etc/dse/dse.yaml</code>

Installer-No Services	<i>install_location/resources/dse/conf/dse.yaml</i>
Tarball installations	<i>install_location/resources/dse/conf/dse.yaml</i>

For `cassandra.yaml` configuration, see [Node and cluster configuration \(cassandra.yaml\)](#).

DSE In-Memory options

```
max_memory_to_lock_fraction: 0.20
# max_memory_to_lock_mb: 10240
```

To use the [DSE In-Memory](#) on page 402, choose one of these options to specify how much system memory to use for all in-memory tables.

max_memory_to_lock_fraction

Specify a fraction of the system memory. The default value of 0.20 specifies to use up to 20% of system memory.

max_memory_to_lock_mb

Specify a maximum amount of memory in MB.

Hive meta store

```
hive_meta_store_enabled: true
```

hive_meta_store_enabled

Enables or disables the Hive meta store via Cassandra. Default: *true*.

Kerberos options

Use these options for configuring security for a DataStax Enterprise cluster using Kerberos. For instructions, see [Authenticating a cluster with Kerberos](#).

```
kerberos_options:
  keytab: path_to_keytab/dse.keytab
  service_principal: dse_user/_HOST@REALM
  http_principal: HTTP/_HOST@REALM
  qop: auth
```

- **keytab:** resources/dse/conf/dse.keytab

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

- **service_principal:** *dse_user/_HOST@REALM*

The service_principal that the Cassandra and Hadoop processes run under must use the form `dse_user/_HOST@REALM`, where `dse_user` is:

- Installer-Services and Package installations: `cassandra`
- Package installations: the name of the UNIX user that starts the service

where:

- `_HOST` is converted to a reverse DNS lookup of the broadcast address.
- `REALM` is the name of your Kerberos realm. In the Kerberos principal, `REALM` must be uppercase.

Set The service_principal must be consistent everywhere: in the `dse.yaml` file, present in the keytab, and in the `cqlshrc` file (where service_principal is separated into `service/hostname`).

- `http_principal: HTTP/_HOST@REALM`

The http_principal is used by the Tomcat application container to run DSE Search. The Tomcat web server uses GSS-API mechanism (SPNEGO) to negotiate the GSSAPI security mechanism (Kerberos). Set `REALM` to the name of your Kerberos realm. In the Kerberos principal, `REALM` must be uppercase.

- `qop - auth`

A comma-delimited list of Quality of Protection (QOP) values that clients and servers can use for each connection. The client can have multiple QOP values, while the server can have only a single QOP value. The valid values are:

- `auth` - Default: Authentication only.
- `auth-int` - Authentication plus integrity protection for all transmitted data.
- `auth-conf` - Authentication plus integrity protection and encryption of all transmitted data.

Encryption using auth-conf is separate and independent of whether encryption is done using SSL. If both auth-conf and SSL are enabled, the transmitted data is encrypted twice. DataStax recommends choosing only one method and using it for both encryption and authentication.

LDAP options

To use these LDAP options, you must set

`com.datastax.bdp.cassandra.auth.LdapAuthenticator` as the authenticator in the `cassandra.yaml` file. For instructions, see [Authenticating with LDAP](#).

```
ldap_options:
    server_host: localhost
    server_port: 389
    search_dn: cn=Admin
    search_password: secret
    use_ssl: false
    use_tls: false
    truststore_path:
    truststore_password:
    truststore_type: jks
    user_search_base: ou=users,dc=example,dc=com
    user_search_filter: (uid={0})
    credentials_validity_in_ms: 0
    connection_pool:
        max_active: 8
        max_idle: 8
```

server_host

The host name of the LDAP server. Default: `localhost`

server_port

The port on which the LDAP server listens. Default: `389`

search_dn

The username of the user that is used to search for other users on the LDAP server.

search_password

The password of the `search_dn` user.

use_ssl

Set to `true` to enable SSL connections to the LDAP server. If set to `true`, you might need to change `server_port` to the SSL port of the LDAP server. Default: `false`

use_tls

Set to `true` to enable TLS connections to the LDAP server. If set to `true`, you might need to change the `server_port` to the TLS port of the LDAP server. Default: `false`

truststore_path

The path to the trust store for SSL certificates.

truststore_password

The password to access the trust store.

truststore_type

The type of trust store. Default: `jks`

user_search_base

The search base for your domain, used to look up users. Set the `ou` and `dc` elements for your LDAP domain. Typically this is set to `ou=users,dc=domain,dc=top_level_domain`. For example, `ou=users,dc=example,dc=com`.

user_search_filter

The search filter for looking up user names. Default: `uid={0}`

credentials_validity_in_ms

The duration period in milliseconds for the credential cache. Default: `0`

search_validity_in_seconds

The duration period in milliseconds for the search cache. Default: `0`

connection_pool

- `max_active`

The maximum number of active connections to the LDAP server. Default: `8`

- `max_idle`

The maximum number of idle connections in the pool awaiting requests. Default: `8`

Scheduler settings for Solr indexes

These settings control the schedulers in charge of querying for and removing expired data.

```
ttl_index_rebuild_options:
  fixed_rate_period: 300
  initial_delay: 20
  max_docs_per_batch: 200
  thread_pool_size: 1
```

ttl_index_rebuild_options

The `ttl_index_rebuild_options` settings control the schedulers in charge of querying for and removing expired data.

fix_rate_period

Schedules how often to check for expired data in seconds. Default: `300`

initial_delay

Speeds start-up time by delaying the first TTL checks in seconds. Default: `20`

max_docs_per_batch

Sets the maximum number of documents to check and delete per batch by the TTL rebuild thread. Default: `200`

thread_pool_size

To manage system resource consumption and prevent many Solr cores from executing simultaneous TTL deletes, define the maximum number of cores that can execute TTL cleanup concurrently. Default: `1`

Solr shard transport options

The shard_transport_options use netty or http for inter-node communication between DSE Search nodes. Also see [Shard transport options for DSE Search communications](#) on page 240.

```
shard_transport_options:
  type: netty
  netty_server_port: 8984
  message_server_port: 8985
  netty_server_acceptor_threads:
  netty_server_worker_threads:
  netty_client_worker_threads:
  netty_client_max_connections:
  netty_client_request_timeout:
  netty_max_frame_length_in_mb:#

# Options specific to the "http" transport type.
#   http_shard_client_conn_timeout: 0
#   http_shard_client_socket_timeout: 0
```

type

- netty TCP-based communication provides lower latency, improved throughput, and reduced resource consumption.
- http uses a standard HTTP-based interface.

When type: netty, define the following netty settings to configure inter-node communication between DSE Search nodes:

netty_server_port (deprecated)

The TCP listen port. This setting is mandatory to use the netty transport now or migrate to it later. To use http transport, comment out this setting or change it to -1. During upgrade to 5.0, netty_server_port is used. After all nodes are running 5.0, message_server_port is used. Default: 8984

message_server_port

The TCP listen port for the inter-node message server. Requests that are coordinated by this node use this port to communicate with other nodes. Default: 8985

netty_server_acceptor_threads

The number of server acceptor threads. Default: *number_of_available_processors*

netty_server_worker_threads

The number of server worker threads. Default: *number_of_available_processors * 8*

netty_client_worker_thread

The number of client worker threads. Default: *number_of_available_processors * 8*

netty_client_max_connections

The maximum number of client connections. Default: 100

netty_client_request_timeout

The client request timeout, in milliseconds, for the maximum cumulative time that a distributed Solr request will wait idly for shard responses. Default: 60000

netty_max_frame_length_in_mb

The maximum length of a message frame, in MB. Default: 256

When type: http, define the following http transport settings to configure http inter-node communication between DSE Search nodes. To avoid blocking operations, DataStax strongly recommends changing these settings to a finite value. These settings are valid across Solr cores:

http_shard_client_conn_timeout

HTTP shard client timeouts in milliseconds. 0 = no timeout. Default: 0

http_shard_client_socket_timeout

HTTP shard client socket timeouts in milliseconds. 0 = no timeout. Default: 0

Solr indexing

DSE Search provides multi-threaded indexing implementation to improve performance on multi-core machines. All index updates are internally dispatched to a per-core indexing thread pool and executed asynchronously, which allows for greater concurrency and parallelism. However, index requests can return a response before the indexing operation is executed.

```
max_solr_concurrency_per_core: 2
# enable_back_pressure_adaptive_nrt_commit: true
# back_pressure_threshold_per_core: 1000
# flush_max_time_per_core: 5
# load_max_time_per_core: 5
```

max_solr_concurrency_per_core

Configures the maximum number of concurrent asynchronous indexing threads per [Solr core](#). If set to 1, DSE Search uses synchronous indexing behavior in a single thread. To achieve optimal performance, assign this value to number of available CPU cores divided by the number of Solr cores. For example, with 12 CPU cores and 3 Solr cores, the suggested value is 4. Also see [Configuring multi-threaded indexing threads](#) on page 267 and [Increasing indexing throughput](#). Default: *number_of_available_CPU_cores*

Note: Dynamic switching to Solr concurrency level at 1 is disallowed.

enable_back_pressure_adaptive_nrt_commit

Allows back pressure system to adapt max auto soft commit time (defined per core in the solrconfig.xml file) to the actual load. Setting is respected only for NRT (near real time) cores. When Solr core is using live indexing with RT (real time) enabled, adaptive commits are disabled regardless of this property value. Default: true

back_pressure_threshold_per_core

The total number of queued asynchronous indexing requests per Solr core. When this number is exceeded, [back pressure](#) prevents excessive resource consumption by throttling new incoming requests. DataStax recommends using the default back_pressure_threshold_per_core value of 1000 to set the threshold at 1000 * number of Solr cores. Default: 1000

flush_max_time_per_core

The maximum time, in minutes, to wait for the flushing of asynchronous index updates, which occurs at Solr commit time or at Cassandra flush time. Expert level knowledge is required to change this value. Always set the value reasonably high to ensure that flushing completes successfully. If the configured value is exceeded, index updates are only partially committed, and the Cassandra commit log is not truncated to ensure data durability.

Note: When a timeout occurs, it usually means this node is being overloaded and cannot flush in a timely manner. Live indexing increases the time to flush asynchronous index updates.

Default: 5

load_max_time_per_core

The maximum time, in minutes, to wait for each Solr core to load on startup or create/reload operations, expressed. This advanced option should be changed only if exceptions happen during core loading. Default: 1 (if not specified)

Cassandra disk failure policy

```
# enable_index_disk_failure_policy: false
```

enable_index_disk_failure_policy

DSE Search activates the configured Cassandra disk failure policy if IOExceptions occur during index update operations. Default: false

Solr CQL query options

Available options for CQL Solr queries.

```
solr_data_dir: /MyDir
cql_solr_query_executor_threads: 2
cql_solr_query_row_timeout: 10000
```

solr_data_dir

The directory to store index data. By default, the Solr data is saved in `cassandra_data_dir/solr.data`, or as specified by the `dse.solr.data.dir` system property.

cql_solr_query_executor_threads

The maximum number of threads for retrieving rows during CQL Solr queries. This value is cross-request and cross-core. Default: number of available processors * 10

cql_solr_query_row_timeout

The maximum time in milliseconds to wait for each row to be read from Cassandra during CQL Solr queries. Default: 10000 milliseconds (10 seconds)

CQL Performance Service options

These settings are used by the Performance Service to configure collection of performance metrics on Cassandra nodes. Performance metrics are stored in the `dse_perf` keyspace and can be queried with CQL using any CQL-based utility, such as [cqlsh](#), [DataStax DevCenter](#), or any application using a Cassandra CQL driver.

```
cql_slow_log_options:
  enabled: true
  threshold_ms: 2000
  ttl_seconds: 259200
cql_system_info_options:
  enabled: false
  refresh_rate_ms: 10000
resource_level_latency_tracking_options:
  enabled: false
  refresh_rate_ms: 10000
db_summary_stats_options:
  enabled: false
  refresh_rate_ms: 10000
cluster_summary_stats_options:
  enabled: false
  refresh_rate_ms: 10000
histogram_data_options:
  enabled: false
  refresh_rate_ms: 10000
  retention_count: 3
user_level_latency_tracking_options:
  enabled: false
  refresh_rate_ms: 10000
  top_stats_limit: 100
  quantiles: false
```

cql_slow_log_options

Report distributed sub-queries for Solr (query executions on individual shards) that take longer than a specified period of time. See [Collecting slow queries](#) on page 353.

enabled

Enables (true) or disables (false) log entries for slow queries. Default: true

threshold_ms

Defines the threshold time in milliseconds. Default: 2000

ttl_seconds

Defines the time, in milliseconds, to keep the slow query log entries. Default: 259200

cql_system_info_options

CQL system information tables settings See [Collecting system level diagnostics](#) on page 354.

enabled

Default: false

refresh_rate_ms

Default: 10000

resource_level_latency_tracking_options

Data resource latency tracking settings. See [Collecting system level diagnostics](#) on page 354.

enabled

Default: false

refresh_rate_ms

Default: 10000

db_summary_stats_options

Database summary statistics settings. See [Collecting database summary diagnostics](#) on page 356.

enabled

Default: false

refresh_rate_ms

Default: 10000

cluster_summary_stats_options

Cluster summary statistics settings. See [Collecting cluster summary diagnostics](#) on page 356.

enabled

Default: false

refresh_rate_ms

Default: 10000

histogram_data_options

Column Family Histogram data tables settings. See [Collecting table histogram diagnostics](#) on page 357.

enabled

Default: false

refresh_rate_ms

Default: 10000

retention_count

Default: 3

user_level_latency_tracking_options

User-resource latency tracking settings. See [Collecting user activity diagnostics](#) on page 358.

enabled

Default: false

refresh_rate_ms

Default: 10000

top_stats_limit

Default: 100

quantiles

Default: false

Spark Performance Service options

These settings are used by the Performance Service. See [Monitoring Spark with Spark Performance Objects](#).

```
spark_cluster_info_options:  
    enabled: false  
    refresh_rate_ms: 10000  
spark_application_info_options:  
    enabled: false  
    refresh_rate_ms: 10000  
    driver:  
        sink: false  
        connectorSource: false  
        jvmSource: false  
        stateSource: false  
    executor:  
        sink: false  
        connectorSource: false  
        jvmSource: false
```

spark_cluster_info_options

- **enabled**

Default: *false*

- **refresh_rate_ms**

Default: 10000 milliseconds

spark_application_info_options

Statistics options.

enabled

Default: false

refresh_rate_ms

Default: 10000 milliseconds

driver

The driver option controls the metrics collected by the Spark Driver.

sink

Enables or disables writing of the metrics that are collected at Spark Driver to the Cassandra database.

Default: false

connectorSource

Enables or disables Spark Cassandra Connector metrics at Spark Driver. Default: false

jvmSource

Enables or disables JVM heap and GC metrics at Spark Driver. Default: false

stateSource

Enables or disables application state metrics. Default: false

Solr Performance Service options

These settings are used by the Performance Service. See [Collecting Solr performance statistics](#) on page 361.

```

solr_indexing_error_log_options:
  enabled: false
  ttl_seconds: 604800
  async_writers: 1
solr_slow_sub_query_log_options:
  enabled: false
  ttl_seconds: 604800
  async_writers: 1
  threshold_ms: 100
solr_update_handler_metrics_options:
  enabled: false
  ttl_seconds: 604800
  refresh_rate_ms: 60000
solr_request_handler_metrics_options:
  enabled: false
  ttl_seconds: 604800
  refresh_rate_ms: 60000
solr_index_stats_options:
  enabled: false
  ttl_seconds: 604800
  refresh_rate_ms: 60000
solr_cache_stats_options:
  enabled: false
  ttl_seconds: 604800
  refresh_rate_ms: 60000
solr_latency_snapshot_options:
  enabled: false
  ttl_seconds: 604800
  refresh_rate_ms: 60000

```

solr_indexing_error_log_options

Enable to collect record errors that occur during Solr document indexing. See [Collecting indexing errors](#) on page 360.

enabled

Default: false

ttl_seconds

Default: 604800

async_writers

Defines the number of server threads dedicated to writing in the log. More than one server thread might degrade performance. Default: 1

solr_slow_sub_query_log_options

See [Collecting slow Solr queries](#) on page 359.

enabled

Default: false

ttl_seconds

Default: 604800

async_writers

Defines the number of server threads dedicated to writing in the log. More than one server thread might degrade performance. Default: 1

threshold_ms

Default: 100

solr_update_handler_metrics_options

See [Collecting handler statistics](#) on page 363.

enabled

Default: false

ttl_seconds

Default: 604800

refresh_rate_ms

Default: 60000

solr_index_stats_options

See [Collecting index statistics](#) on page 362.

enabled

Default: false

ttl_seconds

Default: 604800

refresh_rate_ms

Default: 60000

solr_cache_stats_options

See [Collecting cache statistics](#) on page 362.

enabled

Default: false

ttl_seconds

Default: 604800

refresh_rate_ms

Default: 60000

solr_latency_snapshot_options

See [Collecting Solr performance statistics](#) on page 361.

enabled

Default: false

ttl_seconds

Default: 604800

refresh_rate_ms

Default: 60000

Node health options

```
node_health_options:
    refresh_rate_ms: 60000
    uptime_ramp_up_period_seconds: 86400
    dropped_mutation_window_minutes: 30
```

node_health_options

Node health options are always enabled.

refresh_rate_ms

Default: 60000

uptime_ramp_up_period_seconds

The amount of continuous uptime required for the node's uptime score to advance the [node health score](#) from 0 to 1 (full health), assuming there are no recent dropped mutations. The health score is a composite score based on dropped mutations and uptime. Tip: If a node is repairing after a period of downtime, you might want to increase the uptime period to the expected repair time. Default: 86400 (1 day)

dropped_mutation_window_minutes

The historic time window over which the rate of dropped mutations affect the node health score. Default: 30

Health-based routing

```
enable_health_based_routing: true
```

enable_health_based_routing

Enable replication selection for distributed Solr queries to consider node health when multiple candidates exist for a particular token range. Health-based routing enables a trade-off between index consistency and query throughput. When the primary concern is performance, do not enable health-based routing. Default: true

Reindexing of bootstrapped data

```
async_bootstrap_reindex: false
```

async_bootstrap_reindex

For DSE Search, configure whether to asynchronously re-index bootstrapped data. Default: false

- If enabled, the node joins the ring immediately after bootstrap and re-indexing occurs asynchronously. Do not wait for post-bootstrap reindexing so that the node is not marked down.
- If disabled, the node joins the ring after re-indexing the bootstrapped data.

Encryption and system key settings

Settings for encrypting passwords and sensitive system tables.

```
system_key_directory: /etc/dse/conf
config_encryption_active: false
config_encryption_key_name: system_key
```

system_key_directory

The directory where global encryption keys, called system keys, are stored. Keys that are used for SSTable encryption must be distributed to all nodes. DataStax Enterprise must be able to read and write to this directory. This directory must have 700 permissions and belong to the dse user. Default: /etc/dse/conf

See [Configuring encryption using off-server encryption keys](#) on page 327 and [Configuring encryption using local encryption keys](#) on page 326.

config_encryption_active

When set to true (default: false), the following configuration values **must** be encrypted:

dse.yaml

```
ldap_options.search_password
ldap_options.truststore_password
```

cassandra.yaml

```
server_encryption_options.keystore_password
server_encryption_options.truststore_password
client_encryption_options.keystore_password
client_encryption_options.truststore_password
```

```
ldap_options.truststore_password
```

config_encryption_key_name

The name of the system key for encrypting and decrypting stored passwords in the configuration files. To encrypt keyfiles, use [dsetool createsystemkey](#). When config_encryption_active is *true*, you must provide a valid key with this name for the system_key_directory option. Default: system_key

System encryption settings

```
system_info_encryption:  
  enabled: false  
  cipher_algorithm: AES  
  secret_key_strength: 128  
  chunk_length_kb: 64  
  key_name: system_table_keytab
```

system_info_encryption

Enable to encrypt system tables that might contain sensitive information, such as system.hints, system.batchlog, system.paxos.

enabled

If enabled, system tables that contain sensitive information, such as system.hints, system.batchlog, system.paxos, and commit logs, are encrypted. When you enable system table encryption on a node with existing data, run [nodetool upgradesstable -a](#) on the listed tables. When tracing is enabled, sensitive information is written to the tables in the system_traces keyspace. Configure those tables to encrypt their data by using an encrypting compressor. Default: false

cipher_algorithm

Default: AES

secret_key_strength

Default: 128

chunk_length_kb

Default: 64

key_name

The name of the keys file that is created to encrypt system tables. This file is created in system_key_directory/system/key_name. Comment out when using key_provider: KmipKeyProviderFactory Default: system_table_keytab

key_provider

An alternate key provider for local encryption. Useful for using a KMIP host as a key provider. Default: KmipKeyProviderFactory

kmip_host

When key_provider: KmipKeyProviderFactory, the kmip_groupname that is defined for the [kmip_hosts](#) entry in dse.yaml that describes the KMIP key server or group of KMIP key servers.

DSE Analytics Hive options

```
hive_options:  
  insert_max_retries: 6  
  insert_retry_sleep_period: 50
```

hive_options

Retries setting when Hive inserts data to Cassandra table.

insert_max_retries

Maximum number of retries. Default: 6

insert_retry_sleep_period

Period of time in milliseconds between retries. Default: 50

Spark memory

```
initial_spark_worker_resources: 0.7
```

DataStax Enterprise can control the memory and cores offered by particular Spark Workers in semi-automatic fashion.

Use the initial_spark_worker_resources parameter to specify the fraction of system resources that are made available to the Spark Worker. The available resources are calculated in the following way:

- Spark Worker memory = initial_spark_worker_resources * (total system memory - memory assigned to Cassandra)
- Spark Worker cores = initial_spark_worker_resources * total system cores

The lowest values that you can assign to Spark Worker memory is 64 MB. The lowest value that you can assign to Spark Worker cores is 1 core. If the results are lower, no exception is thrown and the values are automatically limited. The range of the initial_spark_worker_resources value is 0.01 to 1. If the range is not specified, the default value 0.7 is used.

This mechanism is used by default to set the Spark Worker memory and cores. To override the default, uncomment and edit one or both SPARK_WORKER_MEMORY and SPARK_WORKER_CORES options in the spark-env.sh file.

Audit logging options

```
audit_logging_options:
  enabled: false
  logger: SLF4JAuditWriter
  retention_time: 0
```

audit_logging_options

To get the maximum information from data auditing, turn on data auditing on every node. See [Configuring and using data auditing](#) on page 334 and [Configuring audit logging to a logback log file](#) on page 336.

enabled

Default: false

logger

Default: SLF4JAuditWriterfalse

- SLF4JAuditWriter - Logs audit information to the SLF4JAuditWriter logger. Audit logging configuration settings are in the logback.xml file.
- CassandraAuditWriter - Logs audit information to a Cassandra table. This logger can be run synchronously or asynchronously. Audit logs are stored in the dse_audit.audit_log table. See related [cassandra_audit_writer_options](#) configuration entries and [Configuring audit logging to a Cassandra table](#) on page 339.

included_categories or excluded_categories

Specify either included or excluded categories. Specifying both is an error.

Comma separated list of audit event categories to include or exclude from the audit log. Categories are: QUERY, DML, DDL, DCL, AUTH, ADMIN.

included_categories: *comma_separated_list*

or

excluded_categories: *comma_separated_list*

included_keyspaces or excluded_keyspaces

Specify either included or excluded keyspaces. Specifying both is an error.

Use a regular expression to filter keyspaces, or use a comma separated list of keyspaces to be included or excluded from the audit log.

included_keyspaces: *comma_separated_list*

or

excluded_keyspaces: *comma_separated_list*

retention_time

The amount of time, in hours, that audit events are retained by supporting loggers. Only the CassandraAuditWriter supports retention time. Values of 0 or less retain events forever. Default: 0

cassandra_audit_writer_options

Configuration options for the CassandraAuditWriter.

```
cassandra_audit_writer_options:  
  mode: sync  
  batch_size: 50  
  flush_time: 500  
  num_writers: 10  
  queue_size: 10000  
  write_consistency: QUORUM
```

mode

Sets the mode the writer runs in. Default: sync

- sync - A query is not executed until the audit event is successfully written.
- async - Audit events are queued for writing to the audit table, but are not necessarily logged before the query executes. A pool of writer threads consumes the audit events from the queue, and writes them to the audit table in batch queries. While this substantially improves performance under load, if there is a failure between when a query is executed, and its audit event is written to the table, the audit table might be missing entries for queries that were executed.

batch_size

Available only when mode:async.

Must be greater than 0. The maximum number of events the writer will dequeue before writing them out to the table. If warnings in your logs reveal that batches are too large, decrease this value or increase the value of [batch_size_warn_threshold_in_kb](#) in `cassandra.yaml`. Default: 50

flush_time

Available only when mode:async.

The maximum amount of time in milliseconds before an event is removed from the queue by a writer before being written out. This flush time prevents events from waiting too long before being written to the table when there are not a lot of queries happening. Default: 500

num_writers

Available only when mode:async.

The number of worker threads asynchronously logging events to the CassandraAuditWriter. Default: 10

queue_size

queue_size: 10000

The size of the queue feeding the asynchronous audit log writer threads. When there are more events being produced than the writers can write out, the queue fills up, and newer queries block until there is space on the queue. If a value of 0 is used, the queue size is unbounded, which can lead to resource exhaustion under heavy query load. Default: 10000

write_consistency

`write_consistency: QUORUM`

The consistency level that is used to write audit events. Default: QUORUM

KMIP encryption options

Options for KMIP encryption keys and communication between the DataStax Enterprise node and the KMIP key server or key servers.

```
kmip_hosts:
  kmip_groupname:
    hosts: kmip1.yourdomain.com, kmip2.yourdomain.com
    keystore_path: path/to/kmip/keystore.jks
    keystore_type: jks
    keystore_password: password
    truststore_path: path/to/kmip/truststore.jks
    truststore_type: jks
    truststore_password: password
    key_cache_millis: 300000
    timeout: 1000
```

kmip_hosts

Configure options for a `kmip_groupname` section for each KMIP key server or group of KMIP key servers. Using separate key server configuration settings allows use of different key servers to encrypt table data, and eliminates the need to enter key server configuration information in DDL statements and other configurations.

kmip_groupname

A user-defined name for a group of options to configure a KMIP server or servers, key settings, and certificates.

- `hosts`

A comma-separated list of `hosts[:port]` for the KMIP key server. There is no load balancing. In failover scenarios, failover occurs in the same order that servers are listed. For example: `hosts: kmip1.yourdomain.com, kmip2.yourdomain.com`

- `keystore_path`

The path to a java keystore that identifies the DSE node to the KMIP key server. For example: `/path/to/keystore.jks`

- `keystore_type`

The type of key store. The default value is `jks`.

- `keystore_password`

The password to access the key store.

- `truststore_path`

The path to a java truststore that identifies the KMIP key server to the DataStax Enterprise node. For example: `/path/to/truststore.jks`

- `truststore_type`

The type of trust store.

- `truststore_password`

The password to access the trust store.

- `key_cache_milli`

Milliseconds to locally cache the encryption keys that are read from the KMIP hosts. The longer the encryption keys are cached, the fewer requests are made to the KMIP key server, but the longer it takes for changes, like revocation, to propagate to the DSE node. Default: 300000.

- timeout

Socket timeout in milliseconds. Default: 1000.

CQL Solr paging

Option to specify the paging behavior.

```
cql_solr_query_paging: off
```

cql_solr_query_paging

Default: off.

- off - Paging is off. Do not respect driver paging settings for [CQL Solr queries](#). To dynamically enable paging in the query, use the `paging:driver` parameter in [JSON queries](#).
- driver - Respects driver paging settings. Specifies to use [Solr pagination \(cursors\)](#) only when the driver uses pagination. Required for DSE SearchAnalytics workloads when analytics nodes leverage search.

Configuring and using virtual nodes (vnodes)

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

Attention: DataStax Enterprise turns off virtual nodes (vnodes) by default. DataStax does not recommend turning on vnodes for DSE Hadoop or BYOH nodes. Before turning vnodes on for Hadoop, understand the [implications](#). DataStax Enterprise does support turning on vnodes for Spark nodes.

Guidelines for using virtual nodes

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

- Determining the `num_tokens` value:
The recommended initial value for `num_tokens` is 256. For more guidance, see [Setting up virtual nodes](#).
- Migrating existing clusters:
To upgrade existing clusters to vnodes, see [Enabling virtual nodes on an existing production cluster](#).
- Using vnodes in a mixed architecture deployment:
Cassandra supports using virtual node-enabled and non-virtual node datacenters. For example, a single cluster with:
 - A cassandra-only datacenter running OLTP.
 - A analytics datacenter without vnodes
 - A search datacenter with vnodes.

Disabling vnodes

Important: If you do not use vnodes, you must make sure that each node is responsible for roughly an equal amount of data. To ensure that each node is responsible for an equal amount of data, assign each node an `initial-token` value and calculate the tokens for each datacenter as described in [Generating tokens](#).

You can also use the default Murmur3Partitioner and calculate the tokens as described in [Generating tokens](#).

Procedure

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

```
num_tokens: 1
```

2. Uncomment the `initial_token` property and set it to 1 or to the value of a [generated token](#) for a multi-node cluster.

Default file locations for Installer-Services and package installations

The default location of the files depend on how DataStax Enterprise is installed. The DataStax All-in-One Installer installs files differently depending on whether Services or No Services option is selected during installation. When Services is selected, the files are located in the same locations as package installations. When No Services are selected, the files are located in the same locations as the tarball installations.

Directories for `cassandra.yaml` and `dse.yaml`

Directories	Description
<code>/etc/dse/cassandra</code>	cassandra.yaml is the main configuration file for Cassandra.
<code>/etc/dse</code>	dse.yaml is the main configuration file for DataStax Enterprise.

BYOH directories

Directories	Description
<code>/etc/dse</code>	byoh-env.sh is the BYOH configuration file to: <ul style="list-style-type: none"> • Set up the DataStax Enterprise environment • Define the BYOH environment variables

Cassandra directories

Directories	Description
<code>/var/lib/cassandra</code>	commitlog, data, saved_caches
<code>/var/log/cassandra</code>	audit.log, output.log, solrvalidation.log, system.log
<code>/var/run/cassandra</code>	
<code>/usr/share/dse/cassandra</code>	Cassandra environment settings
<code>/usr/share/dse/cassandra/lib</code>	
<code>/usr/bin</code>	
<code>/usr/sbin</code>	
<code>/etc/dse/cassandra/</code>	Cassandra configuration
<code>/etc/init.d</code>	

DataStax Enterprise configuration

Directories	Description
/etc/security/limits.d	
/etc/default/	
/usr/share/doc/dse-libcassandra	Notices and cqlshrc samples, including sample .cqlshrc file with Kerberos configuration

DataStax Enterprise Installer and log file directories

Directories	Description
/usr/share/dse/backups/ <i>log_file_dir/</i> copied_config_files.log	Show Config File Overwrites
/usr/share/dse/backups/ <i>log_file_dir/</i> bitrock_installer.log	View Installation Log
/usr/share/dse/backups/ <i>log_file_dir/</i> install_dependencies.log	View Dependency Installation Log
/usr/share/dse/backups/pfc_results.txt	View Configuration Recommendations and Warnings (Preflight Check Results)
/usr/share/dse	View README
/usr/share/dse	Uninstall DataStax Enterprise

DSE Hadoop directories

Directories	Description
/etc/dse/hadoop	Hadoop configuration
/usr/share/dse/resources/hadoop	Hadoop environment settings
/usr/share/portfolio_manager	Hadoop Portfolio Manager demo

Hive directories

Directories	Description
/etc/dse/hive	Hive configuration
/usr/share/dse/resources/hive	Hive environment settings

Mahout directories

Directories	Description
/etc/dse/mahout	Mahout properties
/usr/share/dse/resources/mahout	Mahout environment settings
/usr/share/demos/mahout	Mahout demo

Pig directories

Directories	Description
/etc/dse/pig	Pig configuration
/usr/share/dse/resources/pig	Pig environment settings
/usr/share/demos/pig	Pig demo

DSE Search directories

Directories	Description
/usr/share/dse/resources/solr/conf	Solr configuration
/usr/share/dse/solr	Solr driver
/usr/share/dse/demos/wikipedia	Search - Wikipedia demo

Spark directories

Directories	Description
/etc/dse/spark	<code>spark-env.sh</code> is the Spark configuration file
/usr/share/dse/spark/work	Spark work directory
/usr/share/dse/spark/logs	Spark Master and Worker logs
/usr/share/dse/demos/spark	Spark Portfolio Manager demo

Sqoop directories

Directories	Description
/etc/dse/sqoop	Sqoop configuration
/usr/share/dse/resources/sqoop	Sqoop environment settings
/usr/share/dse/demos/sqoop	Sqoop demo

Logback-appender directories

Directories	Description
/etc/dse/cassandra	<code>logback.xml</code> is the logback configuration file

Tomcat server logs for DSE Search

Directories	Description
/var/log/tomcat	Default log location. You can change the location of the Tomcat server logs for DSE Search.

OpsCenter directories

Directories	Description
/var/lib/opscenter	SSL certificates for encrypted agent/dashboard communications

Directories	Description
/var/log/opscenter	Log directory
/var/run/opscenter	Runtime
/usr/share/opscenter	JAR, agent, web application, and binary files
/etc/opscenter	<code>opscenterd.conf</code> is the OpsCenter configuration file
/etc	<code>init.d</code> contains the service startup script
/etc/security	<code>limits.d</code> sets OpsCenter user limits

DataStax Agent directories

Directories	Description
/var/lib/datastax-agent/ssl	SSL certificates for encrypted agent and dashboard communications
/var/lib/datastax-agent/conf	Configuration
/var/log/datastax-agent	Log directory
/var/run/datastax-agent	Runtime
/usr/share/datastax-agent	JAR, agent, web application, and binary files
/etc/init.d	Service startup script

Default file locations for Installer-No Services and tarball installations

The default location of the files depend on how DataStax Enterprise is installed. The DataStax All-in-One Installer installs files differently depending on whether Services or No Services option is selected during installation. When Services is selected, the files are located in the same locations as package installations. When No Services are selected, the files are located in the same locations as the tarball installations.

Note: The default `install_location` depends on whether you installed DataStax Enterprise by using the DataStax All-in-One Installer or from the binary tarball:

- **Installer-No Services:** /usr/share/dse
- **Tarball installation:** The `location` where you extracted DataStax Enterprise.

Directories for `cassandra.yaml` and `dse.yaml`

Directories	Description
<code>install_location/resources/cassandra/conf</code>	<code>cassandra.yaml</code> is the main configuration file for Cassandra.
<code>install_location/resources/dse/conf</code>	<code>dse.yaml</code> is the main configuration file for DataStax Enterprise.

BYOH directories

Directories	Description
<i>install_location/bin</i>	byoh-env.sh is the BYOH configuration file to: <ul style="list-style-type: none"> • Set up the DataStax Enterprise environment • Define the BYOH environment variables
<i>install_location/resources/byoh/conf</i>	BYOH configuration

Cassandra directories

Directories	Description
<i>install_location/resources/cassandra/bin</i>	Cassandra commands and utilities, such as nodetool, cqlsh, sstablekeys, and sstableloader
<i>install_location/resources/cassandra/conf</i>	Notices and cqlshrc samples, including sample .cqlshrc file with Kerberos configuration

DataStax Enterprise Installer and log file directories

Directories	Description
<i>install_location/dse/backups/log_file_dir/copied_config_files.log</i>	Show Config File Overwrites
<i>install_location/dse/backups/log_file_dir/bitrock_installer.log</i>	View Installation Log
<i>install_location/dse/backups/log_file_dir/install_dependencies.log</i>	View Dependency Installation Log
<i>install_location/dse/backups/log_file_dir/pfc_results.txt</i>	View Configuration Recommendations and Warnings (Preflight Check Results)
<i>install_location/dse/README.md</i>	View README
<i>install_location/dse/uninstall</i>	Uninstall DataStax Enterprise

DSE Hadoop directories

Directories	Description
<i>install_location/resources/hadoop/conf</i>	Hadoop configuration
<i>install_location/demos/portfolio_manager</i>	Hadoop Portfolio Manager demo

Hive directories

Directories	Description
<i>install_location/resources/hive/conf</i>	Hive configuration

Mahout directories

Directories	Description
<code>install_location/resources/mahout/conf</code>	Mahout properties
<code>install_location/demos/mahout</code>	Mahout demo

Pig directories

Directories	Description
<code>install_location/resources/pig/conf</code>	Pig configuration
<code>install_location/demos/pig</code>	Pig demo

DSE Search directories

Directories	Description
<code>install_location/resources/solr/conf</code>	Solr configuration
<code>install_location/resources/dse/lib</code>	Solr driver
<code>install_location/demos/wikipedia</code>	Search - Wikipedia demo

Spark directories

Directories	Description
<code>install_location/resources/spark/conf</code>	spark-env.sh is the Spark configuration file
<code>install_location/resources/spark/work</code>	Spark work directory
<code>install_location/resources/spark/logs</code>	Spark Master and Worker logs
<code>install_location/demos/spark</code>	Spark Portfolio Manager demo

Sqoop directories

Directories	Description
<code>install_location/resources/sqoop/conf</code>	Sqoop configuration
<code>install_location/demos/sqoop</code>	Sqoop demo

Logback-appender directories

Directories	Description
<code>install_location/resources</code>	logback.xml is the logback configuration file

Tomcat server logs for DSE Search

Directories	Description
<code>/var/log/tomcat</code>	Default log location. You can change the location of the Tomcat server logs for DSE Search .

OpsCenter directories

Directories	Description
<code>install_location/opscenter/agent</code>	Agent installation
<code>install_location/opscenter/bin</code>	Startup and configuration
<code>install_location/opscenter/content</code>	Web application
<code>install_location/opscenter/conf</code>	Configuration
<code>install_location/opscenter/doc</code>	License
<code>install_location/opscenter/lib and /src</code>	Library
<code>install_location/opscenter/log</code>	OpsCenter log
<code>install_location/opscenter/ssl</code>	SSL files for OpsCenter to agent communications

DataStax Agent directories

Directories	Description
<code>install_location/datastax-agent/ssl</code>	SSL certificates for encrypted agent and dashboard communications

Configuring the Tomcat log location

The default location of the Tomcat server logs for DSE Search is `/var/log/tomcat`.

Procedure

To change this location, edit one of these files:

- Set the `TOMCAT_LOGS` environment variable in the `dse.in.sh` file.
- Set the locations in `resources/tomcat/conf/logging.properties`.

Collecting node health and indexing status scores

Node health options are always enabled for all nodes. Node health is a score-based representation of how fit a node is to handle search queries.

The node health composite score is based on dropped mutations and uptime. A dynamic health score between 0 and 1 describes the health of the specified DataStax Enterprise node:

- A higher score indicates better node health. 1 is the highest score.
- A lower score applies to nodes that have a large number of dropped mutations and nodes that are just started.

On DSE Search nodes, the [shard selection](#) algorithm uses account proximity and secondary factors such as active and indexing statuses.

You can examine node health scores and indexing status. The indexing status is INDEXING, FINISHED, or FAILED.

Replication selection for distributed Solr queries can be configured to consider node health when multiple candidates exist for a particular token range. This health-based routing enables a trade-off between index consistency and query throughput. When the primary concern is performance, do not enable health-based routing.

Procedure

1. In the dse.yaml file:

- Customize node health options to increase the node health score from 0 to 1 (full health):

```
node_health_options:  
  refresh_rate_ms: 60000  
  uptime_ramp_up_period_seconds: 86400  
  dropped_mutation_window_minutes: 30
```

node_health_options

Node health options are always enabled.

refresh_rate_ms

Default: 60000

uptime_ramp_up_period_seconds

The amount of continuous uptime required for the node's uptime score to advance the [node health score](#) from 0 to 1 (full health), assuming there are no recent dropped mutations. The health score is a composite score based on dropped mutations and uptime. Tip: If a node is repairing after a period of downtime, you might want to increase the uptime period to the expected repair time. Default: 86400 (1 day)

dropped_mutation_window_minutes

The historic time window over which the rate of dropped mutations affect the node health score. Default: 30

Tip: If a node is repairing after a period of downtime, you might want to increase the `uptime_ramp_up_period_seconds` value to the expected repair time.

- To enable [replication selection](#) for distributed Solr queries to consider node health, enable health-based routing:

```
enable_health_based_routing: true
```

Health-based routing enables a trade-off between index consistency and query throughput. When the primary concern is performance, do not enable health-based routing.

2. To retrieve a dynamic health score between 0 and 1 that describes the specified DataStax Enterprise node, use the [dsetool node_health](#) command.

For example:

```
$ dsetool -h 200.192.10.11 node_health  
Node Health: 0.7
```

If you do not specify the IP address, the default is the local DataStax Enterprise node.

3. To retrieve the dynamic indexing status (INDEXING, FINISHED, or FAILED) of the specified core in a DataStax Enterprise node, use the [dsetool core_indexing_status](#) command.

For example:

```
$ dsetool -h 200.192.10.11 core_indexing_status wiki.solr  
wiki.solr: INDEXING
```

DSE Analytics

DataStax Enterprise analytics includes integration with Apache Spark, BYOH (bring your own Hadoop), and DSE Hadoop.

About DSE Analytics

Use DSE Analytics to analyze huge databases. DSE Analytics includes integration with Apache Spark, BYOH (bring your own Hadoop), and DSE Hadoop:

Apache Spark

A fast alternative to Hadoop. Spark is a distributed, parallel, batch data processing engine based on the Resilient Distributed Datasets (RDD) concept.

BYOH

A [bring your own Hadoop](#) (BYOH) model gives organizations who are already running Hadoop that is implemented by Cloudera or Hortonworks a way to use these implementations with DataStax Enterprise. This model provides better performance through custom, better-tuned Hadoop than earlier DataStax Enterprise versions.

Improved integration of Apache Sqoop

You can import RDBMS data to Cassandra and export Cassandra CQL data to an RDBMS.

DSE Hadoop

Hadoop is integrated with DataStax Enterprise with the following Hive and Pig tools:

- Support for the [native protocol in Hive](#).
- [Auto-creation](#) of Hive databases and external tables for each CQL keyspace and table.
- A `cql3.partition.key` property that maps Hive tables to CQL compound primary keys and composite partition keys.
- Support for [HiveServer2](#).
- Integration of the [Beeline command shell](#).
- Support for expiring data in columns by setting [TTL](#) (time to live) on Hive tables.
- Support for expiring data by setting TTL on Pig data using the `cql://` URL, which includes a prepared statement. See [step 10 of the library demo](#).

DSE Analytics features

No single point of failure

DSE Hadoop supports a peer-to-peer, distributed cluster for running MapReduce jobs. Being peers, any node in the cluster can load data files, and any analytics node can assume the responsibilities of Job Tracker for MapReduce jobs.

Job Tracker management

DSE Hadoop can automatically select Job Tracker and reserve Job Tracker nodes that take over in the event of a problem that would affect availability. The Job Tracker and reserve Job Tracker nodes can also be explicitly set.

Multiple Job Trackers

You can run one or more Job Tracker services across multiple datacenters and create multiple keyspaces per datacenter. Using this capability has performance, data replication, and other benefits.

Hadoop MapReduce with multiple Cassandra File Systems (CFS)

Cassandra File System (CFS) is a Hadoop Distributed File System (HDFS)-compatible storage layer. DataStax Enterprise replaces HDFS with CFS to run MapReduce jobs on Cassandra's peer-to-peer, fault-tolerant, and scalable architecture. You can create additional CFS to organize and optimize data.

Analytics without ETL

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

Hive support

Hive, a data warehouse system, facilitates data summarization, ad hoc queries, and the analysis of large data sets that are stored in Hadoop-compatible file systems. Any ODBC or JDBC compliant user interface connects to Hive from the server. Using the Cassandra-enabled Hive MapReduce client in DataStax Enterprise, you project a relational structure onto Hadoop data in CFS, and query the data using CQL, an SQL-like language.

Pig support

The Cassandra-enabled Pig MapReduce client that is included with DSE Hadoop is a high-level platform for creating MapReduce programs to use with Hadoop. You can analyze large data sets by running jobs in MapReduce mode and Pig programs directly on data that is stored in Cassandra.

Mahout support

Apache Mahout, included with DSE Hadoop, offers machine learning libraries. Machine learning improves a system, such as the system that recreates the Google Priority Inbox, based on past experience or examples.

DSE Analytics and Search integration (experimental)

An integrated DSE SearchAnalytics cluster allows analytics jobs to be performed using [search queries](#). This integration allows finer-grained control over the types of queries that are used in analytics workloads, and improves performance by reducing the amount of data that is processed.

Nodes started in SearchAnalytics mode allow you to create analytics queries that use DSE Search indexes. These queries return RDDs that are used by Spark jobs to analyze the returned data.

The following code shows how to use a DSE Search query from the DSE Spark console.

```
val table = sc.cassandraTable("music", "solr")
val result =
  table.select("id", "artist_name").where("solr_query='artist_name:Miles*'").collect
```

For a detailed example, see [Running the Wikipedia demo with SearchAnalytics](#) on page 123.

Planning and configuring a DSE SearchAnalytics cluster

1. Create DSE SearchAnalytics clusters as new clusters in a datacenter, as described in [Single datacenter deployment per workload type](#) on page 412.

The name of the datacenter is set to `SearchAnalytics` when using the `DseSimpleSnitch`. Do not modify existing search or analytics nodes to be `SearchAnalytics` nodes.

2. Set the `cql_solr_query_paging: driver` option in the `dse.yaml` file. For SearchAnalytics nodes, you must use the `cql_solr_query_paging: driver` setting to make Solr queries from Spark.
3. Perform load testing to ensure your hardware has enough CPU and memory for the additional resource overhead that is required by Spark and Solr.

SearchAnalytics nodes might consume more resources than search or analytics nodes. Resource requirements of the nodes greatly depend on the type of query patterns you are using.

Limitations of DSE SearchAnalytics clusters

While you will be able to query Solr indexes from Spark in a SearchAnalytics datacenter, you will get none of the benefits of workload isolation in that datacenter. DataStax recommends that you do not run real-time Solr queries against SearchAnalytics nodes if Spark jobs are being run on them.

SearchAnalytics clusters are considered experimental, and should not be run in production environments.

About the Cassandra File System (CFS)

A Hive or Pig analytics job requires a Hadoop file system to function. DataStax Enterprise provides a replacement for the Hadoop Distributed File System (HDFS) called the Cassandra File System (CFS). When an analytics node starts up, DataStax Enterprise creates a default CFS rooted at `cfs:/` and an archive file system named `cfs-archive`, which is rooted at `cfs-archive:/`. Cassandra creates a keyspace for the `cfs-archive` file system, and every other CFS file system. The keyspace name is similar to the file system name except the hyphen in the name is replaced by an underscore. For example, the `cfs-archive` file system keyspace is `cfs_archive`. You need to [increase the replication factor](#) of default CFS keyspaces to prevent problems when running Hadoop jobs.

Configuring a CFS superuser

A CFS superuser is the DataStax Enterprise daemon user, the user who starts DataStax Enterprise. A `cassandra` superuser, set up using the CQL [CREATE USER command](#), is also a CFS superuser.

A CFS superuser can modify files in the CFS without any restrictions. Files that a superuser adds to the CFS are password-protected.

Deleting files from the CFS

Cassandra does not immediately [remove deleted data](#) from disk when you use the `dse hadoop fs -rm file` command. Instead, Cassandra treats the deleted data like any data that is deleted from Cassandra. A tombstone is written to indicate the new data status. Data that is marked with a tombstone exists for a configured time period (defined by the `gc_grace_seconds` value that is set on the table). When the grace period expires, the [compaction process](#) permanently deletes the data. You do not have to manually remove expired data.

Checkpointing with the CFS

DataStax Enterprise supports checkpoints into CFS for Spark Streaming jobs that use the Kafka direct API. CFS is used to save the Kafka Direct API offsets for fault tolerance of Spark Streaming jobs. CFS is not suitable for Write Ahead Logging (WAL) or other Spark Streaming checkpointing operations, only the Kafka Direct API.

Managing the CFS consistency level

The default read and write consistency level for CFS is LOCAL_QUORUM or QUORUM, depending on the keyspace replication strategy, SimpleStrategy or NetworkTopologyStrategy, respectively. You can change the consistency level by specifying a value for `dse.consistencylevel.read` and `dse.consistencylevel.write` properties in the `core-site.xml` file.

Using multiple Cassandra File Systems

You can use more than one CFS. Typical reasons for using an additional CFS are:

- To isolate Hadoop-related jobs
- To configure keyspace replication by job

- To segregate file systems in different physical datacenters
- To separate Hadoop data in some other way

Procedure

To create an additional CFS:

1. Open the core-site.xml file for editing.
2. Add one or more property elements to `core-site.xml` using this format:

```
<property>
  <name>fs.cfs-file_system_name.impl</name>
  <value>com.datastax.bdp.hadoop.cfs.CassandraFileSystem</value>
</property>
```

With multiple CFS, you must override the default file system name for the newly created CFS to avoid conflicts with existing CFS on other datacenters. Each datacenter requires a unique default file system. For example, instead of the default value `cfs://127.0.0.1/`, specify a unique file system name for the new CFS, like `cfs-myfs://127.0.0.1/`:

```
<property>
  <name>fs.cfs-myfs.impl</name>
  <value>com.datastax.bdp.hadoop.cfs.CassandraFileSystem</value>
</property>
<property>
  <name>fs.default.name</name>
  <value>cfs-myfs://127.0.0.1/</value>
</property>
```

3. Save the file and restart Cassandra.

DataStax Enterprise creates the new CFS.

4. To access the new CFS, construct a URL using the following format:

```
cfs-file_system_name:path
```

For example, assuming the new file system name is `NewCassandraFS` use the [dse commands](#) to put data on the new CFS.

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

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

Configuring DSE Analytics

Guidelines and steps to configure all DSE Analytics nodes.

Setting the replication factor

The Cassandra File System (CFS) is a Hadoop Distributed File System (HDFS)-compatible storage layer. DataStax Enterprise replaces HDFS with CFS to run MapReduce jobs on Cassandra's peer-to-peer, fault-tolerant, and scalable architecture. CFS is a fundamental piece of infrastructure for all DSE Analytics nodes. For CFS, the three keyspaces are:

- cfs
- cfs_archive
- HiveMetaStore

The default replication factor for the HiveMetaStore, cfs, and cfs_archive system keyspaces is 1.

- A replication factor of 1 is suitable only for development and testing of a single node, but not for a production environment.
- For production clusters, [increase the replication factor](#) to at least 3.

The number of nodes in the cluster determines the replication factor, as discussed in [Choosing keyspace replication options](#). To change the replication factors of these keyspaces:

Procedure

1. Change the replication factor of the cfs and cfs_archive keyspaces from 1 to 3, for example:

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

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

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

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

3. Run [nodetool repair](#) to avoid having missing data problems or data unavailable exceptions.

What to do next

Ensure that you appropriately [configure replication](#) for your environment.

Job Trackers for DSE Hadoop and external Hadoop

Job Trackers are used for analytics nodes that analyze data using Hadoop, including DSE Hadoop and external Hadoop systems.

For each MapReduce job that is submitted to the Job Tracker, DataStax Enterprise schedules a series of tasks on the analytics nodes. One Task Tracker service per node handles the map and reduce tasks that are scheduled for that node. Within a datacenter, the Job Tracker monitors the execution and status of distributed tasks that comprise a MapReduce job.

Note: If the Job Tracker is not manually set after you enable the [automatic Job Tracker](#) setting using the dsetool utility, the Job Tracker is reassigned when the reserve tracker is set.

Using multiple Job Tracker services

You can use multiple Job Tracker nodes in a cluster, one per datacenter. In deployments that have multiple datacenters far away from each other, using multiple Job Trackers and [multiple file systems](#) can improve performance by taking advantage of data locality on each cluster.

Setting the Job Tracker node

There are several ways to set the Job Tracker node for all DSE Analytics nodes.

- You can use the `dsetool setjt` command to explicitly set the Job Tracker nodes.
- You can use the `dsetool autojt` to have DataStax enterprise automatically select Job Trackers.

Hive clients automatically select the correct Job Tracker node on startup. You configure and manage the Job Tracker node for an analytics node using [dsetool commands](#).

About the reserve Job Tracker

DataStax Enterprise nominates a node in the cluster as a reserve Job Tracker for a data center. The reserve Job Tracker becomes the Job Tracker when, for some reason, there is no local node in the datacenter that can function as Job Tracker.

The `dsetool setrjt` command sets the reserve Job Tracker.

Managing the Job Tracker

Several dsetool and dse client-tool commands are useful for managing Job Tracker nodes:

- `dsetool autojt datacenter`
If you do not specify the datacenter name, the command elects Job Trackers for all datacenters. Automatically manage Job Tracker selection and remove manual selections. If the current manually selected tracker is up, the manually selected Job Tracker continues to be used.
- `dsetool jobtracker`
Replaced by the [dse client-tool hadoop job-tracker-address](#) command.
- `dsetool setjt node IP`
Moves the Job Tracker, or the [Spark Master](#), and notifies the Task Tracker nodes of the change.
- `dsetool setrjt node IP`
Moves the reserve Job Tracker and notifies the Task Tracker nodes of the change.
- `dsetool sparkmaster`
For [SparkMaster](#) only. Returns the Job Tracker hostname and port to your location in the datacenter where you issued the command.
- `dsetool listjt`
Lists all Job Tracker nodes grouped by their local datacenter.
- `dsetool ring`
Lists the nodes and types of the nodes in the ring and the following Job Tracker status:
 - (JT) the active Job Tracker
 - (PT) an inactive primary tracker, when the primary tracker is down
 - (RT) an inactive reserve tracker, when the reserve tracker is up while there is a primary tracker

These `dsetool` commands also work for [Spark Master](#), except for `dsetool jobtracker`. The Spark Master equivalent command is `dsetool sparkmaster`.

For `dsetool` commands and options, see [dsetool](#).

Listing Job Trackers example

To determine which nodes in your DataStax Enterprise cluster are Job Tracker nodes, run the following command:

- **Installer-Services and Package installations:**
- ```
$ dsetool jobtracker
```
- **Installer-No Services and Tarball installations:**

```
$ install_location/bin/dsetool jobtracker
```

### Moving the Job Tracker node example

If your primary Job Tracker node fails, move the Job Tracker to another analytics node in the cluster. In-progress MapReduce jobs fail when you move the Job Tracker node or when the node goes down.

### Procedure

1. Log in to a DataStax Enterprise analytics node.
2. Run the `dsetool setjt` command and specify the IP address of the new Job Tracker node in your DataStax Enterprise cluster. For example, to move the Job Tracker to node 110.82.155.4:
  - **Installer-Services and Package installations:**

```
$ dsetool setjt 110.82.155.4
```

- **Installer-No Services and Tarball installations:**

```
$ install_location/bin/dsetool setjt 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 node 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, restart them to pick up the new Job Tracker node information.

### Changing the Job Tracker client port

By default, the Job Tracker listens on port 8012 for client messages. To use a port other than the default port 8012, configure the `mapred.job.tracker` property.

### Procedure

1. Open the `mapred-site.xml` file for editing.
2. Locate the `mapred.job.tracker` property:

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

3. In the `mapred.job.tracker` property, change the placeholder  `${dse.job.tracker}` value to the port number that you want to use. For example, change the port number from the default to 8013:

```
<!-- Auto detect the dse job tracker -->
<property>
 <name>mapred.job.tracker</name>
 <value>8013</value>
 <description>
```

```
The address of the job tracker
</description>
```

## Analyzing data using Spark

Spark is the default mode when you start an analytics node in a packaged installation.

### About Spark

Spark is the default mode when you start an analytics node in a packaged installation. Spark runs locally on each node and executes in memory when possible. Spark uses multiple threads instead of multiple processes to achieve parallelism on a single node, avoiding the memory overhead of several JVMs.

[Apache Spark](#) integration with DataStax Enterprise includes:

- [Spark streaming](#)
- [Spark Java API support](#)
- [DataFrames API](#) to manipulate data within Spark
- [Spark SQL support](#)

Spark runs locally on each node and executes in memory when possible. Spark can cache intermediate RDDs in RAM, disk, or both. Spark stores files for chained iteration in memory, instead of using temporary storage in HDFS like Hadoop does. Spark uses multiple threads instead of multiple processes to achieve parallelism on a single node, avoiding the memory overhead of several JVMs.

### Spark architecture

The software components for a single DataStax Enterprise analytics node are:

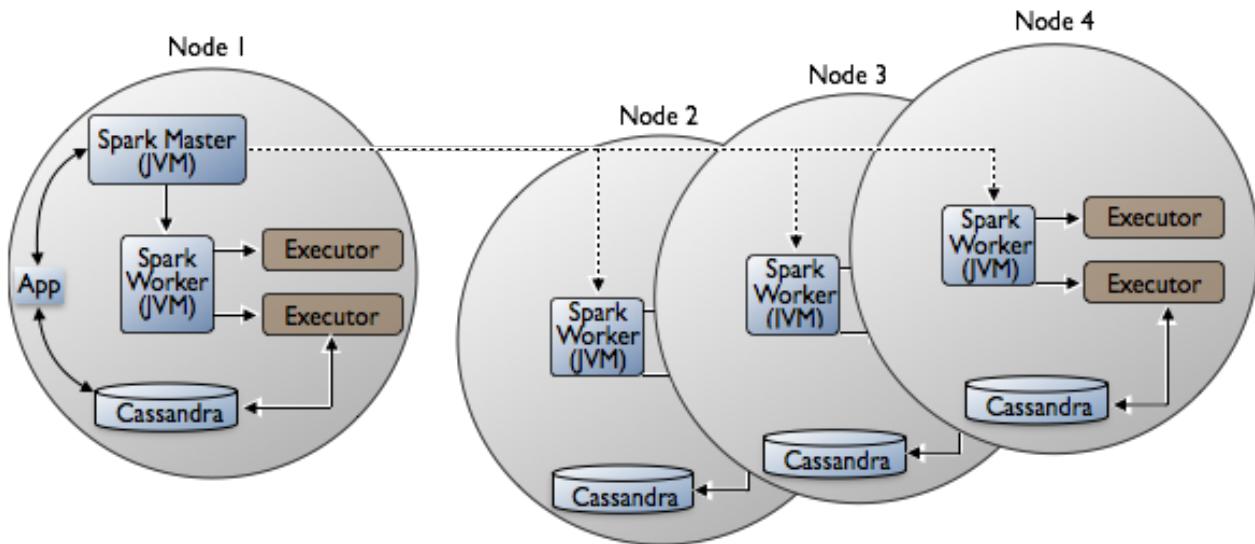
- Spark Worker, on all nodes
- Cassandra File System (CFS)
- Cassandra

A Spark Master controls the workflow, and a Spark Worker launches executors that are responsible for executing part of the job that is submitted to the Spark Master. Spark architecture is described in the [Apache documentation](#).

Spark supports multiple applications. A single application can spawn multiple jobs and the jobs run in parallel. An application reserves some resources on every node and these resources are not freed until the application finishes. For example, every session of Spark shell is an application that reserves resources. By default, the scheduler tries to allocate the application to the highest number of different nodes. For example, if the application declares that it needs four cores and there are ten servers, each offering two cores, the application most likely gets four executors, each on a different node, each consuming a single core. However, the application can get also two executors on two different nodes, each consuming two cores. You can configure the application scheduler. Spark Workers and Spark Master are spawned as separate processes and are very lightweight. Workers spawn other memory-heavy processes that are dedicated to handling queries. Memory settings for those additional processes are fully controlled by the administrator.

In deployment, one analytics node runs the Spark Master, and Spark Workers run on each of the analytics nodes. The Spark Master comes with [automatic high availability](#). Spark executors use native integration to access data in local Cassandra nodes through the [Open Source Spark-Cassandra Connector](#).

**Figure: Spark integration with DataStax Enterprise**



As you run Spark, you can access data in the Hadoop Distributed File System (HDFS) or the Cassandra File System (CFS) by using the URL for one or the other.

## Highly available Spark Master

The Spark Master High Availability mechanism uses a special table in the `dse_system` keyspace to store information required to recover Spark workers and the application. Unlike the high availability mechanism mentioned in Spark documentation, DataStax Enterprise does not use ZooKeeper.

If the original Spark Master fails, the reserved one automatically takes over. To set the reserve Spark Master, use the `dsetool setrjt` command.

If you enable password authentication in Cassandra, DataStax Enterprise creates special users. The Spark Master process accesses Cassandra through the special users, one per analytics node. The user names begin with the name of the node, followed by an encoded node identifier. The password is randomized. Do not remove these users or change the passwords.

In DataStax Enterprise, you manage the location of the Spark Master as you [manage the location of the Hadoop Job Tracker](#). By running a cluster in Spark plus Hadoop mode, the Job Tracker and Spark Master always work on the same node. These `dsetool` commands also work for [Spark Master](#), except for `dsetool jobtracker`. The Spark Master equivalent command is `dsetool sparkmaster`.

## Unsupported features

The following Spark features and APIs are not supported:

- [GraphX](#)
- Writing to blob columns from Spark

Reading columns of all types is supported; however, you must convert collections of blobs to byte arrays before serializing.

## Configuring Spark

Set Spark properties for DataStax Enterprise and Cassandra.

## Configuring Spark nodes

Spark nodes must be configured in their own [datacenters](#). Do not run Spark node types in the same datacenter with nodes running Cassandra node types. [DSE SearchAnalytics clusters](#) can use DSE Search

queries within DSE Analytics jobs. You can run Spark alongside integrated Hadoop or BYOH, but not on the same node. Be sure to follow the workload isolation guidelines.

## Performance

To manage Spark performance and operations:

- Set environment variables
- Protect Spark directories
- Grant access to default Spark directories
- Secure Spark nodes
- Configure Spark memory and cores
- Configure Spark logging options

## Set environment variables

DataStax recommends using the default values of Spark environment variables unless you need to increase the memory settings due to an `OutOfMemoryError` condition or garbage collection taking too long. Use the [Spark memory](#) configuration options in the `dse.yaml` and `spark-env.sh` files.

## Protect Spark directories

After you start up a Spark cluster, DataStax Enterprise creates a Spark work directory for each Spark Worker on worker nodes. A worker node can have more than one worker, configured by the `SPARK_WORKER_INSTANCES` option in `spark-env.sh`. If `SPARK_WORKER_INSTANCES` is undefined, a single worker is started. The work directory contains the standard output and standard error of executors and other application specific data stored by Spark Worker and executors; the directory is writable only by the Cassandra user.

By default, the Spark parent work directory is located in `/var/lib/spark/work`, with each worker in a subdirectory named `worker-number`, where the number starts at 0. To change the parent worker directory, configure `SPARK_WORKER_DIR` in the `spark-env.sh` file.

The Spark RDD directory is the directory where RDDs are placed when executors decide to spill them to disk. This directory might contain the data from the database or the results of running Spark applications. If the data in the directory is confidential, prevent access by unauthorized users. The RDD directory might contain a significant amount of data, so configure its location on a fast disk. The directory is writable only by the Cassandra user. The default location of the Spark RDD directory is `/var/lib/spark/rdd`. The directory should be located on a fast disk. To change the RDD directory, configure `SPARK_LOCAL_DIRS` in the `spark-env.sh` file.

## Grant access to default Spark directories

Before starting up nodes on a tarball installation, you need permission to access the default Spark directory locations: `/var/lib/spark` and `/var/log/spark`. Change ownership of these directories as follows:

```
$ sudo mkdir -p /var/lib/spark/rdd; sudo chmod a+rw /var/lib/spark/rdd; sudo chown -R $USER:$GROUP /var/lib/spark/rdd
$ sudo mkdir -p /var/log/spark; sudo chown -R $USER:$GROUP /var/log/spark
```

In multiple datacenter clusters, use a virtual datacenter to isolate Spark jobs. Running Spark jobs consume resources that can affect latency and throughput. To isolate Spark traffic to a subset of dedicated nodes, follow [workload isolation guidelines](#).

DataStax Enterprise supports the use of Cassandra virtual nodes (vnodes) with Spark.

## Secure Spark nodes

### Client-to-node SSL

Ensure that the truststore entries in `cassandra.yaml` are present as described in [Client-to-node encryption](#), even when client authentication is not enabled.

### JAR files on CFS

When JAR files are on the Cassandra file system (CFS) and authentication is enabled, enable [Spark applications in cluster mode](#).

### Cassandra credentials for the Spark SQL Thrift server

In the `hive-site.xml` file, configure Cassandra authentication credentials for the Spark SQL Thrift server. Ensure that you use the `hive-site.xml` file in the Spark directory:

Installer-Services and Package installations	<code>/etc/dse/spark/hive-site.xml</code>
Installer-No Services and Tarball installations	<code>install_location/resources/spark/conf/hive-site.xml</code>

### Kerberos

Set Kerberos options.

## Configure Spark memory and cores

Spark memory options affect different components of the Spark ecosystem:

### Spark History server and the Spark Thrift server memory

The `SPARK_DAEMON_MEMORY` option configures the memory that is used by the Spark History server and the Spark Thrift server. Add or change this setting in the `spark-env.sh` file on nodes that run these server applications.

### Spark Worker memory

The `SPARK_WORKER_MEMORY` option configures the total amount of memory that you can assign to all executors that are run by a single Spark Worker on the particular node.

### Application executor memory

You can configure the amount of memory that each executor can consume for the application. Spark uses a 512MB default. Use either the `spark.executor.memory` option, described in ["Spark 1.4.1 Available Properties"](#), or the `--executor-memory mem` argument to the `dse spark` command.

## Application memory

You can configure additional Java options that are applied by the worker when spawning an executor for the application. Use the `spark.executor.extraJavaOptions` property, described in [Spark 1.4.1 Available Properties](#). For example: `spark.executor.extraJavaOptions -XX:+PrintGCDetails -Dkey=value -Dnumbers="one two three"`

## Core management

You can manage the number of cores by configuring these options.

- Spark Worker cores

The `SPARK_WORKER_CORES` option configures the number of cores offered by Spark Worker for use by executors. A single executor can borrow more than one core from the worker. The number of cores used by the executor relates to the number of parallel tasks the executor might perform. The number of cores offered by the cluster is the sum of cores offered by all the workers in the cluster.

- Application cores

In the Spark configuration object of your application, you configure the number of application cores that the application requests from the cluster using either the `spark.cores.max` configuration property or the `--total-executor-cores cores` argument to the `dse spark` command.

See the [Spark documentation](#) for details about memory and core allocation.

DataStax Enterprise can control the memory and cores offered by particular Spark Workers in semi-automatic fashion. The initial\_spark\_worker\_resources parameter in the dse.yaml file specifies the fraction of system resources that are made available to the Spark Worker. The available resources are calculated in the following way:

- Spark Worker memory = initial\_spark\_worker\_resources \* (total system memory - memory assigned to Cassandra)
- Spark Worker cores = initial\_spark\_worker\_resources \* total system cores

The lowest values you can assign to Spark Worker memory and cores are 64 MB and 1 core, respectively. If the results are lower, no exception is thrown and the values are automatically limited. The range of the initial\_spark\_worker\_resources value is 0.01 to 1. If the range is not specified, the default value 0.7 is used.

This mechanism is used by default to set the Spark Worker memory and cores. To override the default, uncomment and edit one or both SPARK\_WORKER\_MEMORY and SPARK\_WORKER\_CORES options in the spark-env.sh file.

## Configuring Spark logging options

You can configure Spark logging options for the Spark logs.

### Log directories

The Spark logging directory is the directory where the Spark components store individual log files. DataStax Enterprise places logs in the following locations:

#### Executor logs

- *SPARK\_WORKER\_DIR/worker-n/application\_id/executor\_id/stderr*
- *SPARK\_WORKER\_DIR/worker-n/application\_id/executor\_id/stdout*

#### Spark Master/Worker logs

Spark Master: the global `system.log`

Spark Worker: *SPARK\_WORKER\_LOG\_DIR/worker-n/worker.log*

The default *SPARK\_WORKER\_LOG\_DIR* location is `/var/log/spark/worker`.

#### Default log directory for Spark CQL Thrift server

The default log directory for starting the Spark CQL Thrift server is `$HOME/spark-thrift-server`.

#### Spark Shell and application logs

Spark Driver Shell and application logs are output to the console.

#### Log configuration file

Log configuration files are located in the [same directory](#) as `spark-env.sh`.

## Procedure

To configure Spark logging options:

1. Configure logging options, such as log levels, in the following files:

Option	Description
Executors	<code>logback-spark-executor.xml</code>
Spark Master	<code>logback.xml</code>
Spark Worker	<code>logback-spark-server.xml</code>

Option	Description
<b>Spark Driver (Spark Shell, Spark applications)</b>	logback-spark.xml

- Configure a safe communication channel to access the Spark user interface.

**Note:** When user credentials are specified in plain text on the dse command line, like `$ dse -u username -p password`, the credentials are present in the logs of Spark workers when the driver is run in cluster mode. The Spark Master, Spark Worker, executor, and driver logs might include sensitive information. Sensitive information includes passwords and digest authentication tokens for **Kerberos authentication** mode that are passed in the command line or Spark configuration. DataStax recommends using only safe communication channels like VPN and SSH to access the Spark user interface.

## Configuring the Spark history server

The Spark history server provides a way to load the event logs from Spark jobs that were run with event logging enabled. The Spark history server works only when files were not flushed before the Spark Master attempted to build a history user interface.

### Procedure

To enable the Spark history server:

- Create a directory for event logs in the Cassandra file system (CFS):

```
$ dse hadoop fs -mkdir /spark/events
```

- On each node in the cluster, edit the `spark-defaults.conf` file to enable event logging and specify the directory for event logs:

```
#Turns on logging for applications submitted from this machine
spark.eventLog.dir cfs:/spark/events
spark.eventLog.enabled true
#Sets the logging directory for the history server
spark.history.fs.logDirectory cfs:/spark/events
```

- Start the Spark history server on one of the nodes in the cluster:

The Spark history server is a front-end application that displays logging data from all nodes in the Spark cluster. It can be started from any node in the cluster.

```
$ dse spark-history-server start
```

**Note:** The Spark Master web UI does not show the historical logs. To work around this known issue, access the history from port 18080.

- When event logging is enabled, the default behavior is for all logs to be saved, which causes the storage to grow over time. To enable automated cleanup edit `spark-defaults.conf` and edit the following options:

```
spark.history.fs.cleaner.enabled true
spark.history.fs.cleaner.interval 1d
spark.history.fs.cleaner.maxAge 7d
```

For these settings, automated cleanup is enabled, the cleanup is performed daily, and logs older than seven days are deleted.

## Setting Cassandra-specific properties

Spark integration uses the [Spark Cassandra Connector](#) under the hood. You can use the configuration options defined in that project to configure DataStax Enterprise Spark. Spark recognizes system properties that have the `spark.` prefix and adds the properties to the configuration object implicitly upon creation. You can avoid adding system properties to the configuration object by passing `false` for the `loadDefaults` parameter in the `SparkConf` constructor.

You pass settings for Spark, Spark Shell, and other DataStax Enterprise Spark built-in applications using the intermediate application `spark-submit`, described in [Spark documentation](#).

### Configuring the Spark shell

Pass Spark configuration arguments using the following syntax:

```
$ dse spark [submission_arguments] [application_arguments]
```

- where `submission_arguments` are:

- `--properties-file path_to_properties_file`

The location of the properties file having the configuration settings. By default, Spark loads the settings from `conf/spark-defaults.conf`.

- `--executor-memory memory`

How much memory to allocate on each machine for the application. You can provide the `memory` argument in JVM format using either the `k`, `m`, or `g` suffix.

- `--total-executor-cores cores`

The total number of cores the application uses

- `--conf name=value`

An arbitrary Spark option to the Spark configuration prefixed by `spark`.

- `--help`

Shows a help message that displays all options except DataStax Enterprise Spark shell options.

- `--jars <additional-jars>`

A comma-separated list of paths to additional jar files.

- `--verbose`

Displays which arguments are recognized as Spark configuration options and which arguments are forwarded to the Spark Shell.

- Spark shell application arguments:

- `-i file`

Runs a script from the specified file.

### Configuring Spark applications

You pass the Spark submission arguments using the following syntax:

```
$ dse spark-submit [submission_arguments] application_file
[application_arguments]
```

- All `submission_arguments` and these additional `spark-submit submission_arguments`:
- `-- class class_name`

The full name of the application main class.

- `-- name name`

The application name as displayed in the Spark web application.

- `-- py-files files`

- A comma-separated list of the .zip, .egg, or .py files that are set on PYTHONPATH for Python applications.
  - `--files files`  
A comma-separated list of files that are distributed among the executors and available for the application.
  - `--master master_URL`  
The URL of the Spark Master.
  - `application_file application_arguments`  
*application\_file* is an application file, a JAR file, or a .py file that contains the application being run.  
Passed without any control argument; *application\_file* acts as a separator between Spark configuration arguments and custom application arguments.
- In general, Spark submission arguments are translated into system properties -Dname=value and other VM parameters like classpath. The application arguments are passed directly to the application.

## Spark configuration object

Use the `com.datastax.bdp.spark.DseSparkContext` class to create a Spark context object to connect to DataStax Enterprise clusters. The `DseSparkContext` class is functionally the same as `org.apache.spark.SparkContext`.

```
import com.datastax.bdp.spark.DseSparkContext
import org.apache.spark.SparkConf

object ConfigurationExample extends App {

 def createSparkContext() = {
 val conf = new SparkConf()
 /* set the app name here or by using the --name option when
 you submit the app */
 .setAppName("Configuration example")
 .forDse

 new DseSparkContext.apply(conf)
 }

 val sc = createSparkContext()

 // ...
 sc.stop()
}
```

## Property list

The following Cassandra-specific properties are recognized:

### **spark.cassandra.keyspace**

The default keyspace for Spark SQL.

### **spark.cassandra.connection.native.port**

Default = 9042. Port for native client protocol connections.

### **spark.cassandra.connection.rpc.port**

Default = 9160. Port for thrift connections.

### **spark.cassandra.connection.host**

The host name or IP address to which the Thrift RPC service and native transport is bound. The `rpc_address` property in the `cassandra.yaml`, which is localhost by default, determines the default value of this property.

### Read properties

#### **spark.cassandra.input.split.size**

Default = 100000. Approximate number of rows in a single Spark partition. The higher the value, the fewer Spark tasks are created. Increasing the value too much may limit the parallelism level.

#### **spark.cassandra.input.fetch.size\_in\_rows**

Default = 1000. Number of rows being fetched per roundtrip to Cassandra. Increasing this value increases memory consumption. Decreasing the value increases the number of roundtrips. In earlier releases, this property was `spark.cassandra.input.page.row.size`.

#### **spark.cassandra.input.consistency.level**

Default = LOCAL\_ONE. Consistency level to use when reading.

### Write properties

You can set the following properties in `SparkConf` to fine tune the saving process.

#### **spark.cassandra.output.batch.size.bytes**

Default = auto. Number of bytes per single batch. The default, auto, means the connector adjusts the number of bytes based on the amount of data.

#### **spark.cassandra.output.consistency.level**

Default = LOCAL\_ONE. Consistency level to use when writing.

#### **spark.cassandra.output.concurrent.writes**

Default = 5. Maximum number of batches executed in parallel by a single Spark task.

#### **spark.cassandra.output.batch.size.rows**

Default = 64K. The maximum total size of the batch in bytes.

See the [Spark Cassandra Connector documentation](#) for details on additional, low-level properties.

## Using Spark with DataStax Enterprise

DataStax Enterprise integrates with Apache Spark to allow distributed analytic applications to run using Cassandra data.

### Starting Spark

How you start Spark depends on the installation and if you want to run in Spark mode, Spark and Hadoop mode, or SearchAnalytics:

#### **Installer-Services and Package installations**

To start the Spark trackers on a cluster of analytics nodes, edit the `/etc/default/dse` file to set `SPARK_ENABLED` to 1.

When you [start DataStax Enterprise as a service](#), the node is launched as a Spark node. You can enable additional components.

Mode	Option in <code>/etc/default/dse</code>	Description
Spark	<code>SPARK_ENABLED=1</code>	Start the node in Spark mode.

Mode	Option in /etc/default/dse	Description
SearchAnalytics mode	SPARK_ENABLED=1 SEARCH_ENABLED=1	SearchAnalytics mode is experimental, and is not recommended for production clusters. In dse.yaml, <a href="#">cql_solr_query_paging: driver</a> is required.
Spark and Hadoop mode	SPARK_ENABLED=1 HADOOP_ENABLED=1	Spark and Hadoop mode should be used only for development purposes.

**Installer-No Services and Tarball installations:**

To start the Spark trackers on a cluster of analytics nodes, use the -k option:

```
$ dse cassandra -k
```

**Note:**

Nodes started with -t or -k are automatically assigned to the default Analytics datacenter if you do not configure a datacenter in the snitch property file.

You can enable additional components:

Mode	Option	Description
Spark	-k	Start the node in Spark mode.
SearchAnalytics mode	-k -s	SearchAnalytics mode is experimental, and is not recommended for production clusters. In dse.yaml, <a href="#">cql_solr_query_paging: driver</a> is required.
Spark and Hadoop mode	-k -t	Spark and Hadoop mode should be used only for development purposes.

For example:

To start a node in SearchAnalytics mode, use the -k and -s options.

```
$ dse cassandra -k -s
```

SearchAnalytics mode is experimental, and is not recommended for production clusters.

To start a node in Spark and Hadoop mode, use the -k and -t options:

```
$ dse cassandra -k -t
```

Spark and Hadoop mode should only be used for development purposes.

Starting the node with the Spark or Hadoop option starts a node that is designated as the Job Tracker, as shown by the Analytics(JT) workload in the output of the dsetool ring command:

```
$ dsetool ring
```

Note: Ownership information does not include topology, please specify a keyspace.

Address	DC	Rack	Workload	Status	State	Load
10.160.137.165	Analytics	rack1	Analytics (JT)	Up	Normal	87.04 KB
33.33% -9223372036854775808						
10.168.193.41	Analytics	rack1	Analytics (TT)	Up	Normal	92.91 KB
33.33% -3074457345618258603						
10.176.83.32	Analytics	rack1	Analytics (TT)	Up	Normal	94.9 KB
33.33% 3074457345618258602						

The default location of the `dsetool` command depends on the type of installation:

Package installations	<code>/usr/bin/dsetool</code>
Installer-Services installations	<code>/usr/bin/dsetool</code>
Installer-No Services and Tarball installations	<code>install_location/bin/dsetool</code>

If you use `sudo` to start DataStax Enterprise, remove the `~/.spark` directory before you restart the cluster :

```
$ sudo rm -r ~/.spark
```

## Launching Spark

After starting a Spark node, use `dse` commands to launch Spark.

The default location of the `dse` tool depends on the type of installation:

Package installations	<code>/usr/bin/dse</code>
Installer-Services installations	<code>/usr/bin/dse</code>
Installer-No Services and Tarball installations	<code>install_location/bin/dse</code>

You can use [Cassandra specific properties](#) to start Spark. Spark binds to the `listen_address` that is specified in `cassandra.yaml`.

DataStax Enterprise supports these commands for launching Spark on the DataStax Enterprise command line:

### **dse spark**

Enters interactive Spark shell, offers basic autocompletion.

```
$ dse spark
```

### **dse spark-submit**

Launches applications on a cluster like [spark-submit](#). Replaces the deprecated `dse spark-class` command. Using this interface you can use Spark cluster managers without the need for separate configurations for each application. The syntax is:

```
$ dse spark-submit --class class_name jar_file other_options
```

For example, if you write a class that defines an option named `d`, enter the command as follows:

```
$ dse spark-submit --class com.datastax.HttpSparkStream target/HttpSparkStream.jar -d $NUM_SPARK_NODES
```

**Note:** The directory in which you run the `dse` Spark commands must be writable by the current user.

[Internal authentication](#) is supported.

You can use environment variables to increase security and prevent the user name and passwords from appearing in the Spark log files or in the process list on the Spark Web UI. To specify a user name and password using environment variables, use the following syntax:

```
$ DSE_USERNAME=user DSE_PASSWORD=secret dse spark [-submit]
```

These environment variables are supported for all Spark commands.

To specify a user name and password to run an application, use the following syntax:

```
$ dse [-f config_file] [-u username -p password] [-a jmx_username -b jmx_password] spark[-submit]
```

where:

- `-f config_file` is the path to a configuration file that stores credentials. If not specified, then use `~/.dserc` if it exists.

The configuration file can contain Cassandra and JMX login credentials. For example:

```
username=cassandra
password=cassandra
jmx_username=cassandra
jmx_password=jmx
```

The credentials in the configuration file are stored in clear text. DataStax recommends restricting access to this file only to the specific user.

- `dse -u username` is the user name to authenticate against the configured Cassandra user.
- `dsetool -l username` is the user name to authenticate against the configured Cassandra user.
- `-p password` is the password to authenticate against the configured Cassandra user. If you do not provide a password on the command line, you are prompted to enter one.
- `-a jmx_username` is the user name for authenticating with secure JMX.
- `-b jmx_password` is the password for authenticating with secure JMX. If you do not provide a password on the command line, you are prompted to enter one.

**Note:** To increase security and prevent the user name and passwords from appearing in the Spark log files or in the process list on the Spark Web UI, DataStax recommends using the environment variables instead of passing user credentials on the command line.

## Running Spark commands against a remote cluster

To run Spark commands against a remote cluster, you must copy your Hadoop configuration files from one of the remote nodes to the local client machine.

The default location of the Hadoop configuration files depends on the type of installation:

Installer-Services and Package installations	/etc/dse/hadoop/
Installer-No Services and Tarball installations	<i>install_location/resources/hadoop/conf/</i>

To run a driver application remotely, there must be full public network communication between the remote nodes and the client machine.

### Procedure

1. Copy the files from the remote node to the local machine.

On a services or package install of DataStax Enterprise:

```
$ cd /etc/dse/hadoop
$ scp adminuser@node1:/etc/dse/hadoop/* .
```

2. Optional: Edit the copied XML configuration files to ensure that the IP address for the Cassandra nodes is a publicly accessible IP address.
3. Run the Spark command against the remote node.

```
$ dse spark-submit submit options myApplication.jar
```

To set the driver host to a publicly accessible IP address, pass in the spark.driver.host option.

```
$ dse spark-submit --conf spark.driver.host=IP address myApplication.jar
```

## Accessing Cassandra from Spark

DataStax Enterprise integrates Spark with Cassandra. Cassandra tables are fully usable from Spark.

### Accessing Cassandra from a Spark application

To access Cassandra from a Spark application, follow instructions in the Spark example [Portfolio Manager demo using Spark](#) on page 116.

### Accessing Cassandra from the Spark shell

DataStax Enterprise uses the [Spark Cassandra Connector](#) to provide Cassandra integration for Spark. By running the Spark shell in DataStax Enterprise, you have access to the enriched Spark Context object (sc) for accessing Cassandra directly.

To access Cassandra from the Spark Shell, just run the `dse spark` command and follow instructions in subsequent sections.

```
$ dse spark

Welcome to
 //_/_/_/_/_/_/_
 / \ / \ / \ / \ / \ / \ / \
 / \ / . \ / \ / \ / \ / \ / \
 / \ / \ / \ / \ / \ / \ / \
version 1.4.1

Using Scala version 2.10.3 (Java HotSpot(TM) 64-Bit Server VM, Java
1.7.0_25)
Type in expressions to have them evaluated.
Type :help for more information.
Creating SparkContext...
2015-06-26 22:52:05.295 java[94799:1703] Unable to load realm info from
SCDynamicStore
Created spark context..
Spark context available as sc.
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

### Using the Spark context

To get a Spark RDD that represents a Cassandra table, load data from a Cassandra table into Spark using the `sc.dot` (`sc.`) syntax to call the `cassandraTable` method on the Spark context, where `sc` represents the Spark API `SparkContext` class.

```
sc.cassandraTable("keyspace", "table name")
```

Cassandra data is mapped into Scala objects and DataStax Enterprise returns a `CassandraRDD` [`CassandraRow`]. To use the Spark API for creating an application that runs outside DataStax Enterprise, import `com.datastax.spark.connector.SparkContextCassandraFunctions`.

The following example shows how to load a Cassandra table into Spark and read the table in Cassandra from Spark.

1. Create this keyspace and table in Cassandra using `cqlsh`. Use the Analytics datacenter to create the keyspace.

```
CREATE KEYSPACE test WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'Analytics' : 1};

CREATE TABLE test.words (word text PRIMARY KEY, count int);
```

This example assumes you start a single-node cluster in [Spark mode](#).

2. Load data into the words table.

```
INSERT INTO test.words (word, count) VALUES ('foo', 10);
INSERT INTO test.words (word, count) VALUES ('bar', 20);
```

3. Assuming you started the node in Spark mode, start the Spark shell. Do not use `sudo` to start the shell.

```
$ bin/dse spark
```

The Welcome to Spark output and scala prompt appears.

4. Use the `:showSchema` command to view the user keyspaces and tables in Cassandra.

```
:showSchema
```

Information about all user keyspaces appears.

```
=====
Keyspace: HiveMetaStore
=====

Table: MetaStore

- key : String (partition key column)
- entity : String (clustering column)
- value : java.nio.ByteBuffer

=====
Keyspace: test
=====

Table: words

- word : String (partition key column)
- count : Int

scala> :showSchema test
=====
Keyspace: test
=====

Table: words

- word : String (partition key column)
- count : Int

scala> :showSchema test words
=====
Keyspace: test
=====

Table: words

- word : String (partition key column)
- count : Int
```

5. Get information about only the test keyspace.

```
:showSchema test
```

```
=====
Keyspace: test
=====
Table: words

- word : String (partition key column)
- count : Int
```

6. Get information about the words table.

```
:showSchema test words
```

```
=====
Keyspace: test
=====
Table: words

- word : String (partition key column)
- count : Int
```

7. Define a base RDD to point to the data in the test.words table.

```
val rdd = sc.cassandraTable("test", "words")
```

```
rdd:
com.datastax.spark.connector.rdd.CassandraRDD[com.datastax.spark.connector.CassandraRow] = CassandraRDD[0] at RDD at CassandraRDD.scala:47
```

The RDD is returned in the rdd value. To read the Cassandra table, use this command.

```
rdd.toArray.foreach(println)
```

```
CassandraRow{word: bar, count: 20}
CassandraRow{word: foo, count: 10}
```

Now, you can use methods on the returned RDD to query the test.words table.

## Python support for loading cassandraTables

Python supports loading cassandraTables from a Spark Streaming context and saving a DStream to Cassandra.

## Reading column values

You can read columns in a Cassandra table using the get methods of the CassandraRow object. The get methods access individual column values by column name or column index. Type conversions are applied on the fly. Use `getOption` variants when you expect to receive Cassandra null values.

Continuing with the previous example, follow these steps to access individual column values.

1. Store the first item of the rdd in the firstRow value.

```
val firstRow = rdd.first
```

```
firstRow: com.datastax.spark.connector.CassandraRow = CassandraRow{word:
 foo, count: 10}
```

**2. Get the column names.**

```
rdd.columnNames
```

```
res3: com.datastax.spark.connector.ColumnSelector = AllColumns
```

**3. Use a generic get to query the table by passing the return type directly.**

```
firstRow.get[Int] ("count")
```

```
res4: Int = 10
```

```
firstRow.get[Long] ("count")
```

```
res5: Long = 10
```

```
firstRow.get[BigInt] ("count")
```

```
res6: BigInt = 10
```

```
firstRow.get[java.math.BigInteger] ("count")
```

```
res7: java.math.BigInteger = 10
```

```
firstRow.get[Option[Int]] ("count")
```

```
res8: Option[Int] = Some(10)
```

```
firstRow.get[Option[BigInt]] ("count")
```

```
res9: Option[BigInt] = Some(10)
```

## Reading collections

You can read collection columns in a Cassandra table using the get methods of the `CassandraRow` object. The get methods access the collection column and returns a corresponding Scala collection.

Assuming you set up the test keyspace earlier, follow these steps to access a Cassandra collection.

**1. In the test keyspace, set up a collection set using `cqlsh`.**

```
CREATE TABLE test.users (
 username text PRIMARY KEY, emails SETtext);

INSERT INTO test.users (username, emails)
 VALUES ('someone', {'someone@email.com', 's@email.com'});
```

**2. If Spark is not running, start the Spark shell. Do not use `sudo` to start the shell.**

```
$ bin/dse spark
```

The Welcome to Spark output and scala prompt appears.

3. Define a CassandraRDD[CassandraRow] to access the collection set.

```
val row = sc.cassandraTable("test", "users").toArray.apply(0)
```

```
row: com.datastax.spark.connector.CassandraRow = CassandraRow{username: someone,
emails: {s@email.com, someone@email.com}}
```

4. Query the collection set in Cassandra from Spark.

```
row.getList[String]("emails")
```

```
res2: Vector[String] = Vector(s@email.com, someone@email.com)
```

```
row.get[List[String]]("emails")
```

```
res3: List[String] = List(s@email.com, someone@email.com)
```

```
row.get[Seq[String]]("emails")
```

```
res4: Seq[String] = List(s@email.com, someone@email.com)
```

```
row.get[IndexedSeq[String]]("emails")
```

```
res5: IndexedSeq[String] = Vector(s@email.com, someone@email.com)
```

```
row.get[Set[String]]("emails")
```

```
res6: Set[String] = Set(s@email.com, someone@email.com)
```

```
row.get[String]("emails")
```

```
res7: String = {s@email.com, someone@email.com}
```

## Restricting the number of fetched columns

For performance reasons, you should not fetch columns you don't need. You can achieve this with the `select` method:

To restrict the number of fetched columns:

```
val row = sc.cassandraTable("test", "users").select("username").toArray
```

```
row: Array[com.datastax.spark.connector.CassandraRow] =
Array(CassandraRow{username: someone})
```

## Mapping rows to tuples and case classes

Instead of mapping your Cassandra rows to objects of the `CassandraRow` class, you can directly unwrap column values into tuples of the desired type.

To map rows to tuples:

```
sc.cassandraTable[(String, Int)]("test", "words").select("word",
"count").toArray
```

```

res9: Array[(String, Int)] = Array((bar,20), (foo,10))

sc.cassandraTable[(Int, String)]("test", "words").select("count",
 "word").toArray

res10: Array[(Int, String)] = Array((20,bar), (10,foo))

```

Define a case class with properties of the same name as the Cassandra columns. For multi-word column identifiers, separate each word using an underscore in Cassandra, and use camel case abbreviation on the Scala side.

To map rows to case classes:

```

case class WordCount(word: String, count: Int)

defined class WordCount

scala> sc.cassandraTable[WordCount]("test", "words").toArray

res14: Array[WordCount] = Array(WordCount(bar,20), WordCount(foo,20))

```

You can name columns in Cassandra using these conventions:

- Use the underscore convention and lowercase letters. (Recommended)
- Use the camel case convention, exactly the same as properties in Scala.

The following examples show valid column names.

**Table: Recommended naming convention**

Cassandra column name	Scala property name
count	count
column_1	column1
user_name	userName
user_address	UserAddress

**Table: Alternative naming convention**

Cassandra column name	Scala property name
count	count
column1	column1
userName	userName
UserAddress	UserAddress

## Mapping rows to objects with a user-defined function

Invoke `as` on the CassandraRDD to map every row to an object of a different type. Contrary to `map`, `as` expects a function having the same number of arguments as the number of columns to be fetched. Invoking `as` in this way performs type conversions. Using `as` to directly create objects of a particular type eliminates the need to create `CassandraRow` objects and also decreases garbage collection pressure.

To map columns using a user-defined function:

```
val table = sc.cassandraTable("test", "words")

table:
com.datastax.spark.connector.rdd.CassandraRDD [com.datastax.spark.connector.CassandraRow] = CassandraRDD [9] at RDD at CassandraRDD.scala:47

val total = table.select("count").as((c: Int) => c).sum

total: Double = 30.0

val frequencies = table.select("word", "count").as((w: String, c: Int) =>
(w, c / total)).toArray

frequencies: Array[(String, Double)] = Array((bar,0.6666666666666666), (foo,0.3333333333333333))
```

## Filtering rows on the server

To filter rows, you can use the filter transformation provided by Spark. Filter transformation fetches all rows from Cassandra first and then filters them in Spark. Some CPU cycles are wasted serializing and deserializing objects excluded from the result. To avoid this overhead, CassandraRDD has a method that passes an arbitrary CQL condition to filter the row set on the server.

This example shows how to use Spark to filter rows on the server.

1. [Download](#) and unzip the CQL commands for this example. The commands in this file perform the following tasks:
  - Create a cars table in the test keyspace.
  - Index the color column.
  - Insert some data into the table
2. Run the `test_cars.cql` file using `cqlsh` or DevCenter. For example using `cqlsh`:

```
$ cqlsh -f test_cars.cql
```

3. Filter the rows using Spark:

```
sc.cassandraTable("test", "cars").select("id", "model").where("color = ?",
"black").toArray.foreach(println)
```

```
CassandraRow{id: AS-8888, model: Aston Martin DB9 Volante}
CassandraRow{id: KF-334L, model: Ford Mondeo}
CassandraRow{id: MT-8787, model: Hyundai x35}
CassandraRow{id: MZ-1038, model: Mazda CX-9}
CassandraRow{id: DG-2222, model: Dodge Avenger}
CassandraRow{id: DG-8897, model: Dodge Charger}
CassandraRow{id: BT-3920, model: Bentley Continental GT}
CassandraRow{id: IN-9964, model: Infinity FX}
```

```
sc.cassandraTable("test", "cars").select("id", "model").where("color = ?",
"silver").toArray.foreach(println)
```

```
CassandraRow{id: FR-8877, model: Ferrari FF}
CassandraRow{id: FR-8877, model: Ferrari FF}
CassandraRow{id: HD-1828, model: Honda Accord}
CassandraRow{id: WX-2234, model: Toyota Yaris}
```

## Saving data to Cassandra

With DataStax Enterprise, you can save almost any RDD to Cassandra. Unless you do not provide a custom mapping, the object class of the RDD must be a tuple or have property names corresponding to Cassandra column names. To save the RDD, call the `saveToCassandra` method with a keyspace name, table name, and optionally, a list of columns. Before attempting to use the RDD in a standalone application, import `com.datastax.spark.connector`.

You can also use the [DataFrames API](#) to manipulate data within Spark.

## Saving a collection of tuples

The following example shows how to save a collection of tuples to Cassandra.

```
scala> val collection = sc.parallelize(Seq(("cat", 30), ("fox", 40)))
collection: org.apache.spark.rdd.RDD[(String, Int)] =
 ParallelCollectionRDD[6] at parallelize at console:22

scala> collection.saveToCassandra("test", "words", SomeColumns("word",
 "count"))

scala>
```

At the last scala prompt in this example, no output means that the data was saved to Cassandra.

In cqlsh, query the words table to select all the contents.

```
SELECT * FROM test.words;

 word | count
-----+-----
 bar | 20
 foo | 10
 cat | 30
 fox | 40

(4 rows)
```

## Saving a collection of case class objects to Cassandra

The following example shows how to save a collection of case class objects.

```
scala> case class WordCount(word: String, count: Long)
defined class WordCount

scala> val collection = sc.parallelize(Seq(WordCount("dog", 50),
 WordCount("cow", 60)))
collection: org.apache.spark.rdd.RDD[WordCount] = ParallelCollectionRDD[0]
 at parallelize at console:24

scala> collection.saveToCassandra("test", "words", SomeColumns("word",
 "count"))

scala>
```

In cqlsh, query the words table to select all the contents.

```
SELECT * FROM test.words;

 word | count
```

```

-----+-----
bar | 20
foo | 10
cat | 30
fox | 40
dog | 50
cow | 60

```

## Using non-default property-name to column-name mappings

Mapping rows to tuples and case classes work out-of-the box, but in some cases, you might need more control over Cassandra-Scala mapping. For example, Java classes are likely to use the JavaBeans naming convention, where accessors are named with `get`, `is` or `set` prefixes. To customize column-property mappings, put an appropriate `ColumnMapper[YourClass]` implicit object in scope. Define such an object in a companion object of the class being mapped. The `ColumnMapper` affects both loading and saving data. DataStax Enterprise includes a few `ColumnMapper` implementations.

### Working with JavaBeans

To work with Java classes, use `JavaBeanColumnMapper`. Make sure objects are serializable; otherwise Spark cannot send them over the network. The following example shows how to use the `JavaBeanColumnMapper`.

To use JavaBean style accessors:

```

scala> :paste
// Entering paste mode (ctrl-D to finish)

```

Paste this import command and class definition:

```

import com.datastax.spark.connector.mapper.JavaBeanColumnMapper
class WordCount extends Serializable {
 private var _word: String = ""
 private var _count: Int = 0
 def setWord(word: String) { _word = word }
 def setCount(count: Int) { _count = count }
 override def toString = _word + ":" + _count
}
object WordCount {
 implicit object Mapper extends JavaBeanColumnMapper[WordCount]
}

```

Enter CTRL D to exit paste mode. The output is:

```

// Exiting paste mode, now interpreting.

import com.datastax.spark.connector.mapper.JavaBeanColumnMapper
defined class WordCount
defined module WordCount

scala>

```

Query the `WordCount` object.

```

sc.cassandraTable[WordCount]("test", "words").toArray
res18: Array[WordCount] = Array(cow:60, bar:20, foo:10, cat:30, fox:40,
 dog:50)

```

To save the data, you need to define getters.

## Manually specifying a property-name to column-name relationship

If for some reason you want to associate a property with a column of a different name, pass a column translation map to the DefaultColumnMapper or JavaBeanColumnMapper.

To change column names:

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

import com.datastax.spark.connector.mapper.DefaultColumnMapper
case class WordCount(w: String, c: Int)
object WordCount { implicit object Mapper extends
DefaultColumnMapper[WordCount] (Map("w" -> "word", "c" -> "count")) }
```

Enter CTRL D.

```
// Exiting paste mode, now interpreting.

import com.datastax.spark.connector.mapper.DefaultColumnMapper
defined class WordCount
defined module WordCount
```

Continue entering these commands:

```
scala> sc.cassandraTable[WordCount]("test", "words").toArray
res21: Array[WordCount] = Array(WordCount(cow,60), WordCount(bar,20),
WordCount(foo,10), WordCount(cat,30), WordCount(fox,40), WordCount(dog,50))

scala>
sc.parallelize(Seq(WordCount("bar",20),WordCount("foo",40))).saveToCassandra("test",
"words", SomeColumns("word", "count"))

scala>
```

## Writing custom ColumnMappers

To define column mappings for your classes, create an appropriate implicit object implementing ColumnMapper[YourClass] trait.

### Using the Cassandra context

The Cassandra context was removed in DataStax Enterprise 4.7. Instead, [use the Spark context](#) to create a CassandraRDD.

## Monitoring Spark with the web interface

A web interface, bundled with DataStax Enterprise, facilitates monitoring, debugging, and managing Spark.

**Note:** When user credentials are specified in plain text on the dse command line, like \$ dse -u username -p password, the credentials are present in the logs of Spark workers when the driver is run in cluster mode. The Spark Master, Spark Worker, executor, and driver logs might include sensitive information. Sensitive information includes passwords and digest authentication tokens for [Kerberos authentication](#) mode that are passed in the command line or Spark configuration. DataStax recommends using only safe communication channels like VPN and SSH to access the Spark user interface.

### Using the Spark web interface

To use the Spark web interface:

- Enter the public IP address of the Spark Master node in a browser followed by port number 7080.

- To change the port, modify the `spark-env.sh` configuration file.

 **Spark Master at spark://127.0.0.1:7077**

URL: spark://127.0.0.1:7077  
 REST URL: spark://127.0.0.1:6066 (*cluster mode*)  
 Workers: 1  
 Cores: 6 Total, 6 Used  
 Memory: 8.5 GB Total, 512.0 MB Used  
 Applications: 1 Running, 0 Completed  
 Drivers: 0 Running, 0 Completed  
 Status: ALIVE

#### Workers

Worker Id	Address	State	Cores	Memory
worker-20150826091003-127.0.0.1-62196	127.0.0.1:62196	ALIVE	6 (6 Used)	8.5 GB (512.0 MB Used)

#### Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20150826091021-0000 (kill)	PySparkShell	6	512.0 MB	2015/08/26 09:10:21	russellspitzer	RUNNING	41 min

#### Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

See the Spark documentation for updates on [monitoring](#).

## Spark Worker nodes and debugging logs

- In the Spark Master node page, click the ID of a worker node, in this example `worker-20140314184018-10.168.193.41-41345`. The Spark Worker page for the node appears. In this web interface, you see detailed information about apps that are running.

In this example, the Workers section lists three registered nodes. The misleading summary information in the top left corner of the page covers alive and dead workers.

 **Spark Worker at 127.0.0.1:62196**

ID: worker-20150826091003-127.0.0.1-62196

Master URL: spark://127.0.0.1:7077

Cores: 6 (6 Used)

Memory: 8.5 GB (512.0 MB Used)

[Back to Master](#)

#### Running Executors (1)

ExecutorID	Cores	State	Memory	Job Details	Logs
0	6	LOADING	512.0 MB	ID: app-20150826091021-0000 Name: PySparkShell User: russellspitzer	<a href="#">stdout</a> <a href="#">stderr</a>

- To get debugging information, click the `stdout` or `stderr` links in the Logs column.

## Application: Spark shell

After starting a Spark context, you can see the status of the worker, which can be useful for debugging. The interface also shows the memory that is required for apps that are running, so you can adjust which apps you run to meet your needs.

**Application: Spark shell**

ID: app-20140314223157-0001  
**Name:** Spark shell  
**User:** automaton  
**Cores:** Unlimited (6 granted)  
**Executor Memory:** 512.0 MB  
**Submit Date:** Fri Mar 14 22:31:57 UTC 2014  
**State:** RUNNING  
[Application Detail UI](#)

**Executor Summary**

ExecutorID	Worker	Cores	Memory	State	Logs
2	<a href="#">worker-20140314184630-10.176.83.32-39164</a>	2	512	RUNNING	<a href="#">stdout stderr</a>
1	<a href="#">worker-20140314184018-10.168.193.41-41345</a>	2	512	RUNNING	<a href="#">stdout stderr</a>
0	<a href="#">worker-20140314181117-10.160.137.165-46838</a>	2	512	RUNNING	<a href="#">stdout stderr</a>

## Spark Stages: Application progress

- To see the progress of applications that are running, click the name of application to see every query that was executed with detailed information about how the data got distributed that might be valuable for debugging.
- On a port, not necessarily port 4040 as shown here, you can view Spark stages.

When you run multiple applications at the same time Spark tries to use subsequent ports starting at 4040, for example 4040, 4041, and so on.

The screenshot shows the Spark shell application UI with the title "Spark shell - Spark Stages". The URL in the browser is "54.193.203.204:4040/stages/". The main content area is titled "Spark Stages" and displays the following statistics:

- Total Duration: 23.1 m
- Scheduling Mode: FIFO
- Active Stages: 0
- Completed Stages: 1
- Failed Stages: 0

Below these statistics, there are three sections: "Active Stages (0)", "Completed Stages (1)", and "Failed Stages (0)". Each section has a table with the following columns: Stage Id, Description, Submitted, Duration, Tasks: Succeeded/Total, Shuffle Read, and Shuffle Write.

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write
0	count at <console>:24	2014/03/14 22:51:20	3.2 s	3/3		

## Spark supported types

This table maps [CQL types](#) to Scala types. All CQL types are supported by the DataStax Enterprise Spark integration. Other type conversions might work, but cause loss of precision or not work for all values. Most types are convertible to strings. You can convert strings that conform to the CQL standard to numbers, dates, addresses or uuids. You can convert maps to or from sequences of key-value tuples.

**Table: Supported types**

CQL Type	Scala Type
ascii	String
bigint	Long
blob	ByteBuffer, Array
boolean	Boolean
counter	Long
decimal	BigDecimal, java.math.BigDecimal
double	Double
float	Float
inet	java.net.InetAddress
int	Int
list	Vector, List, Iterable, Seq, IndexedSeq, java.util.List
map	Map, TreeMap, java.util.HashMap

CQL Type	Scala Type
set	Set, TreeSet, java.util.HashSet
text, varchar	String
timestamp	Long, java.util.Date, java.sql.Date, org.joda.time.DateTime
timeuuid	java.util.UUID
uuid	java.util.UUID
varint	BigInt, java.math.BigInteger
nullable values	Option

## Using Spark SQL to query data

You use [Spark SQL](#) to query data that is stored in Cassandra clusters, and execute the queries using Spark. Typically, queries run faster in Spark SQL than in Hive.

### Spark SQL basics

In DataStax Enterprise, Spark SQL allows you to perform relational queries over data stored in Cassandra clusters, and executed using Spark. Spark SQL is a unified relational query language for transversing over Spark Resilient Distributed Datasets (RDDs), and supports a variation of the SQL language used in relational databases. It is intended as a replacement for Shark and Hive, including the ability to run Hive QL queries over RDDs. You can use traditional Spark applications in conjunction with Spark SQL queries to analyze large data sets.

The `SqlContext` class and its subclasses are the entry point for running relational queries in Spark. `SqlContext` instances are created from a `SparkContext` instance. The `CassandraSQLContext` class is a subclass of `SqlContext` and allows you to run these queries against a Cassandra data source.

Spark SQL uses a special type of RDD called `SchemaRDD`, and are similar to tables in a traditional relational database. A `SchemaRDD` consists of object data and a schema that describes the data types of the objects. You can create `SchemaRDD` instances from existing Spark RDDs. Once a `SchemaRDD` has been applied to a `SqlContext`, it can be registered as a table, and SQL queries can be run against it.

### Starting the Spark SQL shell

The Spark SQL shell allows you to interactively perform Spark SQL queries. To start the shell, run `dse spark-sql`:

```
$ dse spark-sql
```

For more information on Spark SQL, see the migration information in the [Spark documentation](#).

### Querying Cassandra data using Spark SQL in Scala

When you start Spark, DataStax Enterprise sets the context to allow you to run Spark SQL queries against Cassandra tables. Use the `setKeyspace` method to connect to a Cassandra keyspace, and then use the `sql` method to execute the query.

### Procedure

1. Start the Spark shell.

```
$ dse spark
```

2. Set the keyspace you'd like to query using the `setKeyspace` method.

```
csc.setKeyspace ("my_keyspace_name")
```

3. Use the `sql` method to pass in the query, storing the result in a variable.

```
val results = csc.sql("SELECT * from my_keyspace_name.my_table")
```

4. Use the returned data.

```
results.collect().foreach(println)
```

```
CassandraRow{type_id: 1, value: 9685.807}
CassandraRow{type_id: 2, value: -9775.808}
```

## Querying Cassandra data using Spark SQL in Java

Java applications that query Cassandra data using Spark SQL first need a Spark configuration instance and Spark context instance.

The Spark context object is used to create a Cassandra-aware Spark SQL context object to connect to Cassandra. We recommend using a `HiveContext` instance, as `HiveContext` is a superset of `SQLContext` and allows you to write more complicated queries using HiveQL. Create an instance of `org.apache.spark.sql.hive.HiveContext` using the `JavaSparkContext` object.

Create the Spark configuration object and Spark context:

```
// create a new configuration
SparkConf conf = new SparkConf()
 .setAppName("My application");
// create a Spark context
JavaSparkContext sc = new JavaSparkContext(conf);
HiveContext hiveContext = new HiveContext(sc.toSparkContext(sc));
```

After the Spark context is created, you can use it to create a `DataFrame` instance from the query. Queries are executed by calling the `SparkContext.sql` method.

```
DataFrame employees = hiveContext.sql("SELECT * FROM company.employees");
employees.registerTempTable("employees");
DataFrame managers = hiveContext.sql("SELECT name FROM employees WHERE role
== 'Manager' ");
```

The returned `DataFrame` object supports the standard Spark operations.

```
employees.collect();
```

## Supported syntax of Spark SQL

The following syntax defines a SELECT query.

```
SELECT [DISTINCT] [column names] | [wildcard]
FROM [keyspace name.]table name
[JOIN clause table name ON join condition]
[WHERE condition]
[GROUP BY column name]
[HAVING conditions]
[ORDER BY column names [ASC | DSC]]
```

A SELECT query using joins has the following syntax.

```
SELECT statement
FROM statement
```

```
[JOIN | INNER JOIN | LEFT JOIN | LEFT SEMI JOIN | LEFT OUTER JOIN | RIGHT
JOIN | RIGHT OUTER JOIN | FULL JOIN | FULL OUTER JOIN]
ON join condition
```

Several select clauses can be combined in a UNION, INTERSECT, or EXCEPT query.

```
SELECT statement 1
[UNION | UNION ALL | UNION DISTINCT | INTERSECT | EXCEPT]
SELECT statement 2
```

**Note:** Select queries run on new columns return ' ', or empty results, instead of None.

The following syntax defines an INSERT query.

```
INSERT [OVERWRITE] INTO [keyspace name.] table name [(columns)]
VALUES values
```

The following syntax defines a CACHE TABLE query.

```
CACHE TABLE table name [AS table alias]
```

You can remove a table from the cache using a UNCACHE TABLE query.

```
UNCACHE TABLE table name
```

## Keywords in Spark SQL

The following keywords are reserved in Spark SQL.

- ALL
- AND
- AS
- ASC
- APPROXIMATE
- AVG
- BETWEEN
- BY
- CACHE
- CAST
- COUNT
- DESC
- DISTINCT
- FALSE
- FIRST
- LAST
- FROM
- FULL
- GROUP
- HAVING
- IF
- IN
- INNER
- INSERT
- INTO
- IS

```
JOIN
LEFT
LIMIT
MAX
MIN
NOT
NULL
ON
OR
OVERWRITE
LIKE
RLIKE
UPPER
LOWER
REGEXP
ORDER
OUTER
RIGHT
SELECT
SEMI
STRING
SUM
TABLE
TIMESTAMP
TRUE
UNCACHE
UNION
WHERE
INTERSECT
EXCEPT
SUBSTR
SUBSTRING
SQRT
ABS
```

## Running HiveQL queries using Spark SQL

Spark SQL supports queries written using HiveQL, a SQL-like language that produces queries that are converted to Spark jobs. HiveQL is more mature and supports more complex queries than Spark SQL. To construct a HiveQL query, first create a new `HiveContext` instance, and then submit the queries by calling the `sql` method on the `HiveContext` instance.

See the [Hive Language Manual](#) for the full syntax of HiveQL.

**Note:** Creating indexes with `DEFERRED REBUILD` is not supported in Spark SQL.

## Procedure

1. Start the Spark shell.

```
$ bin/dse spark
```

2. Use the provided `HiveContext` instance `hc` to create a new query in HiveQL by calling the `sql` method on the `hc` object..

```
scala> val results = hc.sql("SELECT * FROM my_keyspace.my_table")
```

## Getting started with Spark Streaming

[Spark Streaming](#) allows you to consume live data streams from sources, including Akka, Kafka, and Twitter. This data can then be analyzed by Spark applications, and the data can be stored in Cassandra.

You use Spark Streaming by creating a `org.apache.spark.streaming.StreamingContext` instance based on your Spark configuration. You then create a `DStream` instance, or a *discretized stream*, an object that represents an input stream. `DStream` objects are created by calling one of the methods of `StreamingContext`, or using a utility class from external libraries to connect to other sources like Twitter.

The data you consume and analyze is saved to Cassandra by calling one of the `saveToCassandra` methods on the stream object, passing in the keyspace name, the table name, and optionally the column names and batch size.

### Procedure

The following Scala example demonstrates how to connect to a text input stream at a particular IP address and port, count the words in the stream, and save the results to Cassandra.

1. Create a new `StreamingContext` object based on an existing `SparkConf` configuration object, specifying the interval in which streaming data will be divided into batches by passing in a batch duration.

```
val sparkConf =
val ssc = new StreamingContext(sc, Seconds(1)) // Uses the context
automatically created by the spark shell
```

Spark allows you to specify the batch duration in milliseconds, seconds, and minutes.

2. Import the Cassandra-specific functions for `StreamingContext`, `DStream`, and `RDD` objects.

```
import com.datastax.spark.connector.streaming._
```

3. Create the `DStream` object that will connect to the IP and port of the service providing the data stream.

```
val lines = ssc.socketTextStream(server IP address, server port number)
```

4. Count the words in each batch and save the data to the Cassandra table.

```
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)
 .saveToCassandra("streaming_test", "words_table", SomeColumns("word",
"count"))
```

5. Start the computation.

```
ssc.start()
ssc.awaitTermination()
```

## Example

In the following example, you start a service using the `nc` utility that repeats strings, then consume the output of that service using Spark Streaming.

Using cqlsh, start by creating a target keyspace and table for streaming to write into.

```
CREATE KEYSPACE IF NOT EXISTS streaming_test
WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1 }
CREATE TABLE IF NOT EXISTS streaming_test.words_table
(word TEXT PRIMARY KEY, count COUNTER) session.execute(s
"TRUNCATE streaming_test.words_table"
```

In a terminal window, enter the following command to start the service:

```
$ nc -lk 9999
one two two three three three four four four four someword
```

In a different terminal start a Spark shell.

```
$ bin/dse spark
```

In the Spark shell enter the following:

```
val ssc = new StreamingContext(sc, Seconds(1))
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))

val wordCounts = pairs.reduceByKey(_ + _)
wordCounts.saveToCassandra("streaming_test", "words_table",
 SomeColumns("word", "count"))
wordCounts.print()
ssc.start()
ssc.awaitTermination()
exit()
```

Using cqlsh connect to the streaming\_test keyspace and run a query to show the results.

```
$ cqlsh -k streaming_test
cqlsh:streaming_test> select * from words_table;
```

word	count
three	3
one	1
two	2
four	4
someword	1

## What to do next

See the [Spark Streaming Programming Guide](#) for more information, API documentation, and examples.

## Getting started with the Spark Cassandra Connector Java API

The [Spark Cassandra Connector](#) Java API allows you to create Java applications that use Spark to analyze Cassandra data.

## Using the Java API in SBT build files

Add the following library dependency to the `build.sbt` or other SBT build file.

```
libraryDependencies += "com.datastax.spark" %% "spark-cassandra-connector-
java_2.10" % "1.4.0" withSources() withJavadoc()
```

## Using the Java API in Maven build files

Add the following dependencies to the `pom.xml` file:

```
<dependencies>
 <dependency>
 <groupId>com.datastax.spark</groupId>
 <artifactId>spark-cassandra-connector-java_2.10</artifactId>
 <version>1.4.1</version>
 </dependency>
 ...
</dependencies>
```

To use the helper classes included in `dse.jar` in your applications, copy `dse.jar` to `project_directory/lib` and add the following dependency to your `pom.xml` file:

```
<dependency>
 <groupId>com.datastax</groupId>
 <artifactId>dse</artifactId>
 <version>version number</version>
 <scope>system</scope>
 <systemPath>${project.basedir}/lib/dse-version number.jar</systemPath>
</dependency>
```

Alternately, you can manually install `dse.jar` in your local repository.

```
$ mvn install:install-file -Dfile=path/dse-version number.jar -
-DgroupId=com.datastax -DartifactId=dse -Dversion=version number -
Dpackaging=jar
```

And then add the dependency to `pom.xml`:

```
<dependency>
 <groupId>com.datastax</groupId>
 <artifactId>dse</artifactId>
 <version>version number</version>
</dependency>
```

## Accessing Cassandra data in Java applications

To perform Spark actions on Cassandra table data, you first obtain a `CassandraJavaRDD` object, a subclass of the `JavaRDD` class. The `CassandraJavaRDD` is the Java language equivalent of the `CassandraRDD` object used in Scala applications.

To create the `CassandraJavaRDD` object, create a Spark configuration object, which is then used to create a Spark context object.

```
SparkConf conf = new SparkConf()
 .setAppName("My application");
SparkContext sc = new SparkContext(conf);
```

Use the static methods of the `com.datastax.spark.connector.japi.CassandraJavaUtil` class to get and manipulate `CassandraJavaRDD` instances. To get a new `CassandraJavaRDD` instance, call

one of the `javaFunctions` methods in `CassandraJavaUtil`, pass in a context object, and then call the `cassandraTable` method and pass in the keyspace, table name, and mapping class.

```
JavaRDD<String> cassandraRdd = CassandraJavaUtil.javaFunctions(sc)
 .cassandraTable("my_keyspace",
 "my_table", .mapColumnTo(String.class))
 .select("my_column");
```

## Mapping Cassandra column data to Java types

You can specify the Java type of a single column from a table row by specifying the type in when creating the `CassandraJavaRDD<T>` instance and calling the `mapColumnTo` method and passing in the type. Then call the `select` method to set the column name in Cassandra.

```
JavaRDD<Integer> cassandraRdd = CassandraJavaUtil.javaFunctions(sc)
 .cassandraTable("my_keyspace",
 "my_table", .mapColumnTo(Integer.class))
 .select("column1");
```

`JavaBeans` classes can be mapped using the `mapRowTo` method. The `JavaBeans` property names should correspond to the column names following the default mapping rules. For example, the `firstName` property will map by default to the `first_name` column name.

```
JavaRDD<Person> personRdd = CassandraJavaUtil.javaFunctions(sc)
 .cassandraTable("my_keyspace", "my_table",
 mapRowTo(Person.class));
```

`CassandraJavaPairRDD<T, T>` instances are extensions of the `JavaPairRDD` class, and have mapping readers for rows and columns similar to the previous examples. These pair RDDs typically are used for key/value pairs, where the first type is the key and the second type is the value.

When mapping a single column for both the key and the value, call `mapColumnTo` and specify the key and value types, then the `select` method and pass in the key and value column names.

```
CassandraJavaPairRDD<Integer, String> pairRdd =
CassandraJavaUtil.javaFunctions(sc)
 .cassandraTable("my_keyspace", "my_table",
 mapColumnTo(Integer.class), mapColumnTo(String.class))
 .select("id", "first_name");
```

Use the `mapRowTo` method to map row data to a Java type. For example, to create a pair RDD instance with the primary key and then a `JavaBeans` object:

```
CassandraJavaPairRDD<Integer, Person> idPersonRdd =
CassandraJavaUtil.javaFunctions(sc)
 .cassandraTable("my_keyspace", "my_table",
 mapColumnTo(Integer.class), mapRowTo(Person.class))
 .select("id", "first_name", "last_name", "birthdate", "email");
```

## Saving data to Cassandra

To save data from an RDD to Cassandra call the `writerBuilder` method on the `CassandraJavaRDD` instance, passing in the keyspace, table name, and optionally type mapping information for the column or row.

```
CassandraJavaUtil.javaFunctions(personRdd)
 .writerBuilder("my_keyspace", "my_table",
 mapToRow(Person.class)).saveToCassandra();
```

# Using the DataFrames API with PySpark

Spark 1.4 and DataStax Enterprise 4.8 integration with PySpark works more efficiently with the [DataFrames API](#) (for Spark 1.4.1).

**Note:** The DataStax Enterprise wrappers for Spark Cassandra Connector Scala functions are deprecated.

PySpark and DSE PySpark are supported using the more efficient [DataFrames API](#) to manipulate data within Spark. The Spark Cassandra Connector provides an integrated [DataSource](#) to simplify creating Cassandra DataFrames. For more technical details, see the [Spark Cassandra Connector documentation](#) that is maintained by DataStax and the [Cassandra and PySpark DataFrames](#) post.

## PySpark prerequisites

The prerequisites for starting PySpark are:

- Python 2.6 or later
- [Start a DataStax Enterprise node in Spark mode.](#)

## Examples of using DataFrames API

This example shows using the [DataFrames API](#) to read from the Cassandra table `ks.kv` and insert into a different Cassandra table `ks.othertable`.

```
table1 =
 sqlContext.read.format("org.apache.spark.sql.cassandra").options(table="kv",
 keyspace="ks").load()
table1.write.format("org.apache.spark.sql.cassandra").options(table="othertable",
 keyspace = "ks").save(mode ="append")
```

## Run a Python script using dse spark-submit and DataFrames

You run a Python script using the `spark-submit` command. For example, create the following file and save it as `standalone.py`:

```
#standalone.py

from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext

conf = SparkConf().setAppName("Stand Alone Python Script")
sc = SparkContext(conf=conf)
sqlContext = SQLContext(sc)
sqlContext.read.format("org.apache.spark.sql.cassandra").options(table="kv",
 keyspace="ks").load().show()
```

DataStax Enterprise sets the `cassandra.connection.host` environment variable, eliminating the need to set the variable in the Python file. On Linux, for example, from the installation directory, execute `standalone.py` as follows:

```
$ bin/dse spark-submit /path/standalone.py
```

# Using the Spark SQL Thrift server

The Spark SQL Thrift server uses [JDBC and ODBC interfaces](#) for client connections to Cassandra.

When reading or writing large amounts of data, DataStax recommends using Cassandra-backed DataFrames to enable the use of the [Spark Cassandra Connector](#) to leverage the benefits of the tuning parameters that come with it.

## Procedure

- In the `hive-site.xml` file, configure Cassandra authentication credentials for the Spark SQL Thrift server. Ensure that you use the `hive-site.xml` file in the Spark directory:

Installer-Services and Package installations	<code>/etc/dse/spark/hive-site.xml</code>
Installer-No Services and Tarball installations	<code>install_location/resources/spark/conf/hive-site.xml</code>

- Start DataStax Enterprise with Spark enabled as a [service](#) or in a [standalone](#) installation.

**Note:** To run index queries, start the node with both Spark and Hadoop enabled. Running in this mode is experimental and not supported.

- Start the server by entering the `dse spark-sql-thriftserver start` command as a user with permissions to write to the Spark directories.

To override the default settings for the server, pass in the configuration property using the `--hiveconf` option. See the [HiveServer2 documentation](#) for a complete list of configuration properties.

```
$ dse spark-sql-thriftserver start
```

By default, the server listens on port 10000 on the localhost interface on the node from which it was started. You can specify the server to start on a specific port. For example, to start the server on port 10001, use the `--hiveconf hive.server2.thrift.port=10001` option. You can configure the port and bind address in `resources/spark/conf/spark-env.sh`: `HIVE_SERVER2_THRIFT_PORT`, `HIVE_SERVER2_THRIFT_BIND_HOST`.

```
$ dse spark-sql-thriftserver start --hiveconf hive.server2.thrift.port=10001
```

- Use Cassandra-backed DataFrames to read and write large volumes of data. For example, to create the `table_a_cass_df` table that uses a Cassandra-backed DataFrame while referencing `table_a`:

```
CREATE TABLE table_a_cass_df using org.apache.spark.sql.cassandra OPTIONS
 (table "table_a", keyspace "ks")
```

**Note:** With Cassandra-backed DataFrames, compatibility issues exist with `UUID` and `Inet` types when inserting data with the JDBC driver.

- Use the [Spark Cassandra Connector](#) tuning parameters to optimize reads and writes.
- To stop the server, enter the `dse spark-sql-thriftserver stop` command.

```
$ dse spark-sql-thriftserver stop
```

## What to do next

You can now connect your application by using JDBC to the server at the URL:  
`jdbc:hive2://hostname:port number` or use [dse spark-beeline](#).

## SparkSQL inserting into tables with static columns

Static columns are mapped to different columns in SparkSQL and Hive and require special handling. When you run insert query, you must pass data to those columns.

To work around the different columns, set `cql3.output.query` in the insertion hive table properties to limit the columns that are being inserted. In SparkSQL or Hive, alter the external table to configure the prepared statement as the value of the Hive CQL output query. For example, this prepared statement takes values that are inserted into columns `a` and `b` in `mytable` and maps these values to columns `b` and `a`, respectively, for insertion into the new row.

```
spark-sql> ALTER TABLE mytable SET TBLPROPERTIES ('cql3.output.query' =
 'update
 mykeyspace.mytable set b = ? where a = ?');
spark-sql> ALTER TABLE mytable SET SERDEPROPERTIES ('cql3.update.columns' =
 'b,a');
```

## Enabling Spark apps in cluster mode when authentication is enabled

You must enable Spark applications in cluster mode when JAR files are on the Cassandra file system (CFS) and authentication is enabled. When the application is submitted in cluster mode and the JAR files are on the Cassandra File System (CFS), the Spark Worker process is responsible for obtaining the required JAR file. When authentication is required, the Spark Worker process requires the authentication credentials to CFS. The Spark Worker will start executors for unrelated Spark jobs, so giving the Spark Worker process credentials enables all future Spark jobs to pull JAR files from CFS for their dependencies. Credentials that are granted to the Spark Worker must be considered "shared" among all submitted applications, regardless of the submitting user. Shared credentials do not apply to accessing CFS from the application code.

### Procedure

1. To enable Spark applications in cluster mode when JAR files are on the Cassandra file system (CFS) and authentication is enabled, do one of the following:

- Add this statement to the spark-env.sh on every DataStax Enterprise node:

```
SPARK_WORKER_OPTS="$SPARK_WORKER_OPTS -
 -Dspark.hadoop.cassandra.username=username
 -Dspark.hadoop.cassandra.password=password"
```

- Before you start the DataStax Enterprise server process, set the SPARK\_WORKER\_OPTS environment variable in a way that guarantees visibility to DataStax Enterprise server processes.

This environment variable does not need to be passed to applications that are submitted with the `dse spark` or `dse spark-submit` commands.

2. Follow these best practices:

- Create a unique user with privileges only on CFS (access to related CFS keyspace), and then use the unique user credentials for the Spark Worker authentication. This best practice limits the amount of protected information in the Cassandra database that is accessible through user Spark Jobs without explicit permission.
- Create a distinct CFS directory and limit the directory access privileges to read only.

## Setting decimal precision in SparkSQL

SparkSQL uses Hive 0.13 which supports specifying scale and precision when creating tables with the DECIMAL data type.

Use `DECIMAL(precision, scale)` syntax to specify precision and scale. For example, to specify a precision of 8 and a scale of 3 when creating tables with the DECIMAL data type:

```
CREATE TABLE test_decimal_table (pkey DECIMAL(8,3), ckey1 DECIMAL(8,3),
 data1 DECIMAL(8,3)) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS
 TEXTFILE;
```

When scale is not specified, the default scale is 0 with no fractional digits.

When precision is not specified, the default precision is 10.

## Using the Spark Jobserver

DataStax Enterprise includes a bundled copy of the open-source Spark [Jobserver](#), an optional component for submitting and managing Spark jobs, Spark contexts, and JARs on DSE Analytics clusters. See the [release notes](#) to find the version of the Spark Jobserver included in this version of DSE.

Valid [spark-submit options](#) are supported and can be applied to the Spark Jobserver. To use the Jobserver:

```
$ dse spark-jobserver start [any_spark_submit_options] //Start the job server
$ dse spark-jobserver stop //Stop the job server
```

The default location of the Spark Jobserver depends on the type of installation:

Installer-Services and Package installations	/usr/share/dse/resources/spark/spark-jobserver
Installer-No Services and Tarball installations	<i>install_location/resources/spark/spark-jobserver</i>

**Note:** For Installer-Services and Package installations, make sure that the OS user which runs the Jobserver has permission to write into the Spark logs directory (/var/log/spark). This could be accomplished by either starting the Jobserver with `sudo` or by changing the access controls on /var/log/spark. The Jobserver log file is located in /var/log/spark/job-server/spark-jobserver.log.

Beneficial use cases for the Spark Jobserver include sharing cached data, repeated queries of cached data, and faster job starts.

For an example of how to create and submit an application through the Spark Jobserver, see the `spark-jobserver` demo included with DSE.

The default location of the `demos` directory depends on the type of installation:

Installer-Services and Package installations	/usr/share/dse/demos
Installer-No Services and Tarball installations	<i>install_location/demos</i>

## Spark examples

DSE includes Spark example applications that demonstrate different Spark features.

### Portfolio Manager demo using Spark

The Portfolio Manager demo runs an application that is based on a financial use case. You run scripts that create a portfolio of stocks. On the Cassandra OLTP (online transaction processing) side, each portfolio contains a list of stocks, the number of shares purchased, and the purchase price. The demo's pricer utility simulates real-time stock data. Each portfolio gets updated based on its overall value and the percentage of gain or loss compared to the purchase price. The utility also generates 100 days of historical market data (the end-of-day price) for each stock. On the DSE OLAP (online analytical processing) side, a Spark Scala job calculates the greatest historical 10 day loss period for each portfolio, which is an indicator of the risk associated with a portfolio. This information is then fed back into the real-time application to allow customers to better gauge their potential losses.

### Procedure

To run the demo:

**Note:** DataStax Demos do not work with either LDAP or internal authorization (username/password) enabled.

1. Install a single Demo node using the DataStax Installer in [GUI or Text](#) mode with the following settings:

- **Install Options page - Default Interface: 127.0.0.1** (You must use this IP for the demo.)
- **Node Setup page - Node Type: Analytics**
- **Analytic Node Setup page - Analytics Type: Spark + Integrated Hadoop**

2. Start DataStax Enterprise if you haven't already:

- **Installer-Services and Package installations:**

```
$ sudo service dse start
```

- **Installer-No Services and Tarball installations:**

```
$ install_location/bin/dse cassandra -k ## Starts node in Spark mode
```

The default *install\_location* is /usr/share/dse.

3. Go to the Portfolio Manager demo directory.

The default location of the Portfolio Manager demo depends on the type of installation:

Installer-Services and Package installations	/usr/share/dse/demos/portfolio_manager
Installer-No Services and Tarball installations	<i>install_location</i> /demos/portfolio_manager

4. Run the bin/pricer utility to generate stock data for the application:

- To see all of the available options for this utility:

```
$ bin/pricer --help
```

- Start the pricer utility:

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

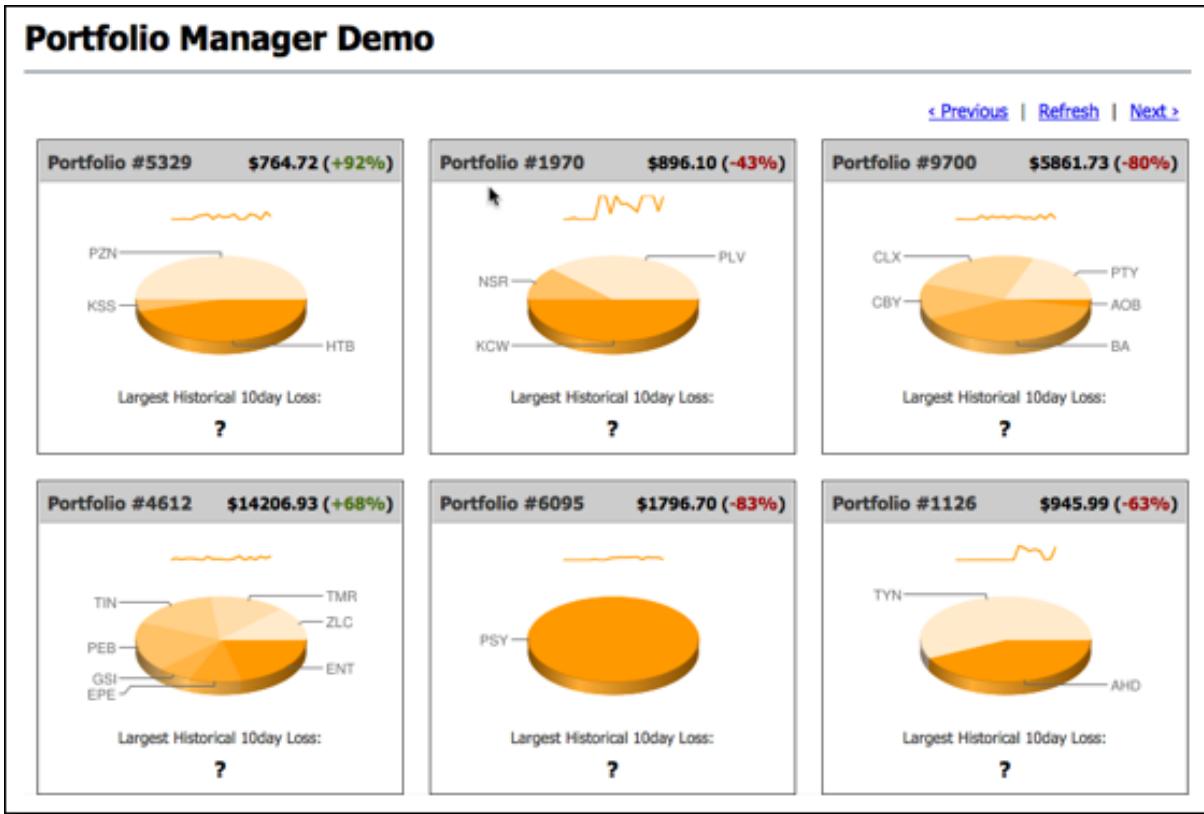
The pricer utility takes several minutes to run.

5. Start the web service:

```
$ cd website
$ sudo ./start
```

6. Open a browser and go to <http://localhost:8983/portfolio>.

The real-time Portfolio Manager demo application is displayed.

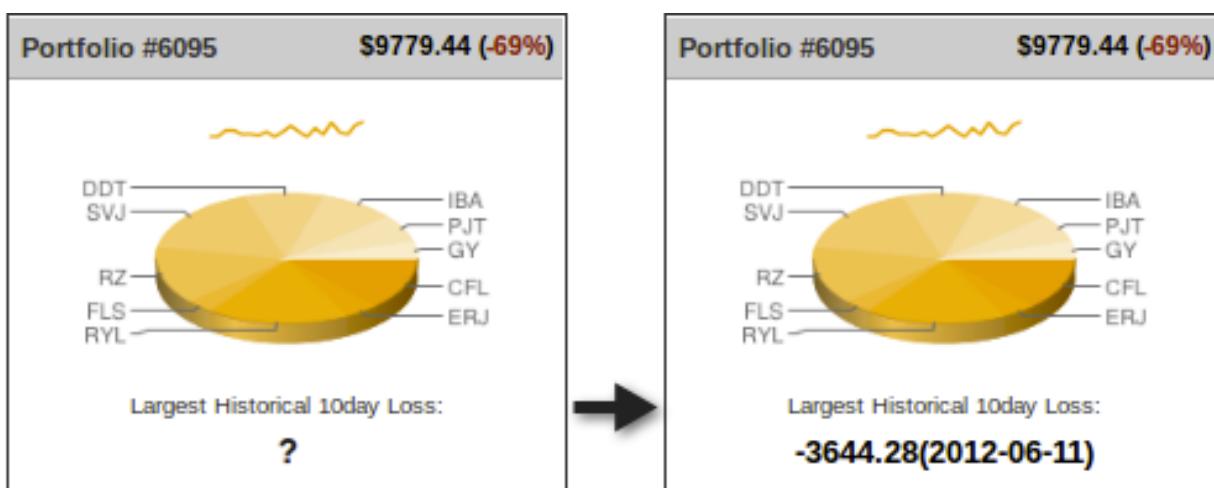


7. Open another terminal.
8. Start Spark by running the `10-day-loss.sh` script.
  - Installer-Services: `$ cd /usr/share/dse/demos/spark; ./10-day-loss.sh`
  - Package installations: `$ cd /usr/share/dse-demos/spark; ./10-day-loss.sh`
  - Installer-No Services and Tarball installations: `$ install_location/demos/spark/10-day-loss.sh`

The Spark application takes several minutes to run.

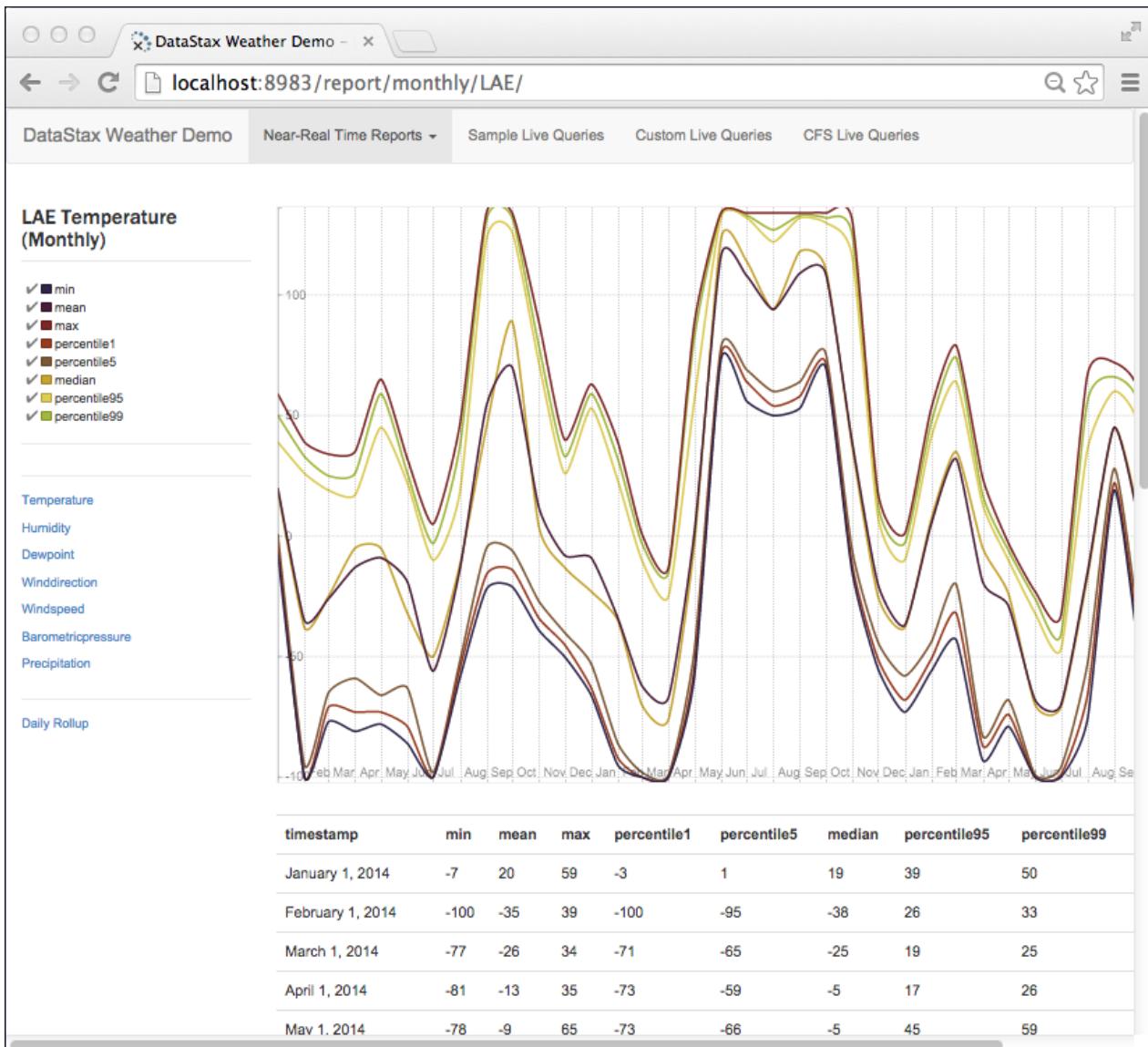
9. After the job completes, refresh the **Portfolio Manager** web page.

The results of the Largest Historical 10 day Loss for each portfolio are displayed.



## Running the Weather Sensor demo

Using the Weather Sensor demo, you can compare how long it takes to run Hive versus Spark SQL queries against aggregated data for a number of weather sensors in various cities. For example, you can view reports using different metrics, such as temperature or humidity, and get a daily roll up.



You run customize Spark SQL or Hive queries using different metrics and different dates. In addition to querying CQL tables, you time Spark SQL and Hive queries against data in the Cassandra File System (CFS).

**Note:** DataStax Demos do not work with either LDAP or internal authorization (username/password) enabled.

## Prerequisites

Before running the demo, install the following source code and tools if you do not already have them:

- Python 2.7

- Debian and Ubuntu

```
$ sudo apt-get install python2.7-dev
```

- RedHat or CentOS

```
$ sudo yum install python27
```

- Mac OS X already has Python 2.7 installed.

- pip installer tool

- Debian and Ubuntu

```
$ sudo apt-get install python-pip
```

- RedHat or CentOS

```
$ sudo yum install python-pip
```

- Mac OS X

```
$ sudo easy_install pip
```

- The libsasl2-dev package

- Debian and Ubuntu

```
$ sudo apt-get install libsasl2-dev
```

- The required Python packages:

- The

If you installed DataStax Enterprise using a tarball or the GUI-no services option, set the PATH environment variable to the DataStax Enterprise installation /bin directory.

```
export PATH=$PATH:install_location/bin
```

You must have

## Start DataStax Enterprise and import data

You start DataStax Enterprise in Spark and Hadoop mode, and then run a script that creates the schema for weather sensor data model. The script also imports aggregated data from CSV files into Cassandra CQL tables. The script uses the hadoop fs command to put the CSV files into the Cassandra File System.

1. Start DataStax Enterprise in [Hadoop and Spark mode](#).

2. Run the create-and-load CQL script in the weather\_sensors/resources directory. On Linux, for example:

```
$ cd install_location/demos/weather_sensors
$ bin/create-and-load
```

The output confirms that the script imported the data into CQL and copied files to CFS.

```
...
10 rows imported in 0.019 seconds.
2590 rows imported in 2.211 seconds.
76790 rows imported in 33.522 seconds.
+ echo 'Copy csv files to Hadoop...'
Copy csv files to Hadoop...
+ dse hadoop fs -mkdir /datastax/demos/weather_sensors/
```

If an error occurs, set the PATH as described in [Prerequisites](#), and retry.

## Starting the Spark SQL Thrift server and Hive

**Note:** Support for Shark is removed. Use [Spark SQL](#) instead.

You start the Spark SQL Thrift server and Hive services on specific ports to avoid conflicts. Start these services using your local user account. Do not use sudo.

1. Start the Spark SQL Thrift server on port 5588. On Linux, for example:

```
$ cd install_location
$ bin/dse start-spark-sql-thriftserver --hiveconf
hive.server2.thrift.port=5588
```

2. Open a new terminal and start the Hive service in DSE on port 5587.

```
$ bin/dse hive --service hiveserver2 --hiveconf
hive.server2.thrift.port=5587
```

If you see a warning message saying, "The blist library is not available, so a pure python list-based set will," ignore it.

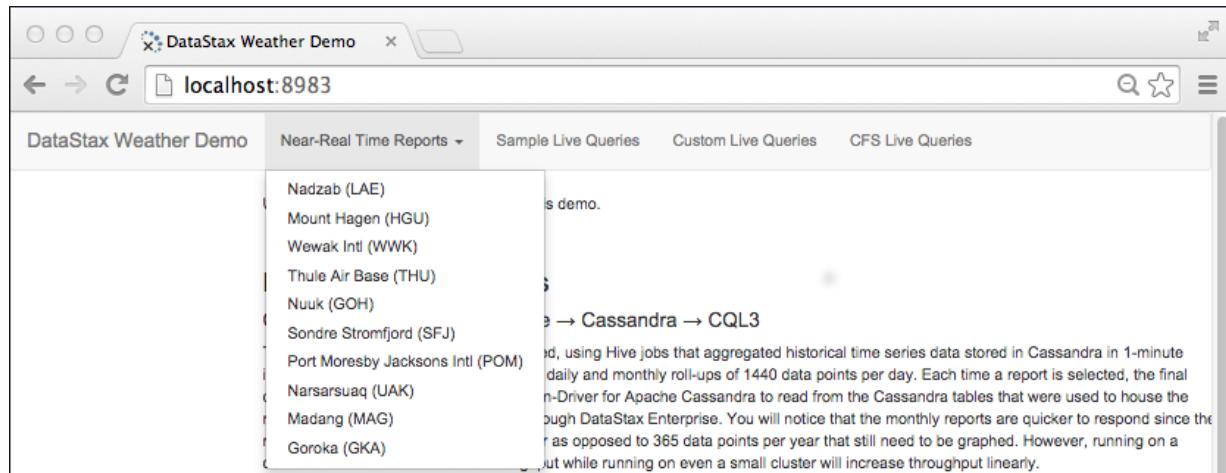
## Start the web app and query the data

1. Open another terminal and start the Python service that controls the web interface:

```
$ cd install_location/demos/weather_sensors
$ python web/weather.py
```

2. Open a browser and go to the following URL: <http://localhost:8983/>

The weather sensors app appears. Select **Near Real-Time Reports** on the horizontal menu. A drop-down listing weather stations appears:



3. Select a weather station from the drop-down, view the graph, and select different metrics from the vertical menu on the left side of the page.
4. On the horizontal menu, click **Sample Live Queries**, then select a sample script. Click the **Spark SQL** button, then click Submit.

The time spent loading results using Spark appears.

The screenshot shows the DataStax Weather Demo interface. At the top, there's a navigation bar with tabs: DataStax Weather Demo, Near-Real Time Reports, Sample Live Queries, Custom Live Queries, and CFS Live Queries. Below the navigation bar, there's a section titled "Select Sample Script" containing several options. One option, "Find correlation of median temperatures between two locations on a daily scale", is highlighted with a gray background. To the right of this section is a large text area labeled "Hive Query" containing a complex SQL query for calculating correlations between station A and station B. Below the query, there are two buttons: "Shark" and "Hive". Underneath these buttons is a "Submit Query" button. At the bottom of the interface, a message says "Time spent loading: 12.0928 seconds for 45 records". Below this message is a table with three columns: "station\_a", "station\_b", and "corr\_temperature". The table contains four rows of data.

station_a	station_b	corr_temperature
LAE	GKA	-0.06787906592935322
SFJ	LAE	0.0653329217238257
UAK	HGU	-0.03175732104186981

- Click the Hive button to see the time spent loading results in Hive.

The screenshot shows the DataStax Weather Demo interface with the "Hive Server" tab selected. It features a "Hive Server" section with "Shark" and "Hive" buttons, and a "Submit Query" button. Below this is a text area with instructions about the query execution. It mentions that the query was submitted using a hive driver and will update as soon as it is completed. It notes that typical times for running against the Shark server are under 20 seconds while Hadoop jobs routinely take 50 seconds when using a local, single-node instance. It also notes that data size, computing power, heap space, and cluster size affect these times drastically. Below this text is a message about handling frozen requests. Further down, there's a link to more information on what's happening in the backend. At the bottom of the interface is a digital clock showing the time as 00:19.

- From the horizontal menu, click Custom Live Queries. Click a Week Day, and then a metric, such as Wind Direction. Click Recalculate Query. The query reflects the selections you made.

- From the horizontal menu, click CFS Live Queries. Click Spark SQL. The time spent loading results from CFS using Spark SQL appears.

The screenshot shows the DataStax Weather Demo interface. At the top, there's a navigation bar with tabs: DataStax Weather Demo, Near-Real Time Reports, Sample Live Queries, Custom Live Queries, and CFS Live Queries. The CFS Live Queries tab is active. Below the navigation bar, there's a section titled "Hive Query" containing a complex SQL query. Underneath the query, there are two buttons: "Shark" and "Hive". A "Submit Query" button is located below these buttons. At the bottom of the interface, a message says "Time spent loading: 2.5111 seconds for 120 records".

## Clean up

To remove all generated data, run the following commands:

```
$ cd install_location/demos/weather_sensors
$ bin/cleanup
```

To remove the keyspace from the cluster, run the following command:

```
$ echo "DROP KEYSPACE weathercql;" | cqlsh
```

## Running the Wikipedia demo with SearchAnalytics

The following instructions describe how to use Solr queries in the Spark console on SearchAnalytics nodes using the Wikipedia demo.

### Prerequisites

You must have created a new SearchAnalytics cluster as described in [the single datacenter deployment scenario](#).

### Procedure

- Start the node or nodes in SearchAnalytics mode.
  - Packages/Services: See [Starting DataStax Enterprise as a service](#).
  - Tarball/No Services: See [Starting DataStax Enterprise as a stand-alone process](#).
- Ensure that the cluster is running correctly by running dsetool. The node type should be SearchAnalytics.

```
$ dsetool ring
```

The default location of the dsetool command depends on the type of installation:

Package installations	/usr/bin/dsetool
Installer-Services installations	/usr/bin/dsetool
Installer-No Services and Tarball installations	<i>install_location</i> /bin/dsetool

3. In a terminal, go to the Wikipedia demo directory.

The default wikipedia demo location depends on the type of installation:

Installer-No Services and Tarball installations	<i>install_location/demos/wikipedia</i>
Installer-Services and Package installations	/usr/share/dse/demos/wikipedia

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

4. Add the schema by running the `1-add-schema.sh` script.

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

5. Create the Solr indexes.

```
$./2-index.sh
```

6. Start the Spark console.

```
$ dse spark
```

7. Create an RDD based on the `wiki.solr` table.

```
scala> val table = sc.cassandraTable("wiki","solr")
```

```
table:
```

```
com.datastax.spark.connector.rdd.CassandraTableScanRDD [com.datastax.spark.connector.CassandraTableScanRDD@12345678 at RDD at CassandraRDD.scala:15]
```

8. Run a query using the title Solr index and collect the results.

```
scala> val result =
 table.select("id","title").where("solr_query='title:Boroph*'").collect
```

Equivalent JSON query:

```
where("solr_query='{"q": "title:Boroph*"}'")
```

```
result:
```

```
Array[com.datastax.spark.connector.CassandraRow] = Array(
 CassandraRow{id: 23729958, title: Borophagus parvus},
 CassandraRow{id: 23730195, title: Borophagus dudleyi},
 CassandraRow{id: 23730528, title: Borophagus hilli},
 CassandraRow{id: 23730810, title: Borophagus diversidens},
 CassandraRow{id: 23730974, title: Borophagus littoralis},
 CassandraRow{id: 23731282, title: Borophagus orc},
 CassandraRow{id: 23731616, title: Borophagus pugnator},
 CassandraRow{id: 23732450, title: Borophagus secundus})
```

## What to do next

For details on using Solr query syntax in CQL, see [Using CQL Solr queries in DSE Search](#) on page 195.

## Running the Spark MLlib demo application

The Spark MLlib demo application demonstrates how to run machine-learning analytic jobs using Spark and Cassandra. The demo solves the classic iris flower classification problem, using the [iris flower data set](#). The application will use the iris flower data set to build a Naive Bayes classifier that will recognize a flower based on four feature measurements.

## Prerequisites

We strongly recommend that you install the BLAS library on your machines before running Spark MLlib jobs. For instructions on installing the BLAS library on your platform, see <https://github.com/fommil/netlib-java/blob/master/README.md#machine-optimised-system-libraries>.

The BLAS library is not distributed with DataStax Enterprise due to licensing restrictions, but improves MLlib performance significantly.

You must have the Gradle build tool installed to build the demo. See <https://gradle.org/> for details on installing Gradle on your OS.

## Procedure

1. Start the nodes in Analytics mode.
  - Installer-Services and Package installations: See [Starting DataStax Enterprise as a service](#).
  - Installer-No Services and Tarball installations: See [Starting DataStax Enterprise as a stand-alone process](#).
2. In a terminal, go to the `spark-mllib` directory located in the Spark demo directory.

The default location of the Spark demo depends on the type of installation:

Installer-Services and Package installations	<code>/usr/share/dse/demos/spark</code>
Installer-No Services and Tarball installations	<code>install_location/demos/spark</code>

3. Build the application using the `gradle` build tool.

```
$ gradle
```

4. Use `spark-submit` to submit the application JAR.

The Spark MLlib demo application reads the *Spark demo directory*/`spark-mllib/iris.csv` file on each node. This file must be accessible in the same location on each node. If some nodes do not have the same local file path, set up a shared network location accessible to all the nodes in the cluster.

To run the application where each node has access to the same local location of `iris.csv`.

```
$ dse spark-submit NaiveBayesDemo.jar
```

To specify a shared location of `iris.csv`:

```
$ dse spark-submit NaiveBayesDemo.jar /mnt/shared/iris.csv
```

## Importing a Text File into a CQL Table

This example shows how to use Spark to import a local or CFS (Cassandra File System)-based text file into an existing CQL table. You use the `saveToCassandra` method present in Cassandra RDDs to save arbitrary RDD to Cassandra.

## Procedure

1. Create a keyspace and a CQL table in Cassandra. For example, use `cqlsh`.

```
CREATE KEYSPACE int_ks WITH replication =
 {'class': 'NetworkTopologyStrategy', 'Analytics':1};
USE int_ks;
CREATE TABLE int_compound (pkey int, ckey1 int, data1 int , PRIMARY KEY
 (pkey,ckey1));
```

**2.** Insert data into the table

```
INSERT INTO int_compound (pkey, ckey1, data1) VALUES (1, 2, 3);
INSERT INTO int_compound (pkey, ckey1, data1) VALUES (2, 3, 4);
INSERT INTO int_compound (pkey, ckey1, data1) VALUES (3, 4, 5);
INSERT INTO int_compound (pkey, ckey1, data1) VALUES (4, 5, 1);
INSERT INTO int_compound (pkey, ckey1, data1) VALUES (5, 1, 2);
```

**3.** Create a text file named normalfill.csv that contains this data.

```
6,7,8
7,8,6
8,6,7
```

**4.** Put the CSV file in the CFS. For example, on Linux:

```
$ bin/dse hadoop fs -put mypath/normalfill.csv /
```

**5.** Start the Spark shell.

**6.** Verify that Spark can access the int\_ks keyspace:

```
scala> :showSchema int_ks
=====
 Keyspace: int_ks
=====
 Table: int_compound

 - pkey : Int (partition key column)
 - ckey1 : Int (clustering column)
 - data1 : Int
```

int\_ks appears in the list of keyspaces.

**7.** Read in the file from the CassandraFS, splitting it on the comma delimiter. Transform each element into an Integer.

```
scala> val normalfill = sc.textFile("/normalfill.csv").map(line =>
 line.split(",").map(_.toInt));
normalfill: org.apache.spark.rdd.RDD[Array[Int]] = MappedRDD[2] at map
at console:22
```

Alternatively, read in the file from the local file system.

```
scala> val file = sc.textFile("file:///local-path/normalfill.csv")
file: org.apache.spark.rdd.RDD[String] = MappedRDD[4] at textFile
at console:22
```

**8.** Check that Spark can find and read the CSV file.

```
scala> normalfill.take(1);
res2: Array[Array[Int]] = Array(Array(6, 7, 8))
```

**9.** Save the new data to Cassandra.

```
scala> normalfill.map(line => (line(0), line(1),
 line(2))).saveToCassandra(
 "int_ks", "int_compound", Seq("pkey", "ckey1", "data1"))

scala>
```

The step produces no output.

10. Check that the data was saved in Cassandra using cqlsh.

```
SELECT * FROM int_ks.int_compound;

pkey | ckey1 | data1
-----+-----+-----+
 5 | 1 | 2
 1 | 2 | 3
 8 | 6 | 7
 2 | 3 | 4
 4 | 5 | 1
 7 | 8 | 6
 6 | 7 | 8
 3 | 4 | 5

(8 rows)
```

## Running spark-submit job with internal authentication

This example shows how to run a spark-submit job with internal authentication.

When you use `dse spark-submit` to submit a Spark job, the Spark Master URL and the Spark Cassandra Connection URL are set automatically. Then use Spark Conf to set the application name. For example:

```
package simpleSpark;
import com.datastax.spark.connector.cql.CassandraConnector;
import org.apache.spark.SparkConf;
import org.apache.spark.SparkContext;
import org.apache.spark.api.java.JavaSparkContext;
public interface SparkConfSetup {
 static public SparkConf getSparkConf() {
 return new SparkConf()
 .setAppName("SimpleSpark");
 }
 static public JavaSparkContext getJavaSparkContext() {
 SparkContext sparkContext = new SparkContext(getSparkConf());
 return new JavaSparkContext(sparkContext);
 }
 static public CassandraConnector getCassandraConnector() {
 return CassandraConnector.apply((getSparkConf()));
 }
}
```

1. Clone the source files from [github](#).
2. Install [Apache Maven](#).
3. Add the DataStax Enterprise JAR file to the local repository:

```
mvn install:install-file -Dfile=/usr/share/dse/dse.jar -DgroupId=com.datastax -DartifactId=dse -Dversion=4.8.3 -Dpackaging=jar
```

4. Add maven to the project so that `pom.xml` looks like this:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

 <modelVersion>4.0.0</modelVersion>
 <groupId>BasicSparkDemo</groupId>
 <artifactId>BasicSparkDemo</artifactId>
 <packaging>jar</packaging>
```

```

<version>0.1</version>
<name>BasicSparkDemo</name>
<url>http://www.datastax.com/</url>

<build>
 <plugins>
 <plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</artifactId>
 <version>2.3.2</version>
 <configuration>
 <source>1.8</source>
 <target>1.8</target>
 </configuration>
 </plugin>
 </plugins>
</build>

<dependencies>
 <dependency>
 <groupId>org.apache.spark</groupId>
 <artifactId>spark-core_2.10</artifactId>
 <version>1.2.1</version>
 <scope>provided</scope>
 </dependency>
 <dependency>
 <groupId>com.datastax.spark</groupId>
 <artifactId>spark-cassandra-connector_2.10</artifactId>
 <version>1.2.5</version>
 <scope>provided</scope>
 </dependency>
 <dependency>
 <groupId>com.datastax.spark</groupId>
 <artifactId>spark-cassandra-connector-java_2.10</artifactId>
 <version>1.2.5</version>
 <scope>provided</scope>
 </dependency>
 <dependency>
 <groupId>com.datastax</groupId>
 <artifactId>dse</artifactId>
 <version>4.7.3</version>
 <scope>provided</scope>
 </dependency>
</dependencies>

<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

</project>

```

##### 5. Edit the code to create the SparkConf object:

```

package simpleSpark;

import com.datastax.spark.connector.cql.CassandraConnector;
import org.apache.spark.SparkConf;
import org.apache.spark.SparkContext;
import org.apache.spark.api.java.JavaSparkContext;
import com.datastax.bdp.spark.DseSparkConfHelper;

/**
 * When you submit a Spark Job using dse spark-submit it automatically
 * sets the Spark Master URL and the Spark Cassandra Connection URL.

```

```

 * The Spark Conf then just needs to set the app name.
 */
public interface SparkConfSetup {

 static public SparkConf getSparkConf() {
 return DseSparkConfHelper.enrichSparkConf(new SparkConf()
 .setAppName("SimpleSpark")
);
 }

 static public JavaSparkContext getJavaSparkContext() {
 SparkContext sparkContext = new SparkContext(getSparkConf());
 return new JavaSparkContext(sparkContext);
 }

 static public CassandraConnector getCassandraConnector() {
 return CassandraConnector.apply((getSparkConf()));
 }
}

```

## 6. Build the package with Maven:

```
mvn clean package
```

## 7. To use internal authentication with spark-submit, specify the user name and password to authenticate against the configured Cassandra authentication schema:

```
dse -u cassandra -p cassandra spark-submit --class
simpleSpark.SparkWordCount ./target/BasicSparkDemo-0.1.jar
```

# Analyzing data using DSE Hadoop

## About DSE Hadoop

You can run analytics on Cassandra data using Hadoop, which is integrated into DataStax Enterprise. The Hadoop component in DataStax Enterprise is not meant to be a full Hadoop distribution, but rather enables analytics to be run across DataStax Enterprise's distributed, shared-nothing architecture. Instead of using the Hadoop Distributed File System (HDFS), DataStax Enterprise uses Cassandra File System (CFS) keyspaces for the underlying storage layer. This provides replication, data location awareness, and takes full advantage of Cassandra's peer-to-peer architecture. DSE Hadoop uses an embedded Apache Hadoop 1.0.4 to eliminate the need to install a separate Hadoop cluster. This is the fastest and easiest option for analyzing Cassandra data using Hadoop.

Unless using [DSE Analytics and Search integration \(experimental\)](#) on page 74, DSE Hadoop workloads are isolated from other workloads that might run in your cluster, Cassandra and Search, never accessing nodes outside of the Analytics datacenter. Therefore, you can run heavy data analysis without affecting performance of your realtime-transactional system.

DataStax Enterprise supports [internal authentication](#) for analyzing data using the following Hadoop components:

- [MapReduce](#)
- [Hive](#) for running HiveQL queries on Cassandra data
- [Pig](#) for exploring very large data sets
- [Apache Mahout](#) for machine learning applications

To get started using DSE Hadoop, run the [Portfolio Manager demo](#).

DataStax Enterprise turns off virtual nodes (vnodes) by default. Before turning vnodes on, understand the [implications of doing so](#).

## Performance enhancement

DataStax Enterprise optimizes performance reading MapReduce files in the Cassandra File System (CFS) by storing files in the page cache, making the files available on the next read.

## Starting a DSE Hadoop node

The way you start up a DSE Hadoop node depends on the type of installation:

- **Installer-Services and Package installations:**

1. Enable Hadoop mode by setting `HADOOP_ENABLED=1` in `/etc/default/dse`.
2. Use this command to start the service:

```
$ sudo service dse start
```

- **Installer-No Services and Tarball installations:**

From the installation directory:

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

## Stopping a DSE Hadoop node

The way you stop a DSE Hadoop node depends on the type of installation:

- **Installer-No Services and Tarball installations:**

1. From the install directory:

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

2. Check that the dse process has stopped.

```
$ ps auwx | grep dse
```

If the dse process stopped, the output should be minimal, for example:

```
jdoe 12390 0.0 0.0 2432768 620 s000 R+ 2:17PM 0:00.00 grep dse
```

If the output indicates that the dse process is not stopped, rerun the `cassandra-stop` command using the process ID (PID) from the top of the output.

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

- **Installer-Services and Package installations:**

```
$ sudo service dse stop
```

## Hadoop getting started tutorial

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

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

## Procedure

- Unzip the downloaded `obama.txt.zip` file into a directory of your choice on your file system.

This file will be the input for the MapReduce job.

- Create a directory in the Cassandra File System (CFS) for the input file using the `dse command` version of the familiar `hadoop fs` command. For example, on Installer-No Services and Tarball installations:

```
$ cd install_location
$ bin/dse hadoop fs -mkdir /user/hadoop/wordcount/input
```

- Copy the input file that you downloaded to the CFS.

```
$ bin/dse hadoop fs -copyFromLocal
path/obama.txt
/user/hadoop/wordcount/input
```

- Check the version number of the `hadoop-examples-version.jar` file, located in:

- Installer-Services installations: `/usr/share/dse/hadoop/lib`
- Installer-No Services installations: `install_location/resources/hadoop`
- Package installations: `/usr/share/dse/hadoop/lib`
- Tarball installations: `install_location/resources/hadoop`

- Get usage information about how to run the MapReduce job from the jar file.

```
$ bin/dse hadoop jar /install_location/resources/hadoop/hadoop-examples-1.0.4.13.jar wordcount
```

The output is:

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

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

- Run the Hadoop word count example in the JAR.

```
$ bin/dse hadoop jar
/install_location/resources/hadoop/hadoop-examples-1.0.4.13.jar
wordcount
/usr/hadoop/wordcount/input
/usr/hadoop/wordcount/output
```

The output is:

```
13/10/02 12:40:36 INFO input.FileInputFormat: Total input paths to process : 0
13/10/02 12:40:36 INFO mapred.JobClient: Running job:
job_201310020848_0002
13/10/02 12:40:37 INFO mapred.JobClient: map 0% reduce 0%
...
13/10/02 12:40:55 INFO mapred.JobClient: FILE_BYTES_WRITTEN=19164
13/10/02 12:40:55 INFO mapred.JobClient: Map-Reduce Framework
```

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

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

The output looks something like this:

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

8. Using the output file name from the directory listing, get more information using the [dsetool utility](#).

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

The output is:

```
Path: cfs://127.0.0.1/user/hadoop/wordcount/output/part-r-00000
INode header:
 File type: FILE
 User: root
 Group: wheel
 Permissions: rwxrwxrwx (777)
 Block size: 67108864
 Compressed: true
 First save: true
 Modification time: Wed Mar 02 12:58:05 PDT 2014
INode:
 Block count: 1
 Blocks: subblocks length start
end
 (B) f2fa9d90-2b9c-11e3-9ccb-73ded3cb6170: 1 24528 0
24528
 f3030200-2b9c-11e3-9ccb-73ded3cb6170: 24528 0
24528
 Block locations:
 f2fa9d90-2b9c-11e3-9ccb-73ded3cb6170: [localhost]
Data:
 All data blocks ok.
```

9. Finally, look at the output of the MapReduce job--the list of word/count pairs using a familiar Hadoop command.

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

The output is:

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

## Analytics node configuration for DSE Hadoop

Important configuration changes, excluding those related to [the Job Tracker](#), are:

- [Disabling virtual nodes](#)
- [Setting the replication factor](#)
- [Configuring the verbosity of log messages](#)
- [Connecting to non-standard Cassandra native port](#)

Advanced users can also configure DataStax Enterprise to [run jobs remotely](#).

DataStax Enterprise turns off virtual nodes (vnodes) by default because using vnodes causes a sharp increase in the Hadoop task scheduling latency. This increase is due to the number of Hadoop splits, which cannot be lower than the number of vnodes in the analytics datacenter. Using vnodes, instead of N splits for tiny data, you have, for example,  $256 * N$  splits, where N number of physical nodes in the cluster. This may raise job latency from tens of seconds to single or even tens of minutes. This increase in job latency is relatively insignificant when running jobs for hours to analyze huge quantities of data that inherently has lots of splits anyway. In this case, vnodes are perfectly fine. You can use vnodes for any Cassandra-only cluster, a Cassandra-only datacenter, a Spark datacenter, or a Search-only datacenter in a mixed Hadoop/Search/Cassandra deployment.

**Attention:** DataStax Enterprise turns off virtual nodes (vnodes) by default. DataStax does not recommend turning on vnodes for DSE Hadoop or BYOH nodes. Before turning vnodes on for Hadoop, understand the [implications](#). DataStax Enterprise does support turning on vnodes for Spark nodes.

## Setting the replication factor

Change the default [replication factor](#) to a production-appropriate value of at least 3.

## Configuring the verbosity of log messages

To adjust the verbosity of log messages for Hadoop map/reduce tasks, add the following settings to the logback.xml file on each analytic node:

```
logback.logger.org.apache.hadoop.mapred=WARN
logback.logger.org.apache.hadoop.filecache=WARN
```

## Connecting to non-standard Cassandra native port

If the Cassandra native port was changed to a port other than the default port 9042, you must change the cassandra.input.native.port configuration setting for Hive and Hadoop to use the non-default port. The following examples change the Cassandra native port protocol connections to use port 9999.

- Inside the Hive shell, set the port after starting the DataStax Enterprise Hive shell:

```
$ dse hive
hive> set cassandra.input.native.port=9999;
```

- General Hive, add cassandra.input.native.port to the hive-site.xml file:

```
<property>
 <name>cassandra.input.native.port</name>
 <value>9999</value>
</property>
```

- For Hadoop, add cassandra.input.native.port to the core-site.xml file:

```
<property>
 <name>cassandra.input.native.port</name>
 <value>9999</value>
</property>
```

## Configuration for running jobs on a remote cluster

This information is intended for advanced users.

### Procedure

To connect to external addresses:

1. Make sure that the hostname resolution works properly on the localhost for the remote cluster nodes.
2. Copy the `dse-core-default.xml` and `dse-mapred-default.xml` files from any working remote cluster node to your local Hadoop conf directory.
3. Run the job using `dse hadoop`.
4. To override the Job Tracker location or if DataStax Enterprise cannot automatically detect the Job Tracker location, define the `HADOOP_JT` environment variable before running the job:

```
$ export HADOOP_JT=jobtracker host:jobtracker port dse hadoop jar
```

5. If you need to connect to many different remote clusters from the same host:
  - a) Before starting the job, copy the remote Hadoop conf directories fully to the local node (into different locations).
  - b) Select the appropriate location by defining `HADOOP_CONF_DIR`.

## Changing the Hadoop log directory

You must add the `HADOOP_LOG_DIR` environment variable to the `dse-env.sh` file to enable DataStax Enterprise to recognize changes to the default log directory used by the Hadoop component integrated into DataStax Enterprise.

**Note:** If you change the default Hadoop log directory environment variable in `hadoop-env.sh` and restart DataStax Enterprise, the change is not recognized.

### Procedure

1. In the `dse-env.sh` file, comments describe where to add the command to configure the environment variable. For example:

```
#!/bin/sh

Add any environment overrides you need here. This is where users
may set third-party variables such as HADOOP_LOG_DIR

export HADOOP_LOG_DIR=/var/log/hadoop/new_log_location

=====
don't change after this.
if [-r "`dirname \"$0`"/dse.in.sh"]; then
 . .
.
```

2. Restart DataStax Enterprise after configuring the new log location.

In a packaged installation, DataStax Enterprise loads the environment variable change using `/usr/share/dse/dse.in.sh` after you restart the node.

# Portfolio Manager demo using DSE Hadoop

The use case is a financial application where users can actively create and manage a portfolio of stocks. On the Cassandra OLTP (online transaction processing) side, each portfolio contains a list of stocks, the number of shares purchased, and the purchase price. The demo's pricer utility simulates real-time stock data where each portfolio updates based on its overall value and the percentage of gain or loss compared to the purchase price. This utility also generates 100 days of historical market data (the end-of-day price) for each stock. On the DSE OLAP (online analytical processing) side, a Hive MapReduce job calculates the greatest historical 10 day loss period for each portfolio, which is an indicator of the risk associated with a portfolio. This information is then fed back into the real-time application to allow customers to better gauge their potential losses.

## Procedure

To run the demo:

**Note:** DataStax Demos do not work with either LDAP or internal authorization (username/password) enabled.

1. Install a single Demo node using the DataStax Installer in [GUI](#) or [Text](#) mode with the following settings:

- **Install Options page - Default Interface:** 127.0.0.1 (You must use this IP for the demo.)
- **Node Setup page - Node Type:** Analytics
- **Analytic Node Setup page - Analytics Type:** Spark + Integrated Hadoop

2. Start DataStax Enterprise if you haven't already:

- **Installer-Services and Package installations:**

```
$ sudo service dse start
```

- **Installer-No Services and Tarball installations:**

```
install_location/bin/dse cassandra -k -t ## Starts node in Spark and
Hadoop mode
```

```
install_location/bin/dse cassandra -t ## Starts node in Hadoop mode
```

The default *install\_location* is /usr/share/dse.

3. Go to the Portfolio Manager demos directory.

The default location of the Portfolio Manager demo depends on the type of installation:

Installer-Services and Package installations	/usr/share/dse/demos/ portfolio_manager
Installer-No Services and Tarball installations	<i>install_location</i> /demos/ portfolio_manager

4. Run the bin/pricer utility to generate stock data for the application:

- To see all of the available options for this utility:

```
$ bin/pricer --help
```

- Start the pricer utility:

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

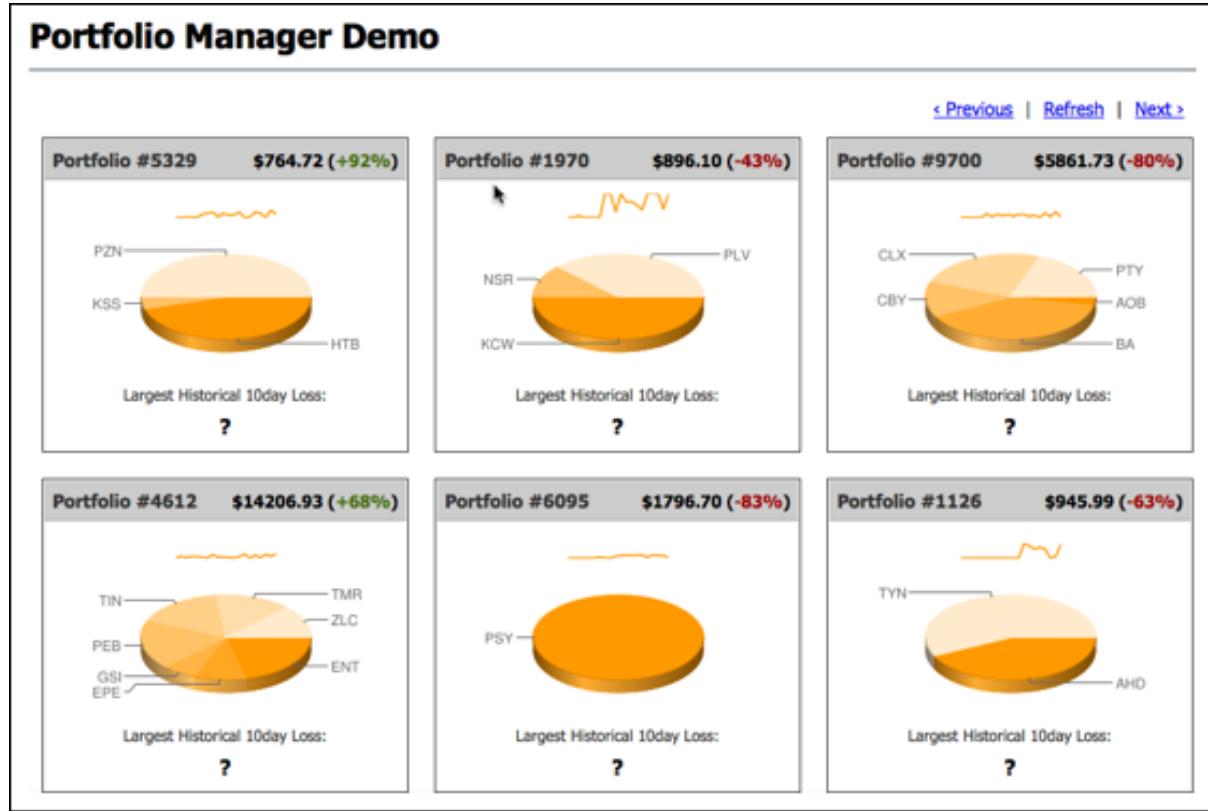
The pricer utility takes several minutes to run.

- Start the web service:

```
$ cd website
$ sudo ./start
```

- Open a browser and go to <http://localhost:8983/portfolio>.

The real-time Portfolio Manager demo application is displayed.



- Open another terminal.
- Start Hive and run the MapReduce job for the demo in Hive.
  - Installer-Services: \$ dse hive -f /usr/share/dse/demos/portfolio\_manager/10\_day\_loss.q
  - Package installations: \$ dse hive -f /usr/share/dse-demos/portfolio\_manager/10\_day\_loss.q
  - Installer-No Services and Tarball installations: \$ *install\_location*/bin/dse hive -f *install\_location*/demos/portfolio\_manager/10\_day\_loss.q

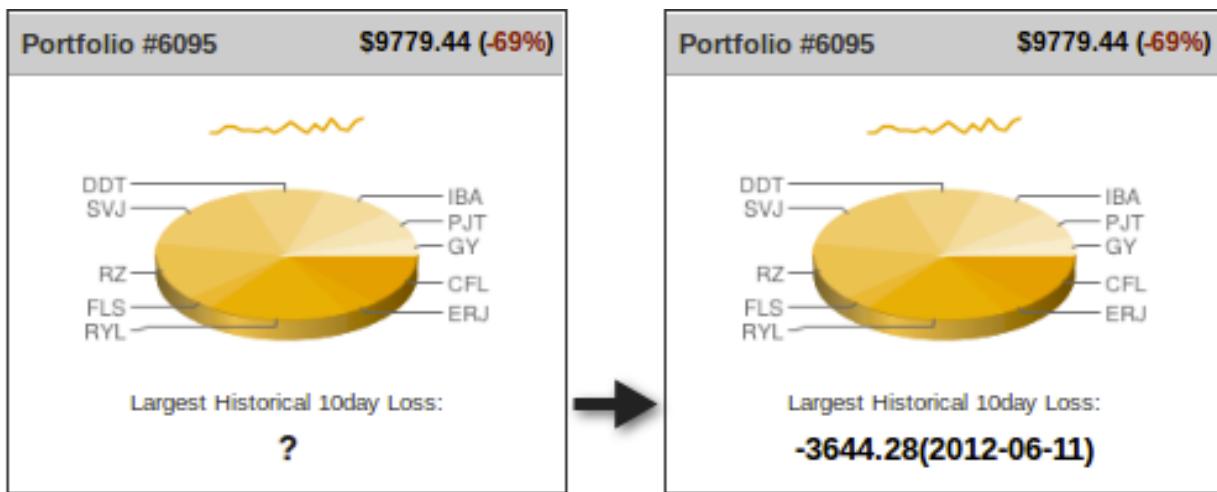
The MapReduce job takes several minutes to run.

- To watch the progress in the Job Tracker node, open the following URL in a browser.

<http://localhost:50030/jobtracker.jsp>

- After the job completes, refresh the **Portfolio Manager** web page.

The results of the Largest Historical 10 day Loss for each portfolio are displayed.



## Using common Hadoop commands

Use common hadoop commands to perform functions in the Cassandra File System (CFS) that correspond to [open source, HDFS file system shell](#) commands. The format of the URI for the CFS is:

```
[cfs-name:] [/ [host]] path
```

- If cfs-name is missing, cfs, which means to access the CFS, is used.
- If host is missing, the address of the local node is used.
- If host is given, the path must start with /

For example, the following paths point to the same path in the CFS:

```
/tmp
///tmp
cfs:/tmp
cfs:///tmp
cfs://localhost/tmp
//localhost/tmp
```

Execute hadoop fs commands on the command line in these directories:

- Installer-Services and Package installations:

```
$ dse hadoop fs option
```

- Installer-No Services and Tarball installations:

```
$ install_location/bin/dse hadoop fs option
```

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

```
$ dse hadoop fs -mkdir /user/hadoop/wordcount/input
$ dse hadoop fs -copyFromLocal $HADOOP_EXAMPLE/data/state_of_union/
state_of_union.txt
 /user/hadoop/wordcount/input
```

To list all options for performing command hadoop HDFS commands:

```
$ dse hadoop fs -help
```

The [DSE command reference](#) lists other commands.

## Using the cfs-archive to store huge files

The Cassandra File System (CFS) consists of two layers, cfs and cfs-archive, that you access using these Hadoop shell commands and URIs:

- cfs:// for the cassandra layer
- cfs-archive:// for the cassandra archive layer

**Note:** You can create other CFS with different names. File systems with the -archive suffix are archive file systems.

Using cfs-archive is highly recommended for long-term storage of huge files, including files with terabytes of data. On the contrary, using cfs is not recommended because the data on this layer undergoes the [compaction process](#) periodically, as it should. Hadoop uses the cfs layer for many small files and temporary data, which need to be cleaned up after deletions occur. When you use the cfs layer instead of the cfs-archive layer, compaction of huge files can take too long, for example, days. Files stored on the cfs-archive layer, on the other hand, do not undergo compaction automatically. You can manually start compaction using the [nodetool compact](#) command.

### Example: Store a file on cfs-archive

This example shows how to store a file on cfs-archive using the Hadoop shell commands from the DataStax Enterprise installation directory on Linux:

1. Create a directory on the cfs-archive layer. You must use an additional forward slash, as [described earlier](#):

```
bin/dse hadoop fs -mkdir cfs-archive:///20140401
```

2. Use the Hadoop shell put command with an absolute path name to store the file on the cfs-archive layer.

```
bin/dse hadoop fs -put big_archive.csv cfs-archive:///20140401/
big_archive.csv
```

3. Verify that the file is stored in the cfs-archive.

```
bin/dse hadoop fs -ls cfs-archive:///20140401/
```

### Example: Migrate a file from SQL to text on cfs-archive

This example shows how to migrate the data from the MySQL table the archive directory `cfs-archive/npa_nxx`.

1. Run the [sqoop demo](#).
2. Use the `dse` command in the bin directory to migrate the data from the MySQL table to text files in the `npa_nxx` directory of cfs-archive. Specify the IP address of the host in the `--target-dir` option.

```
$ sudo ./dse sqoop import --connect
 jdbc:mysql://127.0.0.1/npa_nxx_demo
 --username root
 --password password
 --table npa_nxx
```

```
--target-dir cfs-archive://127.0.0.1/npa_nxx
```

## Using Hive with DSE Hadoop

DataStax Enterprise includes a Cassandra-enabled Hive MapReduce client. [Hive](#) is a data warehouse system for Hadoop that projects a relational structure onto data that is stored in Hadoop-compatible file systems. You use Hive to query Cassandra data using an SQL-like language called HiveQL.

You [start the Hive client](#) on an analytics node and run MapReduce queries directly on data stored in Cassandra. Using the [DataStax Enterprise ODBC driver for Hive](#), a JDBC compliant user interface can connect to Hive from the [Hive server](#).

### Why Hive

By using Hive, you typically eliminate boilerplate MapReduce code and enjoy productivity gains. The large base of SQL users can master HiveQL quickly. Hive has a large set of standard functions, such as mathematical and string functions. You can use Hive for queries that Cassandra as a NoSQL database does not support, such as joins. DataStax Enterprise support of Hive facilitates the migration of data to DataStax Enterprise from a Hive warehouse. Hive capabilities are extensible through a [Hive user-defined function](#) (UDF), which DataStax Enterprise supports.

Typical uses for Hive are:

- Reporting
  - User engagement and impression click count applications
- Ad hoc analysis
- Machine learning
  - Advertising optimization

### Hive in DataStax Enterprise

DSE Analytics nodes store Hive table structures in the Cassandra File System (CFS) instead of in a Hadoop Distributed File System (HDFS). You *layer* a Hive table definition onto a directory in the file system or use Hive to query a [CQL table](#). The Hive table definition describes the layout of the data and is stored in the HiveMetaStore keyspace. DataStax Enterprise implements the Hive metastore as the HiveMetaStore keyspace within Cassandra. Unlike open source Hive, there is no need to run the metastore as a standalone database to support multiple users.

The consistency level of Hadoop nodes is ONE by default, but when processing Hive queries, if DataStax Enterprise can guarantee that all replicas are in the same data center, the consistency level of LOCAL\_ONE is used.

There are two types of Hive tables: external tables and managed tables.

### Automatically created external tables

DataStax Enterprise automatically creates a Hive external table for each existing CQL table when you attempt to use the keyspace/table name in Hive. Exception: After upgrading, you need to enable auto-creation of tables.

### About custom external tables

You can create a custom external table using TBLPROPERTIES and SERDEPROPERTIES when the auto-created table does not suit your needs. The external table data source is external to Hive, located in CQL. When you drop a Hive external table, only the table metadata stored in the HiveMetaStore keyspace is removed. The data persists in CQL.

## Restoring tables after upgrading

You may need to map custom external tables to the new release format after upgrading to DataStax Enterprise 4.8 depending on which version you are upgrading from. DataStax Enterprise provides the `hive-metastore-migrate` tool for mapping the tables to the new format. The tool is in the `hive-metastore-version.jar` in `resources/hive/lib`.

Use the `hive-metastore-migrate` tool only *after* upgrading and only on a *stable* cluster.

To use the `hive-metastore-migrate` tool, perform steps in this order:

1. [Upgrade DataStax Enterprise](#).
2. Verify that the cluster is stable after upgrading.
3. Call the `hive-metastore-migrate` tool using the following options:

### Hive-metastore-migrate tool options

- `-from source_release_num`  
Source release number, for example 4.5.0
- `-help`  
Print `hive-metastore-migrate` command usage
- `-host host`  
Host name
- `-password password`  
Password
- `-port port`  
Port number
- `-to dest_release_num`  
Destination release number, for example 4.8.0
- `-user user`  
User name

This example show how to map Hive custom tables created in DataStax Enterprise 4.5.0 to the format required for a later release, for example 4.8.1:

```
bin/dse hive-metastore-migrate --to 4.8.1 --from 4.5.0
```

In this example, the old Hive tables in 4.5.0 format are mapped to the new 4.8.1 release format.

The `hive-metastore-migrate` tool copies the metadata to a row key using a prefix, for example `4.5.0_`, that you specify using the `-to` option. The tool inserts data for a row key only if there is no data for that row/column.

## Enabling automatic generation of external tables after upgrading

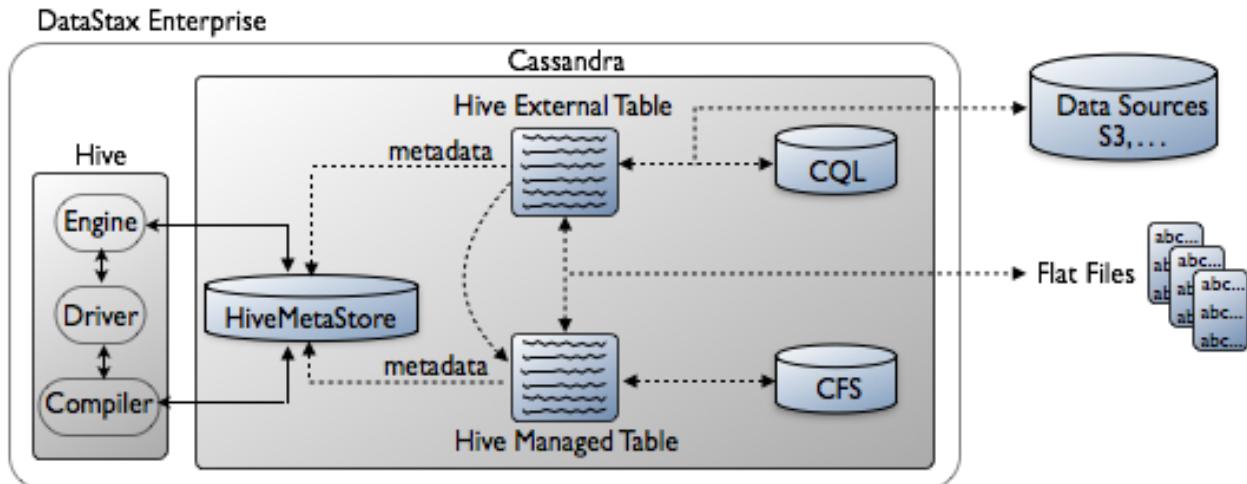
Automatic generation of external tables is disabled. To enable automatic generation of external tables, start Hive and run one of these commands at the Hive prompt to enable automatic generation of external tables:

- `SHOW databases`
- `USE database_name`

## Managed tables

Instead of an external table, you can use a [Hive managed table](#). Hive manages storing and deleting the data in this type of table. DataStax Enterprise stores Hive managed table data in the Cassandra File System (CFS). The data source for the Hive managed table can be a flat file that you put on the CFS using a `dse hadoop -fs` command or the file can be elsewhere, such as on an operating system

file system. To load the managed file, use the LOAD [LOCAL] DATA INPATH, INSERT INTO, or INSERT OVERWRITE Hive commands. You use Hive external tables to access Cassandra or other data sources, such as Amazon S3. Like the Hive managed table, you can populate the external table from flat files on the local hard drive, as well as dumped the data from Hive to a flat file. You can also copy an external table, which represents a Cassandra table, into a Hive managed table stored in CFS. The following diagram shows the architecture of Hive in DataStax Enterprise.



## Hive metastore configuration

The HiveMetastore in DataStax Enterprise supports multiple users and requires no configuration except increasing the default [replication factor](#) of the keyspace. The default replication for system keyspaces is 1. A replication factor of 1 is suitable only for development and testing of a single node, but not for a production environment. To avoid production problems, increase the [replication factor](#) of these system keyspaces to at least 3.

- HiveMetastore
- cfs
- cfs\_archive keyspaces

To prevent missing data problems or data unavailable exceptions after altering keyspaces that contain any data, run [nodetool repair](#) as shown [in these examples](#).

## Supported Hive features

The Hive component in DataStax Enterprise includes querying capabilities, data type mapping, and performance enhancements. The following Hive 0.12 features are supported:

- Windowing functions
  - RANK
  - LEAD/LAG
  - ROW\_NUMBER
  - FIRST\_VALUE, LAST\_VALUE
- Aggregate OVER functions with PARTITION BY and ORDER BY

DataStax Enterprise supports most [CQL](#) and [Cassandra internal data types](#). DataStax provides a [Hive user-defined function \(UDF\)](#) for working with unsupported types, such as blob:

```
org.apache.hadoop.hive.cassandra ql.udf.UDFStringToCassandraBinary
```

This UDF converts from Hive Strings to native Cassandra types. Due to limitations in Hive, the UDF can be used only to convert Hive Strings to string primitives, not collections that are arrays and maps of strings. It

is not possible to use the UDF to convert, for example, an array of strings representing inet addresses to an array of InetAddress columns in Cassandra.

## Running Hive

You can run [Hive as a server](#) or as a client. DataStax Enterprise supports Apache HiveServer and Apache HiveServer2. HiveServer is an optional service for remote clients to submit programmatic requests to Hive. [HiveServer2](#) is an improved version of HiveServer that supports multi-client concurrency and other features. You can use the [Beeline command shell](#) with HiveServer2.

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

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

### Start a Hive client

You can start a Hive client on any analytics node and run MapReduce queries directly on data already stored in Cassandra. You run Hive as a client to perform the examples in this document.

## Procedure

1. Start DataStax Enterprise as an analytics (Hadoop) node.

- Installer-Services and Package installations:

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

```
HADOOP_ENABLED=1
```

2. Use this command to start the service:

```
$ sudo service dse start
```

- Installer-No Services and Tarball installations:

From the installation directory:

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

2. Start a Hive client.

- Installer-Services and Package installations:

```
$ dse hive
```

- Installer-No Services and Tarball installations:

```
$ install_location/bin/dse hive
```

The `hive` prompt appears and you can now enter HiveQL shell commands.

## Browsing through Cassandra tables in Hive

If a keyspace and table exists in Cassandra, you can query the keyspace and table in Hive. For example, create a keyspace in Cassandra using `cqlsh`. Add some data to the table using `cqlsh`, and then access the data in Hive.

```
cqlsh> CREATE KEYSPACE cassandra_keyspace WITH replication =
 {'class': 'NetworkTopologyStrategy', 'Analytics': 1};
cqlsh> USE cassandra_keyspace;
```

```
cqlsh:cassandra_keyspace> CREATE TABLE exampletable
 (key int PRIMARY KEY , data text);
cqlsh:cassandra_keyspace> INSERT INTO exampletable (key, data)
 VALUES (1, 'This data can be read
 automatically in hive');
cqlsh:cassandra_keyspace> quit;
```

At this point, you can [start Hive](#) and query the keyspace and table in Hive.

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

## Creating or altering CQL data from Hive

You need to use a Hive external table to create or alter CQL data from Hive. A counterpart to the Hive database/external table must pre-exist in Cassandra as an keyspace/table. When you use a Hive database name that matches a Cassandra keyspace name, DataStax Enterprise automatically generates a Hive external table for each table in the keyspace. If the auto-created external table does not suit your needs, you create a custom external table using different TBL and SERDEPROPERTIES. Use the `CREATE EXTERNAL TABLE` statement to create such a table.

To use Hive with legacy tables, such as those created using Thrift or the CLI, see [DataStax Enterprise 3.0 documentation](#). Thrift applications require that you configure Cassandra for connection to your application using the `rpc` connections instead of the default `native_transport` connection.

### Creating a custom external table

This example assumes you created the `cassandra_keyspace` and `exampletable` in "[Browsing through Cassandra tables in Hive](#)". A Hive example table is auto-created when you run the `USE cassandra_keyspace` command on the Hive command line. To use a Hive database or table of a different name than the auto-created ones, but with the same or a similar schema, customize the auto-created external table as shown in this example. The example uses the Hive database named `bigdata` instead `cassandra_keyspace`, and the example uses a table named `MyHiveTable` instead of `exampletable`. The example specifies the CQL keyspace and table names in the external table definition using the `TBLPROPERTIES` clause to use the CQL-defined schema.

#### Creating a custom external table

```
hive> CREATE DATABASE bigdata;
hive> USE bigdata;
hive> CREATE EXTERNAL TABLE MyHiveTable
 (key int, data string)
 STORED BY 'org.apache.hadoop.hive.cassandra.cql3.CqlStorageHandler'
 TBLPROPERTIES ("cassandra.ks.name" = "cassandra_keyspace" ,
 "cassandra.cf.name" = "exampletable");
```

### Inspecting an auto-created, external table (DataStax Enterprise 4.0.4 and later)

In Hive, you can use the `SHOW CREATE TABLE CQL_table name` command to see the schema of a auto-created external table. The output of this command can help you construct a custom Hive external

table definition. Assuming you created the table in "Browsing through Cassandra tables in Hive", use the SHOW CREATE TABLE command to see the schema of exampletable.

```
hive> SHOW CREATE TABLE exampletable;
OK
CREATE EXTERNAL TABLE exampletable(
 key int COMMENT 'from deserializer',
 data string COMMENT 'from deserializer')
ROW FORMAT SERDE
 'org.apache.hadoop.hive.cassandra.cql3.serde.CqlColumnSerDe'
STORED BY
 'org.apache.hadoop.hive.cassandra.cql3.CqlStorageHandler'
WITH SERDEPROPERTIES (
 'serialization.format'='1',
 'cassandra.columns.mapping'='key,data')
LOCATION
 'cfs://127.0.0.1/user/hive/warehouse/cassandra_keyspace.db/exampletable'
TBLPROPERTIES (
 'cassandra.partitioner'='org.apache.cassandra.dht.Murmur3Partitioner',
 'cql3.partition.key'='key',
 'cassandra.ks.name'='cassandra_keyspace',
 'cassandra.cf.name'='exampletable',
 'auto_created'='true')
Time taken: 0.028 seconds, Fetched: 18 row(s)
```

## Updating metadata in Hive when altering tables

When you run ALTER TABLE, the metadata in Hive is not updated and subsequent Hive and SparkSQL queries fail.

### Workaround

1. Enter the hive shell:

```
$ dse hive
```

2. In the hive shell, drop the table:

```
hive> DROP TABLE your_keyspace.your_table;
```

3. To allow Hive to refresh the metadata:

```
hive> USE your_keyspace;
```

## Hive to Cassandra type mapping

In the Hive CREATE EXTERNAL TABLE statement, use the Hive data type that corresponds to the Cassandra data type. The following table maps CQL, Cassandra internal storage engine (used by legacy tables), and Hive data types:

CQL	Cassandra Internal	Hive
ascii	AsciiType	string
bigint	LongType	bigint
boolean	BooleanType	boolean
counter	CounterColumnType	bigint
decimal	DecimalType	decimal

CQL	Cassandra Internal	Hive
double	DoubleType	double
float	FloatType	float
inet	InetAddressType	binary
int	Int32Type	int
text	UTF8Type	string
timestamp	TimestampType	date
timestamp	TimestampType	timestamp
timeuuid	TimeUUIDType	binary
uuid	UUIDType	binary
varint	IntegerType	binary
varchar	UTF8Type	varchar
other	other	binary

The InetAddressType stores the raw IP address in network byte order.

## Using TBLPROPERTIES and SERDEPROPERTIES

In an external table definition, the TBLPROPERTIES clause maps the Hive database to a CQL table and can include MapReduce properties, Cassandra database configuration, and native protocol properties for the table. The SERDEPROPERTIES clause specifies the properties used when serializing/deserializing data passed between the Hive table and Cassandra. You can add a WITH SERDEPROPERTIES clause to map meaningful column names in Hive to the Cassandra partition key, column names, and column values. You can change these properties on the fly. Using the Hive SET command, you can configure properties in the hive session. The settings become effective for the next query.

The following table lists general properties used in the TBLPROPERTIES or SERDEPROPERTIES clause or both. The subsequent section lists additional, optional properties for use with the DataStax Java Driver. The TBL/SERDE column of the following table lists how to declare properties in the table definition, as a TBLPROPERTIES (TBL), a SERDEPROPERTIES (SERDE) or both.

**Table: General TBL and SERDE properties**

General Property	TBL/SERDE	Description
cassandra.cf.name	both	Cassandra table name
cassandra.columns.mapping	both	Mapping of Hive to legacy Cassandra columns
cassandra.consistency.level	both	Consistency level - default ONE
cassandra.cql3.type	both	<a href="#">CQL types</a>
cassandra.host	both	IP of a Cassandra node to connect to
cassandra.input.split.size	both	<a href="#">MapReduce split size</a>
cassandra.ks.name	both	Cassandra keyspace name
cassandra.partitioner	both	Partitioner (default = configured partitioner)
cassandra.port	both	Cassandra RPC port - default 9160

General Property	TBL/SERDE	Description
cql3.output.query	TBL	A prepared statement for storing alterations to a CQL users table
cql3.partition.key	both	CQL partition key, a comma-separated list of partition and clustering keys
cql3.pushdown.enable	TBL	True (default) <a href="#">enable pushdown predicate</a>
cql3.update.columns	both	Used with <a href="#">INSERT INTO SELECT</a>

### Required table properties

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

- cassandra.ks.name
- cassandra.cf.name

Other frequently-used properties are:

- cql3.output.query
- cql3.partition.key (DataStax Enterprise 4.0.4 and later)

You use the `SHOW CREATE TABLE CQL_table_name` command at the Hive prompt to see the auto-created external table. The output helps you see how to format the `cql3.partition.key` in your custom external table. For example, the output of a table having following CQL composite partition key, has the '`cql3.partition.key='key,event_id'` Hive property syntax:

```
PRIMARY KEY ((key, event_id), num_responses)
```

### Required storage handler

Also required in the external table definition is the CQL storage handler:

`org.apache.hadoop.hive.cassandra.cql3.CqlStorageHandler`. The storage handler accesses and stores Cassandra data back to Cassandra.

### About the `cassandra.input.split.size`

The `cassandra.input.split.size` property configures the number of CQL partitions processed per mapper (64k rows per split). The default is  $64 * 1024$ . If your tables have large partitions (many distinct values of clustering columns for the same partitioning key), do *not* use the default. Use a lower setting.

### Partitioner use by Hive

You do not need to specify `cassandra.partitioner`. Your configured partitioner is used by Hive. For example, Hive uses this property value if you use the Cassandra 2.1 default partitioner:

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

[Creating or altering CQL data from Hive](#) and [MapReduce performance](#) show examples of using some of these properties.

### Optional native protocol properties

DataStax Enterprise supports the following optional properties for the native protocol.

- `cassandra.input.native.port`
- `cassandra.input.native.core.connections.per.host`
- `cassandra.input.native.max.connections.per.host`
- `cassandra.input.native.min.simult.reqs.per.connection`
- `cassandra.input.native.max.simult.reqs.per.connection`

- cassandra.input.native.connection.timeout
- cassandra.input.native.read.connection.timeout
- cassandra.input.native.receive.buffer.size
- cassandra.input.native.send.buffer.size
- cassandra.input.native.solinger
- cassandra.input.native.tcp.nodelay
- cassandra.input.native.reuse.address
- cassandra.input.native.keep.alive
- cassandra.input.native.auth.provider
- cassandra.input.native.ssl.trust.store.path
- cassandra.input.native.ssl.key.store.path
- cassandra.input.native.ssl.trust.store.password
- cassandra.input.native.ssl.key.store.password
- cassandra.input.native.ssl.cipher.suites

## Using a managed table to load local data

If you do not need to store data in a Cassandra table, use a managed table instead of an external table. The data can be located in the Cassandra File System (CFS) or on the file system. You load the data into the managed table as shown in this example:

1. Create a managed table:

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

2. Load data into a table using the LOAD DATA command. The [HiveQL Manual](#) provides more information about the HiveQL syntax.

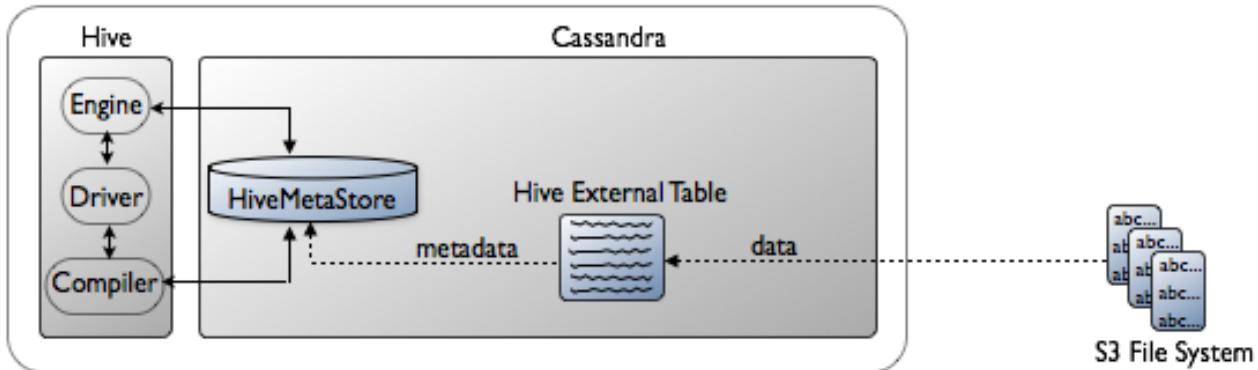
For example, on the Mac OS X:

```
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 tarball distribution.

## Using an external file system

You can map a file in an external file system, such as S3 native file system to a table in Hive. The DSE Hadoop cluster continues to use the Cassandra File System (CFS). The data source is external to Hive, located in S3 for example. You create a Hive external table for querying the data in an external file system. When you drop the external table, only the table metadata that is stored in the HiveMetaStore keyspace is removed. The data persists in the external file system.



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

## Procedure

1. Open the `hive-site.xml` file for editing.
2. Add a property to `hive-site.xml` to set the default file system to be the native S3 block file system. Use `fs.default.name` as the name of the file system and the location of the bucket as the value. For example, if the S3 bucket name is `mybucket`:

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

3. Save the file.
4. Open the `core-site.xml` file for editing.
5. Add these properties to `core-site.xml` to specify the access key ID and the secret access key credentials for accessing the native S3 block file system:

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

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

6. Save the file and restart Cassandra.
7. Create a directory in `s3n://mybucket` named, for example, `mydata_dir`.
8. Create a data file named `mydata.txt`, for example. Delimit fields using =.

```
"key1"=100
"key2"=200
"key3"=300
```

9. Put the data file that you created in `s3n://mybucket/mydata_dir`.

10. Using [cqlsh](#), create a keyspace and a CQL table schema to accommodate the data on S3.

```
cqlsh> CREATE KEYSPACE s3_counterpart WITH replication =
 {'class': 'NetworkTopologyStrategy', 'Analytics': 1};
cqlsh> USE s3_counterpart;
cqlsh:s3_counterpart> CREATE TABLE mytable
 (key text PRIMARY KEY , data int);
```

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

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

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

12. Select all the data in the file on S3.

```
SELECT * from mytable;
OK
key1 100
key2 200
key3 300
```

## Creating a Hive CQL output query

One of the Hive external table properties (TBLPROPERTIES) is the `cql3.output.query`. The value of this property is a prepared statement that the MapReduce job uses to insert data into the corresponding Cassandra table. The prepared query is identical to the CQL statement for altering the table except the binding of the ? is done by Hive. The ? are bound to the hive columns in the order specified in the external table schema.

You can set [TTL](#) (time to live) on data in a column using the `cql3.output.query` property.

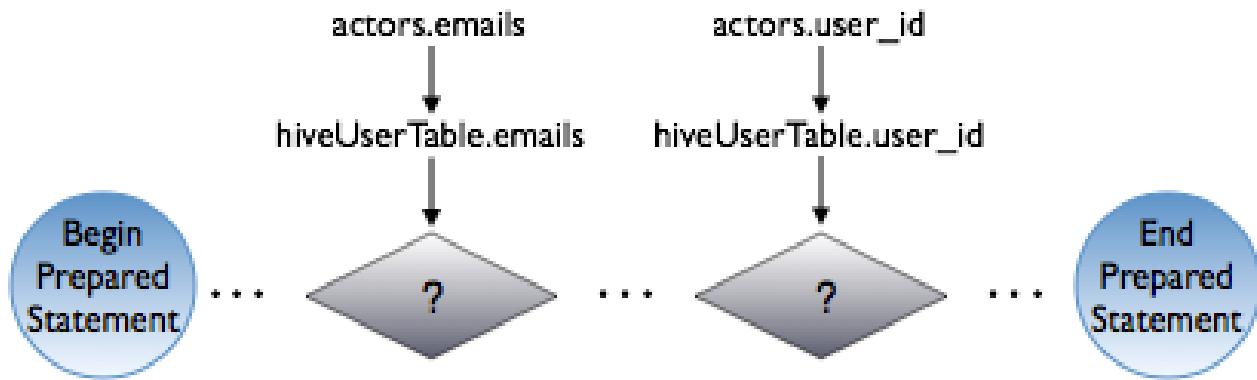
In the [example of using a collection set](#), the external table definition determines the bind variables, '?s', needed in the prepared statements:

```
hive> CREATE EXTERNAL TABLE hiveUserTable
 (emails arraystring,user_id string)
 . . .
```

This external table schema specifies the second column to be the `user_id`; therefore, this `INSERT` statement takes the columns `emails`, `user_id` from the Cassandra `actors` table and maps the data into the Hive `emails` and `user_id` columns:

```
hive> INSERT INTO TABLE hiveUserTable SELECT emails,user_id FROM actors;
```

The following diagram shows the relationship between the tables and the bind variables:



The hiveUserTable includes this prepared query:

```
"cql3.output.query" =
"update cql3ks.users set emails = emails + ? WHERE user_id = ?";
```

The Hive `INSERT` statement starts the MapReduce job that uses the key value from the actors table in the `'WHERE (user_id) ='` clause of prepared statement.

Another example, an abstract one, updates a table having three columns (`x,y,z`) using a prepared statement. The query looks like this internally:

```
create external table (x,y,z) Stored by
(cql3.output.query = "Update cassX = ?(x) cassY=? (y) where cassZ= ? (z) ")
```

## Setting TTL on column data

You can set the TTL on data in an external table. Decoded the following example of how to set TTL using the `cql3.output.query` looks like this:

```
UPDATE users USING TTL 432000 SET 'password' = 'ch@ngem3a' WHERE KEY =
'jsmith';
```

To set TTL on data in an auto-created table, configure a property named `cql.output.query.ttl` for the CQL table. Set the property as you would [set the comment property](#). This action sets the TTL for the entire record.

## Example: Work with an unsupported data type

DataStax Enterprise provides a user defined function (UDF) for converting Hive binary data into string representations of CQL types. Hive cannot auto-create an external table that maps to the unsupported types, such as Cassandra blobs. You have to create a custom external table in Hive and map these types to binary. To read the data in Hive, use a provided UDF to convert the data.

### Create the keyspace and two tables in cqlsh

This example first creates a keyspace and two tables in cqlsh and inserts data of every supported type into the tables.

1. Start `cqlsh`. For example, on Linux.

```
./cqlsh
```

2. In `cqlsh`, create a keyspace:

```
cqlsh> CREATE KEYSPACE cql3ks WITH replication =
{ 'class': 'NetworkTopologyStrategy',
```

```
'Analytics': '1' };
```

3. Using cqlsh, create a table in the cql3ks keyspace having columns of every CQL data type.

```
cqlsh> USE cql3ks;

cql3ks> CREATE TABLE genericAdd (
 key ascii PRIMARY KEY, a bigint, b blob, c boolean,
 d decimal, e double, f float, g inet, h int, i text,
 j timestamp, k uuid, l timeuuid, m varint);
```

4. Insert some data into the table.

```
cql3ks> INSERT INTO genericAdd (
 key,a,b,c,d,e,f,g,h,i,j,k,l,m)
VALUES ('KeyOne', 100005, 0xBEEFEED, true, 3.5,-1231.4,
3.14, '128.2.4.1', 42, 'SomeText', '2008-10-03',
e3d81c40-1961-11e3-8ffd-0800200c9a66,
f078d660-1961-11e3-8ffd-0800200c9a66, 1000000);
```

5. Create a second table, genericToAdd, containing every data type and insert different data into the table.

```
cql3ks> CREATE TABLE genericToAdd (
 id int PRIMARY KEY, key ascii, a bigint, b blob, c boolean,
 d decimal, e double, f float, g inet, h int, i text,
 j timestamp, k uuid, l timeuuid, m varint);
```

6. Insert some data into the second table.

```
cql3ks> INSERT INTO genericToAdd (
 id,key,a,b,c,d,e,f,g,h,i,j,k,l,m)
VALUES (1,'Oneness',1, 0x11111111, true, 1.11,-1111.1,1.11,
'111.1.1.1', 11,'11111','1999-11-01',
e3d81c40-1961-11e3-8ffd-0800200c9a66,
f078d660-1961-11e3-8ffd-0800200c9a66, 1);
```

### Create an external table in Hive

Next, create an *external* table in Hive that maps to the table in Cassandra. You cannot use the auto-created table because Hive cannot represent the blob type in a comprehensible format. After creating the custom external table, you can perform alterations of the CQL tables from Hive. You insert data from the second CQL table into the first CQL table from Hive. Using a UDF, you query the external table in Hive. You need to use the UDF because the data is of the unsupported blob type.

1. Create a table in Hive that includes a cql3.output.query property that has the value of a prepared statement for inserting the data from the second, genericToAdd, table into the first, genericAdd, table.

The last couple of lines in the following statement need to be free of line breaks. If you copy/paste this statement directly from the documentation and do not remove line breaks, an error occurs in the subsequent step.

```
hive> CREATE EXTERNAL TABLE hive_genericadd (key string, a bigint,
 b binary, c boolean, d decimal, e double, f float, g binary, h
 int, i string, j timestamp, k binary, l binary, m binary) STORED BY
 'org.apache.hadoop.hive.cassandra.cql3.CqlStorageHandler' TBLPROPERTIES
 ("cassandra.ks.name" = "cql3ks", "cassandra.cf.name" = "genericadd",
 "cql3.partition.key"="key",
 "cql3.output.query" = "INSERT INTO cql3ks.genericadd
 (key,a,b,c,d,e,f,g,h,i,j,k,l,m) VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
```

2. Use the `INSERT` statement to start the MapReduce job that inserts the data from the second CQL table into the first one.

```
hive> INSERT INTO TABLE hive_genericadd SELECT
 key,a,b,c,d,e,f,g,h,i,j,k,l,m FROM cql3ks.generictoadd;
```

The MapReduce job runs.

```
Total MapReduce jobs = 1
Launching Job 1 out of 1
...
Job 0: Map: 2 HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 33.278 seconds
```

3. Create an alias for the UDF provided by DataStax.

```
hive> CREATE TEMPORARY FUNCTION c_to_string AS
 'org.apache.hadoop.hive.cassandra ql.udf.UDFCassandraBinaryToString';
```

4. Select the data of the unsupported blob type from the Hive table by calling the UDF.

```
hive> select c_to_string(b, 'blob') from hive_genericadd;
```

The MapReduce job runs, and the output correctly displays the values:

```
Total MapReduce jobs = 1
...
Job 0: Map: 2 HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
beeffeed
11111111
```

## INSERT INTO SELECT statement

DataStax Enterprise supports the `INSERT INTO SELECT` statement in Hive. You set a `TBL` and `SERDE` property, and use `INSERT INTO SELECT` to copy data from one table and insert it into another, or the same, table.

Supported `TBL` and `SERDE` properties include the following `SERDE` property:

```
cql3.update.columns
```

You use `cql3.update.columns` in conjunction with the [CQL output query](#) property, `cql3.output.query`.

The following example shows how to configure these properties and use the `INSERT INTO SELECT` statement in Hive to insert selective columns from a table into another row of the same Cassandra table. The `SELECT` statement requires values for each column in the target table. Using fake values satisfies this requirement.

## Procedure

1. Start `cqlsh` and create a Cassandra keyspace and table.

```
cqlsh> CREATE KEYSPACE mykeyspace WITH replication = {'class':
 'SimpleStrategy', 'replication_factor': 3};
cqlsh> USE mykeyspace;
```

```
cqlsh> CREATE TABLE mytable (a INT PRIMARY KEY, b INT, c INT, d INT);
cqlsh> INSERT INTO mytable (a, b, c, d) VALUES (1, 2, 3, 4);
```

2. Start the Hive client.
3. In Hive, use the auto-created database and [external table](#), and select all the data in the table.

```
hive> USE mykeyspace;
hive> SELECT * FROM mytable;
```

Output is:

```
OK 1 2 3 4 Time taken: 0.138 seconds, Fetched: 1 row(s)
```

4. In Hive, alter the external table to configure the [prepared statement](#) as the value of the Hive CQL output query. The prepared statement in this example takes values inserted into columns a and b in mytable and maps them to columns b and a, respectively, for insertion into the new row.

```
hive> ALTER TABLE mytable SET TBLPROPERTIES ('cql3.output.query' = 'update
mykeyspace.mytable set b = ? where a = ?');
hive> ALTER TABLE mytable SET SERDEPROPERTIES ('cql3.update.columns' =
'b,a');
```

5. In Hive, execute an `INSERT INTO SELECT` statement to insert a row of data into mytable. For example, use 4 and 9 as the values to insert into the first two positions (a, b) of the row. The CQL output query will reverse these positions. Use two type-compatible fake values in addition to the values 4 and 9 that you want to insert. In this example, the fake values are an int, 9999, and a column name, d.

```
hive> INSERT INTO TABLE mytable SELECT 4, 9, 9999, d FROM mytable;
```

The MapReduce job runs:

```
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
...
MapReduce Jobs Launched:
Job 0: Map: 2 HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 31.867 seconds
```

6. Check that 4 and 9, and only those values, were inserted:

```
hive> SELECT * FROM mytable;
```

The fake values are inserted as NULL and only the values specified by the CQL output query are inserted. The output query mapped 4 to column b and 9 to column a.

```
OK
1 2 3 4
9 4 NULL NULL
Time taken: 0.131 seconds, Fetched: 2 row(s)
```

## Example: Use a CQL composite partition key

This example first creates a CQL table, and then creates an external table in Hive that maps to the CQL table. You cannot use the auto-created external table because Hive does not support the timeuuid or varint types used in the CQL table. You need to declare these types binary in the external table definition. The Hive table uses a SERDE property and declares a single key followed by the column declarations that correspond to columns in the CQL table. Finally, the example queries the CQL table from Hive.

### Procedure

1. In cqlsh, add a table to the cql3ks [keyspace created earlier](#). Create a table that uses a [composite partition key](#).

```
cql3ks> CREATE TABLE event_table (
 key ascii, factor float, event_type text, event_date timestamp,
 event_id timeuuid, num_responses varint,
 PRIMARY KEY ((key, event_id), num_responses)
);
```

2. Insert data into the table.

```
cql3ks> INSERT INTO event_table (
 key, factor, event_type, event_date, event_id, num_responses)
VALUES ('KeyOne', 3.14, 'Q3-launch', '2014-09-03',
f078d660-1961-11e3-8ffd-0800200c9a66, 1000000
);
```

3. Create a custom external table in Hive named mapped\_table that maps to the CQL event\_table.

```
hive> CREATE EXTERNAL TABLE mapped_table(
 key string, factor float, event_type string,
 event_date timestamp, event_id binary, num_responses binary)
STORED BY
'org.apache.hadoop.hive.cassandra.cql3.CqlStorageHandler'
WITH SERDEPROPERTIES("cassandra.ks.name" = "cql3ks",
"cassandra.cf.name" = "event_table",
'cql3.partition.key'='key,event_id',
"cassandra.cql3.type" = "ascii, float, text, timestamp, timeuuid,
varint"
);
```

4. Trigger a MapReduce job to query the table in Hive.

```
hive> SELECT COUNT(*) FROM mapped_table;
```

The output is:

```
Total MapReduce jobs = 1
Launching Job 1 out of 1
.
.
MapReduce Jobs Launched:
Job 0: Map: 2 Reduce: 1 HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
1
Time taken: 39.929 seconds
```

# Using CQL collections

Hive supports writing to CQL tables, including [tables of collections](#). To store data to a CQL table from Hive, use prepared statements as shown in these examples:

## Prepared statements for a list

```
UPDATE users SET top_places = ? where user_id = ?
UPDATE users SET top_places = ['rivendell', 'rohan'] WHERE user_id =
 'frodo';

UPDATE users SET top_places = ? + top_places where user_id = ?
UPDATE users SET top_places = ['the shire'] + top_places WHERE user_id =
 'frodo';

UPDATE users SET top_places = top_places - ? where user_id = ?
UPDATE users SET top_places = top_places - ['riddermark'] WHERE user_id =
 'frodo';
```

## Prepared statement for a map

Prepared statements for a set are similar to those for a list.

```
UPDATE users SET todo = ? where user_id = ?
UPDATE users
 SET todo = { '2012-9-24' : 'enter mordor',
 '2012-10-2 12:00' : 'throw ring into mount doom' }
 WHERE user_id = 'frodo';
```

The following queries are handled as a regular value instead of tuples:

```
UPDATE users SET top_places[2] = ? where user_id = ?
UPDATE users SET top_places[2] = 'riddermark' WHERE user_id = 'frodo';

UPDATE users SET todo[?] = ? where user_id = ?
UPDATE users SET todo['2012-10-2 12:10'] = 'die' WHERE user_id = 'frodo';
```

## Example: Alter a set collection

Items in a CQL collection are mapped to the Hive types shown in the [Hive to Cassandra type mapping](#) table. The CQL data types not supported in Hive, such as blob, can be used if you transform the fields of that type using a [DataStax-provided UDF](#).

In cqlsh, you create two tables that contain a collection sets and insert data into the tables. In Hive, you create a custom external table that maps to the first CQL table, and then insert data from the second CQL table to the first CQL table. Finally, in cqlsh, you query the second CQL table to verify that the insertion was made.

1. In cqlsh, create the users table shown in the CQL documentation that contains a set collection column, and insert data into the table:

```
cqlsh> CREATE TABLE cql3ks.users (
 user_id text PRIMARY KEY,
 first_name text,
 last_name text,
 emails set text
);
```

```
cqlsh> INSERT INTO cql3ks.users (user_id, first_name, last_name, emails)
```

```
VALUES('frodo', 'Frodo', 'Baggins',
{'f@baggins.com', 'baggins@gmail.com'});
```

2. Create a second table that contains data about actors:

```
cqlsh> CREATE TABLE cql3ks.actors (
 user_id text PRIMARY KEY,
 first_name text,
 last_name text,
 emails settext
);

cqlsh> INSERT INTO cql3ks.actors (user_id, first_name, last_name, emails)
 VALUES ('ejwood', 'Elijah', 'Wood', {'ejwood@hobbit.com'});
```

3. In Hive, create a custom external table named hiveUserTable that maps to the CQL users table. The last couple of lines in the following statement need to be free of line breaks.

```
hive> CREATE EXTERNAL TABLE hiveUserTable (emails arraystring, user_id
 string) STORED BY
 'org.apache.hadoop.hive.cassandra.cql3.CqlStorageHandler'
 TBLPROPERTIES("cassandra.ks.name" = "cql3ks", "cassandra.cf.name" =
 "users", "cql3.partition.key"="user_id", "cql3.output.query" = "update
 cql3ks.users set emails = emails + ? WHERE user_id = ?");
```

4. Add the data from the CQL actors table to the users table:

```
hive> INSERT INTO TABLE hiveUserTable SELECT emails, user_id FROM
 cql3ks.actors;
```

The MapReduce job runs and alters the table.

5. Check that the CQL table contains Elijah Wood's email address:

```
cql3ks> SELECT * FROM cql3ks.users;

user_id | emails
first_name | last_name
-----+-----
+-----+-----+
ejwood | {ejwood@hobbit.com} |
null | null
frodo | {baggins@gmail.com, f@baggins.com, fb@friendsofmordor.org} |
Frodo | Baggins
```

## Using a custom UDF

If the Hive built-in functions do not provide the capability you need, you can include your own Java code in a user-defined function (UDF) and invoke it using a query. [DataStax provides a UDF](#) for working with unsupported data types, for example. The example in this section uses a JAR that converts text from lowercase to uppercase. After downloading the JAR from the [Hadoop tutorial examples repository](#) and setting up the UDF in Hive, you create a Hive table. You insert data into the table from a text file installed with DataStax Enterprise. The contents of the file look like this:

```
238^Aval_238
86^Aval_86
311^Aval_311
27^Aval_27
165^Aval_165
. . .
```

When you execute a SELECT statement, you invoke the UDF to convert text in the file from lowercase to uppercase: val to VAL.

## Procedure

1. [Download the JAR](#) for this example.
2. On the command line, add the JAR to the root Hadoop directory in the Cassandra File System (CFS) using [Hadoop shell commands](#). For example:

```
dse hadoop fs -copyFromLocal local-path-to-jar/myudfs.jar /tmp
```

Substitute the path to the downloaded job in your environment for *local-path-to-jar*.

3. [Start a Hive client](#), and at the Hive prompt, add the JAR file to the Hadoop distributed cache, which copies files to task nodes to use when the files run:

```
hive> add jar cfs:///tmp/myudfs.jar;
```

The output on the Mac OS X is:

```
converting to local cfs:///tmp/myudfs.jar
Added /private/tmp/johndoe/hive_resources/myudfs.jar to class path
Added resource: /private/tmp/johndoe/hive_resources/myudfs.jar
```

4. At the Hive prompt, create an alias for the UDF associated with the JAR.

```
hive> CREATE TEMPORARY FUNCTION myUpper AS 'org.hue.udf.MyUpper';
```

5. Create a Hive table for text data.

```
hive> CREATE TABLE udfctest (foo INT, bar STRING);
```

6. Insert data into the table, substituting the path to the DataStax Enterprise installation in your environment for the *install\_location*. For example, on Mac OS X:

```
hive> LOAD DATA LOCAL INPATH
 'install_location/resources/hive/examples/files/kv1.txt'
 OVERWRITE INTO TABLE udfctest;
```

7. Convert the lowercase text in the table, the instances of *val*, to uppercase by invoking the UDF by its alias in the SELECT statement.

```
hive> SELECT myUpper(bar) from udfctest;
```

The mapper output looks like this:

```
.
.
.
MapReduce Jobs Launched:
Job 0: Map: 1 HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
VAL_238-gg
VAL_86-gg
VAL_311-gg
.
.
```

## Using pushdown predicates

Pushdown predicates resolve expressions as early as possible in the processing pipeline to minimize the amount of data to be processed. You enable pushdown predicates using the `cql3.pushdown.enable` property in the `TBLPROPERTIES` clause of a Hive query. `True` enables the feature and `false` (the default) disables it. Processing of operations on columns of the following types are affected by the setting:

Cassandra type	Hive type
UTF8Type	string
AsciiType	string
CounterColumnType	long
DateType	timestamp
LongType	long
DoubleType	double
FloatType	float
BooleanType	boolean
Int32Type	int

### Recommended usage

When the indexed row is small, enable pushdown predicates; otherwise, disable the feature to avoid a timeout exception or Out-Of-Memory (OOM) condition.

### Pushdown predicate limitations

DataStax Enterprise supports pushdown predicates for indexes only. Primary keys are not supported.

## Using the Hive count function

Using the `Hive TBLPROPERTIES` `cassandra.consistency.level`, set the consistency level to `ALL` before issuing a Hive `SELECT` expression containing the `count` function. Using `ALL` ensures that when you ping one node for a scan of all keys, the node is fully consistent with the rest of the cluster. Using a consistency level other than `ALL` can return resultsets having fewer rows than expected because replication has not finished propagating the rows to all nodes. A count that is higher than expected can occur because tombstones have not yet been propagated to all nodes.

To get accurate results from the `count` function using a consistency level other than `ALL`:

- Repair all nodes.
- Prevent new data from being added or deleted.

## Spatial analytics support

DataStax Enterprise integrates some components of [GIS Tools for Hadoop](#). The [GIS Tools for Hadoop open source project](#) provides several libraries for performing spatial analytics. DataStax Enterprise incorporates the Hive Spatial library of the Spatial Framework for Hadoop and includes a custom tool for importing data in Enclosed JSON format from ArcGIS to a Cassandra table.

DataStax Enterprise supports Environmental Systems Research Institute (ESRI) data types, which map to the following Cassandra CQL types:

ESRI Type	Description	CQL Type
esriFieldTypeSmallInteger	Integer	Int
esriFieldTypeInteger	Long integer	Bigint
esriFieldTypeSingle	Single-precision floating-point number	Float/decimal
esriFieldTypeDouble	Double-precision floating-point number	Double/decimal
esriFieldTypeString	Character string	Text
esriFieldTypeDate	Date	Date
esriFieldTypeOID	Long integer representing an object identifier	Bigint
esriFieldTypeGeometry	Geometry	Blob
esriFieldTypeBlob	Binary large object	Blob
esriFieldTypeRaster	Raster	N/A
esriFieldTypeGUID	Globally unique identifier	Text
esriFieldTypeGlobalID	ESRI global ID	Text
esriFieldTypeXML	XML document	N/A

The DataStax Enterprise custom ESRI import tool supports the Enclosed JSON format. The syntax for using the tool is:

```
esri-import -keyspace keyspace_name -table table_name -dir path_to_files
[options]
```

Options are:

**-dir *path***

Directory of ESRI data files

**-exclude *files***

Files to exclude

**-file *files***

Included files

**-help**

esri-import command usage help

**-host *host***

Host name of node

**-port *port***

Port number on the host node

The example of analyzing data shows how to use the GIS tools for Hadoop.

## Example: Analyzing spatial data

This example shows how to use DataStax Enterprise with the integrated GIS Tools for Hadoop and custom ESRI-import tool for the following tasks:

- Create a CQL table to accommodate ESRI earthquake data.
- Load ESRI earthquake data from a CSV file into Cassandra.
- Load county geographic information from a JSON file into Hive.
- Analyze the data to determine the location of earthquakes.

The example assumes that you [started DataStax Enterprise](#) as a Hadoop-enabled analytics node.

## Procedure

1. [Download](#) the CSV and JSON files from the DataStax web site for this example.
2. Unzip the file into a directory.  
The gis.zip file contains earthquakes.csv and california-counties.json.
3. In `cqlsh`, create and use a keyspace.

```
cqlsh> CREATE KEYSPACE gis WITH replication = {'class': 'NetworkTopologyStrategy', 'Analytics': 1 };

cqlsh> USE gis;
```

4. Create a schema for the earthquake data in earthquakes.csv.

```
cqlsh:gis> CREATE TABLE earthquakes (
 datetime text PRIMARY KEY,
 latitude double,
 longitude double,
 depth double,
 magnitude double,
 magtype text,
 nbstations int,
 gap double,
 distance double,
 rms double,
 source text,
 eventid int
) ;
```

Although the earthquake dates are in ISO 8601 format, the schema uses the text type for the `datetime` column because 1898 - 2011 is outside the [timestamp type range](#).

5. Copy the data in the CSV file to the table using the path that you chose for the CSV file.

```
cqlsh:gis> COPY earthquakes (datetime, latitude, longitude, depth,
 magnitude, magtype, nbstations, gap, distance, rms, source, eventid) FROM
 'path/earthquakes.csv' WITH HEADER = 'true';
```

6. [Start a Hive client](#).
7. From Hive, access the `gis` database in Cassandra.

```
hive> USE gis;
```

8. In Hive, create a managed table named `counties` that defines a schema for the California counties data.

```
hive> CREATE TABLE IF NOT EXISTS counties (
 Area string,
 Perimeter string,
 State string,
 County string,
 Name string,
 BoundaryShape binary
```

```

)
ROW FORMAT SERDE 'com.esri.hadoop.hive.serde.JsonSerde'
STORED AS INPUTFORMAT 'com.esri.json.hadoop.EnclosedJsonInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat';

```

- 9.** Load the ESRI county data into the table. Use the path to the california-counties.json file you downloaded.

```

hive> LOAD DATA LOCAL INPATH 'path/california-counties.json' OVERWRITE
INTO TABLE counties;

```

The output looks something like this:

```

Copying data from file:/Users/me/builds/dse-4.x/bin/california-
counties.json
Copying file: file:/Users/me/builds/dse-4.x/bin/california-counties.json
Loading data to table gis.counties
Table gis.counties stats: [num_partitions: 0, num_files: 1, num_rows: 0,
 total_size: 1028330, raw_data_size: 0]
OK

```

- 10.** In Hive, create temporary functions for the geometry API calls.

```

hive> create temporary function ST_Point as
 'com.esri.hadoop.hive.ST_Point';

hive> create temporary function ST_Contains as
 'com.esri.hadoop.hive.ST_Contains';

```

- 11.** Join the counties and earthquake tables, and query the data to determine the number of earthquakes in each county.

```

hive> SELECT counties.name, count(*) cnt FROM counties
 JOIN earthquakes
 WHERE ST_Contains(counties.boundaryshape,
ST_Point(earthquakes.longitude, earthquakes.latitude))
 GROUP BY counties.name
 ORDER BY cnt desc;

```

The MapReduce job runs, and the output appears.

```

Kern 36
San Bernardino 35
Imperial 28
Inyo 20
Los Angeles 18
Monterey 14
Riverside 14
Santa Clara 12
Fresno 11
San Benito 11
San Diego 7
Santa Cruz 5
San Luis Obispo 3
Ventura 3
Orange 2
San Mateo 1

```

## Handling schema changes

If you change a table in Cassandra, using CQL for example, after creating an external table in Hive that maps to that table in Cassandra, a runtime exception can occur. Changes that occur to the table in Cassandra get out of synch with the mapped table in Hive. The workaround is:

### Procedure

1. In Hive, drop the table.

```
hive> drop table mytable;
```

2. Run SHOW TABLES.

```
hive> show tables;
```

Now, the table in Hive contains the updated data.

## MapReduce performance tuning

You can change performance settings in the following ways:

- In an external table definition, using the **TBLPROPERTIES** or **SERDEPROPERTIES** clauses.
- Using the Hive SET command. For example: `SET mapred.reduce.tasks=32;`
- In the `mapred-site.xml` file.

**Note:** Restart the analytics nodes after you make changes to `mapred-site.xml`.

### Performance changes using `mapred-site.xml`

#### Speeding up map reduce jobs

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

#### Increasing the number of map tasks to maximize performance

You can increase the number of map tasks in these ways:

- Turn off map output compression in the `mapred-site.xml` file to lower memory usage.
- The `cassandra.input.split.size` property specifies rows to be processed per mapper. The default size is 64k rows per split. You can decrease the split size to create more mappers.

#### Out of Memory Errors

When your mapper or reduce tasks fail, reporting Out of Memory (OOM) errors, turn the `mapred.map.child.java.opts` setting in Hive to:

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

## Loading balancing using the Fair Scheduler

The [Hadoop Fair Scheduler](#) assigns resources to jobs to balance the load, so that each job gets roughly the same amount of CPU time.

To enable the fair scheduler, uncomment a section in the `mapred-site.xml` that looks something like this:

```
<property>
 <name>mapred.jobtracker.taskScheduler</name>
```

```

<value>org.apache.hadoop.mapred.FairScheduler</value>
</property>
. .
<value>dse-3.0.2/dse/resources/hadoop/conf/fair-scheduler.xml</value>
</property>
```

You might need to change the value element shown here. If the Fair Scheduler file has a different name, change the name of the file to fair-scheduler.xml. Specify the absolute path to the file.

DataStax Enterprise also supports the [Capacity Scheduler](#).

## Starting the Hive server

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

Installer-Services and Package installations:

```
subcommand hive --service hiveserver
```

Installer-No Services and Tarball installations:

```
$ install_location/bin/dse [-u username -p password] hive --service hiveserver
```

### Starting the HiveServer2

DataStax Enterprise integrates [Apache HiveServer2](#), an improved version of HiveServer that supports multi-client concurrency and other features.

To start HiveServer2, run this command:

```
$ dse [-u username -p password] hive --service hiveserver2
```

After starting HiveServer2, use the [Beeline command shell](#) to connect to the server and run Hive queries.

## Using Beeline

DataStax Enterprise supports the Beeline client for use with the Spark SQL Thrift Server:

- Hive 0.13 Beeline (`dse spark-beeline`) is for Spark SQL Thrift Server, supported in DataStax Enterprise 4.8.0 and later releases
- Hive 0.12 Beeline (`dse beeline`) is for HiveServer2, supported in DataStax Enterprise 4.7.x and earlier releases

**Note:** To show the command line help for `dse spark-beeline`:

```
$ dse spark-beeline --help
```

## Beeline with Spark SQL Thrift Server

1. In a terminal window, start Beeline for Spark SQL Thrift Server.

```
$ dse spark-beeline
```

On Linux, for example:

```
$ install_directory/bin/dse [-u username -p password] spark-beeline
```

The beeline prompt appears.

```
2015-06-19 06:37:22.758 java[46121:1a03] Unable to load realm info from
SCDynamicStore
Beeline version 0.12.0.3-SNAPSHOT by Apache Hive
beeline>
```

2. Connect to the server. On a single-node, development cluster for example:

```
beeline> !connect jdbc:hive2://localhost:10000
```

The HiveServer2 prompt appears.

```
scan complete in 24ms
Connecting to jdbc:hive2://localhost
Enter username for jdbc:hive2://localhost:
```

3. Enter the password.

The hive2 prompt appears.

```
Connected to: Hive (version 0.12.0.3-SNAPSHOT)
Driver: Hive (version 0.12.0.3-SNAPSHOT)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000>
```

4. Run Hive queries.

## Recreating Hive metadata after decommissioning a node

After removing/decommissioning a node that stored the Hive metadata, truncate the Hive metadata table, then recreate the table. In the `hive-site.xml` file, set the parameters as shown in the following example to specify a different keyspace and table for the Hive metastore:

```
<property>
 <name>cassandra.connection.metaStoreKeyspaceName</name>
 <value>newKeyspaceName</value>
</property>
<property>
 <name>cassandra.connection.metaStoreColumnFamilyName</name>
 <value>MetaStore</value>
</property>
```

This action is necessary to prevent an exception in `SemanticAnalyzer.genFileSinkPlan`.

After decommissioning a node in an analytics datacenter, run the `dsetool cleanup_leases` command to clear old LeaderManager entries.

## DataStax ODBC driver for Hive on Windows

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

### Prerequisites

- Windows® 7 Professional or Windows® 2008 R2. Both 32- and 64-bit editions are supported.

- Microsoft Visual C++ 2010 runtime.
- A cluster with a Hadoop node running the Hive server. See [Starting the Hive server](#).

To install the DataStax ODBC driver on a Windows platform:

## Procedure

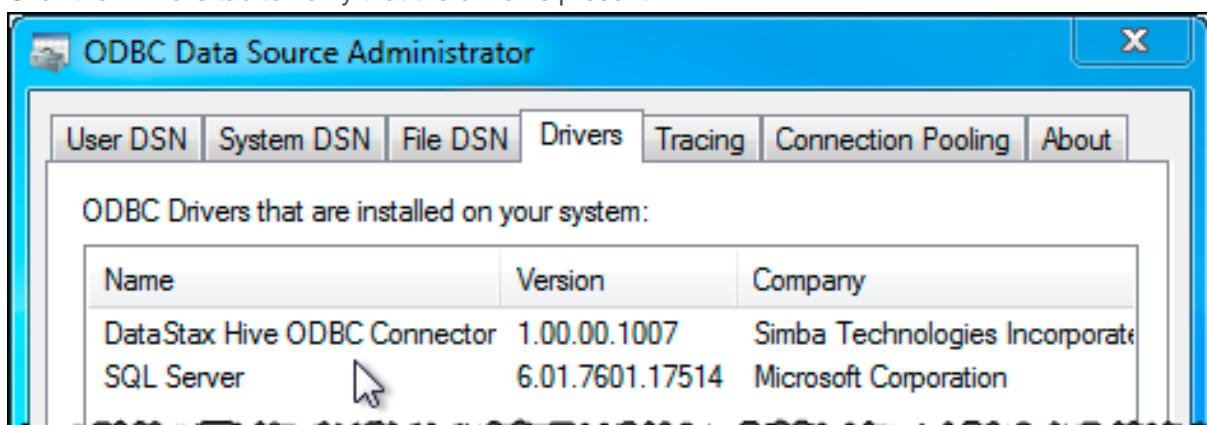
1. Download the driver from [Client Libraries and CQL Drivers](#).
2. Double-click the downloaded file and follow the wizard instructions.

## Configuring the driver

Set up the DataStax ODBC driver for access by your BI tool.

## Procedure

1. Click **Start Program Files > DataStax Hive ODBC Connector > ODBC Driver Manager**.
2. Click the **Drivers** tab to verify that the driver is present.



3. Create either a User or System DSN (data source name) for your BI tool connection.
  - a) Click the **User DSN** or **System DSN** tab.
  - b) Click **Add > DataStax Hive ODBC Connector > Finish**.
  - c) In **DataStax Hive ODBC Connector Setup**, enter the following:

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

- d) Click **Test**.

The test results are displayed.

**Note:** If your DataStax Enterprise cluster is on Amazon EC2, you must open the listening port for the Hive Server.

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

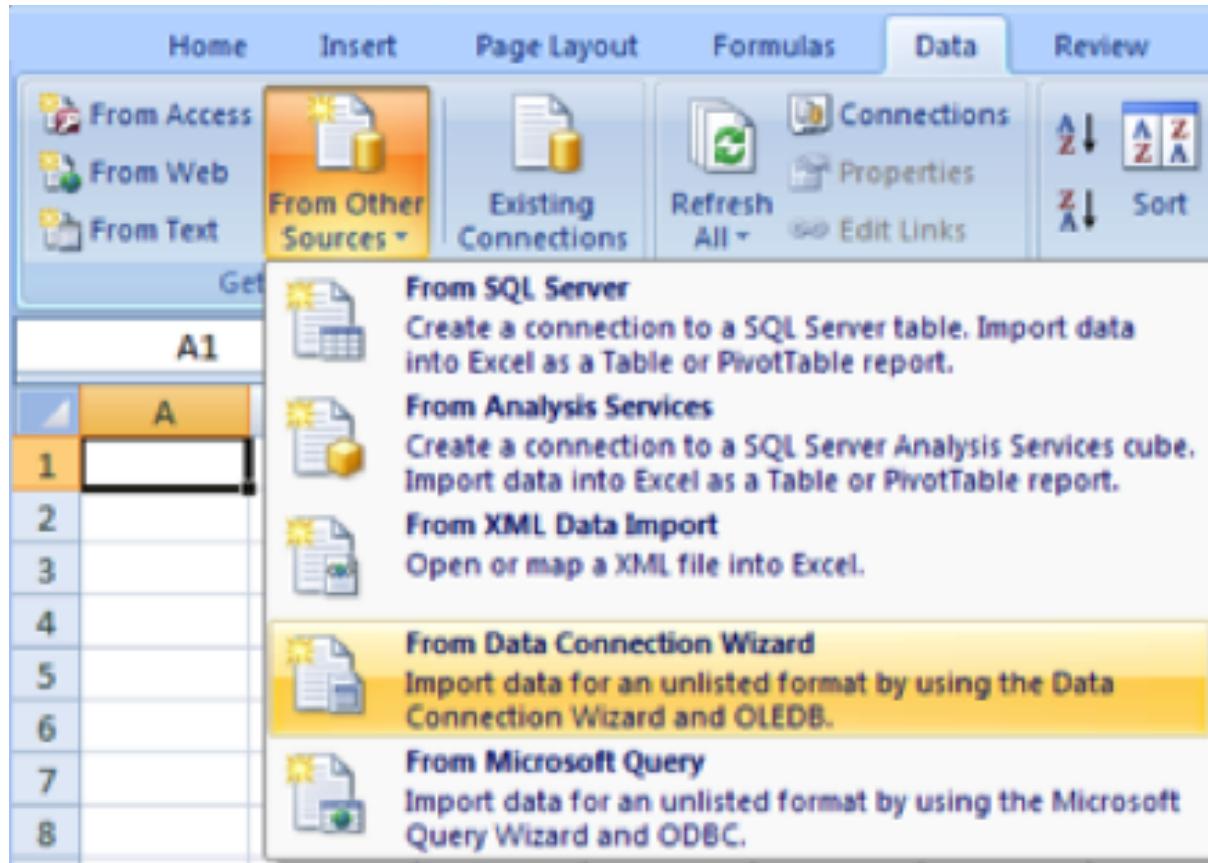
Start > Program Files > DataStax Hive ODBC Connector > User's Guide

## Using the DataStax ODBC driver for Hive

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

### Procedure

1. Use the data connection wizard to select your new ODBC data source:



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

	A	B	C
1	ticker	date	return
2	AAN	2012-05-26	-21.37197212
3	AAN	2012-05-27	12.05748905
4	AAN	2012-05-28	282.616366
5	AAN	2012-05-29	133.431814
6	AAN	2012-05-30	435.8902556
7	AAN	2012-05-31	-149.7094888
8	AAN	2012-06-01	31.77952289
9	AAN	2012-06-02	-747.2978294
10	AAN	2012-06-03	505.1547267
11	AAN	2012-06-04	50.65625204

## Results

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

```
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201208230939_0006,
 Tracking URL = http://localhost:50030/jobdetails.jsp?
jobid=job_201208230939_0006
Kill Command = ./dse hadoop job
-Dmapred.job.tracker=127.0.0.1:8012 -kill job_201208230939_0006
Hadoop job information for Stage-1: number of mappers: 1; number of
reducers: 0
2012-08-23 12:44:39,795 Stage-1 map = 0%, reduce = 0%
2012-08-23 12:44:42,824 Stage-1 map = 100%, reduce = 0%
2012-08-23 12:44:44,833 Stage-1 map = 100%, reduce = 100%
Ended Job = job_201208230939_0006
MapReduce Jobs Launched:
Job 0: Map: 1 HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
```

# Using Mahout

DataStax Enterprise integrates [Apache Mahout](#), a Hadoop component that offers machine learning libraries. Mahout facilitates building intelligent applications that learn from data and user input. Machine learning use cases are many and some, such as the capability of web sites to recommend products to visitors based on previous visits, are notorious.

Currently, Mahout jobs that use Lucene features are not supported.

## Running the Mahout demo

The DataStax Enterprise installation includes a Mahout demo. The demo determines with some percentage of certainty which entries in the input data remained statistically in control and which have not. The input data is time series historical data. Using the Mahout algorithms, the demo classifies the data into categories based on whether it exhibited relatively stable behavior over a period of time. The demo produces a file of classified results. This procedure describes how to run the Mahout demo.

## Procedure

**Note:** DataStax Demos do not work with either LDAP or internal authorization (username/password) enabled.

1. After installing DataStax Enterprise, start an analytics node.
2. Go to the `demos/mahout` directory.

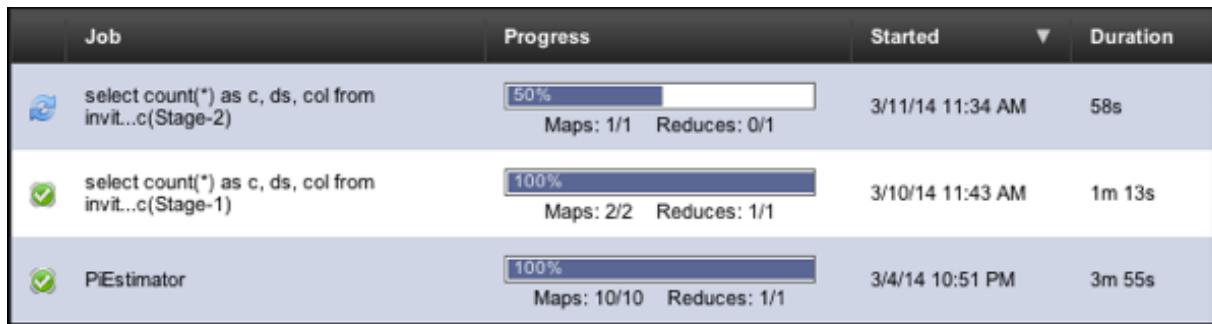
The default location of the `demos/mahout` directory depends on the type of installation:

Installer-Services and Package installations	<code>/usr/share/dse/demos/mahout</code>
Installer-No Services and Tarball installations	<code>install_location/demos/mahout</code>

3. Run the script in the `demos` directory. For example, on Linux:

```
./run_mahout_example.sh
```

If you are running OpsCenter, you can now view the Hadoop job progress:



When the demo completes, a message appears on the standard output about the location of the output file. For example:

```
The output is in /tmp/clusteranalyze.txt
```

# Using Mahout commands in DataStax Enterprise

You can run Mahout commands on the `dse` command line. For example on Mac OS X, to get a list of which commands are available:

```
$ cd install_location
$ bin/dse mahout
```

The list of commands appears.

## Mahout command line help

You use one of these commands as the first argument plus the help option:

```
$ cd install_location
$ bin/dse mahout arff.vector --help
```

The output is help on the `arff.vector` command.

## Add Mahout classes to the class path, execute Hadoop command

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

```
$ cd install_location
$ bin/dse mahout hadoop fs -text mahout_file | more
```

The Apache web site offers an [in-depth tutorial](#).

# Using Pig

DataStax Enterprise includes a [Cassandra File System \(CFS\)](#) enabled Apache Pig Client. [Pig](#) is a high-level programming environment for MapReduce coding. You can explore big data sets using the Pig Latin data flow language for programmers. Relations, which are similar to tables, are constructed of tuples, which correspond to the rows in a table. Unlike a relational database table, Pig relations do not require every tuple to contain the same number of fields. Fields in the same position (column) need not be of the same type. Using Pig, you can devise logic for data transformations, such as filtering data and grouping relations. The transformations occur during the MapReduce phase.

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

## Support for TTL

You can set the [TTL](#) (time to live) on Pig data. You use the `cql://` URL, which includes a prepared statement shown in [step 10 of the library demo](#).

## Support for CQL collections

Pig in DataStax Enterprise supports CQL collections. Pig-supported types must be used.

## Running the Pig demo

Three examples demonstrate how to use Pig to work with CQL tables.

- [How to save Pig relations from/to Cassandra](#)  
Pig uses a single tuple.
- [How to work with a Cassandra compound primary key in Pig](#)  
Pig uses three tuples, one for the partition key and two for the two clustering columns.
- [How to use Pig to set up logic for exploring library data](#)

This example from the [Cassandra and Pig tutorial](#) shows how to copy public library data into Cassandra, add logic to save the data to a Pig relation, execute programs by running MapReduce jobs, and view results in a Cassandra table.

### Start Pig

#### Procedure

**Note:** DataStax Demos do not work with either LDAP or internal authorization (username/password) enabled.

1. Start DataStax Enterprise as an analytics (Hadoop) node:

- **Installer-Services and Package installations:**

1. Set HADOOP\_ENABLED=1 in /etc/default/dse.
2. Start an analytics node:

```
$ sudo service dse start
```

- **Installer-No Services and Tarball installations:**

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

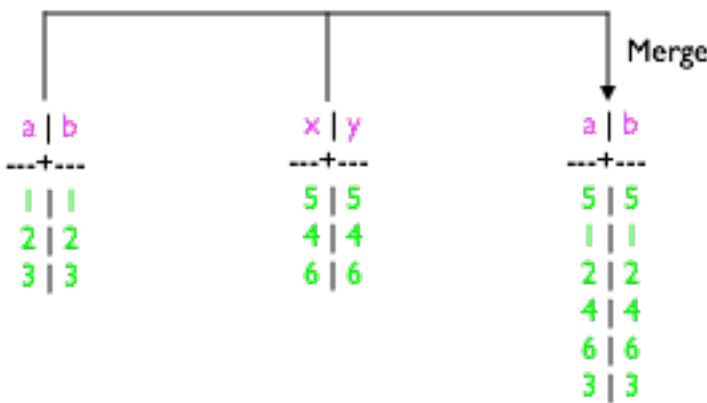
2. Start the Pig shell:

- Installer-Services and Package installations: \$ dse pig
- Installer-No Services and Tarball installations: \$ DSE\_install\_location/bin/dse pig

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

### Example: Save Pig relations from/to Cassandra

For Pig to access data in Cassandra, the target keyspace and table must already exist. Pig can save data from a Pig relation to a table in Cassandra and from a Cassandra table to a pig relation, but it cannot create the table. This example shows how to merge the data from two CQL tables having simple primary keys using Pig.



A [subsequent example](#) shows how to merge data from CQL tables having compound primary keys into one CQL table using Pig.

## Procedure

1. Start [cqlsh](#).
2. Using cqlsh, create and use a keyspace named, for example, cql3ks.

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

cqlsh> USE cql3ks;
```

3. Create a two-column (a and b) Cassandra table named simple\_table1 and another two-column (x and y) table named simple\_table2. Insert data into the tables.

```
cqlsh:cql3ks> CREATE TABLE simple_table1 (a int PRIMARY KEY, b int);
cqlsh:cql3ks> CREATE TABLE simple_table2 (x int PRIMARY KEY, y int);
cqlsh:cql3ks> INSERT INTO simple_table1 (a,b) VALUES (1,1);
cqlsh:cql3ks> INSERT INTO simple_table1 (a,b) VALUES (2,2);
cqlsh:cql3ks> INSERT INTO simple_table1 (a,b) VALUES (3,3);
cqlsh:cql3ks> INSERT INTO simple_table2 (x, y) VALUES (4,4);
cqlsh:cql3ks> INSERT INTO simple_table2 (x, y) VALUES (5,5);
cqlsh:cql3ks> INSERT INTO simple_table2 (x, y) VALUES (6,6);
```

4. Using Pig, add logic to load the data (4, 5, 6) from the Cassandra simple\_table2 table into a Pig relation.

```
grunt> moretestvalues= LOAD 'cql://cql3ks/simple_table2/' USING
CqlNativeStorage;
```

5. Convert the simple\_table2 table data to a tuple. The key column is a chararray, 'a'.

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

6. Save the relation to the Cassandra simple\_table1 table.

```
grunt> STORE insertformat INTO
 'cql://cql3ks/simple_table1?output_query=UPDATE
+cql3ks.simple_table1+set+b+%3D+%3F'
 USING CqlNativeStorage;
```

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

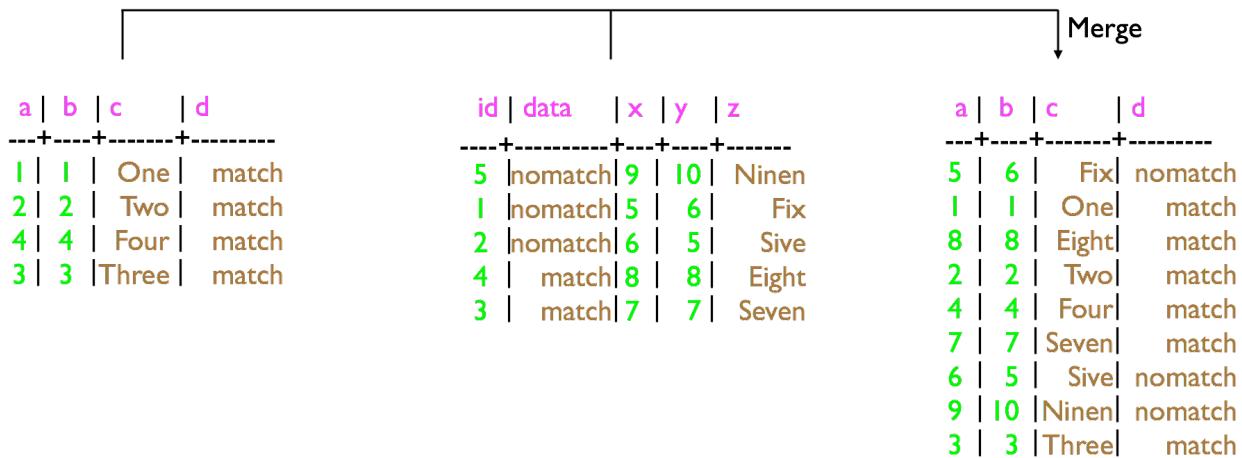
- On the cqlsh command line, check that the simple\_table1 table now contains its original values plus the values from the simple\_table2 table:

```
cqlsh:cql3ks> SELECT * FROM simple_table1;
```

a	b
5	5
1	1
2	2
4	4
6	6
3	3

## Example: Handle a compound primary key

This example, like the previous one, shows you how to work with CQL tables in Pig. The previous example used tables having a simple primary key. The tables in this example use compound primary keys. You create the tables in cqlsh and merge them using Pig.



## Procedure

- Create a four-column (a, b, c, d) Cassandra table named table1 and another five-column (id, x, y, z, data) table named table2.

```
cqlsh:cql3ks> CREATE TABLE table1 (
 a int,
 b int,
 c text,
 d text,
 PRIMARY KEY (a,b,c)
);
cqlsh:cql3ks> CREATE TABLE table2 (
 id int PRIMARY KEY,
```

```

 x int,
 y int,
 z text,
 data text
) ;

```

**2. Insert data into the tables.**

```

cqlsh:cql3ks> INSERT INTO table1 (a, b , c , d)
 VALUES (1,1,'One','match');
cqlsh:cql3ks> INSERT INTO table1 (a, b , c , d)
 VALUES (2,2,'Two','match');
cqlsh:cql3ks> INSERT INTO table1 (a, b , c , d)
 VALUES (3,3,'Three','match');
cqlsh:cql3ks> INSERT INTO table1 (a, b , c , d)
 VALUES (4,4,'Four','match');
cqlsh:cql3ks> INSERT INTO table2 (id, x, y, z,data)
 VALUES (1,5,6,'Fix','nomatch');
cqlsh:cql3ks> INSERT INTO table2 (id, x, y, z,data)
 VALUES (2,6,5,'Sive','nomatch');
cqlsh:cql3ks> INSERT INTO table2 (id, x, y, z,data)
 VALUES (3,7,7,'Seven','match');
cqlsh:cql3ks> INSERT INTO table2 (id, x, y, z,data)
 VALUES (4,8,8,'Eight','match');
cqlsh:cql3ks> INSERT INTO table2 (id, x, y, z,data)
 VALUES (5,9,10,'Ninen','nomatch');

```

**3. Using Pig, add logic to load the data from the Cassandra table2 to a Pig relation.**

```
grunt> moredata = load 'cql://cql3ks/table2' USING CqlNativeStorage;
```

**4. Convert the data to a tuple.**

```

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

```

During the actual data processing, the data is formatted as follows:

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

**5. Save the Pig relation to the Cassandra table1 table. The data from table 1 and table 2 will be merged.**

```
grunt> STORE insertformat INTO 'cql://cql3ks/table1?output_query=UPDATE
%20cql3ks.table1%20SET%20d%20%3D%20%3F' USING CqlNativeStorage;
```

The cql:// URL includes a prepared statement, [described later](#), that needs to be copied/pasted as a continuous string (no spaces or line breaks).

**6. In cqlsh, query table1 to check that the data from table1 and table2 have been merged.**

```
cqlsh:cql3ks> SELECT * FROM table1;
```

a	b	c	d
5	6	Fix	nomatch
1	1	One	match
8	8	Eight	match
2	2	Two	match

4	4	Four	match
7	7	Seven	match
6	5	Sive	nomatch
9	10	Ninen	nomatch
3	3	Three	match

## Example: Explore library data

This example uses library data from the Institute of Library and Museum Services, encoded in UTF-8 format. [Download](#) the formatted data for this example now.

DataStax Enterprise installs files in the following directory that you can use to run through this example using a pig script instead of running Pig commands manually.

- Installer-Services and Package installations: /usr/share/demos/pig/cql
- Installer-No Services and Tarball installations: *install-location*/demos/pig/cql

Using the files is optional. To use the files, copy/paste the commands in steps 2-3 from the `library-populate-cql.txt` file and execute steps 7-10 automatically by running the `library-cql.pig` script.

### Procedure

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

```
$ chmod 777 libdata.csv
```

2. Create and use a keyspace called `libdata`.

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

cqlsh:libdata> USE libdata;
```

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

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

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

```
cqlsh:libdata> COPY libout
 ("STABR", "FSCSKEY", "FSCS_SEQ", "LIBID", "LIBNAME",
 "ADDRESS", "CITY", "ZIP", "ZIP4", "CNTY", "PHONE", "C_OUT_TY",
 "C_MSA", "SQ_FEET", "F_SQ_FT", "L_NUM_BM", "F_BKMOB", "HOURS",
 "F_HOURS", "WKS_OPEN", "F_WKSOPN", "YR_SUB", "STATSTRU", "STATNAME",
 "STATADDR", "LONGITUD", "LATITUDE", "FIPSST", "FIPSCO", "FIPSPLAC",
 "CNTYPOP", "LOCALE", "CENTRACT", "CENBLOCK", "CDCODE", "MAT_CENT",
 "MAT_TYPE", "CBSA", "MICROF") FROM 'libdata.csv' WITH
 HEADER=TRUE;
```

In the FROM clause of the [COPY command](#), use the path to libdata.csv in your environment.

- Check that the libout table contains the data you copied from the downloaded file.

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

17598
```

- Create a table to hold results of Pig relations.

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

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

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

- Add logic to remove data about outlet types other than books-by-mail (BM). The C\_OUT\_TY column uses BM and other abbreviations to identify these library outlet types:

- CE-Central Library
- BR-Branch Library
- BS-Bookmobile(s)
- BM-Books-by-Mail Only

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

- Add logic to filter out the library data that has missing building size data, define the schema for libdata\_buildings, and group data by state. The STABR column contains the state codes. GROUP creates the state\_grouped relation. Pig gives the grouping field the default alias group. Process each row to generate a derived set of rows that aggregate the square footage of each state group.

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

```
grunt> DUMP state_footage;
```

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

```
. . .
(UT,1510353)
(VA,4192931)
(VI,31875)
(VT,722629)
(WA,3424639)
(WI,5661236)
(WV,1075356)
(WY,724821)
```

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

```
grunt> insert_format= FOREACH state_footage GENERATE
 TOTUPLE(TOTUPLE('year',2011),TOTUPLE('state',State)),TOTUPLE(TotalFeet);
grunt> STORE insert_format INTO 'cql://libdata/libsqft?output_query=UPDATE
%20libdata.%20sqft%20USING%20TTL%20300%20SET%20sqft%20%3D%20%3F' USING
CqlNativeStorage;
```

The prepared statement includes a TTL that causes the data to [expire in 5 minutes](#). Decoded the prepared statement looks like this:

```
UPDATE libdata.libsqft USING TTL 300 SET sqft = ?
```

- 11.** In CQL, query the libsqft table to see the Pig results now stored in Cassandra.

```
cqlsh> SELECT * FROM libdata.libsqft;
```

```
year | state | sqft
-----+-----+-----
2011 | AK | 570178
2011 | AL | 2792246
. . .
2011 | WV | 1075356
2011 | WY | 724821
```

## Data access using storage handlers

The DataStax Enterprise Pig driver uses the [Cassandra File System](#) (CFS) instead of the Hadoop distributed file system (HDFS). Apache Cassandra, on the other hand, includes a Pig driver that uses the Hadoop Distributed File System (HDFS).

To execute Pig programs directly on data stored in Cassandra, you use one of the DataStax Enterprise storage handlers:

Table Format	Storage Handler	URL	Description
CQL	CqlNativeStorage()	cql://	Use with DataStax Enterprise 4.7 and later.
CQL	CqlStorage()	cql://	Deprecated.
storage engine	CassandraStorage()	cassandra://	Deprecated.

**Note:** The CqlStorage handler and the CassandraStorage handler are deprecated and will be removed in a future Cassandra release. Use the CqlNativeStorage handler and the cql:// url for new pig applications. DataStax recommends migrating all tables to CqlNativeStorage as soon as possible in preparation for the removal of the CqlStorage and the CassandraStorage handlers.

## Migrating compact tables with clustering columns to CqlNativeStorage format

The CqlNativeStorage handler uses native paging through the DataStax Java driver to communicate with the underlying Cassandra cluster. To use applications that have [compact tables](#) with clustering columns in the CqlStorage format, you need to migrate tables to the CqlNativeStorage format. Attempting to run Pig commands on compact tables in the CqlStorage format results in an exception. You can, however, run Pig commands on non-compact tables in the CqlStorage format.

To migrate tables from CqlStorage to CqlNativeStorage format:

1. Identify Pig functions that interact with compact tables in CqlStorage format. For example, suppose you identify a command that adds logic to load the data to a Pig relation from the compact table tab in keyspace ks.

```
x = LOAD 'cql://ks/tab' USING CqlStorage(); -- Old function
```

```
x = LOAD 'cql://ks/tab' USING CassandraStorage(); -- Old function
```

2. Change CqlStorage() or CassandraStorage() to USING CqlNativeStorage().

```
x = LOAD 'cql://ks/tab' USING CqlNativeStorage(); -- New function
```

### URL format for CqlNativeStorage

The URL format for CqlNativeStorage is:

```
cql:// [username:password@] keyspace/table [?
[page_size=size]
[&columns=col1,col2]
[&output_query=prepared_statement_query]
[&cql_input=prepared_statement_query]
[&where_clause=clause]
[&split_size=size]
[&partitioner=partitioner]
[&use_secondary=true|false]
[&init_address=host]
[&native_port=port]]
```

where:

- page\_size -- the number of rows per page
- columns -- the select columns of CQL query
- output\_query -- the CQL query for writing in a prepared statement format
- input\_cql -- the CQL query for reading in a prepared statement format
- where\_clause -- the where clause on the index columns, which needs URL encoding
- split\_size -- number of rows per split
- partitioner -- Cassandra partitioner
- use\_secondary -- to enable pig filter partition push down
- init\_address -- the IP address of the target node
- native\_port -- the listen address of the target node

where:

- page\_size -- the number of rows per page

- columns -- the select columns of CQL query
- output\_query -- the CQL query for writing in a prepared statement format
- where\_clause -- the where clause on the index columns, which needs URL encoding
- split\_size -- number of rows per split
- partitioner -- Cassandra partitioner
- use\_secondary -- to enable pig filter partition push down
- init\_address -- the IP address of the target node
- rpc\_port -- the listen address of the target node

### Working with legacy Cassandra tables

Use the `CqlNativeStorage()` handler and `cfs://` URL to work with Cassandra tables that are in the storage engine (CLI/Thrift) format in Pig. Legacy tables are created using Thrift, CLI, or using the `WITH COMPACT STORAGE` directive in CQL. Thrift applications require that you configure Cassandra for connection to your application using the `rpc` connections instead of the default `native transport` for `CqlNativeStorage` connection.

## CQL data access

Use the `CqlNativeStorage` handler with the `input_cql` statement or the `output_query` statement. To access data in the CassandraFS, the target keyspace and table must already exist. Data in a Pig relation can be stored in a Cassandra table, but Pig will not create the table.

The Pig `LOAD` function pulls Cassandra data into a Pig relation through the storage handler as shown in this examples:

```
pig_relation_name = LOAD 'cql://keyspace/table'
 USING CqlNativeStorage();
```

DataStax Enterprise supports these Pig data types:

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

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

```
pig_relation_name = LOAD 'cql://keyspace/table'
 USING CqlNativeStorage();
```

The [Pig demo](#) examples include using the `LOAD` command.

### LOAD schema

The `LOAD Schema` is:

```
(colname:colvalue, colname:colvalue, ...)
```

where each `colvalue` is referenced by the Cassandra column name.

## CQL pushdown filter

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

- [Create an index](#) for the Cassandra table.  
For Pig pushdown filtering, the secondary index must have the same name as the column being indexed.
- Include the [use\\_secondary option](#) with a value of true in the url format for the storage handler. The option name reflects the term used to be used for a Cassandra index: secondary index. For example:

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

## Saving a Pig relation to Cassandra

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

```
STORE relation_name INTO 'cql://keyspace/column_family?prepared_statement'
USING CqlNativeStorage();
```

### Store schema

The input schema for Store is:

```
(value, value, value)
```

where each value schema has the name of the column and value of the column value.

The output schema for Store is:

```
((name, value), (name, value)), (value ... value), (value ... value))
```

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

## Creating a URL-encoded prepared statement

The Pig demo examples show the steps required for setting up a prepared CQL query using the output\_query statement:

### Procedure

#### 1. Format the data

The example of [saving Pig relations from/to Cassandra](#) shows the output schema: the name of the simple\_table1 table primary key 'a', represented as a chararray in the relation is paired with a value in the simple\_table2 table. In this case, the key for simple\_table1 table is only a partitioning key, and only a single tuple is needed.

The Pig statement to add (moredata) fields to a tuple is:

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

The example of [exploring library data](#) works with more complicated data, a partition key and clustering column:

```
grunt> insertformat = FOREACH moredata GENERATE
```

```
TOTUPLE(TOTUPLE('a',x),TOTUPLE('b',y),TOTUPLE('c',z)),TOTUPLE(data);
```

## 2. Construct the prepared query

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

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

```
'cql://cq13ks/simple_table1?output_query=UPDATE+cq13ks.simple_table1+set+b
+%3D+%3F'
```

The key values of the simple\_table1 table are automatically transformed into the 'WHERE (key) =' clause to form the output\_query portion of a prepared statement.

## 3. Execute the query

To update the simple\_table1 table using the values in the simple\_table2 (4-6), the prepared statement is executed using these WHERE clauses when the MapReduce job runs:

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

This output\_query in Pig statement forms the '...' url-encoded portion of the prepared statement:

```
grunt> STORE insertformat INTO
 'cql://cq13ks/simple_table1?output_query=UPDATE
+cq13ks.simple_table1+set+b+%3D+%3F'
 USING CqlNativeStorage;
```

Decoded the [UPDATE statement](#) is:

```
UPDATE cq13ks.simple_table1 SET b = ?
```

The prepared statement represents these queries:

```
UPDATE cq13ks.test SET b = 5 WHERE a = 5;
UPDATE cq13ks.test set b = 4 WHERE a = 4;
UPDATE cq13ks.test set b = 6 WHERE a = 6;
```

# Analyzing data using external Hadoop systems

## About BYOH

Hadoop is a software framework for distributed processing of large data sets using MapReduce programs. DataStax Enterprise works with these external Hadoop systems in a bring your own Hadoop (BYOH) model. Use BYOH to run DSE Analytics with a separate Hadoop cluster, from a different vendor. Supported vendors are:

- Hadoop 2.x data warehouse implementations Cloudera 4.5, 4.6, 5.0.x, and 5.2.x
- Hortonworks 1.3.3, 2.0.x, 2.1, and 2.2

You can use Hadoop in one of the following modes:

- External Hadoop

Uses the Hadoop distribution provided by Cloudera (CDH) or Hortonworks (HDP).

- Internal Hadoop

Uses the DSE Hadoop integrated with DataStax Enterprise.

For legacy purposes, DataStax Enterprise includes **DSE Hadoop** 1.0.4 with built-in Hadoop trackers.

Use cases for BYOH are:

- Bi-directional data movement between Cassandra in DataStax Enterprise and the Hadoop Distributed File System (HDFS)
- Hive queries against Cassandra data in DataStax Enterprise
- Data combination (joins) between Cassandra and HDFS data
- ODBC access to Cassandra data through Hive

## Components

This table compares DSE Hadoop with the external Hadoop system in the BYOH model:

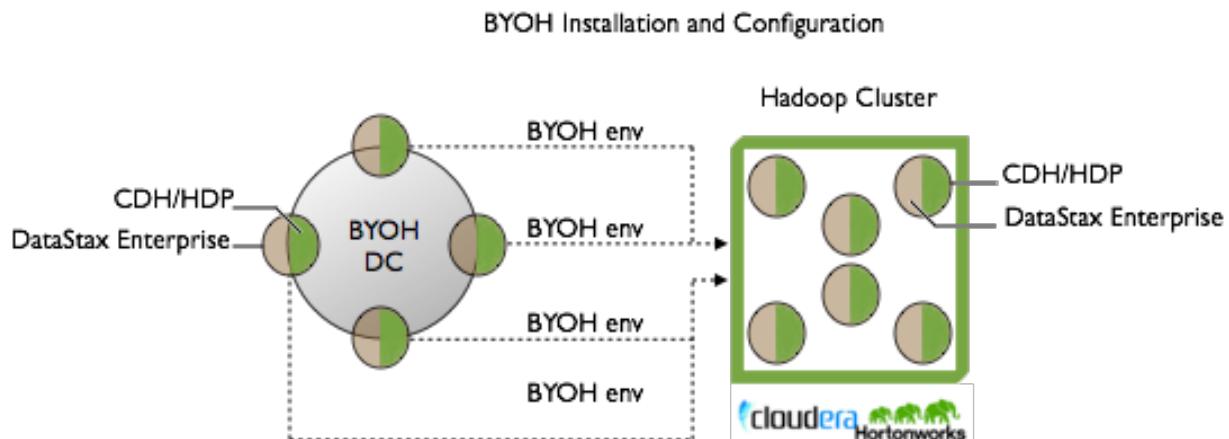
**Table: Comparison of DSE Hadoop and the BYOH model**

Component	DSE-integrated Hadoop owner	BYOH owner	DSE interaction
Job Tracker	DSE Cluster	Hadoop Cluster	Optional
Task Tracker	DSE Cluster	Hadoop Cluster	Co-located with BYOH nodes
Pig	Distributed with DSE	Distribution chosen by operator	Can launch from Task Trackers
Hive	Distributed with DSE	Distribution chosen by operator	Can launch from Task Trackers
HDFS/CFS	CFS	HDFS	Block storage

## BYOH installation and configuration

The [procedure for installing and configuring](#) DataStax Enterprise for BYOH is straight-forward.

- Ensure that you meet the [prerequisites](#).
- Install DataStax Enterprise on all nodes in the Cloudera or Hortonworks cluster and on additional nodes outside the Hadoop cluster.
- Install several Cloudera or Hortonworks components on the additional nodes and deploy those nodes in a virtual BYOH datacenter.
- [Configure DataStax Enterprise](#) BYOH environment variables on each node in the BYOH datacenter to point to the Hadoop cluster, as shown in the following diagram:



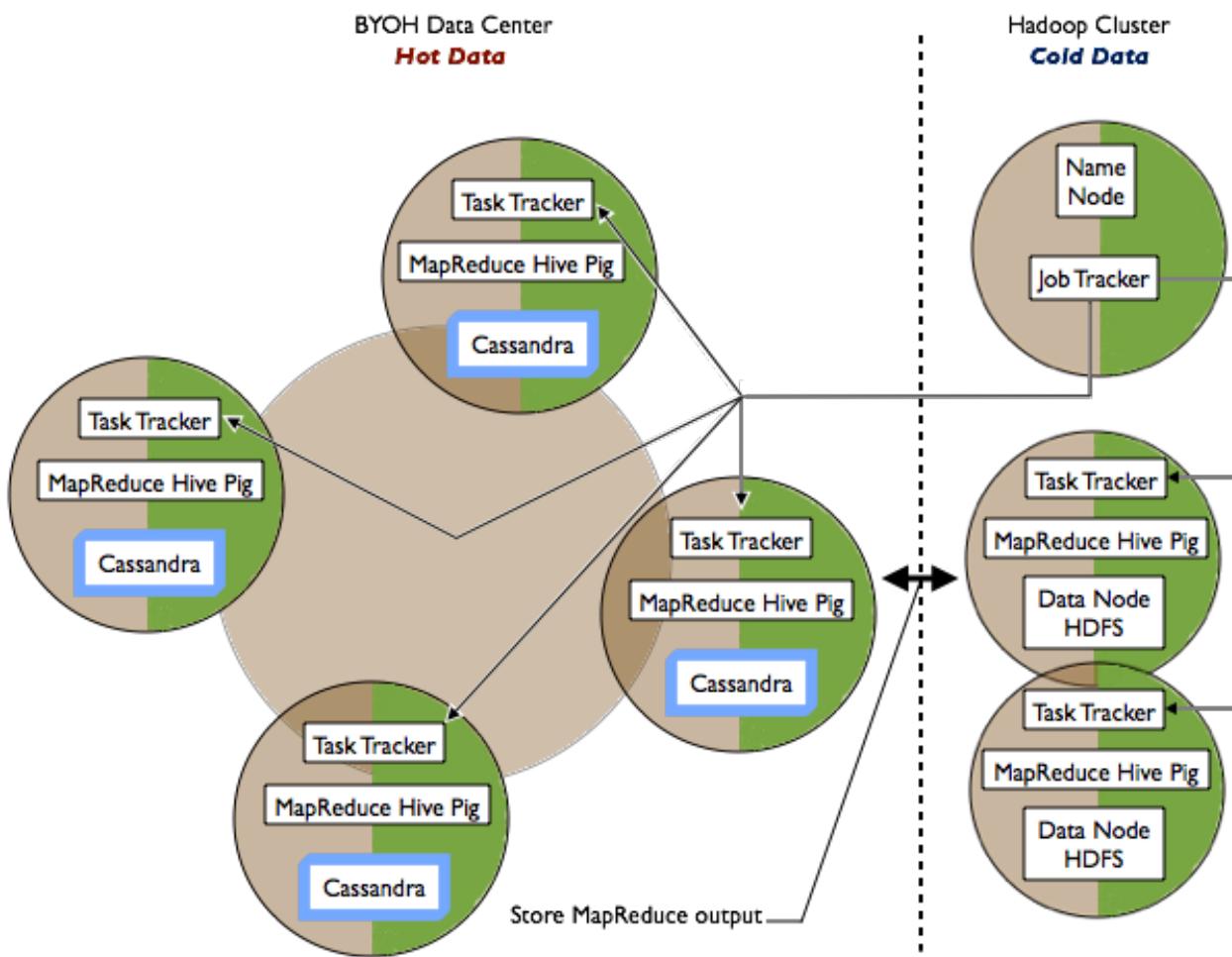
DataStax Enterprise runs only on BYOH nodes, and uses Hadoop components to integrate BYOH and Hadoop. You never start up the DataStax Enterprise installations on the Hadoop cluster.

### MapReduce process

In a typical Hadoop cluster, Task Tracker and Data Node services run on each node. A Job Tracker service running on one of the master nodes coordinates MapReduce jobs between the Task Trackers, which pull data locally from data node. For the latest versions of Hadoop using YARN, Node Manager services replace Task Trackers and the Resource Manager service replaces the Job Tracker.

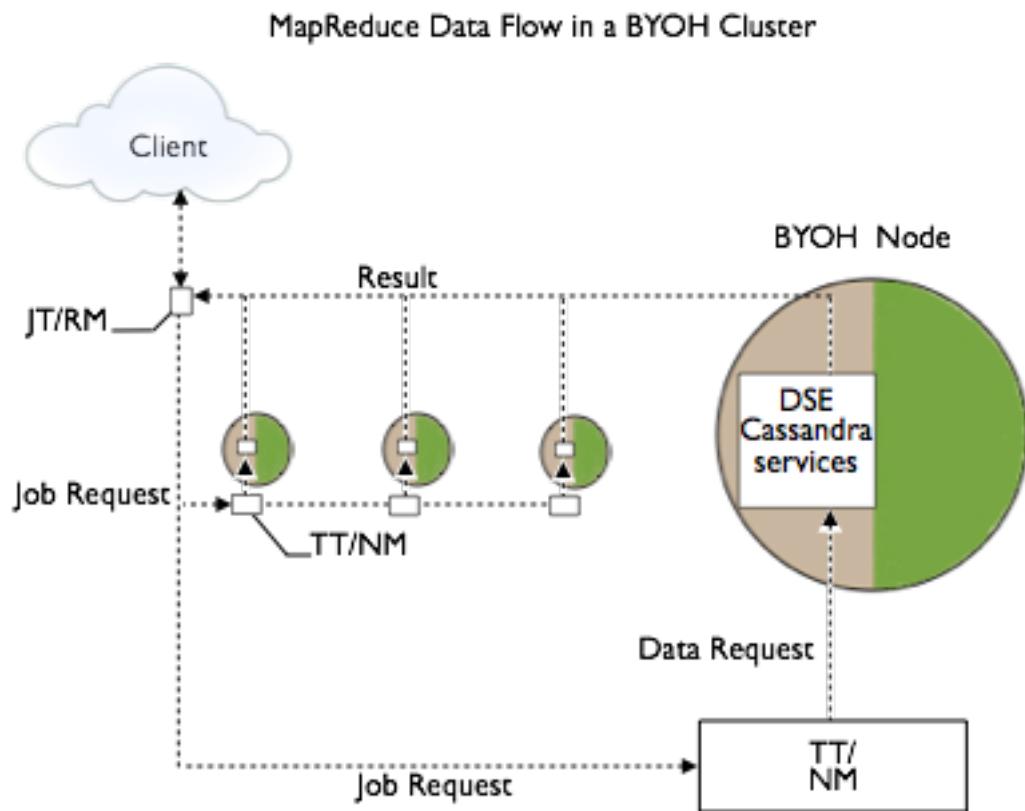
In contrast with the typical Hadoop cluster, in the BYOH model DSE Cassandra services can take the place of the Data Node service in MapReduce jobs, providing data directly to the Task Trackers/Node Managers, as shown in the following diagram. For simplicity purposes, the diagram uses the following nomenclature:

- Task Tracker--Means Task Tracker or Node Manager.
- Job Tracker--Means Job Tracker or Resource Manager.



A MapReduce service runs on each BYOH node along with optional MapReduce, Hive, and Pig clients. To take advantage of the performance benefits offered by Cassandra, BYOH handles frequently accessed hot data. The Hadoop cluster handles less-frequently and rarely accessed cold data. You design the MapReduce application to store output in Cassandra or Hadoop.

The following diagram shows the data flow of a job in a BYOH datacenter. The Job Tracker/Resource Manager (JT/RM) receives MapReduce input from the client application. The JT/RM sends a MapReduce job request to the Task Trackers/Node Managers (TT/NM) and optional clients, MapReduce, Hive, and Pig. The data is written to Cassandra and results sent back to the client.



## BYOH workflow

BYOH clients submit Hive jobs to the Hadoop Job Tracker or ResourceManager in the case of YARN. If Cassandra is the source of the data, the Job Tracker evaluates the job, and the ColumnFamilyInputFormat creates input splits and assigns tasks to the various Task Trackers in the Cassandra node setup (giving the jobs local data access). The Hadoop job runs until the output phase.

During the output phase if Cassandra is the target of the output, the HiveCqlOutputFormat writes the data back into Cassandra from the various reducers. During the reduce step, if data is written back to Cassandra, locality is not a concern and data gets written normally into the cluster. For Hadoop in general, this pattern is the same. When spilled to disk, results are written to separate files, partial results for each reducer. When written to HDFS, the data is written back from each of the reducers.

Intermediate MapReduce files are stored on the local disk or in temporary HDFS tables, depending on configuration, but never in CFS. Using the BYOH model, Hadoop MapReduce jobs can access Cassandra as a data source and write results back to Cassandra or Hadoop.

## BYOH Prerequisites and installation

You must install DataStax Enterprise on all the nodes, nodes in the Hadoop cluster, and additional nodes outside the Hadoop cluster. Configure the additional nodes in one or more BYOH datacenters to **isolate workloads**. Run sequential data loads, not random OLTP loads or Solr data loads in a BYOH datacenter.

### Prerequisites

The prerequisites for installing and using the BYOH model are:

- Installation of a functioning CDH or HDP Hadoop cluster.
- Installation and configuration of these master services on the Hadoop cluster:
  - Job Tracker or Resource Manager (required)

- HDFS Name Node (required)
- Secondary Name Node or High Availability Name Nodes (required)
- At least one set of HDFS Data Nodes (required externally)

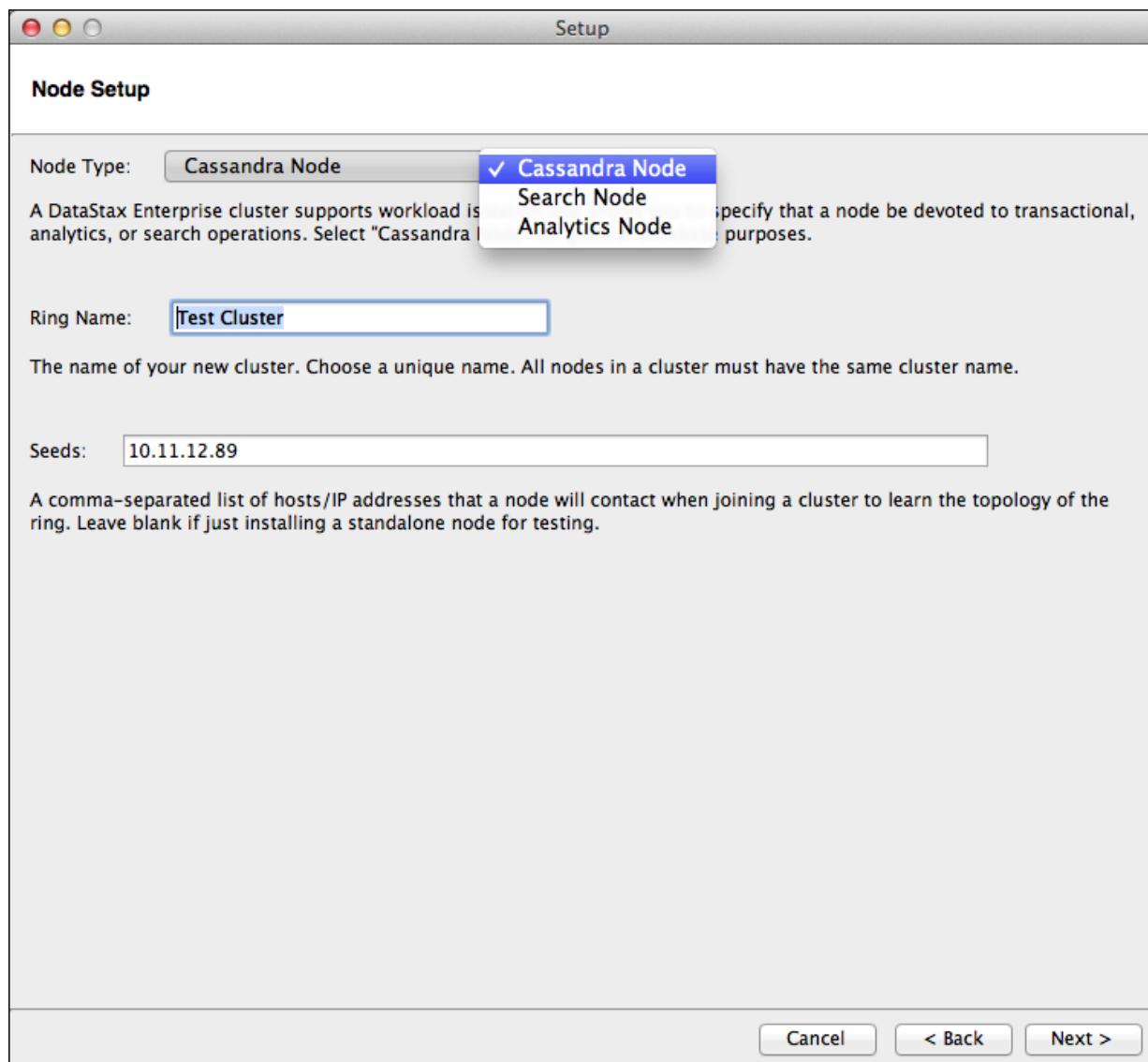
The BYOH nodes must be able to communicate with the HDFS Data Node that is located outside the BYOH data center.

During the installation procedure, you install only the required Hadoop components in the BYOH datacenter: Task Trackers/Node Managers and optional clients, MapReduce, Hive, and Pig. Install Hadoop on the same paths on all nodes. CLASSPATH variables that are used by BYOH need to work on all nodes.

## Installation procedure

To install DataStax Enterprise:

1. Ensure that you meet the prerequisites.
2. On each node in the BYOH and Hadoop cluster, install but do not start up **DataStax Enterprise**. Install DataStax Enterprise as a plain Cassandra node, not to run CFS, Solr, or integrated Hadoop. If you are using the GUI installer, on Node Setup, select Cassandra Node for Node Type.



3. On packaged installations on the Hadoop cluster only, remove the `init.d` startup files for DataStax Enterprise and DataStax Enterprise Agent. For example, as root, stop DSE processes if they started up automatically, and then remove the files:

```
$ sudo /etc/init.d/dse stop
$ sudo /etc/init.d/datastax-agent stop
$ sudo rm -rf /etc/init.dse
$ sudo rm /etc/init.d/datastax-agent
```

Removing the startup files prevents accidental start up of DataStax Enterprise on the Hadoop cluster.

4. Deploy only the BYOH nodes in a virtual datacenter.
5. After configuring the `cassandra.yaml` and `dse.yaml` files as described in [instructions for deploying the datacenter](#), copy both files to the nodes in the Hadoop cluster, overwriting the original files.
6. Observe [workload isolation](#) best practices. Do not enable `vnodes`.
7. Install the following Hadoop components and services on the BYOH nodes.
  - Task Tracker or Node Manager (required)
  - MapReduce (required).
  - Clients you want to use: Hive or Pig, for example (optional)

Including the HDFS Data Node in the BYOH datacenter is optional, but not recommended.

### Separating workloads

Use separate datacenters to [deploy mixed workloads](#). Within the same datacenter, do not mix nodes that run DSE Hadoop integrated Job Tracker and Task Trackers with external Hadoop services. In the BYOH mode, run external Hadoop services on the same nodes as Cassandra. Although you can [enable CFS](#) on these Cassandra nodes as a startup option, CFS as a primary data store is not recommended.

## Configuring an external Hadoop system

You perform a few configuration tasks after installation of DataStax Enterprise.

- Configure Kerberos on the Hadoop cluster.
- Configure Java on the Hadoop cluster.
- Install Hive 0.12 on the Hadoop cluster.
- Configure BYOH environment variables on nodes in the BYOH datacenter.

### Configuring Kerberos (optional)

To use Kerberos to protect your data, configure Hadoop security under Kerberos on your Hadoop cluster. For information about configuring Hadoop security, see "[Using Cloudera Manager to Configure Hadoop Security](#)" or the [Hortonworks documentation](#).

### Configuring Java

BYOH requires that the external Hadoop system use the same Java version as DataStax Enterprise. Ensure that the Cloudera and Hortonworks clusters are configured to use it.

### Configuring Hive

Configure nodes to use Hive or Pig, generally the one that is provided with Cloudera or Hortonworks. Additional configuration is not required for BYOH with Apache and Cloudera versions of Hive versions 0.11 to 0.14.

1. If your Hadoop distribution is a version of Hive other than 0.11 to 0.14, follow these steps to install one of the supported versions.
2. For example, download Hive 0.12 <http://apache.mirrors.pair.com/hive/hive-0.12.0/hive-0.12.0.tar.gz>.

3. Unpack the archive to install Hive 0.12.

```
$ tar -xzvf hive-0.12.0.tar.gz
```

4. If you move the Hive installation, avoid writing over the earlier version that was installed by Cloudera Manager or Ambari. For example, rename the Hive fork if necessary.
5. Move the Hive you installed to the following location:

```
$ sudo mv hive-0.12.0 /usr/lib/hive12
```

After making the changes, restart the external Hadoop system. For example, restart the CDH cluster from the Cloudera Manager-Cloudera Management Service drop-down. Finally, configure BYOH environment variables before using DataStax Enterprise.

## Configuring BYOH environment variables

The DataStax Enterprise installation includes the `byoh-env.sh` configuration file that sets up the DataStax Enterprise environment. Make these changes on all nodes in the BYOH datacenter. BYOH automatically extracts the Hive version from `$HIVE_HOME/lib/hive-exec*.jar` file name.

1. Open the `byoh-env.sh` file.
2. Set the `DSE_HOME` environment variable to the DataStax Enterprise installation directory.

- Package installations:

```
export DSE_HOME="/etc/dse"
```

- Installer-Services installations:

```
export DSE_HOME="/usr/share/dse"
```

- Installer-No Services and Tarball installations:

```
export DSE_HOME="install_location"
```

3. Edit the `byoh-env.sh` file to point the BYOH configuration to the Hive version and the Pig version.

```
HIVE_HOME="/usr/lib/hive"
PIG_HOME="/usr/lib/pig"
```

**Note:** You can manually change the Hive version in the `HIVE_VERSION` environment variable in `hive-env.sh`.

4. Check that other configurable variables match the location of components in your environment.
5. Configure the `byoh-env.sh` for using Pig by editing the IP addresses to reflect your environment. On a single node, cluster for example:

```
export PIG_INITIAL_ADDRESS=127.0.0.1
export PIG_OUTPUT_INITIAL_ADDRESS=127.0.0.1
export PIG_INPUT_INITIAL_ADDRESS=127.0.0.1
```

6. If a Hadoop data node is not running on the local machine, configure the `DATA_NODE_LIST` and `NAME_NODE` variables as follows:

- `DATA_NODE_LIST`

Provide a comma-separated list of Hadoop data node IP addresses this machine can access. The list is set to `mapreduce.job.hdfs-servers` in the client configuration.

- `NAME_NODE`

Provide the name or IP address of the name node. For example:

```
export DATA_NODE_LIST="192.168.1.1, 192.168.1.2, 192.168.1.3"
export NAME_NODE="localhost"
```

If a Hadoop data node is running on the local machine, leave these variables blank. For example:

```
export DATA_NODE_LIST=
export NAME_NODE=
```

## Starting up the BYOH datacenter

After you install and configure DataStax Enterprise on all nodes, start the seed nodes first, and then start the rest of the nodes, as described in [Multiple datacenter deployment](#).

### Installer-Services and Package installations:

1. Check the `/etc/default/dse` file to ensure that DSE Hadoop and DSE Search are disabled:

- `HADOOP_ENABLED=0` - Disables the DSE Hadoop integrated Job Tracker and Task Tracker services.
- `SOLR_ENABLED=0` - Disables the capability to run DSE Search workloads.

DataStax does not support using the `SOLR_ENABLED` and `HADOOP_ENABLED` options in BYOH deployments.

2. Start each BYOH node using the following command.

```
$ sudo service dse start
```

3. Check that the BYOH cluster is up and running.

```
$ dsetool status
```

### Installer-No Services and Tarball installations:

Start DataStax Enterprise in Cassandra mode, not Analytics mode.

1. From the installation directory, start up each BYOH node in Cassandra mode.

```
$ bin/dse cassandra
```

Do not use the `-t` option to start a BYOH node.

2. Check that the BYOH cluster up and running.

```
$ cd install_location
$ bin/dsetool status
```

## Using BYOH

Usage patterns for BYOH are the same as typical MapReduce usage patterns. Hadoop jobs run through Pig, Hive, or other MapReduce jobs. To access Cassandra data when working with the external Hadoop system, use the `byoh` command. For example, on Linux in the `bin` directory, prepend `byoh` to a Pig or Hive command. You can access the following data:

- Cassandra data in CQL or Thrift format using an application or utility, such as `cqlsh`.
- Data stored in HDFS through Pig or Hive.

## Using CFS

DataStax does not recommend using the CFS as a primary data store. However, if you need to use CFS as a data source, or as the output destination for a BYOH job, you can run the [dse command](#) with the `-c` option when you start nodes. This option enables CFS, but not the integrated DSE Job Trackers and task trackers.

To migrate data from the CFS to HDFS, use [distcp](#), or an alternative tool. Copy data from one HDFS to another either before or after the transition to BYOH.

## Running the DSE Analytics Demos

You can run the [portfolio demo](#) against your installation of BYOH to test it.

## Using Hive with BYOH (Deprecated)

BYOH Hive is deprecated and will be removed in a future release.

[Apache Hive](#) is a data warehouse system for Hadoop that projects a relational structure onto data stored in Hadoop-compatible file systems. Documentation about DataStax Enterprise [DSE Hadoop](#) provides a general introduction to Hive for new users.

BYOH capabilities connect DataStax Enterprise to a Hive MapReduce client in the external Hadoop system for querying the data using a SQL-like language called HiveQL.

[Start Hive](#) on a Cassandra BYOH node, and then run MapReduce queries directly on data outside or inside Cassandra. Use a Hive managed table to query data outside of Cassandra. Hive manages storing and deleting the data in a Hive managed table. Use a Hive external table to query data in Cassandra. Cassandra manages storing and deleting the data in a Hive external table.

## Starting Hive

To start Hive use this byoh command:

```
$ bin/byoh hive
```

The output should look something like this:

```
/usr/lib/dse/resources/cassandra/conf

Logging initialized using configuration in jar:file:/usr/lib/hive12/lib/
hive-common-0.12.0.jar!/hive-logback.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-
logback12-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hive12/lib/slf4j-
logback12-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/dse/resources/dse/lib/slf4j-
logback12-1.7.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.LogbackLoggerFactory]
hive>
```

## Accessing data outside Cassandra

At the Hive prompt, you can create and [query the Hive managed table](#). For example, you can query a flat file that you put on the HDFS (using a hadoop -fs command) or the file can be elsewhere, such as on an operating system file system.

## Accessing data in Cassandra

Use the DataStax Enterprise custom metastore in the BYOH model to map Cassandra tables to Hive tables automatically. The keyspace and table must pre-exist in Cassandra. You create a schema representing your table using the `dse hive-schema` command. The command dumps your entire schema, or part of it, to standard output. Next, in the Hive client, you pass the table containing the map to Hive using the `byoh hive -f` command. DataStax Enterprise creates the Hive external table. Finally, create or alter CQL data from Hive.

The syntax of the `hive-schema` command is:

```
bin/dse hive-schema -keyspace testks -table testa testb -exclude testc testd
```

The `hive-schema` command options are:

**-all**

Include all keyspaces and tables

**-decimal**

Decimal parameters in form precision, scale for Hive 0.13 and later

**-exclude**

Exclude these tables

**-help**

Provide `hive-schema` command usage

**-keyspace**

Include these keyspaces

**-table**

Include these tables

To dump all Cassandra keyspaces and tables to a file called `byoh_automap`, for example, use this command:

```
$ dse hive-schema -all > byoh_automap
```

To start Hive and pass the `hive-schema`:

```
$ byoh hive -f byoh_automap
```

## Running the Hive demo

The Hive demo creates a keyspace and table in Cassandra using `cqlsh`, creates a Hive external table, and then queries the table from Hive.

**Note:** DataStax Demos do not work with either LDAP or internal authorization (username/password) enabled.

1. Create a Cassandra keyspace and table using `cqlsh`.

```
cqlsh> CREATE KEYSPACE cassandra_keyspace WITH replication =
 {'class': 'NetworkTopologyStrategy', 'Cassandra': 1};
cqlsh> use cassandra_keyspace;
cqlsh:cassandra_keyspace> CREATE TABLE exampletable (key int PRIMARY
 KEY , data text);
cqlsh:cassandra_keyspace> INSERT INTO exampletable (key, data) VALUES
 (1, 'This data can be read automatically in hive');
```

2. On the command line, use the dse hive-schema command to create an automap file:

```
$ bin/dse hive-schema -keyspace cassandra_keyspace -table exampletble
```

The output is:

```
CREATE DATABASE IF NOT EXISTS cassandra_keyspace;

USE cassandra_keyspace;

CREATE EXTERNAL TABLE IF NOT EXISTS exampletble (
 key int COMMENT 'Auto-created based on
 org.apache.cassandra.db.marshall.Int32Type from Column Family meta data',
 data string COMMENT 'Auto-created based on
 org.apache.cassandra.db.marshall.UTF8Type from Column Family meta data')
ROW FORMAT SERDE
 'org.apache.hadoop.hive.cassandra.cql3.serde.CqlColumnSerDe'
STORED BY
 'org.apache.hadoop.hive.cassandra.cql3.CqlStorageHandler'
WITH SERDEPROPERTIES (
 'serialization.format'='1',
 'cassandra.columns.mapping'='key,data')
TBLPROPERTIES (
 'auto_created' = 'true',
 'cassandra.partitioner' = 'org.apache.cassandra.dht.Murmur3Partitioner',
 'cql3.partition.key' = 'key',
 'cassandra.ks.name' = 'cassandra_keyspace',
 'cassandra.cf.name' = 'exampletble');
```

3. To start Hive and pass the hive-schema:

```
$ byoh hive -f byoh_automap
SLF4J: Found binding in [jar:file:/home/automaton/dse-4.6.0/resources/dse/
lib/slf4j-logback12-1.7.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.LogbackLoggerFactory]
OK
Time taken: 5.15 seconds
OK
Time taken: 0.008 seconds
OK
Time taken: 3.085 seconds
```

4. Start Hive using the byoh hive command to access the Cassandra table.

```
$ bin/byoh hive
```

5. In Hive, use the Cassandra keyspace and query the Cassandra table.

```
hive> use cassandra_keyspace;
OK
Time taken: 5.264 seconds

hive> select * from exampletble;
OK
1 This data can be read automatically in hive
Time taken: 3.815 seconds, Fetched: 1 row(s)
```

## Using Pig

The external Hadoop system includes an Apache Pig Client that you enable through BYOH. [Pig](#) is a high-level programming environment for MapReduce coding. Using Pig under BYOH is straight-forward. You start the Pig client through BYOH. On the grunt command line, access Pig using the same [data access commands](#), [CQL pushdown filter](#), and URL-encoded [prepared statements](#) as used by DataStax Enterprise integrated Hadoop. [Store Pig relations](#) to Cassandra in the same manner also.

Generally, Pig examples work as shown in the documentation of DataStax Enterprise integrated Hadoop. For example, to run the [Pig library demo](#), the only change to the steps is how you start Pig. To start Pig, use the byoh preface. On Linux, for example:

```
$ bin/byoh pig
grunt>
```

## Using Mahout with external Hadoop

Apache Mahout is a Hadoop component that offers machine learning libraries. You can use Apache Mahout with external Hadoop systems and DataStax Enterprise.

If Mahout is installed to its default location of `/usr/lib/mahout`, the `byoh-env.sh` file is already configured correctly. If Mahout is installed in a different location, open `byoh-env.sh` in a text editor and set `MAHOUT_HOME` to the correct location of Mahout.

```
export MAHOUT_HOME="/usr/local/lib/mahout"
```

### Related tasks

[Using Mahout](#) on page 168

### Running the demo with external Mahout

The DataStax Enterprise installation includes a Mahout demo. The demo determines with some percentage of certainty which entries in the input data remained statistically in control and which have not. The input data is time series historical data. Using the Mahout algorithms, the demo classifies the data into categories based on whether it exhibited relatively stable behavior over a period of time. The demo produces a file of classified results. This procedure describes how to run the Mahout demo.

### Procedure

**Note:** DataStax Demos do not work with either LDAP or internal authorization (username/password) enabled.

1. Go to the Hadoop home directory and make the test data directory.

```
$ cd Hadoop_home
$ bin/hadoop fs -mkdir testdata
```

2. Add the data from the demo directory to Mahout.

```
$ bin/hadoop fs -put DSE_home/demos/mahout/synthetic_control.data testdata
```

3. Go to the DSE home directory and run the demo's analysis job using byoh.

```
$ bin/byoh mahout org.apache.mahout.clustering.syntheticcontrol.canopy.Job
```

The job will take some time to complete. You can monitor the process of the job in OpsCenter if you have it installed.

4. When the job completes, output the classified data into a file in a temporary location.

```
$ bin/byoh mahout clusterdump --input output/clusters-0-final --pointsDir
output/clusteredPoints --output /tmp/clusteranalyze.txt
```

5. Open the /tmp/clusteranalyze.txt output data file and look at the results.

# DSE Search

DataStax Enterprise Search (DSE Search) simplifies using search applications for data that is stored in a Cassandra database. DSE Search is an enterprise grade search solution that is scalable to work across multiple datacenters and the cloud.

## About DSE Search

DSE Search (DataStax Enterprise Search) simplifies using search applications for data that is stored in a Cassandra database. DSE Search is an enterprise grade search solution that is scalable to work across multiple datacenters and the cloud.

The benefits of running enterprise search functions through DataStax Enterprise and DSE Search include:

- A fully fault-tolerant, no-single-point-of-failure search architecture across multiple datacenters.
- [Add search capacity](#) just like you add capacity in Cassandra.
- Ability to isolate transactional, analytic, and search workloads to prevent competition for resources.
- [Live indexing](#) increases indexing throughput, reduces Lucene reader latency, and enables queries to be made against recently indexed data. Live indexing enables queries to be made against recently indexed data. Live indexing, also known as RT (real time) indexing, improves index throughput and reduces Lucene reader latency while supporting all Solr functionality. Enable live indexing on only one Solr core per cluster.
- Near real-time query capabilities.
- [Commands](#) for creating, reloading, and managing Solr core resources.
- Read/write to any DSE Search node and automatically index stored data.
- Selective [updates](#) of one or more fields and restricted query routing.
- Examine and aggregate real-time data in multiple ways using CQL or the Solr compatible HTTP API.
- Fault-tolerant queries, efficient deep paging, and advanced search node resiliency.
- Support of [virtual nodes \(vnodes\)](#).
- [Manage](#) where the Solr data files are saved on the server.

## Indexing

DSE Search allows Cassandra columns to be automatically indexed by Solr through its secondary index API. Each insert or update of a Cassandra row triggers a new indexing on DSE Search, inserting or updating the document that corresponds to that Cassandra row. Using CQL, DSE Search supports partial document updates that enable you to modify existing information while maintaining a lower transaction cost.

Indexing DSE Search documents requires the `schema.xml` and `solrconfig.xml` resources. DSE can automatically generate these resources, or you can [use custom](#) resources.

## Solr resources

DSE Search supports all Solr tools and APIs. See these resources for more information on using Open Source Solr.

- [Apache Solr documentation](#)

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

## Starting and stopping DSE Search

To install a DSE Search node, use the same [installation procedure](#) as you use to install any other type of node. To use real-time (Cassandra), analytics (Hadoop/Spark), or DSE Search nodes in the same cluster, [segregate the different nodes](#) into separate datacenters. Using the default DSESimpleSnitch automatically puts all the DSE Search nodes in the same datacenter, so you need to change the snitch from the default to another type for multiple datacenter deployment.

### Starting and stopping a DSE Search node

The way you start a DSE Search node depends on the type of installation:

- **Installer-No Services and Tarball installations:**

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

```
$ bin/dse cassandra -s
```

The node starts up.

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

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

- **Installer-Services and Package installations:**

1. Enable DSE Search mode by setting this option in the `/etc/default/dse` file:

```
SOLR_ENABLED=1
```

2. Start the dse service using this command:

```
$ sudo service dse start
```

The DSE Search node starts.

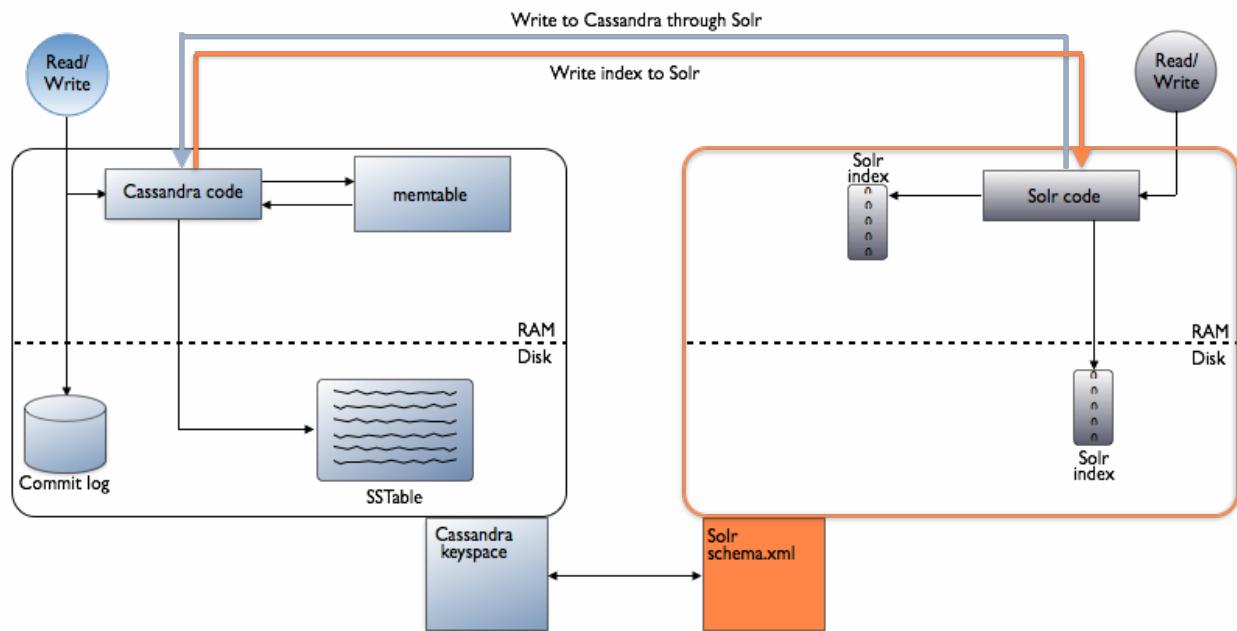
You stop a node using this command:

```
$ sudo service dse stop
```

## DSE Search architecture

In a distributed environment, such as DataStax Enterprise and Cassandra, the data is spread over multiple nodes. In a [mixed-workload cluster](#), DSE Search nodes are in a separate datacenter. Deploy DSE Search nodes in a single datacenter to run DSE Search on all nodes.

A Solr API client writes data to Cassandra first, and then Cassandra updates indexes.



When you update a table using CQL, the Solr document is updated. Re-indexing occurs automatically after an update. Writes are durable. All writes to a replica node are recorded in memory and in a commit log before they are acknowledged as a success. If a crash or server failure occurs before the memory tables are flushed to disk, the commit log is replayed on restart to recover any lost writes.

**Note:** DSE Search does not support JBOD mode.

## DSE Search terms

In DSE Search, there are several names for an index of documents and configuration on a single node:

- A Solr core
- A collection
- One shard of a collection

Each document in a Solr core/collection is considered unique and contains a set of fields that adhere to a user-defined [schema](#). The schema lists the field types and how they should be indexed. DSE Search maps Solr cores/collections to Cassandra tables. Each table has a separate Solr core/collection on a particular node. Solr documents are mapped to Cassandra rows, and document fields to columns. The shard is analogous to a partition of the table. The Cassandra keyspace is a prefix for the name of the Solr core/collection and has no counterpart in Solr.

This table shows the relationship between Cassandra and Solr concepts:

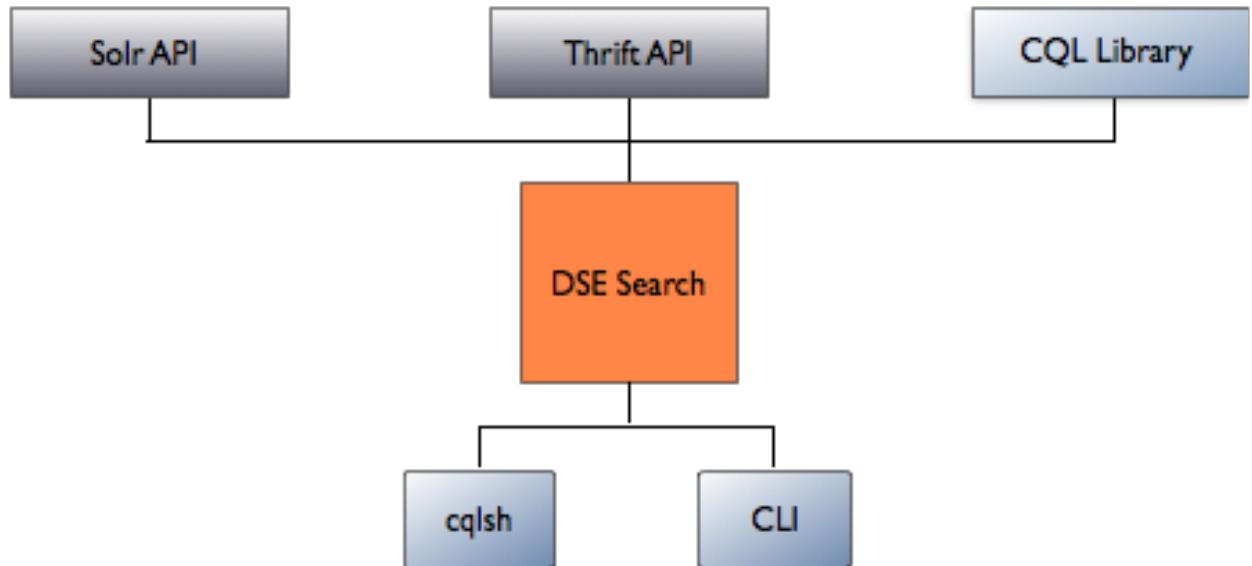
Cassandra	Solr single node environment
Table	Solr core or collection
Row	Document
Primary key	Unique key
Column	Field
Node	N/A
Partition	N/A
Keyspace	N/A

With Cassandra replication, a Cassandra node or Solr core contains more than one partition (shard) of table (collection) data. Unless the replication factor equals the number of cluster nodes, the Cassandra node or Solr core contains only a portion of the data of the table or collection.

**Note:** Do not mix Solr indexes with Cassandra secondary indexes. Attempting to use both indexes on the same table is not supported.

## Queries

DSE Search hooks into the Cassandra Command Line Interface (CLI), Cassandra Query Language (CQL) library, the cqlsh tool, existing [Solr APIs](#), and Thrift



APIs.

Avoid querying nodes that are indexing. For responding to queries, DSE Search ranks the nodes that are not performing Solr indexing higher than indexing ones. If only indexing nodes can satisfy the query, the query will not fail but instead will return potentially partial results.

## Using CQL Solr queries in DSE Search

DataStax Enterprise supports production-grade implementation of CQL Solr queries in DSE Search. You can develop CQL-centric applications supporting full-text search without having to work with Solr-specific APIs. Only full text search queries are supported in this release. Using CQL, DSE Search supports partial document updates that enable you to modify existing information while maintaining a lower transaction cost. Before using CQL Solr queries in DSE Search, [configure solrconfig.xml](#) to handle CQL queries.

### Required configuration for CQL Solr queries in DSE Search

Using CQL solr\_query syntax is supported only on nodes where search is enabled.

When you automatically generate resources, the `solrconfig.xml` file already contains the request handler for running CQL Solr queries in DSE Search. If you do not automatically generate resources and want to run CQL Solr queries using custom resources, you can verify or manually add the `CqlSearchHandler` handler to the `solrconfig.xml` file:

```

<requestHandler
 class="com.datastax.bdp.search.solr.handler.component.CqlSearchHandler"
 name="solr_query" />

```

If the requestHandler is not already set, the CQLSearchHandler is automatically inserted.

## CQL Solr considerations

- CQL Solr queries are defaulted to an equivalent LIMIT 10.
- The row retrieval phase of CQL Solr queries uses the LOCAL\_ONE consistency level for reads. In contrast, HTTP Solr queries use local/internal reads.
- Pagination is not on by default. When using a driver with a CQL Solr query, you can specify to use pagination (also called cursors) only when the driver uses pagination. Set the `cql_solr_query_paging` option in the `dse.yaml` file is set to driver. To turn off cursors with CQL Solr queries, set the `cql_solr_query_paging` option to off in the `dse.yaml` file. You can use the `paging:driver` parameter in CQL and JSON queries to [dynamically enable pagination](#).
- [Solr restrictions apply to pagination](#).
- Queries with smaller result sets will see increased performance with paging off.

**Note:** Limitations and known Apache Solr issues apply to DSE Search queries. For example, incorrect `SORT` results for tokenized text fields.

## CQL Solr query syntax

You can run CQL Solr queries using the SELECT statement that includes the search expression.

### Synopsis

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

There are two types of search expressions:

- Search queries with CQL
- [Search queries with JSON](#)

## Search queries with CQL

The Solr query expression uses the syntax supported by the `Solr q parameter`. For example:

```
SELECT * FROM keyspace.table WHERE solr_query='name: cat name: dog -
name:fish'
```

When you name specific columns, DSE Search retrieves only the specified columns and returns the columns as part of the resulting rows. DSE Search supports projections (SELECT a, b, c...) only, not functions, for the select expression. The following example retrieves only the name column:

```
SELECT name FROM keyspace.table WHERE solr_query='name:cat name:dog -
name:fish'
```

Use the LIMIT clause to specify how many rows to return. The following example retrieves only 1 row:

```
SELECT * FROM keyspace.table WHERE solr_query='name:cat name:dog -name:fish'
 LIMIT 1
```

Use the count() function in CQL Solr queries to return the number of rows that satisfy the Solr query:

```
SELECT count(*) FROM table WHERE solr_query = '...';
```

You cannot use CQL Solr queries to set the consistency level, ordering, or specify WHERE clauses other than the solr\_query one. The consistency level for CQL Solr queries is ONE by default and should not be changed; otherwise, the query returns an error.

Using count() in combination with LIMIT or facets results in an error.

## Queries for tuples and UDTs

DSE Search supports indexing and querying of advanced data types, including [tuples](#) and [user-defined types](#) (UDT).

- The tuple data type holds fixed-length sets of typed positional fields. Use a tuple as an alternative to a user-defined type.
- A user-defined type (UDT) facilitates handling multiple fields of related information in a table. UDTs are a specialization of tuples. All examples and documentation references to tuples apply to both tuples and UDTs.

Applications that require multiple tables can be simplified to use fewer tables by using a user-defined type to represent the related fields of information instead of storing the information in a separate table.

[Configuration and schema](#) requirements apply. See [UDT query examples](#) on page 216.

## Using CQL partition key restrictions with Solr queries

Solr CQL queries support restriction to a single partition key. Partition key restrictions work only when `_partitionKey` is explicitly indexed or the schema explicitly includes all of the components of the Cassandra partition key. In your schema, you can [override `\_partitionKey`](#) when not using joins.

Example:

```
SELECT id, date, value FROM keyspace.table WHERE id = 'series1' AND
solr_query='value:bar*'"
```

CQL partition key restrictions work only with fully specified partition keys. For example, with this table:

```
CREATE TABLE vtbl (k1 text, k2 text, valuetext, PRIMARYKEY ((k1, k2)))
```

Avoid using a query like this:

```
SELECT * FROM vtbl WHERE k1 = '50'AND solr_query='value:*''
```

Use a filter query against the partially specified composite partition key:

```
SELECT * FROM valuetable WHERE solr_query='{"q": "value:*", "fq": "k1:50"}'
```

## Using the Solr token function

Solr CQL queries support limited use of the CQL [token](#) function. The token function enables targeted search that restricts the nodes queried to reduce latency.

**Note:** Using the Solr token function is for advanced users only and is supported only in specific use cases.

Example:

```
SELECT id, value FROM keyspace.table WHERE token(id) >= -3074457345618258601
AND token(id) <= 3074457345618258603 AND solr_query='id:*''
```

Example with an open range:

```
SELECT id, value FROM keyspace.table WHERE token(id) >= 3074457345618258604
AND solr_query='id:*''
```

Constraints apply to using the token function with Solr CQL queries:

- `token()` cannot be used with `route.range` or `route.partition`

- Wrapping token() ranges are not supported
- A specified token() range must be owned by a single node; ranges cannot span multiple nodes
- Because DSE uses the Solr **single-pass** queries, only the fields that are declared in the Solr schema are returned in the query results. If you have columns that do not need to be indexed, but still need to be returned by using a token-restricted query, you can declare the columns as stored non-indexed fields in your `schema.xml` file.

## Secondary indexes (2i) in queries

Use `solr_query` to query secondary indexes (2i) that are created by Solr. Use:

```
SELECT * from users WHERE solr_query = '{"q":"irc:jdoe"}';
```

The secondary indexes created by Solr cannot be used as a Cassandra 2i index in a cqlsh query. For example, this syntax fails: `SELECT * FROM users WHERE irc = 'jdoe'.`

## Search queries with JSON

DataStax Enterprise supports JSON-based query expressions.

### JSON query syntax

The JSON query expression syntax is a JSON string. The JSON-based query expression supports **local parameters** in addition to the following parameters:

```
{
 "fq": filter_query_expression(s) (string_or_array_of_strings),
 "sort": sort_expression (string),
 "start": start_index(number),
 "tz": zoneID, // Any valid zone ID in java TimeZone class
 "facet": facet_query_expression (object),
 "commit": true/false (boolean), "q": query_expression (string),
 "query.name": query_name (string)
 "paging": "driver" (string),
 "distrib.singlePass": true/false (boolean),
 "shards.failover": true/false (boolean), // Default: true
 "shards.tolerant": true/false (boolean), // Default: false
 "route.partition": partition_routing_expression (array_of_strings),
 "route.range": range_routing_expression (array_of_strings),
}
```

For example:

```
SELECT id FROM nhanes_ks.nhanes WHERE solr_query=' {"q":"ethnicity:Asian"} ';
```

```
SELECT id FROM nhanes_ks.nhanes WHERE solr_query=' {"q":"ethnicity:Mexi*", "sort":"id asc"} ' LIMIT 3;
```

```
SELECT * FROM mykeyspace.mysolr WHERE solr_query=' {"q" : "(!edismax}quotes:yearning or kills")' ;
```

**Note:** To use Solr Extended DisMax Query Parser (eDisMax) with `solr_query`, you must include `defaultSearchField` in your schema.

## Making distributed queries tolerant of shard failures

Since distributed queries contact many shards, making queries more tolerant of shard failures ensures more successful completions. Use `shards.failover` and `shards.tolerant` parameters to define query failover and tolerance of shard failures during JSON queries:

Valid configurations	Description
<code>"shards.failover": true,</code> <code>"shards.tolerant": false,</code>	This default configuration enables query failover and disables fault tolerance. Attempt to retry the failed shard requests when errors indicate that there is a reasonable chance of recovery. If any of the nodes (shards) that we scatter to fail before the query is complete, retry the shard query against a replica.
<code>"shards.failover": false,</code> <code>"shards.tolerant": true,</code>	Disable query failover. Enable fault tolerance. Make the query succeed, even if the query only partially succeeded, and did not succeed for all nodes.
<code>"shards.failover": false,</code> <code>"shards.tolerant": false,</code>	Disable query failover. Disable fault tolerance.

Failover and tolerance of partial results cannot coexist in the same query. Queries support enabling tolerance for only one parameter.

Other fault tolerance configuration options include: `netty_client_request_timeout` in `dse.yaml` and `read_request_timeout_in_ms` in `cassandra.yaml`.

## JSON queries with literal characters that are Lucene/Solr special characters

Lucene supports escaping [special characters](#) that are part of the query syntax. Special characters are: +, -, &&, ||, !, (, ), ", ~, \*, ?, and :. Using JSON with `solr_query` requires additional syntax for literal characters that are Lucene special characters.

Syntax for a simple search string:

Simple search string	mytestuser1?
Solr query	name:mytestuser1\?
CQL Solr Query	<code>solr_query='{"q": "name:mytestuser1\\\\?"}'</code>

Syntax for a complex search string:

Complex search string	(1+1):2
Solr query	e:(1+1)\:2
CQL Solr Query	<code>solr_query='{"q": "e:\\(1\\+1\\\\\\)\\:2"}'</code>

## Escape characters in queries

Solr queries require escaping [special characters](#) that are part of the query syntax. To escape these characters, use a slash (\) before the character to escape. For example, to search for a literal double quotation mark ("") character, escape the " for Solr with \".

For queries that contain double quotation marks, use triple slashes \\\:

- For query syntax: One slash \ to escape the "
- For the JSON string syntax: Two slashes \\ to escape the \

Triple slashes \\\ escape both characters in \" to produce \\ (an escaped escape) and \\\" (an escaped double quote).

Query type	Example
Exact phrase query	<p>For a row that looks like this, with an email address that includes a double quotation mark greenr"q@example.com:</p> <pre>INSERT INTO users(id, email) VALUES(1,     'greenr"q@example.com')"</pre> <p>Perform a phrase query to search for the email address that is enclosed in double quotation marks:</p> <pre>SELECT * FROM users where solr_query = ' { "q": "*:*", "fq": "email:\\"greenr\\\\\"q@example.com \""}' ';</pre>
Fuzzy query	<p>For a row that looks like this, with the same email address that includes a double quotation mark greenr"q@example.com:</p> <pre>cqlsh&gt; select * from test.users where     solr_query='{"q":"email:r\\\\\"q@example"}' ;       id        email             solr_query -----+-----+       1        greenr"q@example.com   null (1 rows)</pre> <p>For a term query (fuzzy search) for all email addresses that include r"q@example, remove the double quotation marks but retain triple quotation marks for the escaped double quotation character that is part of the email address:</p> <pre>SELECT * FROM users where solr_query = ' { "q": "*:*", "fq": "email:r\\\\\"q@example"}' ';</pre>

## Overriding the default TimeZone (UTC) in search queries

Specify the TZ parameter to overwrite the default TimeZone (UTC) that is used for adding and rounding in date math. The local rules for the specified time zone, including the start and end of daylight saving time (DST) if any, determine when each arbitrary day starts. The time zone rules impact the rounding and adding of DAYs, but also cascades to rounding of HOUR, MIN, MONTH, and YEAR. For example, specifying a different time zone changes the result:

Date math	Result
2016-03-10T12:34:56Z/YEAR	Default TZ 2016-01-01T00:00:00Z
	TZ=America/Los_Angeles 2016-01-01T08:00:00Z
2016-03-10T08:00:00Z+1DAY	Default TZ

Date math	Result
	2016-03-11T08:00:00Z
	TZ=America/Los_Angeles
	2016-03-11T07:00:00Z

The value of the TZ parameter can be any zone ID that is supported by the [java TimeZone class](#).

## Field, query, and range facetting with a JSON query

Specify the facet parameters inside a facet JSON object to perform field, query, and range facetting inside Solr queries. Distributed pivot facetting is supported. The query syntax is less verbose to specify facets by:

- Specifying each facet parameter without the facet prefix that is required by HTTP APIs.
- Expressing multiple facet fields and queries inside a JSON array.

### Faceted search example

```
SELECT * FROM solr WHERE solr_query='{"q":"id:*", "facet":{"field":"type"}}';
```

### Query facet example

```
SELECT * FROM solr WHERE solr_query='{"q":"id:*", "facet":{"query":["type:0"]}}';
```

### Multiple queries example

```
SELECT * FROM solr WHERE solr_query='{"q":"id:*", "facet":{"query":["type:0", "type:1"]}}';
```

### Distributed pivot facetting example

```
SELECT id FROM table WHERE solr_query='{"q":"id:*", "facet":{"pivot":{"type,value"}, "limit":"-1"}}'
```

### Range facet example

```
SELECT * FROM solr WHERE solr_query='{"q":"id:*", "facet":{"range":{"type", "f.type.range.start": -10, "f.type.range.end": 10, "range.gap": 1}}}';
```

The returned result is formatted as a single row with each column corresponding to the output of a facet (either field, query, or range). The value is represented as a JSON blob because facet results can be complex and nested. For example:

facet_fields		facet_queries
-----+-----		
{"type": {"0": 2, "1": 1}}		{"type:0": 2, "type:1": 1}

### Range by date facet example

```
SELECT * FROM solr WHERE solr_query='{"q":"business_date:*", "facet": {"range": "business_date", "f.business_date.range.gap": "+1MONTH"} }';
```

**Warning:** Solr range facets before, after, and between might return incorrect and inconsistent results on multinode clusters. See [SOLR-6187](#) and [SOLR-6375](#).

### Interval facet example

```
SELECT * FROM solr WHERE solr_query='{"q":"id:*", "facet": {"interval": "id", "interval.set": "[*,500]"}';
```

## JSON single-pass distributed query

Single-pass distributed queries are supported in CQL Solr queries.

To use a single pass distributed query instead of the standard two-pass query, specify the `distrib.singlePass` Boolean parameter in the JSON query expression:

```
SELECT * FROM ks.cf WHERE solr_query = '{"q" : "*:*", "distrib.singlePass" : true}'
```

Using a single-pass distributed query has an operational cost that includes potentially more disk and network overhead. With single-pass queries, each node reads all rows that satisfy the query and returns them to the coordinator node. An advanced feature, a single-pass distributed query saves one network round trip transfer during the retrieval of queried rows. A regular distributed query performs two network round trips, the first one to retrieve IDs from Solr that satisfy the query and another trip to retrieve only the rows that satisfy the query from Cassandra, based on IDs from the first step. Single-pass distributed queries are most efficient when most of the documents found are returned in the search results, and they are not efficient when most of the documents found will not be returned to the coordinator node.

For example, a distributed query that only fans out to a single node from the coordinator node will likely be most efficient as a single-pass query.

Single pass distributed queries for CQL are supported when the additional `distrib.singlePass` boolean parameter is included in the JSON query.

With single-pass queries, there is a limitation that only document fields that are defined in the Solr schema are returned as query results. This limitation also applies to map entries that do not conform to the [dynamic field mapping](#).

## JSON query name option

Using the following syntax to name your queries to support metrics and monitoring for performance objects. Naming queries can be useful for tagging and JMX operations, for example.

```
SELECT id FROM nhanes_ks.nhanes WHERE solr_query=' {"query.name":"Asian subjects", "q":"ethnicity:Asia*"}' LIMIT 50;
```

## JSON query commit option

If you are executing custom queries after bulk document loading, and the normal [auto soft commit](#) is disabled or extremely infrequent, and you want the latest data to be visible to your query, use the JSON query commit option to ensure that all pending updates are soft-committed before the query runs. By default, the commit option is set to false.

For example:

```
SELECT id FROM nhanes_ks.nhanes WHERE solr_query=' {"q":"ethnicity:Asia*", "commit":true}' LIMIT 50;
```

**Warning:** Do not use the JSON commit option for live operations against a production cluster. DataStax recommends using the JSON commit option only when you would otherwise be forced to issue a commit

though the Solr HTTP interface. The commit option is not a replacement for the normal auto soft commit process.

## Queries for tuples and UDTs

DSE Search supports indexing and querying of advanced data types, including [tuples](#) and [user-defined types](#) (UDT).

- The tuple data type holds fixed-length sets of typed positional fields. Use a tuple as an alternative to a user-defined type.
- A user-defined type (UDT) facilitates handling multiple fields of related information in a table. UDTs are a specialization of tuples. All examples and documentation references to tuples apply to both tuples and UDTs.

Applications that require multiple tables can be simplified to use fewer tables by using a user-defined type to represent the related fields of information instead of storing the information in a separate table.

[Configuration and schema](#) requirements apply. See [UDT query examples](#) on page 216.

## Queries to dynamically enable paging

When the `cql_solr_query_paging` option is off in the `dse.yaml` file, use the `paging:driver` parameter to dynamically enable pagination.

```
cqlsh> select id from wiki.solr where solr_query='{"q":"*", "sort":"id asc", "paging":"driver"}';
```

## Using the Solr HTTP API

You can use the Solr HTTP API to query data that is indexed in DSE Search just as you would search for data indexed in Solr.

**Note:** Limitations and known Apache Solr issues apply to DSE Search queries. For example, incorrect [SORT](#) results for tokenized text fields.

HTTP Solr queries use local/internal reads. In contrast, the row retrieval phase of CQL Solr queries uses the LOCAL\_ONE consistency level for reads.

For use with the HTTP API only, you can define the default number of rows in the `solrconfig.xml` file:

```
<requestHandler
 class="com.datastax.bdp.search.solr.handler.component.CqlSearchHandler"
 name="solr_query">
 <lst name="defaults">
 <int name="rows">10</int>
 </lst>
</requestHandler>
```

### Solr HTTP API example

Assuming you performed [the example of using a collection set](#), to find the titles in the `mykeyspace.mysolr` table that begin with the letters Succ in XML, use this URL:

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

The response is:

```
<response>
<lst name="responseHeader">
 <int name="status">0</int>
```

```

<int name="QTime">2</int>
<lst name="params">
 <str name="f1">title</str>
 <str name="q">title:Success*</str>
</lst>
</lst>
<result name="response" numFound="2" start="0">
 <doc>
 <str name="title">Success</str>
 </doc>
 <doc>
 <str name="title">Success</str>
 </doc>
</result>
</response>

```

## Using Solr pagination (cursors)

DataStax Enterprise integrates native driver paging with Solr cursor-based paging. Pagination, also called cursors, supports using a cursor to scan results. [Solr pagination restrictions](#) apply. You can use CQL Solr queries and the Solr HTTP API.

**Note:** When using CQL Solr queries with Cassandra pagination enabled, you might experience a performance slowdown because Solr is not able to use its query result cache when pagination is configured. If you do not want to paginate through large result sets, disable pagination when running CQL Solr queries. See the [driver](#) documentation.

### Using cursors with CQL Solr queries

When using a driver with a CQL Solr query, you can specify to use pagination (also called cursors) only when the driver uses pagination. Set the [cql\\_solr\\_query\\_paging](#) option in the `dse.yaml` file is set to `driver`. To turn off cursors with CQL Solr queries, set the [cql\\_solr\\_query\\_paging](#) option to `off` in the `dse.yaml` file. You can use the `paging:driver` parameter in CQL and JSON queries to [dynamically enable pagination](#).

See the driver documentation for details. It is not mandatory to use a sort clause. However, if a sort clause is not provided, sorting is undefined.

#### Examples

```
SELECT * from ks.cf where solr_query='{"q":"*:*", "sort":"id asc, id2 asc"}'
```

```
SELECT * from ks.cf where solr_query='{"q":"*:*"}'
```

### Using cursors with the HTTP API

To use cursors with the Solr HTTP API, it is not mandatory to provide a sort clause. However, if a sort clause is not provided, sorting is undefined. Do not make assumptions on sorting. Follow the steps in [Using CQL Solr queries](#).

## Inserting/updating data using the Solr HTTP API

Updates to a CQL-based [Solr core](#) replace the entire row. You cannot replace only a field in a CQL table. The deprecated `replacefields` parameter for inserting into, modifying, or deleting data from CQL Solr cores is not supported. The `replacefields` parameter is supported for updating indexed data in a non-CQL table and in Solr. Use the parameter in this way:

```
$ curl http://host:port/solr/keyspace.table/update?
replacefields=false -H 'Content-type: application/json' -d
'json string'
```

To update a CQL-based core, use the following procedure:

## Procedure

Building on the [collections example](#), insert data into the mykeyspace.mytable data and Solr index. Use this curl command:

```
$ curl http://localhost:8983/solr/mykeyspace.mysolr/update -H 'Content-
type: application/json' -d '[{"id":"130", "quotes":"Life is a beach.",
"name":"unknown", "title":"Life"}]'
```

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

When you use CQL or CLI to update a field, DSE Search implicitly sets replacefields to false and updates individual fields in the Solr document. The re-indexing of data occurs automatically.

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

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

The Lucene merge policy is very efficient. Using the optimize command is no longer necessary and using the optimize command in a URL can cause nodes to fail.

## Querying a CQL collection set

DataStax Enterprise supports CQL collections. In this example, you create a table containing a [CQL collection set](#) of famous quotations. You insert data into the table by copying/pasting INSERT commands from a file that you download.

Next, you insert a collection into Cassandra, index the data in DSE Search, and finally, query the search index.

## Procedure

1. [Start DataStax Enterprise in DSE Search mode](#).
2. [Start cqlsh](#).
3. Create a keyspace and a table consisting of a set collection column and other columns, and then, insert some data for DSE Search to index.

```
CREATE KEYSPACE mykeyspace
 WITH REPLICATION = {'class':'NetworkTopologyStrategy', 'Solr':1};

USE mykeyspace;

CREATE TABLE mysolr (
 id text PRIMARY KEY,
 name text,
 title text,
 quotes set text
```

```
) ;
```

4. Download the **INSERT commands** in the `quotations.zip` file. Unzip the `quotations.zip` file that you downloaded, copy the insert commands, and paste the commands on the `cqlsh` command line.
5. Run the following command, which is located in the `bin` directory of tarball installations. For example, from a tarball installation:

```
$ install_location/bin/dsetool create_core mykeyspace.mysolr
generateResources=true reindex=true
```

If you are recreating the `mykeyspace.mysolr` core, use the `reload_core` instead of the `create_core` command.

There is no output from this command. You can search Solr data after indexing finishes.

6. In `cqlsh`, search Solr-indexed data to find titles like `Succ*`.

```
SELECT * FROM mykeyspace.mysolr WHERE solr_query='title:Succ*';
```

Because you created the core using automatically generated resources, the `solrconfig` defines the **request handler** for using CQL for Solr queries.

7. Using a browser, search Solr-indexed data using the Solr HTTP API to find titles like `Succ*`.

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

```
{
 "response": { "numFound":2, "start":0, "docs": [
 {
 "id": "126",
 "title": "Success",
 "quotes": ["If A is success in life, then A equals x plus y plus z. Work is x; y is play; and z is keeping your mouth shut."],
 "name": "Albert Einstein"
 },
 {
 "id": "125",
 "title": "Success",
 "quotes": ["Always bear in mind that your own resolution to succeed is more important than any one thing.", "Better to remain silent and be thought a fool than to speak out and remove all doubt."],
 "name": "Abraham Lincoln"
 }
]}
```

## Spatial queries

Performing spatial queries that include polygon shapes requires the JTS Topology Suite JAR. Dynamic fields for spatial subfields use **prefix naming conventions** to enable using Cassandra map types to store geospatial data. DSE Search includes the Solr Spatial4j library that adds advanced spatial types like polygons to search indexes.

### Solr spatial field types

Solr spatial field types (`solr.PointType` and `solr.LatLonType`) rely on **dynamic fields** to store latitude and longitude data. You must declare one field of type `solr.TrieDoubleField` in the schema for each spatial field type. The prefix subfield naming policy uses Cassandra map types to store geospatial data.

## Advanced spatial queries

Performing spatial queries that include polygon shapes requires that you install the JTS (Java Topology Suite) library into the DataStax Enterprise Solr library directory. Download version 1.13 of the `jts.jar` file from <http://central.maven.org/maven2/com/vividsolutions/jts/1.13/> and install in the Solr library path:

The default Solr library path location depends on the type of installation:

Installer-Services	<code>/usr/share/dse/resources/solr/lib</code>
Package installations	<code>/usr/share/dse/solr/lib</code>
Installer-No Services and Tarball installations	<code>install_location/resources/solr/lib</code>

## Spatial predicates

DataStax Enterprise supports these spatial predicates:

- Intersects
- IsWithin
- IsDisjointTo
- Contains

## Examples

### Intersects

```
fq=geo:"Intersects(-74.093 41.042 -69.347 44.558)"
```

### IsWithin

```
fq=geo:"IsWithin(POLYGON((-10 30, -40 40, -10 -20, 40 20, 0 0, -10 30)) distErrPct=0")
```

### IsDisjointTo

```
fq=geo:"IsDisjointTo(POLYGON((-10 30, -40 40, -10 -20, 40 20, 0 0, -10 30)) distErrPct=0")
```

### Contains

```
fq=geo:"Contains(POLYGON((-10 30, -40 40, -10 -20, 40 20, 0 0, -10 30)) distErrPct=0")
```

## Using dynamic fields

Using dynamic fields, you can index content in fields that are not explicitly defined by the schema. Dynamic fields allow you to process multiple Solr fields the same way. A common use case for dynamic fields is to identify fields that should not be indexed or to implement a schema-less index.

In CQL-based [Solr cores](#), the Solr schema fields that are dynamic and multivalued are not supported.

## Spatial subfields prefix naming conventions

Dynamic fields for spatial subfields use prefix naming conventions to enable using Cassandra map types to store geospatial data:

```
<types>
```

```

<fieldType class="solr.LatLonType" multiValued="false" name="LatLonType"
subFieldPrefix="llt_"/>
<fieldType name="tdouble" class="solr.TrieDoubleField" precisionStep="8"
positionIncrementGap="0"/>
</types>
<fields>
 <dynamicField indexed="true" name="latmap*" stored="true"
type="LatLonType"/>
 <dynamicField name="llt_*" type="tdouble" indexed="true"
stored="true"/>
</fields>

```

## Best practices

- Avoid or limit the use of dynamic fields.  
Lucene allocates memory for each unique field (column) name, so if you have a row with columns A, B, C, and another row with B, D, E, Lucene allocates 5 chunks of memory. For millions of rows, the heap is unwieldy.
- Instead of using dynamic fields, use [Copy fields](#) instead and then perform queries against the combined field.
- Use the [FieldInputTransformer \(FIT\)](#) API as an option.

## To use a dynamic field

- Include a Solr dynamic field in `schema.xml`.  
Name the field using a wildcard at the beginning or end of the field. For example, an asterisk prefix or suffix in the field name in the schema designates a dynamic field.
  - `dyna_*`
  - `*_s`
- In CQL, to define the [map collection](#) column, use the same base name (no asterisk) that you used for the field in `schema.xml`.  
For example, use `dyna_*` in `schema.xml` and `dyna_` for the name of the CQL map collection.
- Use type `text` for the map key. For example:

```

CREATE TABLE my_dynamic_table (
 . . .
 dyna_ map<text, int>,
 . . .
);

```

- Using CQL, insert data into the map using the base name as a prefix or suffix in the first component of each map pair. The format of the map using a prefix is:

```
{ prefix_literal : literal, prefix_literal : literal, . . . }
```

For example, the CQL map looks like this:

```
'dyn_' : {dyn_1 : 1, dyn_2 : 2, dyn_3 : 3}
```

DSE Search maps the Solr dynamic field to a Cassandra map collection column, as shown in the [advanced tutorial](#).

## Deleting by id

Delete by id removes the document with a specified id and is more efficient than delete by query. The id is the value of the `uniqueKey` field declared in the schema. The id can be a synthetic id that represents a

Cassandra compound primary key, such as the one used in the [Basic tutorial](#). To delete by id, the following example builds on the example in [running a simple search](#). After clicking Execute Query, a list of results appears. Each result includes a \_uniqueKey in JSON format. The uniqueKey is the first line of each result and looks like this:

```
<str name="_uniqueKey">["47336", "29"]</str>
```

In this example, ["47336", "29"] are the values of the id, age compound primary key. The following delete by id query shows the HTTP API command you use to remove that particular record from the Solr index:

```
$ curl http://localhost:8983/solr/nhanes_ks.nhanes/update --data
'<delete><id>["47336", "29"]</id></delete>' -H 'Content-type:text/xml;
charset=utf-8'
```

After deleting the record, [run a simple search](#) on the Solr tutorial data again. The Solr Admin shows that the number of documents has been reduced by one. Query the Cassandra table using cqlsh:

```
cqlsh:nhanes_ks> SELECT * FROM nhanes WHERE id=47336;
```

The cqlsh output also confirms that the data was removed. Null values appear instead of the data.

## Deleting by query

After you issue a delete by query, documents start getting deleted immediately and deletions continue until all documents are removed. For example you can delete the data that you inserted using this command on the operating system command line:

```
$ curl http://localhost:8983/solr/mykeyspace.mysolr/update --data
'<delete><query>*:*</query></delete>' -H
'Content-type:text/xml; charset=utf-8'
```

Using &allowPartialDeletes parameter set to false (default) prevents deletes if a node is down. Using &allowPartialDeletes set to true causes the delete to fail if a node is down and the delete does not meet a consistency level of quorum. Delete by queries using \*:\* are an exception to these rules. These queries issue a truncate, which requires all nodes to be up in order to succeed.

## Joining cores

DataStax Enterprise supports the [OS Solr query time join](#) through a custom implementation. You can join Solr documents, including those having different Solr cores under these conditions:

- Solr cores need to have the same keyspace and same Cassandra partition key.
- Both Cassandra tables that support the Solr cores to be joined have to be either Thrift- or CQL-compatible. You cannot have one that is Thrift-compatible and one that is CQL-compatible.
- The type of the unique key (Cassandra key validator of the partition key) are the same.
- The order of table partition keys and schema unique keys are the same.

DataStax Enterprise 4.5.0 and later provides faster DocValues-based joins than earlier versions of DataStax Enterprise, such as 4.0.2. In the earlier version, using the simplified syntax shown in the next section for a join query requires re-indexing the CQL Solr core, but not the Thrift Solr core. In DataStax Enterprise, using the simplified syntax automatically takes advantage of faster joins in the case of a CQL Solr core. In the case of a Thrift Solr core, to use the simplified syntax, re-index, and in the from field of the query, use docValues=true.

## Simplified syntax

This simplified syntax is recommended for joining Solr cores:

```
q={!join fromIndex=test.from}field:value
```

The custom implementation eliminates the need to use to/from parameters required by OS Solr. Based on the key structure, DataStax Enterprise can determine what the parameters are. For backward compatibility with applications, the verbose, [legacy syntax](#) is also supported.

## Example of using a query time join

This example creates two tables, songs and lyrics. The tables use the same partition key. The songs table uses a simple primary key, the UUID of a song. The primary key of the songs table is its partition key. The lyrics table uses a compound primary: id and song, both of type UUID. After joining cores, you construct a single query to retrieve information about songs having lyrics that include "love".

You can copy CQL commands, Solr HTTP requests, and the query from the downloaded commands.txt file.

1. [Download and unzip the file](#) containing the Solr schemas, Solr configuration files, and commands for this example.

This action creates /songs and /lyrics directories, schemas, and Solr configuration files for indexing data in the songs and lyrics tables.

2. Start cqlsh, and then create and use a keyspace named internet.

You can copy/paste from the downloaded commands.txt file.

3. Create two tables, song and lyrics, that share the internet keyspace and use the same partition key.

```
cqlsh> CREATE TABLE songs (song uuid PRIMARY KEY, title text, artist text);
cqlsh> CREATE TABLE lyrics (song uuid, id uuid, words text, PRIMARY KEY (song, id));
```

Both tables share the song partition key, a uuid. The second table also contains the id clustering column.

4. Insert the data from the downloaded file into the songs table.
5. Insert data into the lyrics table.

The lyrics of songs by Big Data and John Cedrick mention love.

6. Navigate to the songs directory that you created in step 1, and take a look at the Solr schema.xml. Navigate to the lyrics directory and take a look at the schema. Notice that the order of the unique key in the schema and the partition key of the lyrics table are the same: (song, id). Using (id, song) does not work.

```
<schema name="songs_schema" version="1.5">
 <types>
 <fieldType name="uuid" class="solr.UUIDField" />
 <fieldType name="text" class="solr.TextField">
 <analyzer>
 <tokenizer class="solr.StandardTokenizerFactory"/>
 </analyzer>
 </fieldType>
 </types>
 <fields>
 <field name="song" type="uuid" indexed="true" stored="true"/>
 <field name="title" type="text" indexed="true" stored="true"/>
 <field name="artist" type="text" indexed="true" stored="true"/>
 </fields>
 <defaultSearchField>artist</defaultSearchField>
```

```

<uniqueKey>song</uniqueKey>
</schema>

<schema name="lyrics_schema" version="1.5">
 <types>
 <fieldType name="uuid" class="solr.UUIDField" />
 <fieldType name="text" class="solr.TextField" >
 <analyzer>
 <tokenizer class="solr.StandardTokenizerFactory"/>
 </analyzer>
 </fieldType>
 </types>
 <fields>
 <field name="song" type="uuid" indexed="true" stored="true"/>
 <field name="id" type="uuid" indexed="true" stored="true"/>
 <field name="words" type="text" indexed="true" stored="true"/>
 </fields>
 <defaultSearchField>words</defaultSearchField>
 <uniqueKey>(song, id)</uniqueKey>
</schema>
```

7. In the songs directory, post `solrconfig.xml` and `schema.xml` for the `internet.songs` core, and create the Solr core for `internet.songs`.
8. In the lyrics directory, post the `solrconfig.xml` and `schema.xml` for the `internet.lyrics` core, and create the Solr core for `internet.lyrics`.
9. Search for songs that have lyrics about love.

```
http://localhost:8983/solr/internet.songs/select/?q={!join
+fromIndex=internet.lyrics}words:love&indent=true&wt=json
```

The output includes two songs having the word "love" in the lyrics, one by Big Data and the other by John Cedrick:

```
"response": {"numFound":2, "start":0, "docs": [
 {
 "song": "a3e64f8f-bd44-4f28-b8d9-6938726e34d4",
 "title": "Dangerous",
 "artist": "Big Data" },
 {
 "song": "8a172618-b121-4136-bb10-f665cfcc469eb",
 "title": "Internet Love Song",
 "artist": "John Cedrick"}]
```

## Recursive join support

You can nest a join query to use the result of one join as an input for another join, and another, recursively. All joined data must reside on the same partition. To embed one query in the Solr query string of another, use the magic field name `_query_`.

Use the following syntax to construct a query that recursively joins cores.

```
F1:V1 AND _query_:"{!join fromIndex=keyspace.table} (F2:V2 AND _query_:\\"{!
join fromIndex=keyspace.table} (F3:V3)\\"")"
```

Where the top level from query includes a nested join query. The nested join in this example is:

```
query:\\"{!join fromIndex=keyspace.table} (F3:V3)\\"
```

Like an SQL `SELECT IN ... (SELECT IN ...)` query, Solr executes the nested join queries first, enabling multiple nested join queries if required.

A Solr join query is not a relational join where the values from the nested join queries are returned in the results.

## Example of a recursive join query

This example builds on the solr query time join example. Embed in the query to join songs and lyrics having words:"love" a second query to join award-winning videos using AND \_query\_:"award:true".

You can copy CQL commands, Solr HTTP requests, and the query from the downloaded commands.txt file.

1. In cqlsh, create a videos table that shares the internet keyspace and uses the same partition key as the songs and lyrics tables.

```
cqlsh> CREATE TABLE videos (song uuid, award boolean, title text, PRIMARY KEY (song));
```

All three tables use the song partition key, a uuid.

2. Insert the data from the downloaded file into the videos table. The video data sets the award field to true for the videos featuring songs by Big Data and Brad Paisley.
3. Navigate to the videos directory that was created when you unzipped the downloaded file.
4. In the videos directory, post solrconfig.xml and schema.xml, and create the Solr core for internet.videos.
5. Use a nested join query to recursively join the songs and lyrics documents with the videos document, and to select the song that mentions love and also won a video award.

```
http://localhost:8983/solr/internet.songs/select/?q=
 {!join+fromIndex=internet.lyrics}words:love AND _query_:
 {!join+fromIndex=internet.videos}award:true&indent=true&wt=json
```

Output is:

```
"response": {"numFound":1, "start":0, "docs": [
 {
 "song": "a3e64f8f-bd44-4f28-b8d9-6938726e34d4",
 "title": "Dangerous",
 "artist": "Big Data"
 }
]}
```

## Support for the legacy join query

DataStax Enterprise supports using the legacy syntax that includes to/from fields in the query. The requirements for using the legacy syntax are:

- Tables do not use [composite partition keys](#).
- The query includes the force=true local parser parameter, as shown in this example that joins mytable1 and mytable2 in mykeyspace.

### Legacy syntax example

```
curl 'http://localhost:8983/solr/mykeyspace.mytable1/select/?q=\{\!join
+from=id+to=id+fromIndex=mykeyspace.mytable2+force=true\}'
```

## Querying multiple tables

To map multiple Cassandra tables to a single [Solr core](#), use the Solr HTTP API. Specify multiple tables using the shards parameter. For example:

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

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

## Using HTTP API SolrJ and other Solr clients

Solr clients work with DataStax Enterprise. If you have an existing Solr application, using it with DataStax Enterprise is straight-forward. Create a schema, then import your data and query using your existing Solr tools. The [Wikipedia demo](#) is built and queried using SolrJ. The query is done using pure Ajax. No Cassandra API is used for the demo.

You can also use any Thrift API, such as Pycassa or Hector, to access DSE Search. Pycassa supports [Cassandra indexes](#). You can use indexes in Pycassa just as you use the solr\_query expression in DSE Search.

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

## Working with advanced data types: tuples and UDTs

DSE Search supports indexing and querying of advanced data types, including [tuples](#) and [user-defined types](#) (UDT).

- The tuple data type holds fixed-length sets of typed positional fields. Use a tuple as an alternative to a user-defined type.
- A user-defined type (UDT) facilitates handling multiple fields of related information in a table. UDTs are a specialization of tuples. All examples and documentation references to tuples apply to both tuples and UDTs.

Applications that require multiple tables can be simplified to use fewer tables by using a user-defined type to represent the related fields of information instead of storing the information in a separate table.

### Performance and memory

Tuples and UDTs are read and written as a single unit of information. Consider performance and memory impact when working with tuples and UDTs. Subfields are managed as the full tuple or UDT, and are not handled individually.

### Limitations

- Tuples and UDTs that are used inside primary key declarations are not supported.
- Tuples and UDTs that are used as CQL map values are not supported. Use a [workaround to simulate a map-like data model](#).

## Configuring tuples and UDTs in the Solr schema

DSE Search automatic schema generation from CQL tables supports tuples and UDTs.

- Use the [automated procedure for creating resources](#) that are based on a CQL table.

- Use the `dsetool infer_solr_schema` command to automatically infer and propose a schema that is based on the specified keyspace and table with the tuples and UDTs.

## Inserting data

Data that is inserted using CQL is automatically recognized by DSE Search and is indexed like any other data. However, data that is inserted by using the HTTP interface requires the tuple/UDT field as a string in its JSON representation. For example:

```
{"alternativeAddressCollection": [{"city": "NY", "street": "sesame1"}, {"city": "SF", "street": "sesame2"}], "mainAddress": {"street": "Sesame", "city": "Atlanta"}}
```

## Procedure

If you do not automatically generate the Solr schema, you must follow these steps to manually declare tuples and UDTs in the Solr schema:

1. Declare the actual tuple or UDT column of this type:

`com.datastax.bdp.search.solr.core.types.TupleField`.

2. Declare each tuple or UDT field with its own type.

The field name must be composed by the corresponding tuple or UDT name, plus the actual field index for tuples, or name for UDTs, concatenated by dots.

## Tuple configuration example

Example steps to configure a fullname tuple for DSE Search.

### In the Solr schema, declare the TupleField class

UDTs are a specialization of tuples. The TupleField class applies to both UDTs and tuples.

```
<fieldType class="com.datastax.bdp.search.solr.core.types.TupleField"
 name="TupleField"/>
```

### Create a table with the tuple

```
CREATE TABLE Person (nickname text, fullname <tuple<text, text>>)
```

### Configure the TupleField in the Solr schema

```
<field name="fullname" type="TupleField" indexed="true" stored="true"/>
<field name="fullname.field1" type="text" indexed="true" stored="true"/>
<field name="fullname.field2" type="text" indexed="true" stored="true"/>
```

## UDT configuration example

Example steps to configure a UDT for DSE Search.

### In the Solr schema, declare the UDTField class

```
<fieldType class="com.datastax.bdp.search.solr.core.types.TupleField"
 name="UDTField"/>
```

## Create a type with the UDT

You must create a type for UDTs.

```
CREATE TYPE Address (street text, city text)
```

## Create a table with the tuple

```
CREATE TABLE Location (id text primary key, address frozen<Address>);
```

## Configure the UDTField in the Solr schema

```
<field name="address" type="UDTField" indexed="true" stored="true"/>
<field name="address.street" type="text" indexed="true" stored="true"/>
<field name="address.city" type="text" indexed="true" stored="true"/>
```

## Nesting tuples and UDTs

DSE Search supports queries for nested tuples and UDTs. For example, you can nest and declare tuples and UDTs inside CQL lists and sets. You cannot nest tuples and UDTs inside maps or keys.

## Inserting data

Data that is inserted using CQL is automatically recognized by DSE Search and is indexed like any other data. However, data that is inserted by using the HTTP interface requires the tuple/UDT field as a string in its JSON representation. For example:

```
{"alternativeAddressCollection": [{"city": "NY", "street": "sesame1"}, {"city": "SF", "street": "sesame2"}] ,
"mainAddress": {"street": "Sesame", "city": "Atlanta"}}
```

## Create a type with the Address tuple

```
CREATE TYPE Address (street text, city text, residents set<tuple<text,
text>>)
```

## Create a table with the Address tuple

```
CREATE TABLE Location (id text, address Address)
```

## In the Solr schema, declare the TupleField class

UDTs are a specialization of tuples. The TupleField class applies to both UDTs and tuples.

```
<fieldType class="com.datastax.bdp.search.solr.core.types.TupleField"
name="TupleField"/>
```

## In the Solr schema, declare the TupleField and the nested TupleField

```
<field name="address" type="TupleField" indexed="true" stored="true"/>
<field name="address.street" type="text" indexed="true" stored="true"/>
<field name="address.city" type="text" indexed="true" stored="true"/>
```

```
<field name="address.residents" type="TupleField" indexed="true"
 stored="true" multiValued="true"/>
<field name="address.residents.field1" type="text" indexed="true"
 stored="true"/>
<field name="address.residents.field2" type="text" indexed="true"
 stored="true"/>
```

The residents nested tuple is TupleField. Each nested field is concatenated with each parent tuple or UDT by using dots.

See

## Tuples and UDTs as CQL map values

DSE Search does not support using tuples and UDTs as CQL map values. Use this workaround to simulate a map-like data model.

### Procedure

1. Declare a collection of tuples or UDTs that have a type field that represents what would have been the map key:

Create the tuple type. The tuple type applies to tuples and UDTs.

```
CREATE TYPE Address (type text, street text, city text)
```

Create table for UDT:

```
CREATE TABLE Person (name text primary key, addresses
set<frozen<address>>)
```

Or create a table for a tuple:

```
CREATE TABLE Person (name text primary key, addresses
set<frozen<tuple<text, text, text>>>)
```

2. Using this collection of tuples or UDTs as a map-like data model, it is possible to query for person addresses of a given type (key).

For example, to query for persons whose home address is in London:

```
{!tuple}addresses.type:Home AND
addresses.city:London
```

## UDT query examples

You can query [nested tuples and UDTs](#) inside CQL lists and sets. UDTs are a specialization of tuples. `{!tuple}` in these examples applies to UDTs.

**Note:** Selecting an entire UDT column in the CQL SELECT clause is supported. Selecting individual fields of a UDT is not supported.

More examples are available in the DataStax Developer Blog post [Tuple and UDT support in DSE Search](#).

### Querying fields

```
{!tuple}address.street:sesame
```

## Querying dynamic fields

```
<dynamicField name="user.position_*" type="text" indexed="true"
 stored="true"/>
{!tuple}user.position_day1:second
{!tuple}user.position_day2:first
```

## Querying collections

```
{!tuple}user.hobbies:swim
```

## Querying across UDT/tuple fields

```
({!tuple}father.name.firstname:Sam AND {!tuple}mother.name.firstname:Anne)
```

## Querying UDT/tuple fields with several conditions

You can find a tuple that satisfies several conditions. For example:

```
{!tuple}address.residents.field1:Alice AND address.residents.field2:Smith
```

**Note:** The difference in syntax specifies to search across tuples or within tuples.

- `{!tuple}condition1 AND {!tuple}condition2 AND {!tuple}conditionN` searches for docs that satisfy all conditions, but not necessarily satisfied by the same single tuple/UDT
- `{!tuple}condition1 AND condition2 AND conditionN` searches for documents within a tuple that satisfy all conditions

## Querying for nested tuples and UDTs

To query nested tuples and UDTs, use the same dot notation and the tuple query parser. Because UDTs are a specialization of tuples, you use the tuple query parser for tuples and UDTs. In this example, the dot notation identifies `address.resident` as a UDT.

For the example nested `address.residents` tuple, you can query for locations with a resident that has the first name Alice:

```
{!tuple}address.residents.field1:Alice
```

You can query for locations with a resident that has the first name Alice and second name Smith:

```
{!tuple}address.residents.field1:Alice AND address.residents.field2:Smith
```

**Note:** Tuples and UDTs are modelled internally as nested documents. The Solr block join is used internally to query them. Parents are identified with the `_parent_=true` field. Children are identified with `_parent_=false`. For certain types of queries, including negative queries and empty field queries, you might need to use the `_parent_` field.

## Querying for empty firstnames

The negation (-) and inclusion (+) operators must precede the `{!tuple}` directive.

```
- {!tuple}_parent_=false AND user.name.firstname:[* TO *]
```

## Negative queries

Negative queries use this syntax:

```
select * from demo where solr_query=' -{!tuple}name.firstname:*'
```

Negative queries with more than one condition must follow the Solr rules. Use this syntax:

```
{!tuple}address.street: * NOT (address.street:sesame AND address.number:32)
```

## Schema and data modeling

### Creating a schema and data modeling

This topic provides an overview of the Solr schema. For details about all the options and Solr schema settings, see [the Solr wiki](#). A Solr schema defines the relationship between data in a table and a [Solr core](#). The schema identifies the columns to index in Solr and maps column names to Solr types.

#### Table and schema definition

A CQL table must be created in Cassandra before creating the Solr core. DSE Search maps the schema fields and the unique key specification to the Cassandra key components, and generates a synthetic unique key for Solr, as shown in the following excerpts from the [Basic tutorial](#).

```
CREATE TABLE nhanes (
 "id" INT,
 "num_smokers" INT,
 "age" INT,
 .
 .
 PRIMARY KEY ("id", "age")
);
```

```
<schema name="solr_quickstart" version="1.1">
<types>
. . .
<fields>
<field name="id" type="int" indexed="true" stored="true"/>
<field name="num_smokers" type="int" indexed="true" stored="true"/>
<field name="age" type="int" indexed="true" stored="true"/>
. . .
<uniqueKey>(id,age)</uniqueKey>
. . .
```

The schema must have a unique key. The unique key is like a primary key in SQL. It maps to the Cassandra primary key, which DataStax Enterprise uses to route documents to cluster nodes.

Fields with `indexed="true"` are indexed and searched by Lucene. The indexed fields are stored in Cassandra, not in Lucene, regardless of the value of the `stored` attribute value, with the exception of [copy fields](#).

- To store a field with `indexed="false"` in Cassandra and enable the field to be returned on search queries, set `stored="true"`.
- To ignore the field, set both `indexed="false"` and `stored="false"`.

## Defining the unique key

DataStax Enterprise supports CQL tables using simple or compound primary keys, as shown in the Solr query join example, and composite partition keys.

When using legacy type mappings, the unique key must be defined as a string in the Solr schema.

## Sample schema

The following example from [Querying a CQL collection set](#) on page 205 uses a simple primary key.

```
<schema name="my_search_demo" version="1.5">
 <types>
 <fieldType class="solr.StrField" multiValued="true"
 name="StrCollectionField"/>
 <fieldType name="string" class="solr.StrField"/>
 <fieldType name="text" class="solr.TextField"/>
 <fieldType class="solr.TextField" name="textcollection"
 multiValued="true">
 <analyzer>
 <tokenizer class="solr.StandardTokenizerFactory"/>
 </analyzer>
 </fieldType>
 </types>
 <fields>
 <field name="id" type="string" indexed="true" stored="true"/>
 <field name="quotes" type="textcollection" indexed="true"
 stored="true"/>
 <field name="name" type="text" indexed="true" stored="true"/>
 <field name="title" type="text" indexed="true" stored="true"/>
 </fields>
 <defaultSearchField>quotes</defaultSearchField>
 <uniqueKey>id</uniqueKey>
</schema>
```

DSE Search indexes the id, quotes, name, and title fields.

## Mapping CQL primary keys and Solr unique keys

If the field is a compound primary key or composite partition key column in Cassandra, you must enclose the unique keys in parentheses in the Solr schema. The schema for this kind of table requires a different syntax than the simple primary key:

- List each compound primary key column that appears in the CQL table in the Solr schema as a field, just like any other column.
- Declare the unique key using the key columns enclosed in parentheses.
- Order the keys in the uniqueKey element as the keys are ordered in the CQL table.
- When using composite partition keys, do not include the extra set of parentheses in the Solr uniqueKey.

Cassandra Partition Key	CQL Syntax	Solr uniqueKey Syntax
Simple CQL primary key	CREATE TABLE ( . . . a <type> PRIMARY KEY, . . . );  (a is both the partition key and the primary key)	<uniqueKey>a</uniqueKey>  <b>Note:</b> Parenthesis are not required for a single key.
Compound primary key	CREATE TABLE ( . . . PRIMARY KEY ( a, b, c ) );	<uniqueKey>(a, b, c)</uniqueKey>

Cassandra Partition Key	CQL Syntax	Solr uniqueKey Syntax
The <a href="#">Basic tutorial</a> contains a schema for a Cassandra table that uses a CQL compound primary key.	(a is the partition key and a b c is the primary key)	
Composite partition key	CREATE TABLE ( . . . PRIMARY KEY ( ( a, b), c ) ;  (a b is the partition key and a b c is the primary key)	<uniqueKey>(a, b, c)</uniqueKey>

## Overriding `_partitionKey` when not using joins

The special `_partitionKey` field is used internally for joins. If you do not plan on using joins, you can override this field declaration in the `schema.xml` file for only the `docValues` and `indexed` properties.

```
<field name="_partitionKey" type="string" indexed="false"/>
```

To disable doc values, add `docValues="false"`.

```
<fieldname="_partitionKey" type="string" docValues="false"/>
```

## Document level boosting (Deprecated)

Document-level boosting is deprecated and will be removed in a future release. To add document-level boosting on CQL tables, add a column named "`_docBoost`" of type float to the table. Fields belonging to that document are boosted at indexing time.

## Changing a schema

Changing the Solr schema makes [reloading the Solr core](#) necessary. Re-indexing can be disruptive. Users can be affected by performance hits caused by re-indexing. Changing the schema is recommended only when absolutely necessary and during scheduled down time.

## Mapping of Solr types

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

Solr Type	CQL type	Cassandra Validator	Description
AsciiStrField	ascii	AsciiType	Indexed as a standard Solr StrField
BCDIntField	int	Int32Type	Binary-coded decimal (BCD) integer
BCDLongField	bigint	LongType	BCD long integer
BCDStrField	text, varchar	UTF8Type	BCD string
BinaryField	blob	BytesType	Binary data
BoolField	boolean	BooleanType	True (1, t, or T) or False (not 1, t, or T)

Solr Type	CQL type	Cassandra Validator	Description
ByteField	int	Int32Type	Contains an 8-bit number value
DateField	timestamp	DateType	Point in time with millisecond precision
DecimalStrField	decimal	DecimalType	Indexed as a standard Solr StrField
DoubleField	double	DoubleType	Double (64-bit IEEE floating point)
EnumField	text, varchar	UTF8Type	A closed set having a pre-determined sort order
ExternalFileField	text, varchar	UTF8Type	Values from disk file
FloatField	float	FloatType	32-bit IEEE floating point
GeoHashField	text, varchar	UTF8Type	Geohash lat/lon pair represented as a string
InetField	inet	InetAddressType	InetField is currently implemented and indexed as a standard Solr StrField.
IntField	int	Int32Type	32-bit signed integer
LatLonType	text, varchar	UTF8Type	Latitude/Longitude 2-D point, latitude first
LongField	bigint	LongType	Long integer (64-bit signed integer)
PointType	text, varchar	UTF8Type	Arbitrary n-dimensional point for spatial search
RandomSortField	text, varchar	UTF8Type	Dynamic field in random order
ShortField	int	Int32Type	Short integer
SortableDoubleField	double	DoubleType	Numerically sorted doubles
SortableFloatField	float	FloatType	Numerically sorted floating point
SortableIntField	int	Int32Type	Numerically sorted integer
SortableLongField	bigint	LongType	Numerically sorted long integer
SpatialRecursivePrefixTreeFieldType	text, varchar	UTF8Type	Spatial field type for a geospatial context
StrField	text, varchar	UTF8Type	String (UTF-8 encoded string or Unicode)
TextField	text, varchar	UTF8Type	Text, usually multiple words or tokens
TrieDateField	timestamp	DateType	Date field for Lucene TrieRange processing
TrieDoubleField	double	DoubleType	Double field for Lucene TrieRange processing
TrieField	n/a	n/a	Same as any Trie field type

Solr Type	CQL type	Cassandra Validator	Description
TrieFloatField	float	FloatType	Floating point field for Lucene TrieRange processing
TrieIntField	int	Int32Type	Int field for Lucene TrieRange processing
TrieLongField	bigint	LongType	Long field for Lucene TrieRange processing
UUIDField	uuid, timeuuid	UUIDType	Universally Unique Identifier (UUID)
VarIntStrField	varint	IntegerType	Indexed as a standard Solr StrField
Other	text, varchar	UTF8Type	Indexed as a standard Solr StrField

For efficiency in operations such as range queries, using Trie types is recommended. Keep the following information in mind about these types:

- **UUIDField**  
DataStax Enterprise supports the Cassandra TimeUUID type. A value of this type is a Type 1 UUID that includes the time of its generation. Values are sorted, conflict-free timestamps. For example, use this type to identify a column, such as a blog entry, by its timestamp and allow multiple clients to write to the same partition key simultaneously. To find data mapped from a Cassandra TimeUUID to a Solr UUIDField, users need to search for the whole UUID value, not just its time component.
- **BCD**  
A relatively inefficient encoding that offers the benefits of quick decimal calculations and quick conversion to a string.
- **SortableDoubleField/DoubleType**  
If you use the plain types (DoubleField, IntField, and so on) sorting will be lexicographical instead of numeric.
- **TrieField**  
Used with a type attribute and value: integer, long, float, double, date.

### Mapping of CQL collections

DSE Search maps collections as follows:

- Collection list and set: multi-valued field. See [Managing the field cache memory](#) on page 271 and [Example: copy fields and docValues](#) on page 224
- Collection maps: dynamic field. See [Using dynamic fields](#) on page 207.

The name of the dynamic field minus the wildcard is the map name. For example, a map column name dyna\* is mapped to dyna. Inner keys are mapped to the full field name.

## Legacy mapping of Solr types

Configure [legacy mapping of Solr types](#) if you created indexes in DataStax Enterprise 3.0.x or earlier. DataStax Enterprise 3.0.x and earlier use the legacy type mapping by default.

Solr Type	Cassandra Validator
TextField	UTF8Type
StrField	UTF8Type

Solr Type	Cassandra Validator
LongField	LongType
IntField	Int32Type
FloatField	FloatType
DoubleField	DoubleType
DateField	UTF8Type
ByteField	BytesType
BinaryField	BytesType
BoolField	UTF8Type
UUIDField	UUIDType
TrieDateField	UTF8Type
TrieDoubleField	UTF8Type
TrieField	UTF8Type
TrieFloatField	UTF8Type
TrieIntField	UTF8Type
TrieLongField	UTF8Type
All Others	UTF8Type

If you use legacy type mappings, the [solr schema](#) needs to define the unique key as a string.

## Changing Solr Types

Changing a Solr type mapper is rarely if ever done and is not recommended; however, for particular circumstances, such as converting [Solr types](#) such as the Solr LongField to TrieLongField, you configure the `dseTypeMappingVersion` using the force option.

The Cassandra internal validation classes of the types you are converting to and from must be compatible. Also, the actual types you are converting to and from must be valid types. For example, converting a legacy Trie type to a new Trie type is invalid because corresponding Cassandra validators are incompatible. The output of the `CLI` command, `DESCRIBE keyspace_name`, shows the validation classes assigned to columns.

For example, the `org.apache.cassandra.db.marshall.LongType` column validation class is mapped to `solr.LongType`. You can force this column to be of the `TrieLongField` type by using `force="true"` in the `solrconfig.xml`, and then performing a [Solr core](#) reload with re-indexing.

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

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

To use DSE Search data from a 3.0 release or earlier, use the legacy type mapping.

## Using copy fields

DSE Search supports the `copyField` directive with `stored=false` in the `schema.xml` file. Ingested data is copied by the copy field mechanism to the destination field for search, but is not stored in Cassandra. When you add a new `copyField` directive to the `schema.xml`, pre-existing and newly ingested data is re-indexed when copied as a result of the new directive.

The Solr `stored=true` `copyField` directive is removed. DataStax recommends that you upgrade an existing core by changing the directive and reloading the core as follows:

1. Change the `stored` attribute value of a `copyField` directive from true to false in the `schema.xml` file.
2. [Post the `solrconfig.xml` and the modified `schema.xml`](#).
3. [Reload the Solr core](#), specifying an in-place re-index.

Old data and Cassandra columns remain intact, but stored copy fields are not applied to new data.

### Using a copy field and multivalued field

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

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

### Using docValues and copy fields for faceting

Using `docValues` can [improve performance](#) of faceting, grouping, filtering, sorting, and other operations described on the [Solr Wiki](#).

For faceting to use `docValues`, the schema needs to specify `multiValued="true"` even if the field is a single-value facet field. The field needs to include `docValues="true"`. You also need to use a field type that supports being counted by Solr. The `text` type, which tokenizes values, cannot be used, but the `string` type works fine. DataStax Enterprise supports all aspects of copy fields except:

- The `maxChars` attribute is not supported.
- Copying from/to the same dynamic field is not supported.

## Example: copy fields and docValues

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

Step 9 covers how to see information about the per-segment field cache and filter cache. DataStax Enterprise moves the DSE per-segment filter cache off-heap by using native memory, hence reducing on-heap memory consumption and garbage collection overhead. The off-heap filter cache is enabled by default, but can be disabled by passing the following JVM system property at startup time: -  
`Dsolr.offheap.enable=false`.

### Procedure

1. If you did not already create a directory named `solr_tutorial46` that contains a `schema.xml` and `solrconfig.xml`, do so now. You can use the `schema.xml` and `solrconfig.xml` from the `demos/wikipedia` directory by copying these files to `solr_tutorial46`.

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

```

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

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

```

3. Run a CQL BATCH command, [as explained earlier](#), if the schema includes a multivalue field.

```

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

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

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

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

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

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

APPLY BATCH;

```

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

```

<schema name="my_search_demo" version="1.5">
 <types>
 <fieldType name="string" class="solr.StrField"/>
 <fieldType name="text" class="solr.TextField">
 <analyzer>
 <tokenizer class="solr.StandardTokenizerFactory"/>
 </analyzer>
 </fieldType>
 </types>
 <fields>

```

```

<field name="id" type="string" indexed="true" stored="true"/>
<field name="name" type="string" indexed="true" stored="true"/>
<field name="email" type="string" indexed="true" stored="true"/>
<field name="skype" type="string" indexed="true" stored="true"/>
<field name="irc" type="string" indexed="true" stored="true"/>
<field name="twitter" type="string" indexed="true" stored="true"/>
<field name="all" type="string" docValues="true" indexed="true"
 stored="false" multiValued="true"/>
</fields>
<defaultSearchField>name</defaultSearchField>
<uniqueKey>id</uniqueKey>
<copyField source="id" dest="all"/>
<copyField source="email" dest="all"/>
<copyField source="skype" dest="all"/>
<copyField source="irc" dest="all"/>
<copyField source="twitter" dest="all"/>
</schema>
```

- On the command line in the `solr_tutorial46` directory, upload the `schema.xml` and `solrconfig.xml` to Solr. Create the [Solr core](#) for the keyspace and table, `user_info.users`.

```

$ curl http://localhost:8983/solr/resource/user_info.users/solrconfig.xml
--data-binary @solrconfig.xml -H 'Content-type:text/xml; charset=utf-8'

$ curl http://localhost:8983/solr/resource/user_info.users/schema.xml
--data-binary @schema.xml -H 'Content-type:text/xml; charset=utf-8'

$ curl "http://localhost:8983/solr/admin/cores?
action=CREATE&name=user_info.users"
```

- In a browser, search Solr to identify the user, alias, and id of users having an alias smitty.

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

The output is:

```

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

- Run this query:

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

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

```

. . .
</result>
<lst name="facet_counts">
 <lst name="facet_queries"/>
 <lst name="facet_fields">
 <lst name="name">
```

```

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

```

8. Now you can [view the status](#) of the field cache memory to see the RAM usage of docValues per Solr field. Results look something like the example shown in [Example 2](#).
9. In the Solr Admin, after selecting a Solr core from the drop-down menu, click **Plugins / Stats**. Expand **dseFieldCache** and **dseFilterCache** to see information about the per-segment field cache and filter cache.

Choose **Watch Changes** or **Refresh Values** to get updated information.

The screenshot shows the Solr Admin interface at [localhost:8983/solr/admin](http://localhost:8983/solr/admin). The left sidebar has a 'nhanes\_ks.nha...' dropdown set to 'nhanes\_ks.nha...'. The 'Plugins / Stats' option is selected. The main content area shows the 'CACHE' section expanded, with 'dseFieldCache' and 'dseFilterCache' collapsed. 'dseFilterCache' is expanded, showing its class as 'com.datastax.bdp.search.solr.FilterCacheMBean', version as '1.0', and a detailed description: 'Provides an aggregate view of per-segment filter caches'. It also lists statistics: lookups (357), hits (357), hitratio (1.00), inserts (22), evictions (0), and size (726). Other collapsed sections include 'fieldCache', 'fieldValueCache', and 'queryResultCache'. At the bottom, there are links to Documentation, Issue Tracker, IRC Channel, Community forum, and Solr Query Syntax.

## Configuring DSE Search

DSE Search configuration includes configuring nodes, search components, threading, global filter cache, indexing, and completing other configuration tasks.

**Note:** By default, DataStax Enterprise turns off [virtual nodes](#) (vnodes). If you use vnodes on DSE Search nodes, DataStax recommends a maximum of 32 vnodes. DSE Search with vnodes increases overhead by approximately 30%. You must configure vnodes before starting to use DSE Search. You must configure vnodes before starting to use DSE Search.

## DSE Search configuration file (`solrconfig.xml`)

The `solrconfig.xml` resource file is the primary configuration file for configuring Solr for use with DSE Search.

- For DataStax Enterprise configuration, see [dse.yaml](#).

- For node and cluster configuration, see [cassandra.yaml](#).

You can use [custom resources](#) or [automatically create resources](#), including the `solrconfig.xml` file. The `solrconfig.xml` resource is persisted in the `solr_admin.solr_resources` database table.

[Reload a Solr core](#) after you modify the `solrconfig.xml` file. Changes apply only to the node where you reload the core. Do not make schema changes on production systems.

## Parameters

You might need to modify the following parameters to tune DSE Search. For full details, see the [Apache Solr Reference Guide](#).

### autoSoftCommit

For live indexing, ensure that the autoSoftCommit time is 100ms.

```
<autoSoftCommit>
 <maxTime>100</maxTime>
</autoSoftCommit>
```

See [Increasing indexing throughput](#) on page 236.

### directoryFactory

The directory factory to use for search indexes.

### dseTypeMappingVersion

The Solr type mapping version defines how Solr types are mapped to Cassandra Thrift or Cassandra types. Changing a Solr type mapper is rarely if ever done and is not recommended; however, for particular circumstances, such as converting [Solr types](#) such as the Solr LongField to TrieLongField, you configure the `dseTypeMappingVersion` using the force option. See [Changing Solr Types](#) on page 223 and [Configuring the Solr type mapping version](#) on page 238. Use this option only if you are an expert and have confirmed that the Cassandra internal validation classes of the types involved in the conversion are compatible. You can force a column to change type by using `force="true"`. For example:

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

After changing the type mapping, you must [reload](#) the Solr core with re-indexing.

### dseUpdateRequestProcessorChain

You can configure a custom URP to extend the Solr [UpdateRequestProcessor](#). See [Field input/output transformer example](#) on page 275.

### enableLazyFieldLoading

Do not change the default value of true:

```
<enableLazyFieldLoading>true</enableLazyFieldLoading>
```

A Solr bug [SOLR-8858](#) in earlier versions of Solr restricted changing this field.

### fieldInputTransformer

The field output transformer API is an option to the input/output transformer support in Solr.

### fieldOutputTransformer

The field output transformer API is an option to the input/output transformer support in Solr. See [Field input/output transformer example](#) on page 275 and [an Introduction to DSE Field Transformers](#).

### indexConfig

You can change the size of the RAM buffer and increase the soft commit time to tune the performance of re-indexing and index building. See [Configuring re-indexing](#) on page 267.

### lib

The location for library files in DataStax Enterprise is not the same location as open source Solr. Contrary to the examples shown in the `solrconfig.xml` file that indicate support for relative paths, DataStax Enterprise does not support the relative path values that are set for the `<lib>` property. DSE Search fails to find files in directories that are defined by the `<lib>` property. The workaround is to place custom code or Solr contrib modules in the Solr library directories. See [Configuring the Solr library path](#) on page 242.

### **maxBooleanClauses**

Defines the maximum number of clauses in a boolean query. If you change this value, restart the nodes to make the change effective. See [Changing maxBooleanClauses](#) on page 244.

### **mergeScheduler**

The default mergeScheduler settings are not appropriate for DSE Search near real time (NRT) indexing production use on a typical size server. DataStax recommends these settings as a starting point, and then adjust as appropriate to your environment:

- `maxThreadCount` = to the number of CPU cores divided by 2
- `maxMergeCount` = `maxThreadCount * 2`

For example, for 24 CPU cores:

```
<indexConfig>
 ...
<mergeScheduler
 class="org.apache.lucene.index.ConcurrentMergeScheduler">
 <int name="maxThreadCount">12</int>
 <int name="maxMergeCount">24</int>
 </mergeScheduler>
 ...

```

### **queryExecutorThreads**

You can set the query executor threads parameter in the `solrconfig.xml` file to enable multi-threading for filter queries, normal queries, and doc values facets:

```
<queryExecutorThreads>4</queryExecutorThreads>
```

See [Configuring multi-threaded queries](#) on page 239.

### **queryResponseWriter**

For performance, you can configure DSE Search to parallelize the retrieval of a large number of rows.

```
<queryResponseWriter name="javabin" class="solr.BinaryResponseWriter">
 ...
 <str
 $Factory</str>
 </queryResponseWriter>
```

See [Parallelizing large Cassandra row reads](#) on page 266.

### **ramBufferSizeMB**

The default value is 512 MB.

- To tune for performance, see [Configuring re-indexing](#) on page 267.
- To configure live indexing, increase the RAM buffer size to 2000.

```
<ramBufferSizeMB>2000</ramBufferSizeMB>
```

See [Increasing indexing throughput](#) on page 236.

### **requestHandler**

The correct search handler is required for CQL Solr queries in DSE Search.

When you [automatically generate resources](#), the `solrconfig.xml` file already contains the request handler for running CQL Solr queries in DSE Search. If you do not automatically generate resources and want to run CQL Solr queries using custom resources, the `CqlSearchHandler` handler is automatically inserted:

```
<requestHandler
name="solr_query" />
```

For recommendations for the basic configuration for the search handler, and an example that shows adding a search component, see [Configuring search components](#) on page 237.

In this example, to [configure the Data Import Handler](#), you can add a request handler element that contains the location of `data-config.xml` and data source connection information.

#### **rt**

To enable live indexing (also known as RT), add `<rt>true</rt>` to the `<indexConfig>` attribute.

```
<indexConfig>
 <rt>true</rt>
 ...
```

See [Increasing indexing throughput](#) on page 236.

#### **SolrFilterCache**

The DSE Search configurable global filter cache, SolrFilterCache, can reliably bound the filter cache memory usage for a Solr core. This implementation contrasts with the default Solr implementation which defines bounds for filter cache usage per segment. See [Configuring global filter cache for searching](#) on page 241.

#### **updateHandler**

You can configure per-document or per-field TTL. See [Expiring a DSE Search column](#) on page 253.

## Indexing resources

Indexing DSE Search documents requires these resources:

### **Schema.xml**

Describes the fields to index in Solr and types associated with them. These fields map to Cassandra columns.

### **Solrconfig.xml**

Holds configuration information for the index.

You can [automatically generate](#) these resources or you can [use custom resources](#). The `schema.xml` and `solrconfig.xml` resources are persisted in the `solr_admin.solr_resources` database table.

**Note:** When you post schema or configuration files simultaneously, schema disagreements might occur and cause Solr errors.

## Using automatically generated resources

Use the [automated procedure](#) for creating resources based on a CQL table. DataStax Enterprise generates `schema.xml` and `solrconfig.xml` resources for a CQL-based core, and then creates the core.

You can [customize automatic resource generation](#) using a number of options, such as the directory factory used for generating the resource. Use `dsetool` commands to reload a core or output the resource.

## Creating a core with automatic resource generation

The prerequisite for creating a core with automatic resource generation is an existing CQL table. DSE Search automatically generates default `solrconfig.xml` and `schema.xml` files that are based on the table metadata. You can generate resources automatically, using an HTTP POST request or a `dsetool` command and the `generateResources` option. The following list describes the options for generating resources automatically:

### `dsetool` syntax

The `dsetool` syntax for generating resources automatically and creating the Solr core is:

```
$ dsetool create_core keyspace.table generateResources=true [option ...]
```

This command preserves the case of keyspace and table names. You must use the correct case for the keyspace and table names.

User credentials can be provided in several ways, see [Providing credentials for authentication](#) on page 344.

The following example shows the `dsetool` method of automatically generating resources for the `nhanes_ks` keyspace and `nhanes` table used in the [Tutorial: Basics](#) on page 281:

```
$ dsetool create_core nhanes_ks.nhanes generateResources=true
```

By default, when you automatically generate resources, existing data is not re-indexed. You can check and customize the resources before indexing.

See [dsetool create\\_core](#) for details on command options.

To override the default and re-index existing data, use the `reindex=true` option:

```
$ dsetool create_core nhanes_ks.nhanes generateResources=true reindex=true
```

DataStax Enterprise uses the type mapping in [Mapping of Solr types](#) to generate the CQL-based core and resources. To generate resources automatically, the CQL table can consist of keys and columns of any CQL data type. However, decimal and varint are indexed as strings. Lucene does not support the precision required by these numeric types. Range and sorting queries do not work as expected if a table uses these types.

**Note:** If one or more nodes fail to create the core in distributed operations, an error message indicates a list of the failing node or nodes. Distributed operations fail if the core creation or core reload fails on one of the nodes. To solve the problem, follow the resolution steps in the error message. For example, if a core creation or core reload succeeds in all nodes except one, an error message suggests a core reload to solve the issue. The error message includes the failing node or nodes.

### Customizing automatic resource generation

You can customize `solrconfig.xml` and `schema.xml` generation by providing a yaml-formatted file of options:

#### `auto_soft_commit_max_time`

The maximum auto soft commit time in milliseconds.

#### `default_query_field`

The schema field to use when no field is specified in queries.

#### `type_mapping_version`

The Solr/Cassandra type mapping version.

#### `directory_factory_class`

The class name of the directory factory.

**enable\_string\_copy\_fields**

Specify to enable generation of non-stored string copy fields for non-key text fields. Text data can be tokenized or non tokenized. The enable\_string\_copy fields is false by default. True creates a non-stored, non-tokenized copy field, so that you can have text both ways.

**generate\_docvalues\_for\_fields**

Define the fields to automatically configure doc values in the generated schema. Specify '\*' to add all possible fields:

```
generate_docvalues_for_fields: '*' ## You can omit this parameter or not specify a value
```

or specify a comma-separated list of fields, for example:

```
generate_docvalues_for_fields: uuidfield, bigintfield
```

**index\_merge\_factor**

The index merge factor.

**index\_ram\_buffer\_size**

The index ram buffer size in megabytes.

**rt**

Enable live indexing to increase indexing throughput. Enable live indexing on only one Solr core per cluster.

```
rt=true
```

**Example to customize the solrconfig and yaml**

For example, create a yaml file that lists the following options to customize the solrconfig and yaml:

```
default_query_field: name
auto_soft_commit_max_time: 1000
generate_docvalues_for_fields: '*'
enable_string_copy_fields: false
```

Use the dsetool command to create the core and customize the solrconfig and schema generation. Use coreOptions to specify the YAML file, for example:

```
$ dsetool create_core nhanes_ks.nhanes generateResources=true
coreOptions=config.yaml
```

**Example to create a core with live indexing on**

To create the core with live indexing (also known as RT), use the dsetool command to create the core and use coreOptions to specify a YAML file that includes the rt: true setting, for example:

```
$ dsetool create_core udt_ks.users generateResources=true reindex=true
coreOptions=rt.yaml
```

where the contents of the rt.yaml are rt: true

You can verify that DSE Search created the solrconfig and schema by reading core resources using dsetool.

**Reloading a Solr core using dsetool**

After you modify schema.xml or solrconfig.xml or upload resource files (like a synonym file), reload a Solr core instead of creating a new one.

To simplify Solr code reloading, use `dsetool reload_core`. The syntax of the command is:

```
$ dsetool reload_core keyspace.table [option ...]
```

where *option* is one or more of the following options:

Option	Settings	Default	Description
schema=	<i>filepath</i>	n/a	Path of the schema file
solrconfig=	<i>filepath</i>	n/a	Path of the solrconfig.xml file
distributed=	true or false	true	<ul style="list-style-type: none"> <li>• true distributes and applies the reload operation to all nodes in the local DC.</li> <li>• false applies the reload operation only to the node it was sent to.</li> </ul>
reindex=	true or false	false	<ul style="list-style-type: none"> <li>• true re-indexes the data.</li> <li>• false does not re-index the data.</li> </ul>
deleteAll=	true or false	false	<ul style="list-style-type: none"> <li>• true deletes the already existing index before re-indexing; search results will return either no or partial data while the index is rebuilding.</li> <li>• false does not delete the existing index, causing the re-index to happen in-place; search results will return partially incorrect results while the index is updating.</li> </ul>

**Note:** To reload the core and prevent re-indexing, accept the default values `reindex=false` and `deleteAll=false`.

This command preserves the case of keyspace and table names. You must use the correct case for the keyspace and table names.

None of these options is mandatory. If the `schema.xml` or `solrconfig.xml`, or both, are provided, DataStax Enterprise uploads the files before reloading the core. You use these options, described in [Creating a core with automatic resource generation](#), the same way with the `dsetool` command or with an HTTP RELOAD request.

When you make a change to the schema, the compatibility of the existing index and the new schema is questionable. If the change to the schema made changes to a field's type, the index and schema will be incompatible. Changes to a field's type can occur in subtle ways, occasionally without a change to the `schema.xml` file itself. For example, a change to other configuration files, such as synonyms, can change the schema. If such an incompatibility exists, a full re-index, which includes deleting all the old data, of the Solr data is required. In these cases, anything less than a full re-index renders the schema changes ineffective. Typically, a change to the Solr schema requires a full re-indexing.

**Note:** If one or more nodes fail to reload the core in distributed operations, an error message indicates a list of the failing node or nodes. Issue the reload again only on those failing nodes using `distributed=false`.

## Re-indexing in place

Setting `reindex=true` and `deleteAll=false` re-indexes data and keeps the existing Lucene index. During the uploading process, user searches yield inaccurate results. To perform an in-place re-index, use this syntax:

```
$ dsetool reload_core keyspace.table reindex=true deleteAll=false
```

## Re-indexing in full

Setting `reindex=true` and `deleteAll=true` deletes the Lucene index and re-indexes the dataset. User searches initially return no or partial documents as the Solr cores reload and data is re-indexed.

```
$ dsetool reload_core keyspace.table reindex=true deleteAll=true
```

## Verifying indexing status

Use the Solr Admin to [check the indexing status](#).

### Unloading a Solr core

To disable full text search on a core, unload the core without removing its backing table.

To simplify Solr code unloading, use `dsetool unload_core`. The syntax of the command is:

```
$ dsetool unload_core keyspace.table [option ...]
```

where `option` is one or more of the following options:

Option	Settings	Description
<code>deleteDataDir=</code>	true or false	If true, deletes the underlying Cassandra data.
<code>deleteResources=</code>	true or false	If true, deletes the resources associated with the core, the <code>Solrconfig.xml</code> and schema.
<code>distributed=</code>	true or false	If true and <code>deleteDataDir=true</code> , deletes the index data directory on all nodes.

This command preserves the case of keyspace and table names. You must use the correct case for the keyspace and table names.

When authentication is enabled, the `dsetool` command is not available. Instead, use:

```
$ curl -u username:password "http://localhost:8983/solr/admin/cores?action=UNLOAD&name=keyspace.table"
```

**Note:** If one or more nodes fail to unload the core in distributed operations, an error message indicates the failing node or nodes. Issue the unload again.

### Reading core resources using dsetool

To simplify accessing the `Solrconfig.xml` and `schema.xml` files, use the `dsetool` command.

The syntax of the `dsetool get_core_schema` command is:

```
$ dsetool get_core_schema keyspace.table [current=true|false]
```

The syntax of the `dsetool get_core_config` command is:

```
$ dsetool get_core_config keyspace.table [current=true|false]
```

If the value of the current option is `false` (the default), DSE Search outputs the last `Solrconfig.xml` or `schema.xml` file that was uploaded; otherwise, DataStax Enterprise outputs the `solrconfig` or `schema` currently in use by the core.

For example, to output the latest uploaded schema, use this command:

```
$ dsetool get_core_schema nhanes_ks.nhanes
```

The generated schema for the `nhanes` table in the `nhanes_ks` keyspace appears.

For example, to output the current uploaded `Solrconfig.xml`, use this command:

```
$ dsetool get_core_config Solrconfig.xml current=true
```

## Using custom resources

Use custom index resources when requiring custom schema.xml or Solrconfig.xml changes.

### Uploading the schema and configuration

After writing [schema.xml](#) and [solrconfig.xml](#) files, use dsetool to post them to a DSE Search node in the DataStax Enterprise cluster to create a Solr index. You can also post additional resource files.

Resource files are stored in Cassandra database, not in the file system. The schema.xml and solrconfig.xml resources are persisted in the solr\_admin.solr\_resources database table.

### Procedure

- Post the configuration file:

```
$ dsetool write_resource keyspace.table name=schema.xml file=schemaFile.xml
```

- Post the schema file:

```
$ dsetool write_resource keyspace.table name=solrconfig.xml file=solrconfigFile.xml
```

- Post any other resources that you might need.

```
$ dsetool write_resource keyspace.table name=ResourceFile.xml file=schemaFile.xml
```

You can specify a path for the resource file:

```
$ dsetool write_resource keyspace.table name=ResourceFile.xml file=myPath1/myPath2/schemaFile.xml
```

- To verify the resources after they are posted:

```
$ dsetool read_resource keyspace.table name=ResourceFile.xml
```

### Creating a Solr core

You cannot create a [Solr core](#) unless you first upload the schema and configuration files. If you are creating a CQL-based Solr core, the table must exist in Cassandra before creating the core.

#### Creating a Solr core

Use dsetool to create a Solr core:

```
$ dsetool create_core keyspace.table [option ...]
```

This command preserves the case of keyspace and table names. You must use the correct case for the keyspace and table names.

Creating a Solr core on one node automatically creates the core on other DSE Search nodes, and DSE Search stores the files on all the Cassandra nodes. Core creation is per datacenter. If multiple DSE Search datacenters are present, repeat the Solr core creation on each datacenter. See [dsetool create\\_core](#) for details on command options.

**Note:** If one or more nodes fail to create the core in distributed operations, an error message indicates the failing node or nodes. If it failed to create the core immediately, issue the create again. If it failed to create on some nodes, issue a reload for those nodes to load the newly created core.

By default, the cassandra user has full permissions on all keyspaces. If you specify a non-default user to create a Solr core, the specified user must have the necessary Cassandra permissions. When specifying a user to create a Solr core, ensure that the user has either:

- CREATE permission on all keyspaces
- All permissions on all keyspaces (superuser)

See [dsetool create\\_core](#) for command option details.

## Increasing indexing throughput

Live indexing enables queries to be made against recently indexed data. Live indexing, also known as RT (real time) indexing, improves index throughput and reduces Lucene reader latency while supporting all Solr functionality. Live indexing works for all DSE Search applications. Fields that are sorted on must be DocValues, otherwise the field cache is used and is inefficient with live indexing.

DocValues are a way of recording field values internally that is more efficient for some purposes, such as sorting and faceting, than traditional indexing. To use docValues, enable it for a field; define a field type and then define fields of that type with docValues enabled in `schema.xml`.

### Procedure

1. Enable live indexing on only one Solr core per cluster.
2. To enable live indexing (also known as RT), add `<rt>true</rt>` to the `<indexConfig>` attribute of the `solrconfig.xml` file.  
`<rt>true</rt>`
3. To configure live indexing, edit the `solrconfig.xml` file and change these settings:

- Increase the RAM buffer size to enable faster indexing.

```
<ramBufferSizeMB>2000</ramBufferSizeMB>
```

- Ensure that the `autoSoftCommit/maxTime` is 100ms:

```
...
<autoSoftCommit>
 <maxTime>100</maxTime>
</autoSoftCommit>
```

The `autoSoftCommit/maxTime` for live indexing saturates at 3000 ms, any higher value is overridden to limit the maximum time to 3000ms.

4. Increase the heap size. For live indexing, DataStax recommends a heap size of at least 20 GB for use with Java 1.8 and G1GC. A larger heap size allows you to allocate more RAM buffer size, which contributes to faster live (RT) indexing. Enable live indexing on only one Solr core per cluster.
5. In the `dse.yaml` file, define the number of indexing threads per Solr core with the `max_solr_concurrency_per_core` option. To achieve optimal performance, assign this value to number of available CPU cores divided by the number of Solr cores. For example, with 12 CPU cores and 3 Solr cores, the suggested value is 4.
6. Restart DataStax Enterprise to use live indexing with the increased heap size.

# Configuring search components

The wikipedia demo `solrconfig.xml` configures the Search Handler as follows:

```
<requestHandler name="search" class="solr.SearchHandler" default="true">
```

DataStax recommends using this basic configuration for the Search Handler.

## Configuring additional search components

To configure the search handler for managing additional search components, you generally need to add the additional component to the array of last-components to preserve the default configured components. Distributed search does not work properly if you fail to preserve the default configured components. Unless otherwise specified in Solr documentation, declare the additional component as described in the following example.

### How to add a search component

This example shows the configuration of an additional search component for spellchecking and how to add that component to the last-components array of the search handler. The additional component specifies the Java spelling checking package JaSpell:

#### Component configuration

```
<searchComponent class="solr.SpellCheckComponent" name="suggest_jaspell">
 <lst name="spellchecker">
 <str name="name">suggest</str>
 <str name="classname">org.apache.solr.spelling.suggest.Suggester</str>
 <str
 name="lookupImpl">org.apache.solr.spelling.suggest.jaspell.JaspellLookup</
 str>
 <str name="field">suggest</str>
 <str name="storeDir">suggest</str>
 <str name="buildOnCommit">true</str>
 <float name="threshold">0.0</float>
 </lst>
</searchComponent>
```

To add the spell check component to the last-components array:

#### Last-components declaration

```
<requestHandler class="org.apache.solr.handler.component.SearchHandler"
 name="/suggest">
 <lst name="defaults">
 <str name="spellcheck">true</str>
 <str name="spellcheck.dictionary">suggest</str>
 <str name="spellcheck.collate">true</str>
 <str name="spellcheck.extendedResults">true</str>
 </lst>
 <arr name="last-components">
 <str>suggest_jaspell</str>
 </arr>
</requestHandler>
```

## Segregating workloads in a cluster

A common question is how to use real-time (Cassandra), integrated Hadoop or Spark (DSE Analytics), an external Hadoop system, or DSE Search nodes in the same cluster. Within the same datacenter, attempting to run DSE Search on some nodes and real-time queries, analytics, or external Hadoop on other nodes does not work. The answer is to [organize the nodes](#) running different workloads into virtual datacenters.

### Replicating data across datacenters

You set up replication for DSE Search nodes exactly as you do for other nodes in a Cassandra cluster, by [creating a keyspace](#). You can [change the replication](#) of a keyspace after creating it.

## Configuring the Solr type mapping version

The Solr type mapping version defines how Solr types are mapped to [Cassandra Thrift](#) or [Cassandra types](#).

DataStax Enterprise 3.2 and later releases use type mapping version 2. New tables, including those using [compact storage](#), created using CQL 3 require type mapping version 2. [CQL 3](#) is the default mode in Cassandra 2.x, which is based on CQL specification 3.1.0.

Tables that were migrated from previous DataStax Enterprise installations can keep the old mapping versions. Newly created non-CQL 3 tables require type mapping version 1.

During and after upgrades from 3.2.x to 4.x, tables that were created with DataStax Enterprise 3.0.x require type mapping version 0. DataStax Enterprise 3.0 used type mapping version 0, also known as legacy mapping. DataStax Enterprise 3.1 used type mapping version 1.

To change the type mapping, configure `dseTypeMappingVersion` in the `solrconfig.xml` file:

```
<dseTypeMappingVersion>2</dseTypeMappingVersion>
```

Switching between versions is not recommended after the Solr core has been created successfully: attempting to load a `solrconfig.xml` file with a different `dseTypeMappingVersion` configuration and reloading the Solr core will cause an error.

## Securing a DSE Search cluster

DataStax Enterprise supports secure enterprise search using Apache Solr and Lucene. The [security table](#) summarizes the security features of DSE Search and other integrated components.

For optimal security, DataStax recommends using [CQL Solr queries in DSE Search](#).

DSE Search data is completely or partially secured by using DataStax Enterprise security features:

### Object permission management

Access to Solr documents, excluding cached data, can be limited to users who have been granted access permissions. [Object permission management](#) also secures tables used to store Solr data.

### Transparent data encryption

Data at rest in Cassandra tables, excluding cached and Solr-indexed data, can be encrypted. [Transparent data encryption](#) occurs on the Cassandra side and impacts performance slightly.

### Internode (node-to-node) encryption

Internal, node-to-node Solr communication uses the netty transport. Enable and set the [server\\_encryption\\_options](#) in the `cassandra.yaml` file to secure your Solr internode communication.

### Client-to-node encryption

You can use SSL to encrypt HTTP access to Solr data. Enable SSL [Client-to-node encryption](#) on the DSE Search node.

### Strong cipher suites

If you use strong cipher suites to ensure secure node-to-node and client-to-node communication, you must [install JCE](#) to ensure support for all encryption algorithms. Some of the cipher suites in the default set of [server\\_encryption\\_options](#) in `cassandra.yaml` are included only in the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files. To ensure support for all encryption algorithms, install the JCE Unlimited Strength Jurisdiction Policy Files.

### Kerberos authentication: required for DSE Search in production

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

### Cassandra internal authentication or DataStax Enterprise LDAP authentication

To configure DSE Search to use Cassandra internal authentication:

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

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

2. [Configure the replication strategy for the system\\_auth keyspace](#).
3. Start the server.
4. Open a browser, and go to the service web page. For example, assuming you ran the [Wikipedia demo](#), go to `http://localhost:8983/demos/wikipedia/`.

When prompted, provide the Cassandra username and password.

**Note:** Security configuration on a DSE Search node is automatic and internal. Additional configuration is not required for Tomcat and Solr. Change your `web.xml` or `server.xml` files only for custom advanced setups.

### HTTP Basic Authentication (testing and development only)

You can use HTTP Basic Authentication, but this is not recommended for production. To secure DSE Search in production, enable DataStax Enterprise [Kerberos](#) authentication or [search using CQL](#) instead.

When you enable Cassandra [internal authentication](#) by specifying `authenticator: org.apache.cassandra.auth.PasswordAuthenticator` in `cassandra.yaml`, clients must use [HTTP Basic Authentication](#) to provide credentials to Solr services. Due to the stateless nature of HTTP Basic Authentication, this can have a significant performance impact as the authentication process must be executed on each HTTP request. For this reason, DataStax does not recommend using internal authentication on DSE Search clusters in production.

## Configuring multi-threaded queries

You can set the `query executor threads` parameter in the `solrconfig.xml` file to enable multi-threading for filter queries, normal queries, and doc values facets.

```
<queryExecutorThreads>4</queryExecutorThreads>
```

## Shard transport options for DSE Search communications

A custom, TCP-based communications layer for Solr is the default type in DataStax Enterprise. The TCP-based type, netty, is an alternative to the slower more resource intensive HTTP-based, Tomcat-backed interface. The netty communications layer improves DSE Search inter-node communications in several ways:

- Lowers latency
- Reduces resource consumption
- Increases throughput even while handling thousands of concurrent requests
- Provides nonblocking I/O processing

To avoid distributed deadlock during queries, switch from the HTTP-based communications to the netty non-blocking communications layer.

The TCP-based communications layer for DSE Search supports client-to-node and node-to-node encryption using SSL, but does not support Kerberos.

Configure the shard transport options in the `dse.yaml` file to select HTTP- or TCP-based communication.

**Note:** Shard transport options work only in datacenters where the replication factor is not equal to the number of nodes. You can verify or [change the replication factor](#) of the keyspace.

The `shard_transport_options` in the `dse.yaml` file for managing inter-node communication between DSE Search nodes are:

- `type: netty` or `http`

The default type, netty, configures TCP-based Solr communications. Choosing `http` configures Solr communication that uses the standard HTTP-based communications interface. Accept the netty default so that the following netty options are applicable.

- `netty_server_port: 8984`

The TCP listen port, mandatory to use the netty type. To use the `http` type indefinitely, either comment `netty_server_port` or set it to `-1`.

- `netty_server_acceptor_threads`

The number of server acceptor threads. The default is the number of available processors.

- `netty_server_worker_threads`

The number of server worker threads. The default is the number of available processors times 8.

- `netty_client_worker_threads`

The number of client worker threads. The default is the number of available processors times 8.

- `netty_client_max_connections`

The maximum number of client connections. The default is 100.

- `netty_client_request_timeout`

The client request timeout in milliseconds. The default is 60000.

### Upgrading to use the netty type

For upgrades to DataStax Enterprise 4.0 or later, perform the upgrade procedure using the shard transport type of your old installation, and after the upgrade, change the shard transport type to netty. Start the cluster using a rolling restart.

# Changing Tomcat web server settings

To configure DSE Search security, you can make changes in the Tomcat `server.xml` file to change the IP address for client connections to DSE Search using the HTTP and Solr Admin interfaces, and other changes, including specifying SSL ciphers and ports.

## Procedure

Several configuration changes in the Tomcat `server.xml` file:

1. You can change the IP address for client connections to DSE Search.

The default IP address that the HTTP and Solr Admin interface uses to access DSE Search is defined with `rpc_address` in the `cassandra.yaml` file.

- The default `rpc_address` value `localhost` enables Tomcat to listen on the `localhost` only.
- To enable Tomcat to listen on all configured interfaces, set `rpc_address` to `0.0.0.0`.

To change the IP address for client connections to DSE Search using the HTTP and Solr Admin interfaces, change the client connection `rpc_address` in the `cassandra.yaml` file or create a Tomcat connector.

Option	Description
Create a Tomcat connector	In the <code>&lt;Service name="Solr"&gt;</code> section of the <code>server.xml</code> file:

```
<Connector
 port="PORT"
 protocol="HTTP/1.1"
 address="IP_ADDRESS"
 connectionTimeout="20000"
 redirectPort="8443"
 />
```

### Change the `rpc_address`

Change `rpc_address` in the `initialization_options` section of the `cassandra.yaml` file. The `rpc_address` is read on startup only.

2. For advanced users only: In the Tomcat `server.xml` file, you can specify a different client connection port, other than the default port 8983. If you specify a connection port other than the default, the automatic SSL connection configuration that is typically performed by DataStax Enterprise is not done. You must provide the valid connector configuration, including keystore path and password. See the DataStax Support article [Configuring the DSE Solr HTTP/HTTPS port](#).
3. After making changes, restart the node.

# Configuring global filter cache for searching

The DSE Search configurable global filter cache, `SolrFilterCache`, can reliably bound the filter cache memory usage for a Solr core. This implementation contrasts with the default Solr implementation which defines bounds for filter cache usage per segment. `SolrFilterCache` bounding works by evicting cache entries after the configured per core high water mark is reached, and stopping after the configured lower water mark is reached.

To configure the global filter cache thresholds, change the `filterCache` element in the `solrconfig.xml` file.

## Procedure

1. In your `solrconfig.xml` file, set the class attribute of the `filterCache` element to `solr.search.SolrFilterCache`.
2. To define the low and high watermark for cache eviction, set the `lowWaterMarkMB` and `highWaterMarkMB` attributes.

### **lowWaterMarkMB**

The minimum memory use in MB to stop cache eviction.

### **highWaterMarkMB**

The maximum memory use in MB to trigger the cache eviction.

Use the following configuration to set the lower bound cache usage of 128 MB and the upper bound of cache usage of 256 MB:

```
<filterCache
 class="solr.search.SolrFilterCache"
 lowWaterMarkMB="128"
 highWaterMarkMB="256" />
```

## Configuring the Solr library path

The location for library files in DataStax Enterprise is not the same location as open source Solr. Contrary to the examples shown in the `solrconfig.xml` file that indicate support for relative paths, DataStax Enterprise does not support the relative path values for `<lib>` directives. DSE Search fails to find files in directories that are defined with the `<lib>` directive.

The workaround is to place the JAR file, custom code, or the Solr contrib modules in the Solr library directories.

The default Solr library path location depends on the type of installation:

Installer-Services	<code>/usr/share/dse/resources/solr/lib</code>
Package installations	<code>/usr/share/dse/solr/lib</code>
Installer-No Services and Tarball installations	<code>install_location/resources/solr/lib</code>

When the plugin JAR file is not in the directory that is defined by the `<lib>` property. Attempts to deploy custom Solr libraries in DataStax Enterprise fail with `java.lang.ClassNotFoundException` and an error in the `system.log` like this:

```
ERROR [http-8983-exec-5] 2015-12-06 16:32:33,992 CoreContainer.java (line 956) Unable to create core: boogle.main
org.apache.solr.common.SolrException: Error loading class
'com.boogle.search.CustomQParserPlugin'
at org.apache.solr.core.SolrCore.(SolrCore.java:851)
at org.apache.solr.core.SolrCore.(SolrCore.java:640)
at
com.datastax.bdp.search.solr.core.CassandraCoreContainer.doCreate(CassandraCoreContainer
at
com.datastax.bdp.search.solr.core.CassandraCoreContainer.create(CassandraCoreContainer
at
com.datastax.bdp.search.solr.core.SolrCoreResourceManager.createCore(SolrCoreResourceMa
at
com.datastax.bdp.search.solr.handler.admin.CassandraCoreAdminHandler.handleCreateAction
...
Caused by: org.apache.solr.common.SolrException: Error loading class
'com.boogle.search.CustomQParserPlugin'
```

```

at
 org.apache.solr.core.SolrResourceLoader.findClass(SolrResourceLoader.java:474)
at
 org.apache.solr.core.SolrResourceLoader.findClass(SolrResourceLoader.java:405)
at org.apache.solr.core.SolrCore.createInstance(SolrCore.java:541)
...
Caused by: java.lang.ClassNotFoundException:
 com.boogle.search.CustomQParserPlugin
at java.net.URLClassLoader$1.run(Unknown Source)
at java.net.URLClassLoader$1.run(Unknown Source)
...

```

## Workaround

Using the class in this example with the JAR filename

`com.boogle.search.CustomQParserPlugin-1.0.jar`, follow these steps to get the custom plugin working on all DSE Search nodes.

1. Define the parser in the `solrconfig.xml` file:

```

<queryParser name="myCustomQP"
 class="com.boogle.search.CustomQParserPlugin"/>

```

2. Place custom code or Solr contrib modules in the Solr library directories
3. Deploy the JAR file on all DSE Search nodes in the cluster in the appropriate `lib/` directory.

For example, package installations: `/usr/share/dse/solr/lib/com.boogle.search.CustomQParserPlugin-1.0.jar`

4. Reload the Solr core with the new configuration.

## Limiting columns indexed and returned by a query

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

DataStax Enterprise supports CQL tables. When using dynamic fields, the default column limit applies only if the table is created using the deprecated method of automatically creating a table on core creation or creating a compact storage table.

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

```

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

```

## Configuring autocomplete/spellcheck

By default, the `solrconfig.xml` does not include configuration for the Solr suggestor. After [configuring the search component](#) in the `solrconfig.xml` for `/suggest`, you can issue a query specifying the autocomplete/spellcheck behavior using the `shards.qt=` parameter. For example, to test the suggestor:

```

curl "http://localhost:8983/solr/mykeyspace.mysolr/select?shards.qt=/
 suggest&qt=/suggest&q=testin"

```

## Changing maxBooleanClauses

The `maxBooleanClauses` parameter defines the maximum number of clauses in a boolean query. An exception is thrown if this value is exceeded. If you change the `maxBooleanClauses` parameter in `solrconfig.xml`, restart the nodes to make the change effective. Reloading the [Solr cores](#) does not make this change effective.

**Note:** Limitations and known Apache Solr issues apply to DSE Search queries, including the 1024 `maxBoolean` clause limit in [SOLR-4585](#).

## Configuring the Data Import Handler (deprecated)

The [Data Import Handler](#) is deprecated and will be removed in a future release.

You can import data into DSE Search from data sources, such as XML and RDBMS. The configuration-driven method to import data differs from the method that is used by open source Solr. See the [Migrating from MySQL to Cassandra Using Spark](#) post.

## Operations

You can run Solr on one or more nodes. DataStax does not support running Solr and Hadoop on the same node, although it's possible to do so in a development environment. In production environments, [separate workloads](#) by running real-time (Cassandra), analytics (Hadoop), or DSE Search (Solr) nodes on separate nodes and in separate datacenters.

## Adding, decommissioning, repairing a DSE search node

To add, decommission, or repair a DSE Search node, use the same methods as you would for a Cassandra node.

### Procedure

- To increase the number of DSE Search nodes in a datacenter, [add a search node](#) to the cluster, and then use OpsCenter to [rebalance the cluster](#).  
The default DSESIMPLESnitch automatically puts the DSE Search nodes in the same data center.
- To repair a DSE Search node, see [Repairing nodes](#) in the Cassandra documentation.  
DataStax recommends using the [subrange repair method](#).
- To decommission a DSE Search node, see [Removing a node](#) in the Cassandra documentation.

## Enabling the disk failure policy

You can configure DSE Search to respond to an I/O exception during any index update by enabling the indexing disk failure policy. When enabled, DSE Search uses the configured Cassandra [disk failure policy](#), which by default shuts down gossip and other processes, rendering the node dead. When disabled, DSE Search ignores the Cassandra disk failure policy. The node does not shut down.

## Procedure

1. In the dse.yaml file, locate this section:

```
Applies the configured Cassandra disk failure policy to index write
failures.
Default is disabled (false).
#
enable_index_disk_failure_policy: false
```

2. Uncomment the last line and change false to true:

```
enable_index_disk_failure_policy: true
```

## Restricted query routing

DSE Search restricted query routing is designed for applications that have a data model that supports restricting common queries to a single partition. This feature is for use by experts only and should be used with care. You can restrict queries based on a list of partition keys to a limited number of nodes. You can also restrict queries based on a single token range. Use token range routing only if you thoroughly understand cluster token placement.

### Partition key routing

To specify routing by partition keys, use the route.partition query parameter and set its value to one or more partition keys. DSE Search queries only the nodes that own the given partition keys. The vertical line delimiter separates components of a composite key. The comma delimiter separates different partition keys.

For example:

```
route.partition=k1c1|k1c2,k2c1|k2c2 . . .
```

If the actual partition key value contains a delimiter character, use a backslash character to escape the delimiter.

### Examples

You can route [Solr HTTP API](#) and [Solr CQL](#) queries. This example shows how to use the route queries on a table with a composite partition key, where "nike" and "2" are composite key parts.

```
http://localhost:8983/solr/test.route/select?
q=*:*&indent=true&shards.info=true&route.partition=nike|2,reebok|2
```

Or, in CQL:

```
SELECT * FROM test.route WHERE solr_query='{"q" : "*:*", "route.partition" :
["nike|2", "reebok|2"]}'
```

### Token range routing

For simplicity, routing queries by partition range is recommended over routing by token range. To specify routing by token range, use the route.range query parameter and set its value to the two token values that represent the range, separated by comma:

For example:

```
route.range=t1,t2
```

DSE Search queries only the nodes in the given token range.

## Shuffling shards to balance the load

DataStax Enterprise uses a shuffling technique to balance the load, and also attempts to minimize the number of shards that are queried as well as the amount of data that is transferred from non-local nodes.

To balance the load in a distributed environment, choose from several strategies for shuffling the shards. The shard shuffling strategy specifies how one node is selected over others for reading the Solr data. The value of the `shard.shuffling.strategy` parameter must be one of the following values:

- host  
Shards are selected based on the host that received the query.
- query  
Shards are selected based on the query string.
- host\_query  
Shards are selected by host x query.
- random  
Different random set of shards are selected with each request (default).
- SEED  
Selects the same shard from one query to another.

### Methods for selecting shard shuffling strategy

- Append `shard.shuffling.strategy = strategy` to the HTTP API query. For example:

```
http://localhost:8983/solr/wiki.solr/select?
q=title:natio*&shard.shuffling.strategy=host
```

Issuing this query determines the shard shuffling strategy for this query only.

- Create a `dse-search.properties` file and POST it to Solr. For example:

1. Create the `dse-search.properties` file with the following contents:

```
shard.shuffling.strategy=query
```

2. Post the command to DSE Search. For example:

```
curl -v --data-binary @dse-search.properties
http://localhost:8983/solr/resource/wiki.solr/dse-search.properties
```

Posting the command determines the shard shuffling strategy for all queries to the specified [Solr core](#). The strategy is propagated to all nodes and saved in Solr core metadata.

- Set the following parameters to use the SEED strategy:

1. Pass the `shard.shuffling.strategy=SEED` as a request parameter.
2. Specify a request parameter, such as an IP address or any string, using the `shard.shuffling.seed` parameter. When you reuse the same seed value between queries on a stable cluster, the same shard strategy will be in effect.

Every time you pass the same string, the same list of shards is queried, regardless of the target node you actually query; if you change the string, a different list of shards are queried.

3. Verify that the strategy was maintained by passing the `shards.info=true` request parameter. For example:

```
curl "http://localhost:8983/solr/demo.solr/select??
q=text:search&shards.info=true&shard.shuffling.strategy=SEED&shard.shuffling.seed=1"
```

Shuffling does not always result in the node selection you might expect. For example, using a replication factor of 3 with six nodes, the best and only solution is a two-shard solution where half of the data is read from the originator node and half from another node. A three-shard solution would be inefficient.

## Shard routing for distributed queries

On DSE Search nodes, the shard selection algorithm for distributed queries uses a series of criteria to route sub-queries to the nodes most capable of handling them. The best node is determined by a chain of node comparisons. Selection occurs in the following order using these criteria:

1. Is node active? Preference to active nodes.
2. Is the requested core indexing, or has it failed to index? If node is not in either of these states, select the best node.
3. Node health rank, an exponentially increasing number between 0 and 1 that describes the health node so that, all the previous criteria being equal, a node with a better score is chosen first. This node health rank value is exposed as a JMX metrics under ShardRouter.

How do the nodes compare on uptime and dropped mutations? The node health rank is calculated by the formula below:

```
node_health = uptime / (1 + drop_rate)
```

where:

- drop\_rate = the rate of dropped mutations per minute over a sliding window of configurable length. To configure the historic time window, set [dropped\\_mutation\\_window\\_minutes](#) in dse.yaml. A high dropped mutation rate indicates an overloaded node; for example, Cassandra insertions and updates.
  - uptime = a score between 0 and 1 that weights recent downtime more heavily than less recent downtime.
4. Is node close to the node that is issuing the query? Node selection uses Cassandra endpoint snitch proximity. Give preference to closer nodes.

After using these criteria, node selection is random.

## Managing the location of Solr data

Data that is added to a DSE Search node is locally indexed in the Cassandra node. Data changes to one node also apply to the other node. Like Cassandra data files, DSE Search has its own indexing files. You can control where the Search indexing data files are saved on the server. By default, the Solr data is saved in `cassandra_data_dir/solr.data`, or as specified by the `dse.solr.data.dir` system property.

### Procedure

1. Shut down the search node.
2. Move the `solr.data` directory to the new location.
3. Specify the location:

Option	Description
From the command line	<pre>\$ cd install_location \$ bin/dse cassandra -s -Ddse.solr.data.dir=/solr_data_dir</pre>

In <code>dse.yaml</code>	<pre>solr_data_dir: solr_data_dir</pre>
--------------------------	-----------------------------------------

4. Start the node.

## Results

The location change is permanent when set in `dse.yaml`. The command-line argument must be used consistently or DSE reverts to the default data directory.

## Changing the Solr connector port

To change the Solr port from the default, 8983, change the `http.port` setting in the `catalina.properties` file that is installed with DSE in `install_location/tomcat/conf`, usually `/usr/share/dse/tomcat/conf/catalina.properties`.

## Deleting Solr data

To delete a Cassandra table and its data, including the data indexed in Solr, from a DSE Search node, drop the table using CQL. The following example assumes that you ran the example of [using a collection set](#). List the Solr files on the file system, drop the table named `mysolr` that the demo created, and then verify that the files are deleted from the file system:

Wait until you finish working through all the examples before you delete the example data.

## Procedure

1. List the Solr data files on the file system.

- Installer-Services and Package installations:

```
ls /usr/local/var/lib/dse/data/solr.data/mykeyspace.mysolr/index/
```

- Installer-No Services and Tarball installations:

```
ls /var/lib/cassandra/data/solr.data/mykeyspace.mysolr/index
```

The output looks something like this:

```
_33.fdt _35_nrm.cfe _38_Lucene40_0.tim
_33.fdx _35_nrm.cfs _38_Lucene40_0.tip
_33.fnm _36.fdt _38_nrm.cfe
. . .
```

2. Launch `cqlsh` and execute the CQL command to drop the table named `solr`.

```
DROP TABLE mykeyspace.mysolr;
```

3. Exit `cqlsh` and verify that the files are deleted from the file system. For example:

```
ls /var/lib/cassandra/data/solr.data/mykeyspace.mysolr/index
```

The output is:

```
ls: /var/lib/cassandra/data/solr.data/mykeyspace.mysolr/index: No such
file or directory
```

## Viewing the Solr core status

You can use the Solr API to view the status of the [Solr core](#). For example, to view the status of the wiki.solr core after running the wikipedia demo, use this URL:

```
http://localhost:8983/solr/#/~cores/wiki.solr
```

The screenshot shows the Apache Solr Admin interface for the core named "wiki.solr". The left sidebar has a "Core Admin" section selected. The main panel displays the "Core" and "Index" sections. The "Core" section shows the start time as "about 2 hours ago", the instance directory as "solr/", and the data directory as "/var/lib/cassandra/data/solr.data/wiki.solr/". The "Index" section shows the last modified time as "about 2 hours ago", the version as 532, the number of documents as 3579, the maximum document ID as 3579, and the deleted documents count as -. It also shows the current status as "no" and the indexing directory as "org.apache.lucene.store.NRTCachingDirectory: NRTCachingDirectory(org.apache.lucene.store.NIOFSDirectory@ /private/var/lib/cassandra/data/solr.data/wiki.solr/index lockFactory=org.apache.lucene.store.NativeFSLockFactory@e3f6d; maxCacheMB=48.0 maxMergeSizeMB=4.0)".

### Status of all Solr cores

To view the status of all Solr cores:

```
http://localhost:8983/solr/admin/cores?action=STATUS
```

For example, the status of the wiki.solr core looks like this:

```
{
 "defaultCoreName": "default.1371321667755813000",
 "initFailures": {},
 "status": {
 "wiki.solr": {
 "name": "wiki.solr",
 "isDefaultCore": false,
 "instanceDir": "solr/",
 "dataDir": "/var/lib/cassandra/data/solr.data/wiki.solr/",
 }
 }
}
```

```

"config":"solrconfig.xml",
"schema":"schema.xml",
"startTime":"2013-06-16T21:05:54.894Z",
"uptime":7212565,
"index": {
 "numDocs":3579,
 "maxDoc":3579,
 "deletedDocs":0,
 "version":532,
 "segmentCount":15,
 "current":false,
 "hasDeletions":false,
 "directory":"org.apache.lucene.store.
 NRTCachingDirectory:NRTCachingDirectory
 (org.apache.lucene.store.NIOFSDirectory
 @/private/var/lib/cassandra/data/solr.data/wiki.solr/index
lockFactory=
 org.apache.lucene.store.NativeFSLockFactory@e3f6d;
 maxCacheMB=48.0 maxMergeSizeMB=4.0)",
 "userData": {"commitTimeMSec":"1371416801053"}, ,
 "lastModified": "2013-06-16T21:06:41.053Z",
 "sizeInBytes":8429726,
 "size":"8.04 MB"},
 "indexing":false}}}

```

## Solr log messages

DSE Search logs Solr errors, warnings, debug, trace, and info messages in the Cassandra system log:

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

### Changing the Solr logging level

You can control the granularity of Solr log messages, and other log messages, in the Cassandra system.log file by configuring the logback.xml file.

To set log levels, specify one of these values:

- All - turn on all logging
- OFF - no logging
- FATAL - severe errors causing premature termination
- ERROR - other runtime errors or unexpected conditions
- WARN - use of deprecated APIs, poor use of API, near errors, and other undesirable or unexpected runtime situations
- DEBUG - detailed information on the flow through the system
- TRACE - more detailed than DEBUG
- INFO - highlight the progress of the application at a coarse-grained level

### Accessing the validation log

DSE Search stores validation errors that arise from non-indexable data that is sent from nodes other than DSE Search nodes:

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

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

## Adding and viewing index resources

DSE Search includes a REST API for viewing and adding resources that are associated with an index. You can look at the contents of the existing Solr resource by loading its URL in a web browser or using HTTP get. Retrieving and viewing resources returns the last uploaded resource, even if the resource is not the one currently in use.

Use this URL to post a file to Solr:

```
http://host:port/solr/resource/keyspace.table/filename.ext
```

If you upload a new schema, and then request the schema resource before reloading, Solr returns the new schema even though the [Solr core](#) continues to use the old schema.

Generally, you can post any resource that is 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.

## Checking indexing status

You can check the indexing status using dsetool, the Core Admin, or the logs.

If you use HTTP to post the files to a pre-existing table, DSE Search starts indexing the data. If you use HTTP to post the files to a non-existent column keyspace or table, DSE Search creates the keyspace and table, and then starts indexing the data. For example, you can change the `stopwords.txt` file, repost the schema, and the index updates.

### Checking the indexing status using dsetool

To check the indexing status with dsetool:

```
$ dsetool core_indexing_status keyspace.table
```

This command retrieves the dynamic indexing status (INDEXING, FINISHED, or FAILED) of the specified core in the local DSE Search node.

To retrieve the indexing status of all Solr cores:

```
$ dsetool core_indexing_status keyspace.table --all
```

To retrieve the indexing status only for a specific host, use:

```
$ dsetool -h IP_address core_indexing_status keyspace.table
```

### Checking the indexing status using the Core Admin

To check the indexing status, open the Solr Admin and click **Core Admin**.

The screenshot shows the Solr Admin interface for a core named 'wiki.solr'. The left sidebar has a 'Core Admin' section selected. The main panel shows the 'Core' section with fields: startTime (5 minutes ago), instanceDir (solr/), and dataDir (/var/lib/cassandra/data/solr.data/wiki.solr/). The 'Index' section shows: lastModified (4 minutes ago), version (189), numDocs (3579), maxDoc (3579), current (green checkmark), indexing (no), and directory (org.apache.lucene.store.NRTCachingDirectory: NRTCachingDirectory(org.apache.lucene.store.NIOFSDirectory@/private/var/lib/cassandra/data/solr.data/wiki.solr/index lockFactory=org.apache.lucene.store.NativeFSLockFactory@3ab6a5fb; maxCacheMB=48.0 maxMergeSizeMB=4.0)).

## Checking the indexing status using the logs

You can also check the logs to get the indexing status. For example, you can check information about the plugin initializer:

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

Or you can check the SecondaryIndexManager.java information:

```
INFO Thread-38 2013-08-26 19:31:28,498 SecondaryIndexManager.java (line 136)
 Submitting index build of wiki.solr for data in SSTableReader(path='/mnt/
 cassandra/data/wiki/solr/wiki-solr-ic-5-Data.db'), SSTableReader(path='/mnt/
 cassandra/data/wiki/solr/wiki-solr-ic-6-Data.db')

FINISH INDEXING -
INFO Thread-38 2013-08-26 19:38:10,701 SecondaryIndexManager.java (line 156)
 Index build of wiki.solr complete
```

## Fast repair

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

### To repair a subrange

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

- Run the [dsetool list\\_subranges command](#), using the approximate number of rows per subrange, the beginning of the partition range (token), and the end of the partition range of the node.

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

The output lists the subranges.

Start Token	End Token
Estimated Size	
113427455640312821154458202477256070485	
132425442795624521227151664615147681247	11264
132425442795624521227151664615147681247	
151409576048389227347257997936583470460	11136
151409576048389227347257997936583470460	0
11264	

- Use the output of the previous step as input to the [nodetool repair command](#).

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

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

## Excluding hosts from Solr-distributed queries

To exclude hosts from Solr-distributed queries, perform these steps on each node that you want to send queries to.

- Navigate to the `solr/conf` directory.

The default Solr conf location depends on the type of installation:

Installer-Services and Package installations	/usr/share/dse/resources/solr/conf
Installer-No Services and Tarball installations	<code>dse_install_location/resources/solr/conf</code>

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

## Expiring a DSE Search column

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

- [Configuring the high-performance update handler](#)

Configuring per-document TTL causes removal of the entire document. Configuring per-field TTL causes removal of the field only.

- [Using the Solr HTTP API](#)
- [Using CQL to set TTL](#)

If you configure TTL in the `solrconfig.xml` file, and then use the Solr HTTP API or CQL to configure a different TTL, the Solr HTTP API or CQL takes precedence.

### Configuring expiration using the Solr HTTP API

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

- Using the `ttl` parameter

Specifies per-document TTL. For example:

```
curl http://host:port/solr/mykeyspace.mytable/update?ttl=86400
```

- Using the `ttl.field name` parameter

Specifies per-field TTL. For example:

```
curl http://host:port/solr/mykeyspace.mytable/update?ttl.myfield=86400
```

### Configuring expiration using CQL

Using a CQL `INSERT` or `UPDATE` operation, you can set the TTL property. For example, continuing with the example of [using a collection set](#), insert a 5 minute (300 seconds) TTL property on the all columns of the Einstein data:

```
INSERT INTO mysolr (id, name, title, body)
VALUES ('126', 'Albert Einstein', 'Success', 'If A is success
in life, then A equals x plus y plus z. Work is x; y is play;
and z is keeping your mouth shut.')
USING TTL 300;
```

After a few seconds, check the remaining time-to-live on the data:

```
SELECT TTL (name) FROM mykeyspace.mysolr WHERE id = '126';
```

The output after 9 seconds expire is:

```
ttl(name)

291
```

After the remaining time has passed, the data expires, and querying the data returns no results. If you refresh the Solr Admin console, the number of documents is 3 instead of 4.

### Configuring expiration scope

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

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

1. Configure the high-performance update handler section of the `solrconfig.xml`.

- For per-document TTL, add these lines to the high-performance updateHandler section:

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

- ```
</lst>
• For per-field TTL, add these lines to the updateHandler section:
```

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

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

Managing expired columns

After Cassandra expires a column using the time-to-live (TTL) mechanism, DSE Search can still find the expired column. The column data remains in the index until one of the following conditions is met:

- Re-indexing occurs due to a DSE Search ttl rebuild timeout.
Set the [ttl rebuild timeout properties](#) in the [dse.yaml](#) file.
- All columns in a row expire due to the Cassandra [time-to-live \(TTL\) mechanism](#), triggering removal of the entire row/Solr document from the index.

Setting the ttl rebuild timeout properties is the recommended method for managing expired columns.

Changing the HTTP interface to Apache JServe Protocol

In addition to the widely-used HTTP interface, you can configure DSE search to use the AJP (Apache JServe Protocol). AJP is an optimized, binary version of HTTP that facilitates Tomcat communication with an Apache web server using mod_jk. This capability is typically used where https serves a web application and DSE Search powers the backend.

By default the AJP connector is disabled. To enable the AJP connector, uncomment the connector configuration in the Tomcat server.xml file. For example, remove the comments as follows:

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
Connector port="8009" protocol="AJP/1.3" redirectPort="8443"
```

Backing up Solr indexes

Use these steps as a practical starting point to create backups for Solr indexes. These steps apply when the backups are intended to restore a cluster with the same token layout, and the backups can be created in a rolling fashion.

Procedure

For each node:

1. Drain the node to ensure that the Solr cores are in sync with their backing Cassandra tables. This command forces a memtable flush that forces a Solr hard commit:

```
$ nodetool drain
```

2. [Shut down](#) the node.
3. Manually back up your data directories. The default location for Solr index files is /var/lib/cassandra/data/solr.data.
4. [Restart](#) the node.

Performance tuning

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

- [Using Cassandra table compression](#)
- [Configuring the Search Handler](#)
- [Configuring the update handler and autoSoftCommit](#)
- [Changing the stack size and memtable space](#)
- [Managing the data consistency level](#)
- [Configuring the available indexing threads](#)
- [Adding replicas to increase read performance](#)
- [Changing the replication factor](#)
- [Configuring re-indexing and repair](#)
- [Performance impact when using deep paging with CQL Solr queries](#)

Using metrics MBeans

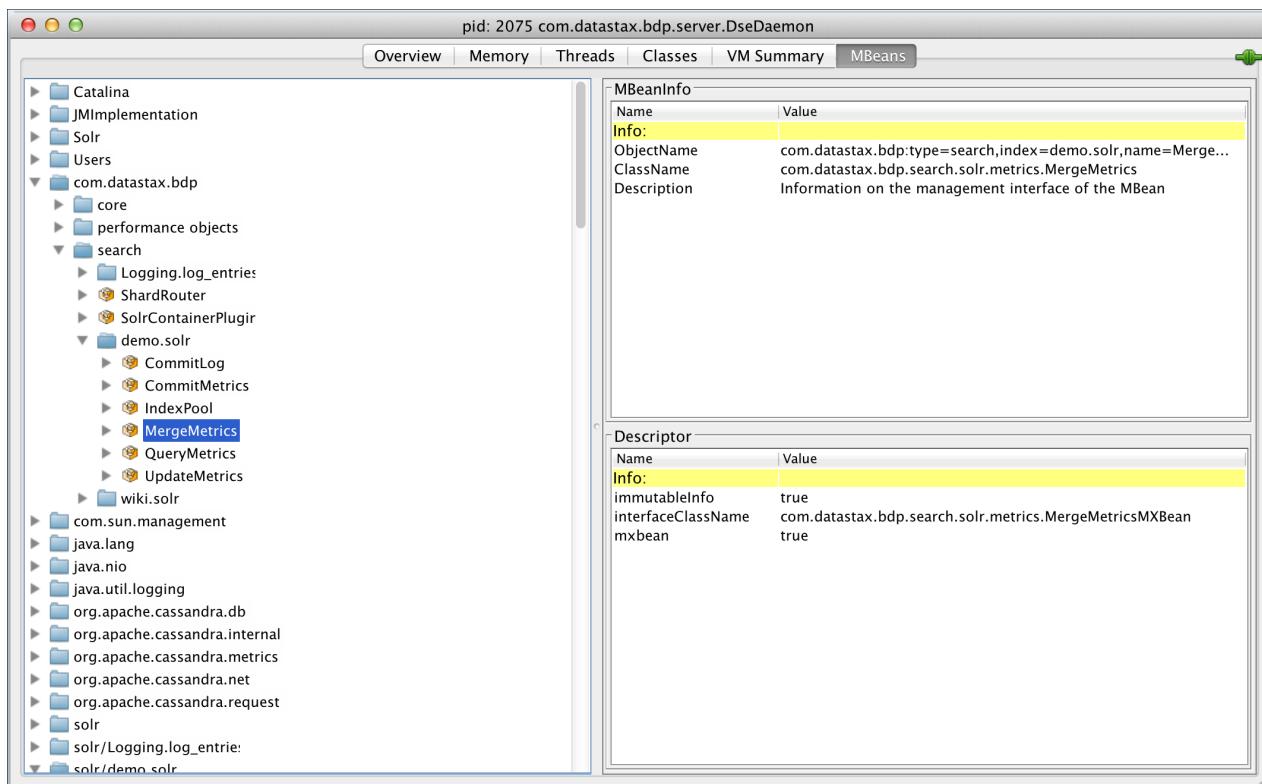
DataStax Enterprise provides commit, merge, query, update, and index pool metrics MBeans for troubleshooting, and for tuning performance and consistency issues.

The following paths identify the MBeans:

```
type=search,index=core,name=CommitMetrics  
type=search,index=core,name=MergeMetrics  
type=search,index=core,name=QueryMetrics  
type=search,index=core,name=UpdateMetrics  
type=search,index=core,name=IndexPool
```

where *core* is the name of the [Solr core](#) that is referenced by the metrics.

For example, the following figure shows the com.datastax.bdp merge metrics MBean in JConsole. The demo.solr core under search is expanded.



Using MBeans to evaluate performance

Example steps to use the MBeans on Linux to obtain information about performance while running the DataStax Solr stress test demo.

Procedure

1. Start a single DSE Search node.
2. Start JConsole using the PID of the DSE Search node:

```
sudo jconsole 1284
```

3. In JConsole, connect to a DSE Search node. For example, connect to the Local Process com.datastax.bdp.DseModule.
4. Change to the `demos` directory.

The default location of the `demos` directory depends on the type of installation:

Installer-Services and Package installations	/usr/share/dse/demos
Installer-No Services and Tarball installations	<code>install_location/demos</code>

5. Make `demos/solr_stress` your current directory.
6. Execute this script to create the schema:

```
./1-add-schema.sh [options]
```

where the script options are:

CQL table creation options

```
--ssl use SSL for Cassandra table creation over cqlsh
```

Solr HTTP options

```
-e CA_CERT_FILE use HTTPS with the provided CA certificate
-E CLIENT_CERT_FILE use the provided client certificate
-h HOST hostname or IP for Solr HTTP requests
-a enable Kerberos
-u USERNAME Kerberos username
-p PASSWORD Kerberos password
```

The script creates the Cassandra schema and posts the `solrconfig.xml` and `schema.xml` files to these locations:

- `http://localhost:8983/solr/resource/demo.solr/solrconfig.xml`
- `http://localhost:8983/solr/resource/demo.solr/schema.xml`

The script then creates the core/index by posting to the following location:

- `http://localhost:8983/solr/admin/cores?action=CREATE&name=demo.solr`

You can override the script defaults by specifying command line parameters:

```
-x schemafile.xml -t tableCreationFile.cql -r solrCofgFile.xml -k solrCore
```

7. Execute this script to run the benchmark:

```
./run-benchmark.sh [--clients=clients_count] [--loops=loops_count] [--fetch=fetch_size] [--solrCore=solr_core] [--testData=test_data_file]
[--url=url1,url2,url3,...] [--qps=qps] [--stats=true|false] [--seed=seed_value]
```

where the script options are:

--clients

The number of client threads to create.

Default: 1

--loops

The number of times the commands list gets executed if running sequentially or the number of commands to run if running randomly.

Default: 1

--fetch

Fetch size for CQL pagination (disabled by default). Only the first page is retrieved.

--solrCore

Solr core name to run the benchmark against.

--testData

Name of the file that contains the test data.

--seed

Value to set the random generator seed to.

--qps

Maximum number of queries per second allowed.

--stats

Specifies whether to gather statics during runtime and create a csv file with the recorded values.

Default: false

--url

A comma delimited list of servers to run the benchmark against. For example: --url=http://localhost:8983,http://192.168.10.45:8983,http://192.168.10.46:8983

Default: http://localhost:8983

The demo creates a Solr core named demo.solr and indexes 50,000 documents.

Example CQL commands:

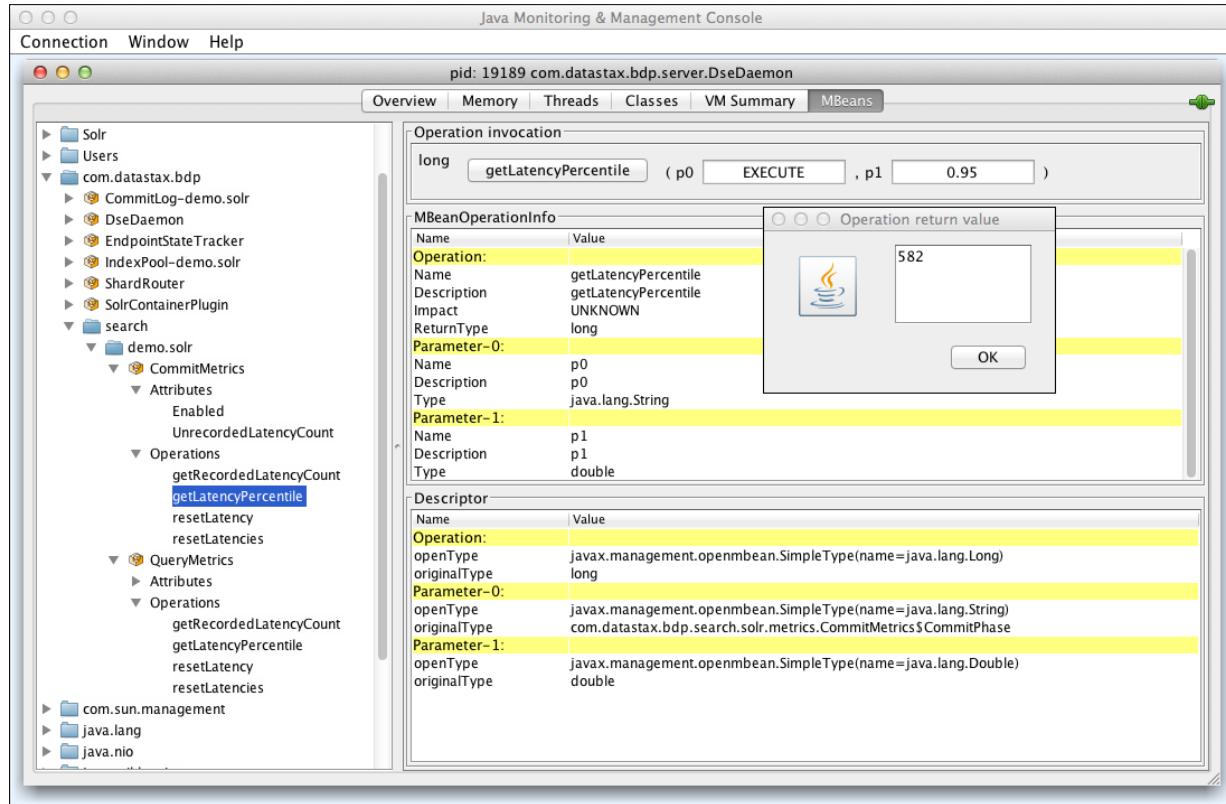
```
./run-benchmark.sh --url=http://localhost:8983 --testData=resources/testCqlQuery.txt --solrCore=demo.solr
```

```
./run-benchmark.sh --url=http://localhost:8983 --testData=resources/testCqlWrite.txt --solrCore=demo.solr
```

See /demos/solr_stress/README.txt for execution modes and sample script commands.

8. In JConsole, expand **com.datastax.bdp > search > demo.solr** to view the MBeans. The CommitMetrics and QueryMetrics MBeans are present.
9. In JConsole, in **Search > demo.solr > CommitMetrics > Operations > getLatencyPercentile**, type EXECUTE in the p0 text entry box and 0.95 in the p1 text entry box. Click the **getLatencyPercentile** button.

The Operation return value, 582 microseconds, appears:



Commit metrics MBean

The commit metrics MBean is useful for troubleshooting index performance as well as data consistency issues caused by asynchronous commits between different index replicas. Using this MBean is also useful for fine-tuning indexing **back pressure**. The commit metrics MBean records the amount of time that is spent to execute two **main phases** of a commit operation on the index.

Main operational phases

The main phases of a commit operation on the index are:

FLUSH

Comprising the time spent by flushing the async indexing queue.

EXECUTE

Comprising the time spent by actually executing the commit on the index.

Commit metrics MBean operations use the FLUSH and EXECUTE commit phase names.

Commit metrics MBean set operations

The commit metrics MBean measures latency in microseconds. You can set these commit metrics MBean operations:

- `setEnabled(boolean enabled)`
Enables/disables metrics recording. Enabled by default.
- `resetLatency(String phase)`
Resets latency metrics for the given commit phase.
- `resetLatencies()`
Resets all latency metrics.

Commit metrics MBean get operations

The commit metrics MBean measures latency in microseconds. You can get these commit metrics Mbean operations:

- `isEnabled()`
Checks that metrics recording is enabled.
- `getLatencyPercentile(String phase, double percentile)`
Gets a commit latency percentile by its phase.
- `getRecordedLatencyCount(String phase)`
Gets the total count of recorded latency metrics by its commit phase.
- `getUnrecordedLatencyCount()`
Gets the total count of unrecorded latency values due to exceeding the maximum tracked latency, which is 10 minutes.

Merge metrics MBean

The merge metrics MBean tracks the time that Solr/Lucene spends on merging segments that accumulate on disk. Segments are files that store new documents and are a self-contained index. When data is deleted, Lucene does not remove it, but instead marks documents as deleted. For example, during the merging process, Lucene copies the data from 100 segment files into a single new file. Documents that are marked deleted are not included in the new segment files. Next, Lucene removes the 100 old segment files, and the single new file holds the index on disk.

After segments are written to disk, they are immutable.

In a high throughput environment, a single segment file is rare. Typically, there are several files and Lucene runs the merge metric operation concurrently with inserts and updates of the data using a merge policy and merge schedule.

Merge operations are costly and can impact the performance of Solr queries. A huge merge operation can cause a sudden increase in query execution time.

Main operational phases

The main phases of a merge operation on the index are:

INIT

How long it takes to initialize the merge process.

EXECUTE

How long it takes to execute the merge process.

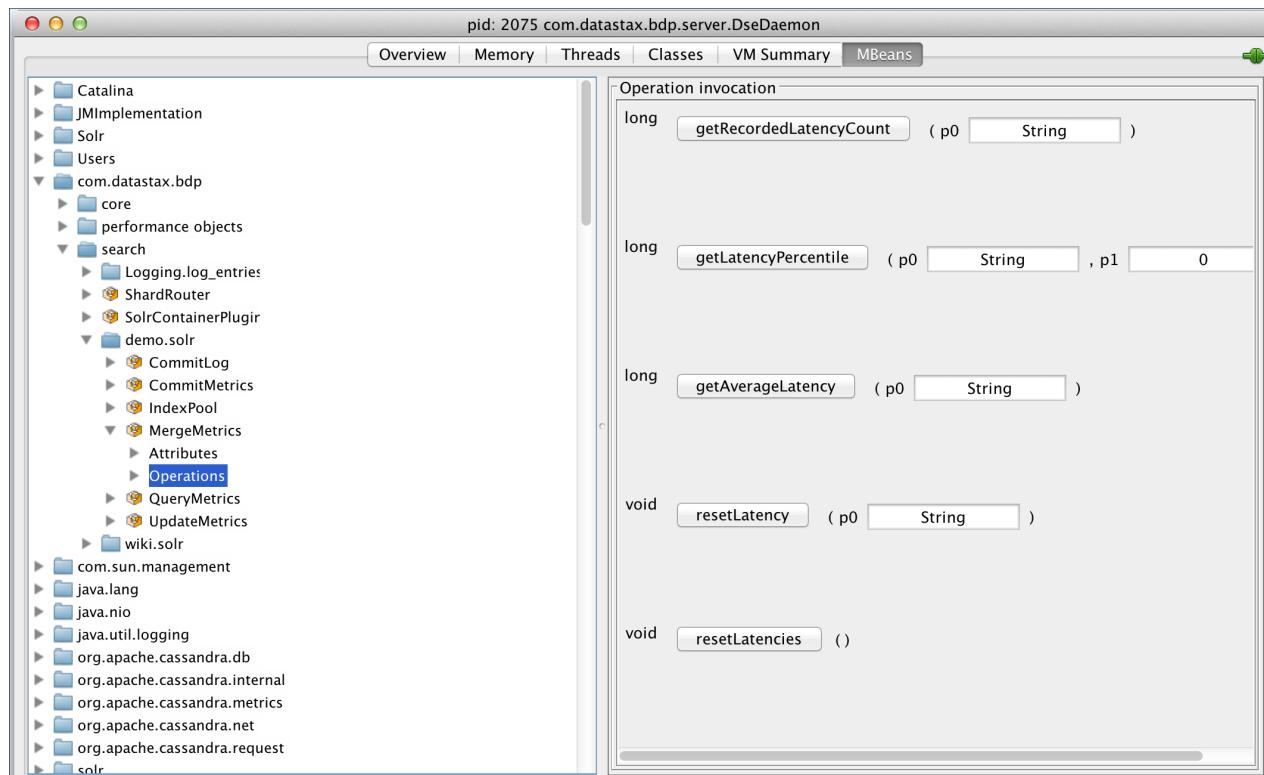
WARM

How long it takes to warm up segments to speed up cold queries.

WARM time is part of EXECUTE time: EXECUTE time = WARM time + other operations. For example, if the EXECUTE phase is 340 ms, and the WARM phase is 120 ms, then other operations account for the remainder, 220 ms.

The merge metrics MBean operations, as shown in the following figure, are:

- getRecordedLatencyCount
- getLatencyPercentile
- getAverageLatency
- resetLatency
- resetLatencies



To get merge metrics, insert one of the phases of the merge operation and select a phase, for example EXECUTE.

Query metrics MBean

The query metrics MBean is useful for troubleshooting query performance, tuning the Solr configuration, such as the schema and caches, and tuning server resources, such as the JVM heap. The query metrics

MBean records the amount of time spent to execute several **main phases** of a distributed query on the index.

The query metrics MBean measures latency in microseconds. Metrics can be grouped by query, by providing an additional query.name parameter. For example, assuming you are using a Solr core named demo.solr and have indexed a field named type, this URL provides the additional query.name parameter:

```
http://localhost:8983/solr/demo.solr/select/?q=type:1&query.name=myquery
```

All metrics collected under a given query name are recorded and retrieved separately. If a query name is not provided, all metrics are recorded together.

Main operational phases

The main phases of a distributed query operation are:

COORDINATE

Comprises the total amount of time spent by the coordinator node to distribute the query and gather/process results from shards. This value is computed only on query coordinator nodes.

EXECUTE

Comprises the time spent by a single shard to execute the actual index query. This value is computed on the local node executing the shard query.

RETRIEVE

Comprises the time spent by a single shard to retrieve the actual data from Cassandra. This value will be computed on the local node hosting the requested data.

Query metrics MBean set operations

Operations are:

- `setEnabled(boolean enabled)`
Enables/disables metrics recording. Enabled by default.
- `isEnabled()`
Checks if metrics recording is enabled.
- `getLatencyPercentile(String phase, String query, double percentile)`
Gets a query latency percentile by its query name, which is optional and can be null, and phase.
- `getRecordedLatencyCount(String phase, String query)`
Gets the total count of recorded latency metrics by its query name, which is optional and can be null, and phase.
- `getUnrecordedLatencyCount()`
Gets the total count of unrecorded latency values due to exceeding the maximum tracked latency, which is 10 minutes.
- `resetLatency(String query)`
Resets latency metrics for the given query name, which is optional and can be null.
- `resetLatencies()`
Resets all latency metrics.

Update metrics MBean

The update metrics MBean is identified by the following path:

`type=search,index=core,name=UpdateMetrics`, where core is the Solr core name that the metrics reference.

This MBean records the amount of time spent to execute an index update, split by the following main phases:

WRITE

Comprising the time spent to convert the Solr document and write it into Cassandra (only available when indexing via the SOLRj HTTP APIs).

QUEUE

Comprising the time spent by the index update task into the index pool.

PREPARE

Comprising the time spent preparing the actual index update.

EXECUTE

Comprising the time spent to actually execute the index update on Lucene.

The update metrics MBean can be useful to guide tuning of all factors affecting indexing performance, such as [back pressure](#), indexing threads, RAM buffer size and merge factor.

Set

You can get the following metrics from the Mbean:

The following MBean operations are provided:

- `setEnabled(boolean enabled)`
Enables/disables metrics recording (enabled by default).
- `isEnabled()`
Checks if metrics recording is enabled.
- `getLatencyPercentile(String phase, double percentile)`
Gets a commit latency percentile by its phase.
- `getRecordedLatencyCount(String phase)`
Gets the total count of recorded latency metrics by its phase.
- `getUnrecordedLatencyCount()`
Gets the total count of unrecorded latency values, because exceeding the max tracked latency.
- `resetLatency(String phase)`
Resets latency metrics for the given phase.
- `resetLatencies()`
Resets all latency metrics.

The maximum tracked latency is 10 minutes. Latency values are in microseconds.

IndexPool MBean

The IndexPool MBean exposes metrics around the progress of indexing tasks as they move through the pipeline, and provides a mechanism to tweak the flushing, concurrency, and back-pressure behavior of a core indexing thread pool.

The index pool MBean is useful for controlling task submission and flush with these properties:

Configurable concurrency

The maximum number of concurrent workers is predefined at construction time. The actual concurrency can be dynamically configured between 1 (synchronous execution) and the given max concurrency.

Flow control via back pressure

To reduce memory consumption in case of fast producers, back pressure throttles incoming tasks based on a configurable queue threshold and max pause per task. The implementation is re-entrant if a worker submits a task on the pool, the task is not subjected to any pause.

Path

The index pool MBean is identified by the following path:

`type=search, index=core, name=IndexPool`, where core is the Solr core name that the metrics reference.

IndexPool MBean attributes that you can modify

The attributes are effective only until the node is restarted. To make the change permanent, you must change the corresponding option in `dse.yaml`.

- `FlushMaxTime`

The maximum time, in milliseconds, to wait before flushing asynchronous index updates, which occurs at Solr commit time or at Cassandra flush time. In `dse.yaml` (in minutes): [flush_max_time_per_core](#).

- `BackPressureThreshold`

The back pressure threshold is the target total number of queued asynchronous indexing requests per core; the back pressure mechanism will throttle incoming requests to keep the queue size as close to the threshold as possible. In `dse.yaml`: [back_pressure_threshold_per_core](#).

- `Concurrency`

The maximum number of concurrent asynchronous indexing threads. In `dse.yaml`: [max_solr_concurrency_per_core](#).

IndexPool MBean view-only attributes

You can get the following MBean operations:

- `MaxConcurrency`

Get the predefined max concurrency level.

- `QueueSize`

Get the current size of each processing queues.

- `TotalQueueSize`

Get the total size of all processing queues.

- `TaskProcessingTimeNanos`

Get the last processing time for all workers. Could be 0 in case the clock resolution is too coarse.

- `ProcessedTasks`

Get the total number of processed tasks for all workers.

- `BackPressurePauseNanos`

Get the average back pressure pause.

- `IncomingRate`

Get the 1-minute rate of ingested tasks per second.

- `OutgoingRate`

Get the 1-minute rate of processed tasks per second.

- `Throughput`

The 1-minute rate of work throughput per second.

Using table compression

Search nodes typically engage in read-dominated tasks, so maximizing storage capacity of nodes, reducing the volume of data on disk, and limiting disk I/O can improve performance. In Cassandra 1.0 and later, you can [configure data compression](#) on a per-table basis to optimize performance of read-dominated tasks.

Configuration affects the compression algorithm for compressing SSTable files. For read-heavy workloads, such as those carried by Enterprise Search, LZ4 compression is recommended.

Compression using the LZ4 compressor is enabled by default when you create a table. You can change [compression options](#) using CQL. Developers can also implement custom compression classes using the org.apache.cassandra.io.compress.ICompressor interface. You can configure the compression chunk size for read/write access patterns and the average size of rows in the table.

Configuring the update handler and autoSoftCommit

You need to configure the `solrconfig.xml` to use near real-time capabilities in Solr by setting the default high-performance update handler flag.

For example, the Solr configuration file for the Wikipedia demo sets this flag as follows and uncomments the `autoSoftCommit` element:

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

. . .

<autoSoftCommit>
    <maxTime>3000</maxTime>
</autoSoftCommit>
</updateHandler>
```

The `autoCommit` element is removed to prevent hard commits that hit the disk and flush the cache. The soft commit forces uncommitted documents into internal memory. When data is committed, it is immediately available after the commit.

For [live indexing](#) only, the `autoSoftCommit/maxTime` saturates at 3000 ms, any higher value is overridden to limit the maximum time to 3000ms. Live indexing is on when `<rt>true</rt>` is in `solrconfig.xml`.

The `autoSoftCommit` element uses the `maxTime` update handler attribute. The update handler attributes enable near real-time performance and trigger a soft commit of data automatically, so checking synchronization of data to disk is not necessary. This table describes both update handler options.

Attribute	Default	Description
<code>maxDocs</code>	No default	Maximum number of documents to add since the last soft commit before automatically triggering a new soft commit.
<code>maxTime</code>	3000	Maximum expired time in milliseconds between the addition of a document and a new, automatically triggered soft commit.

For more information about the update handler and modifying `solrconfig.xml`, see the [Solr documentation](#).

Configuring update performance

If updates take too long and the value of `autoSoftCommit` is higher than the default (1000ms), reset `autoSoftCommit` to the default value. This setting is in `solrconfig.xml`.

Parallelizing large Cassandra row reads

For performance, you can configure DSE Search to parallelize the retrieval of a large number of rows. First, configure the [queryResponseWriter](#) in the `solrconfig.xml` as follows:

```
<queryResponseWriter name="javabin" class="solr.BinaryResponseWriter">
  <str name="resolverFactory">com.datastax.bdp.search.solr.response.ParallelRowResolver
$Factory</str>
</queryResponseWriter>
```

By default, the parallel row resolver uses up to x threads to execute parallel reads, where x is the number of CPUs. Each thread sequentially reads a batch of rows equal to the total requested rows divided by the number of CPUs:

Rows read = Total requested rows / Number of CPUs

You can change the batch size per request, by specifying the `cassandra.readBatchSize` HTTP request parameter. Smaller batches use more parallelism, while larger batches use less.

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 `-Xss256k` 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_heap_space_in_mb](#) and [memtable_offheap_space_in_mb](#) properties in the `cassandra.yaml` file.

Managing the consistency level of write in Cassandra on the client side

You can specify the consistency level of write in Cassandra on the client side. 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 with the `cl` HTTP parameter that you can send with Solr data to tune consistency. The format of the URL is:

```
$ curl "http://host:port/solr/keyspace.table/update?cl=ONE"
```

The `cl` parameter specifies the consistency level of the write in Cassandra on the client side. The default consistency level for write is QUORUM. To globally change the default on the server side, use the Cassandra drivers and client libraries.

Note: For DSE Search read queries, only CL=ONE is supported.

Setting the consistency level of the write in Cassandra on the client side using SolrJ

SolrJ does not allow setting the consistency level parameter using a Solr update request. To set the consistency level using SolrJ, use this command instead:

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

See the [Data Consistency in DSE Search](#) blog post.

Configuring multi-threaded indexing threads

DSE Search provides multi-threaded asynchronous indexing with a back pressure mechanism to avoid saturating available memory and to maintain stable performance. Multi-threaded indexing improves performance on machines that have multiple CPU cores. All index updates are internally dispatched to a per CPU core indexing thread pool and executed asynchronously.

This multi-threaded indexing implementation allows for greater concurrency and parallelism, but as a consequence, index requests return a response before the indexing operation is actually executed.

DSE Search also provides advanced JMX-based, configurability, and visibility through the IndexPool JMX MBean.

Procedure

1. In the `dse.yaml` file, define the number of indexing threads per Solr core with the `max_solr_concurrency_per_core` option. To achieve optimal performance, assign this value to number of available CPU cores divided by the number of Solr cores. For example, with 12 CPU cores and 3 Solr cores, the suggested value is 4.
If set to 1, DSE Search uses the legacy synchronous indexing implementation.
2. In the `dse.yaml` file, define the number of buffered asynchronous index updates per Solr core before the back-pressure is activated with the `back_pressure_threshold_per_core` option. The default value is 1000 times the number of available CPU cores.
When the back pressure threshold is reached, new incoming requests are throttled up to a maximum of 80% of the `write_request_timeout_in_ms` pause per request. The `write_request_timeout_in_ms` in the `cassandra.yaml` file defines how long the coordinator should wait for writes to complete.
3. To monitor the indexing performance of each Solr core, use these JMX attributes in the `com.datastax.bdp:type=search, index=solr_core, name=IndexPool mbean:`
 - `TotalQueueSize`: Total number of buffered asynchronous index updates.
 - `Throughput`: Current one minute rate of executed index updates per second.
 - `BackPressurePauseNanos`: Current one minute average of applied back pressure pause per request.

Configuring re-indexing

When running the RELOAD command using the `reindex` or `deleteAll` options, a long delay might indicate that tuning is needed. Tune the performance of re-indexing and index rebuilding by making a few changes in the `solrconfig.xml` file.

Procedure

1. Increase the size of the RAM buffer, which is set to 512 MB by default. For example, increase to 2000.

```
<indexConfig>
  <useCompoundFile>false</useCompoundFile>
  <ramBufferSizeMB>2000</ramBufferSizeMB>
  <mergeFactor>10</mergeFactor>
  . . .

```

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

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

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

Note: A higher value for autoSoftCommit, such as 10000, is suitable when [live indexing](#) is not enabled. DataStax recommends using the default value when live indexing is enabled.

Tuning index size and range query speed

In DataStax Enterprise, you can trade off Solr index size for range query speed and vice versa. You make this tradeoff to suit a particular use case and on a core-by-core basis by setting up the precision step of two special token field types that are used by DataStax Enterprise.

Use extreme care when performing this tuning. This advanced tuning feature is recommended for use in rare cases. In most cases, using the default values is the best. To perform this tuning, you change the precision step of one or both DataStax Enterprise internal field types:

- `token_long`
Used for filtering over token fields during query routing.
- `ttl_long`
Used for searching for expiring documents.

Change the precision step as follows:

1. In the `fieldType` definition, set the `class` attribute of `token_long` and `ttl_long` to `solr.TrieLongField`.
2. Set the `precisionStep` attribute from the default 8 to another number. Choose this number based on an understanding of its impact. Usually, a smaller precision step increases the index size and range query speed, while a larger precision step reduces index size, but potentially reduces range query speed.

The following snippet of the `schema.xml` shows an example of the required configuration of both field types:

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema name="test" version="1.0">
  <types>
    . . .
    <fieldType name="token_long" class="solr.TrieLongField"
      precisionStep="16" />
    <fieldType name="ttl_long" class="solr.TrieLongField"
      precisionStep="16" />
    . . .
  </types>
  <fields>
    . . .
  </fields>
</schema>
```

DataStax Enterprise ignores one or both of these field type definitions and uses the default precision step if you make any of these mistakes:

- The field type is defined using a name other than `token_long` or `ttl_long`.
- The class is something other than `solr.TrieLongField`.
- The precision step value is not a number. DataStax Enterprise logs a warning.

The definition of a fieldType alone sets up the special field. You do not need to use `token_long` or `ttl_long` types as fields in the `<fields>` tag.

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 strategy class, the names of your datacenters, and the number of replicas. For example, you can add replicas using the NetworkTopologyStrategy replica placement strategy. To configure this strategy, use CQL.

Procedure

For example, if you are using a PropertyFileSnitch, perform these steps:

1. Check the datacenter names of your nodes using the nodetool command.

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

The datacenter names, DC1 and DC2 in this example, must match the datacenter 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.

Set the number of replicas in datacenters, one replica in datacenter 1 and three in datacenter 2. For more information about adding replicas, see [Choosing Keyspace Replication Options](#).

Changing the replication factor for a Solr keyspace

This example assumes the `solrconfig.xml` and `schema.xml` files have already been posted using `mykeyspace.mysolr` in the URL, which creates a keyspace named `mykeyspace` that has a default replication factor of 1. You want three replicas of the keyspace in the cluster, so you need to change the keyspace replication factor.

Procedure

To change the keyspace replication factor

1. Check the name of the datacenter of the Solr/Search nodes.

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

The output tells you that the name of the datacenter for your node is, for example, `datacenter1`.

2. Use CQL to [change the replication factor](#) of the keyspace from 1 to 3.

```
ALTER KEYSPACE mykeyspace WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy', 'datacenter1' : 3 };
```

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

Managing caching

The DSENRTCachingDirectoryFactory is deprecated. If you use DSENRTCachingDirectoryFactory or the NRTCachingDirectoryFactory, modify the `solrconfig.xml` to use the StandardDirectoryFactory. For example, change the `directoryFactory` element in the `solrconfig.xml` file as follows:

```
<directoryFactory class="solr.StandardDirectoryFactory"
  name="DirectoryFactory"/>
```

Capacity planning

Using DSE Search is memory-intensive. Solr rereads the entire row when updating indexes, and can impose a significant performance hit on spinning disks. Use solid-state drives (SSD) for applications that have very aggressive insert and update requirements.

This capacity planning discovery process helps you develop a plan for having sufficient memory resources to meet the operational requirements.

Overview

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

Capacity planning requires a significant effort by operations personnel to achieve these results:

- Optimal heap size per node.
- Estimate of the number of nodes that are required for your application.
- Increase the replication factor to support more queries per second.

Note: The [Pre-flight tool](#) can detect and fix many invalid or suboptimal configuration settings.

Prerequisites

A node with:

- The amount of RAM that is determined during capacity planning
- SSD or a spinning disk with its own dedicated disk. A dedicated SSD is recommended, but is not required.

Input data:

- N documents indexed on a single test node
- A complete set of sample queries to be executed
- The maximum number of documents the system will support

Procedure

1. Create the `schema.xml` and `solrconfig.xml` files.
2. Start a node.
3. Add N docs.
4. Run a range of queries that simulate a production environment.

5. View the status of [the field cache memory](#) to discover the memory usage.
6. View the size of the index (on disk) included in the [status information](#) about the Solr core.
7. Based on the server's system IO cache available, set a maximum index size per server.
8. Based on the memory usage, set a maximum heap size required per server.
 - For JVM memory to provide the required performance and memory capacity, DataStax recommends a heap size of 14 GB or larger.
 - For live indexing, DataStax recommends a heap size of at least 20 GB for use with Java 1.8 and G1GC. A larger heap size allows you to allocate more RAM buffer size, which contributes to faster live (RT) indexing. Enable live indexing on only one Solr core per cluster.
9. Calculate the maximum number of documents per node based on steps 6 and 7.

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

Managing the field cache memory

The Solr field cache caches values for all indexed documents, which if left unchecked, can result in out-of-memory errors. For example, when performing faceted queries using multi-valued fields the multiValue fields are multi-segmented (as opposed to single segmented single-valued fields), resulting in an inefficient near real time (NRT) performance. You can use densely packed DocValue field types and per-segment docsets. Facet queries will be per-segment, which improves real-time search performance problems.

To ensure that the JVM heap can accommodate the cache, monitor the status of the field cache and take advantage of the Solr option for storing the cache on disk or on the heap. To view the status of the field cache memory usage, append &memory=true to the URL used to view the status of Solr cores. For example, to view the field cache memory usage of the DSE Search quick start example after running a few facet queries, use this URL:

```
http://localhost:8983/solr/admin/cores?action=STATUS&memory=true
```

Example 1

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

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

. . .

"memory": {
  "unInvertedFields": {
    "totalSize": 0,
    "totalReadableSize": "0 bytes",
    "multiSegment": {
      "multiSegment": "StandardDirectoryReader(segments_3:532:nrt _6p(4.6):C3193 _71(4.6):C161 _6i(4.6):C15 _6n(4.6):C21 _6e(4.6):C16 _6k(4.6):C19 _6t(4.6):C17 _6g(4.6):C10 _77(4.6):C12 _6v(4.6):C9 _7c(4.6):C66 _72(4.6):C14 _6x(4.6):C7 _6y(4.6):C7 _6w(4.6):C12)",
      "fieldCache": {
        "entriesCount": 0,
        "totalSize": 0,
        "totalReadableSize": "0 bytes"
      }
    },
    "segments": {
      "_6p": {
        "segment": "_6p",
        "docValues": {
          " . . .

```

```

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

```

Example 2

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

```

. . .

"fieldCache": {
    "entriesCount":1,
    "id": {
        "cacheType": "org.apache.lucene.index.SortedDocValues",
        "size":260984,
        "readableSize": "254.9 KB"}},
"totalSize":260984,
"totalReadableSize": "254.9 KB"}},
"segments": {

. . .

"fieldCache": {
    "entriesCount":2,
    "age": {
        "cacheType": "int",
        "size":3832,
        "readableSize": "3.7 KB"}},
    "id": {
        "cacheType": "int",
        "size":3832,
        "readableSize": "3.7 KB"}},
"totalSize":59232,
"totalReadableSize": "57.8 KB"}},
"totalSize":524648,
"totalReadableSize": "512.4 KB"}},
"totalMemSize":524648,
"totalReadableMemSize": "512.4 KB"}}

```

Using the field cache

In Lucene-Solr 4.5 and later, docValues are mostly disk-based to avoid the requirement for large heap allocations in Solr. If you use the field cache in sort, stats, and other queries, make those fields [docValues](#).

Tuning near-real-time (NRT) indexing

The default mergeScheduler settings are not appropriate for DSE Search near real time (NRT) indexing production use on a typical size server.

Lucene merge scheduling and lack of parallelism might cause periods of 0 throughput.

Procedure

The `solrconfig.xml` resource file is the primary configuration file for configuring Solr for use with DSE Search.

1. If NRT indexing throughput is not performant, change or add the `mergeScheduler` settings in the `solrconfig.xml` file.
2. DataStax recommends using this formula to define the settings:
 - `maxThreadCount` = to the number of CPU cores divided by 2
 - `maxMergeCount` = `maxThreadCount * 2`
3. Adjust the values as appropriate to your environment.

For example, for 24 CPU cores:

```
<indexConfig>
  ...
    <mergeScheduler
      class="org.apache.lucene.index.ConcurrentMergeScheduler">
        <int name="maxThreadCount">12</int>
        <int name="maxMergeCount">24</int>
    </mergeScheduler>
  ...

```

4. Restart the node for the changes to be recognized.

Update request processor and field transformer

DataStax Enterprise supports the [classic Solr update request processor](#) (URP), a custom URP chain for processing requests and transforming data, and a field input/output transformer (FIT) API. The DataStax Enterprise custom URP implementation provides similar functionality to the Solr URP chain, but appears as a plugin to Solr. The classic URP is invoked when updating a document using HTTP, the custom URP when updating a table using Cassandra. If both classic and custom URPs are configured, the classic version is executed first. The custom URP chain and the FIT API work with CQL and HTTP updates.

A field input/output transformer, an alternative for handling update requests, is executed later than a URP at indexing time. For more information, see the DataStax Developer Blog post [An Introduction to DSE Field Transformers](#).

Examples are provided for using the custom URP and the field input/output transformer API.

Custom URP example

DSE Search includes the released version of a plugin API for Solr updates and a plugin to the `CassandraDocumentReader`. The plugin API transforms data from the secondary indexing API before data is submitted to Solr. The plugin to the `CassandraDocumentReader` transforms the results data from Cassandra to Solr.

Using the API, applications can tweak a Solr Document before it is mapped and indexed according to the `schema.xml` file. The API is a counterpart to the [input/output transformer support](#) in Solr.

The field input transformer (FIT) requires a trailing Z for date field values.

Procedure

To use the API:

1. Configure the custom URP in the solrconfig.xml.

```
<dseUpdateRequestProcessorChain name="dse">
<processor
  class="com.datastax.bdp.search.solr.functional.DSEUpdateRequestProcessorFactoryExamp
</processor>
</dseUpdateRequestProcessorChain>
```

2. Write a class to use the custom URP that extends the Solr [UpdateRequestProcessor](#). For example:

```
package com.datastax.bdp.search.solr.functional;

import
  com.datastax.bdp.search.solr.handler.update.CassandraAddUpdateCommand;
import
  com.datastax.bdp.search.solr.handler.update.CassandraCommitUpdateCommand;
import
  com.datastax.bdp.search.solr.handler.update.CassandraDeleteUpdateCommand;
import java.io.IOException;

import org.apache.solr.update.AddUpdateCommand;
import org.apache.solr.update.CommitUpdateCommand;
import org.apache.solr.update.DeleteUpdateCommand;
import org.apache.solr.update.MergeIndexesCommand;
import org.apache.solr.update.processor.UpdateRequestProcessor;

public class TestUpdateRequestProcessor extends UpdateRequestProcessor
{
    public boolean cprocessAdd = false;
    public boolean processAdd = false;

    public boolean cprocessDelete = false;
    public boolean processDelete = false;

    public boolean cprocessCommit = false;
    public boolean processCommit = false;

    public TestUpdateRequestProcessor(UpdateRequestProcessor next)
    {
        super(next);
    }

    public void processAdd(AddUpdateCommand cmd) throws IOException
    {
        if (cmd instanceof CassandraAddUpdateCommand)
        {
            cprocessAdd = true;
        }
        else
        {
            processAdd = true;
        }
        super.processAdd(cmd);
    }

    public void processDelete(DeleteUpdateCommand cmd) throws IOException
    {
        if (cmd instanceof CassandraDeleteUpdateCommand)
        {
            cprocessDelete = true;
        }
        else
        {
```

```

        processDelete = true;
    }
    super.processDelete(cmd);
}

public void processMergeIndexes(MergeIndexesCommand cmd) throws
IOException
{
    super.processMergeIndexes(cmd);
}

public void processCommit(CommitUpdateCommand cmd) throws IOException
{
    if (cmd instanceof CassandraCommitUpdateCommand)
    {
        cprocessCommit = true;
    }
    else
    {
        processCommit = true;
    }
    super.processCommit(cmd);
}
}
}

```

3. Export the class to a JAR, and place the JAR in this location:

- Installer-No Services and Tarball installations: `install-location/resources/solr/lib`
- Installer-Services and Package installations: `/usr/share/dse/solr/lib`

The JAR is added to the CLASSPATH automatically.

4. Test your implementation. For example:

```

package com.datastax.bdp.search.solr.functional;

import
    com.datastax.bdp.search.solr.handler.update.DSEUpdateProcessorFactory;
import org.apache.solr.core.SolrCore;
import org.apache.solr.update.processor.UpdateRequestProcessor;

public class DSEUpdateRequestProcessorFactoryExample extends
    DSEUpdateProcessorFactory
{
    SolrCore core;

    public DSEUpdateRequestProcessorFactoryExample(SolrCore core) {
        this.core = core;
    }

    public UpdateRequestProcessor getInstance(
        UpdateRequestProcessor next)
    {
        return new TestUpdateRequestProcessor(next);
    }
}

```

Field input/output transformer example

Use the field input/output transformer API as an option to the input/output transformer support in Solr. An [Introduction to DSE Field Transformers](#) provides details on the transformer classes.

DSE Search includes the released version of a plugin API for Solr updates and a plugin to the CassandraDocumentReader. The plugin API transforms data from the secondary indexing API before data is submitted to Solr. The plugin to the CassandraDocumentReader transforms the results data from Cassandra to Solr.

Using the API, applications can tweak a Solr Document before it is mapped and indexed according to the schema.xml. The API is a counterpart to the input/output transformer support in Solr.

The field input transformer (FIT) requires a trailing Z for date field values.

Procedure

To use the API:

1. Define the plugin in the solrconfig.xml for a Cassandra table ([Solr core](#)).

```
<fieldInputTransformer name="dse" class="com.datastax.bdp.cassandra.index.solr.functional.BinaryFieldInputTransformer">
</fieldInputTransformer>

<fieldOutputTransformer name="dse" class="com.datastax.bdp.cassandra.index.solr.functional.BinaryFieldOutputTransformer">
</fieldOutputTransformer>
```

2. Write a transformer class something like this [reference implementation](#) to tweak the data in some way.
3. Export the class to a JAR, and place the JAR in this location:
 - Installer-No Services and Tarball installations: *install-location/resources/solr/lib*
 - Installer-Services and Package installations: */usr/share/dse/solr/lib*
 The JAR is added to the CLASSPATH automatically.
4. Test your implementation using something like the reference implementation.

FIT reference implementation

The DataStax Developer Blog provides an [introduction to DSE Field Transformers](#).

Here are examples of field input and output transformer (FIT) classes.

Input transformer example

```
package com.datastax.bdp.search.solr.functional;

import java.io.IOException;

import org.apache.commons.codec.binary.Hex;
import org.apache.commons.lang.StringUtils;
import org.apache.lucene.document.Document;
import org.apache.solr.core.SolrCore;
import org.apache.solr.schema.SchemaField;

import com.datastax.bdp.search.solr.FieldOutputTransformer;
import org.apache.solr.schema.IndexSchema;

public class BinaryFieldInputTransformer extends FieldInputTransformer
{
    @Override
    public boolean evaluate(String field)
    {
```

```
        return field.equals("binary");
    }

    @Override
    public void addFieldToDocument(SolrCore core,
        IndexSchema schema,
        String key,
        Document doc,
        SchemaField fieldInfo,
        String fieldValue,
        float boost,
        DocumentHelper helper)
        throws IOException
    {
        try
        {
            byte[] raw = Hex.decodeHex(fieldValue.toCharArray());
            byte[] decomp = DSP1493Test.decompress(raw);
            String str = new String(decomp, "UTF-8");
            String[] arr = StringUtils.split(str, ",");
            String binary_name = arr[0];
            String binary_type = arr[1];
            String binary_title = arr[2];

            SchemaField binaryNameField =
core.getSchema().getFieldOrNull("binary_name");
            SchemaField binaryTypeField =
core.getSchema().getFieldOrNull("binary_type");
            SchemaField binaryTitleField =
core.getSchema().getFieldOrNull("binary_title");

            helper.addFieldToDocument(core, core.getSchema(), key, doc,
binaryNameField, binary_name, boost);
            helper.addFieldToDocument(core, core.getSchema(), key, doc,
binaryTypeField, binary_type, boost);
            helper.addFieldToDocument(core, core.getSchema(), key, doc,
binaryTitleField, binary_title, boost);
        }
        catch (Exception ex)
        {
            throw new RuntimeException(ex);
        }
    }
}
```

Output transformer example

```
package com.datastax.bdp.search.solr.functional;

import java.io.IOException;
import org.apache.commons.lang.StringUtils;
import org.apache.lucene.index.FieldInfo;
import org.apache.lucene.index.StoredFieldVisitor;
import com.datastax.bdp.search.solr.FieldOutputTransformer;

public class BinaryFieldOutputTransformer extends FieldOutputTransformer
{
    @Override
    public void binaryField(FieldInfo fieldInfo, byte[] value,
                           StoredFieldVisitor visitor, DocumentHelper helper) throws
IOException
    {
        byte[] bytes = DSP1493Test.decompress(value);
```

```

        String str = new String(bytes, "UTF-8");
        String[] arr = StringUtils.split(str, ",");
        String binary_name = arr[0];
        String binary_type = arr[1];
        String binary_title = arr[2];

        FieldInfo binary_name_fi = helper.getFieldInfo("binary_name");
        FieldInfo binary_type_fi = helper.getFieldInfo("binary_type");
        FieldInfo binary_title_fi = helper.getFieldInfo("binary_title");

        visitor.stringField(binary_name_fi, binary_name);
        visitor.stringField(binary_type_fi, binary_type);
        visitor.stringField(binary_title_fi, binary_title);
    }
}

```

Interface for custom field types

DataStax Enterprise implements a `CustomFieldType` interface that marks Solr custom field types and provides their actual stored field type. The custom field type stores an integer trie field as a string representing a comma separated list of integer values: when indexed, the string is split into its integer values, each one indexed as a trie integer field. This class effectively implements a multi-valued field based on its string representation.

To use the `CustomFieldType` interface:

1. Implement a custom field type class something like the following reference implementation.
2. Export the class to a JAR, and place the JAR in this location:
 - Package installations: `usr/share/dse`
 - Installer-No Services and Tarball installations: `install_location/resources/dse/lib`

The JAR is added to the CLASSPATH automatically.

Reference implementation

Here is an example of a custom field type class:

```

package com.datastax.bdp.search.solr.functional;

import com.datastax.bdp.search.solr.CustomFieldType;
import java.util.ArrayList;
import java.util.List;
import org.apache.lucene.index.IndexableField;
import org.apache.solr.schema.FieldType;
import org.apache.solr.schema.SchemaField;
import org.apache.solr.schema.StrField;
import org.apache.solr.schema.TrieField;

public class CustomTestField extends TrieField implements CustomFieldType
{
    public CustomTestField()
    {
        this.type = TrieField.TrieTypes.INTEGER;
    }

    @Override
    public FieldType getStoredFieldType()
    {
        return new StrField();
    }
}

```

```

@Override
public boolean multiValuedFieldCache()
{
    return true;
}

@Override
public ListIndexableField createFields(SchemaField sf, Object value,
float boost)
{
    String[] values = ((String) value).split(" ");
    ListIndexableField fields = new ArrayListIndexableField();
    for (String v : values)
    {
        fields.add(createField(sf, v, boost));
    }
    return fields;
}

@Override
public String toInternal(String value)
{
    return value;
}

@Override
public String toExternal(IndexableField f)
{
    return f.stringValue();
}
}

```

Unsupported features

Unsupported features include Cassandra and Solr features. CQL-based Solr cores require a [new type mapping version 2](#). A [CQL table](#) must be created in Cassandra before creating the Solr core. The schema corresponding to a CQL table using a compound primary key requires a [special syntax](#).

Limitations

Apache Solr and Lucene limitations apply to DSE Search. For example, the 2 billion records per node limitation as described in [Lucene limitations](#).

Limitations and known Apache Solr issues apply to DSE Search queries. For example, incorrect **SORT** results for tokenized text fields.

Unsupported Cassandra features

- Cassandra [static columns](#)
- Cassandra [compound primary keys](#) for **COMPACT STORAGE** tables
- Cassandra counter columns
- Cassandra super columns
- Cassandra Thrift-compatible tables with column comparators other than UTF-8 or ASCII.

Unsupported Solr features

- DSE Search does not support [Solr Managed Resources](#).
- Solr schema fields that are both dynamic and multivalued for CQL-based Solr cores (only)

- The deprecated replaceFields request parameters on document updates for CQL-based Solr cores. Use the [suggested procedure](#) for inserting/updating data.
- Block joins based on the Lucene BlockJoinQuery in Solr indexes and CQL tables
- Schemaless mode
- Partial schema updates through the REST API after Solr resources are uploaded. For example, to update individual fields of a schema using the REST API to add a new field to a schema, you must change the schema.xml file, upload it again to Solr, and reload the core (same for copy fields).
- org.apache.solr.spelling.IndexBasedSpellChecker and org.apache.solr.spelling.FileBasedSpellChecker (org.apache.solr.spelling.DirectSolrSpellChecker is supported for spell checking)
- The commitWithin parameter
- The SolrCloud CloudSolrServer feature of SolrJ for endpoint discovery and round-robin load balancing

Other unsupported features

- Tuples and UDTs used inside primary key declarations.
- DSE Search does not support JBOD mode.
- The commit log replaces the Solr updatelog. The Solr updatelog is not supported in DSE Search. Consequently, features that require the updateLog are not supported. Instead of using [atomic updates](#), partial document updates are available by running the update with CQL.

DSE Search versus Open Source Solr

By virtue of its integration into DataStax Enterprise, differences exist between DSE Search and Open Source Solr (OSS).

Major differences

The major differences in capabilities are:

Capability	DSE Search	OS Solr	Description
Includes a database	yes	no	A user has to create an interface to add a database to OSS.
Indexes real-time data	yes	no	Cassandra ingests real-time data and Solr indexes the data.
Provides an intuitive way to update data	yes	no	DataStax provides a SQL-like language and command-line shell, CQL, for loading and updating data. Data added to Cassandra shows up in Solr
Indexes Hadoop output without ETL	yes	no	Cassandra ingests the data, Solr indexes the data, and you run MapReduce against that data in one cluster.
Supports data distribution	yes	yes [1]	DataStax Enterprise distributes Cassandra real-time, Hadoop, and Solr data to multiple nodes in a cluster transparently.
Balances loads on nodes/shards	yes	no	Unlike Solr and Solr Cloud loads can be rebalanced efficiently .
Spans indexes over multiple datacenters	yes	no	A cluster can have more than one datacenter for different types of nodes.
Automatically re-indexes Solr data	yes	no	The only way to re-index data in Solr is to have the client re-ingest everything.

Capability	DSE Search	OS Solr	Description
Stores data added through Solr in Cassandra	yes	no	Data updated using the Solr API shows up in Cassandra.
Makes durable updates to data	yes	no	Updates are durable and written to the Cassandra commit log regardless of how the update is made.
Upgrades of Lucene preserve data	yes	no	DataStax integrates Lucene upgrades periodically and when you upgrade DSE, data is preserved. Solr users must re-ingest all their data after upgrading to Lucene.
Security	yes	no	DataStax has extended SolrJ to protect internal communication and HTTP access. Solr data can be encrypted and audited. For example, use Kerberos or SSL security for a DSE instance and then run secure queries of that DSE instance by using CQL or HTTP.

[1] Requires using Zookeeper.

DSE Search tutorials and demos

Use the tutorials and demos to learn how to use DSE Search.

Tutorial: Basics

Setting up DSE Search for this tutorial involves the same basic tasks as setting up a typical application:

- [Create a Cassandra table.](#)
- [Import data.](#)
- [Create resources automatically.](#)

After finishing the setup tasks, you perform these tasks:

- [Explore the Solr Admin.](#)
- [Search the data using the Solr Admin.](#)
- [Search the data using CQL.](#)

In this tutorial, you use some sample data from a health-related census.

Start DSE Search and download files

This setup assumes you started DataStax Enterprise in [DSE Search mode](#) and downloaded the sample data and tutorial files. The tutorial files include a CQL table definition, which uses a compound primary key. The partitioning key is the id column and the clustering key is the age column.

Procedure

1. [Download the sample data and tutorial files.](#)
2. Unzip the files you downloaded in the DataStax Enterprise installation home directory.
A solr_tutorial146 directory is created that contains the following files.

- `copy_nhances.cql`
The COPY command you use to import data
 - `create_nhances.cql`
The Cassandra CQL table definition
 - `nhances52.csv`
The CSV (comma separated value) data
 - `schema.xml` and `solrconfig.xml`
The Solr schema and configuration file for the [advanced tutorial](#)
3. Take a look at these files using your favorite editor.

Create a Cassandra table

Procedure

1. Ensure that your configuration is appropriate, and that you know the snitch for your cluster. See [Configuring replication](#) and verify the status of your node:

```
$ nodetool status
```

2. Start `cqlsh`, and create a keyspace. Use the keyspace.

```
cqlsh> CREATE KEYSPACE nhances_ks WITH REPLICATION =
          {'class':'NetworkTopologyStrategy', 'Solr':1};

cqlsh> USE nhances_ks;
```

3. Copy the CQL table definition from the downloaded `create_nhances.cql` file, and paste it on the `cqlsh` command line.
This action creates the `nhances` table in the `nhances_ks` keyspace.

Import data

Procedure

1. Copy the `cqlsh` COPY command from the downloaded `copy_nhances.cql` file.
2. Paste the COPY command on the `cqlsh` command line, but do not run the command yet.
3. Change the FROM clause to match the path to `/solrTutorial46/nhances52.csv` that you downloaded to your computer, and then run the command.
This action imports the data from the CSV file into the `nhances` table in Cassandra. Output is:

```
20050 rows imported in 27.524 seconds.
```

Generate resources automatically

You can [generate solrconfig and schema resources](#) automatically when creating a core. You can use either a `dsetool` command or an [HTTP-post command](#) to automatically generate resources, or you can create the core from custom resources using the classic manual method shown in the [advanced tutorial](#).

Follow these steps to create resources automatically using the `dsetool` command.

Procedure

1. Exit cqlsh.
2. Run the following command, which is located in the bin directory of tarball installations. On a tarball installation:

```
$ bin/dsetool create_core nhanes_ks.nhanes generateResources=true  
reindex=true
```

There is no output from this command. You can search Solr data after indexing finishes.

Explore the Solr Admin

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

```
http://localhost:8983/solr/
```

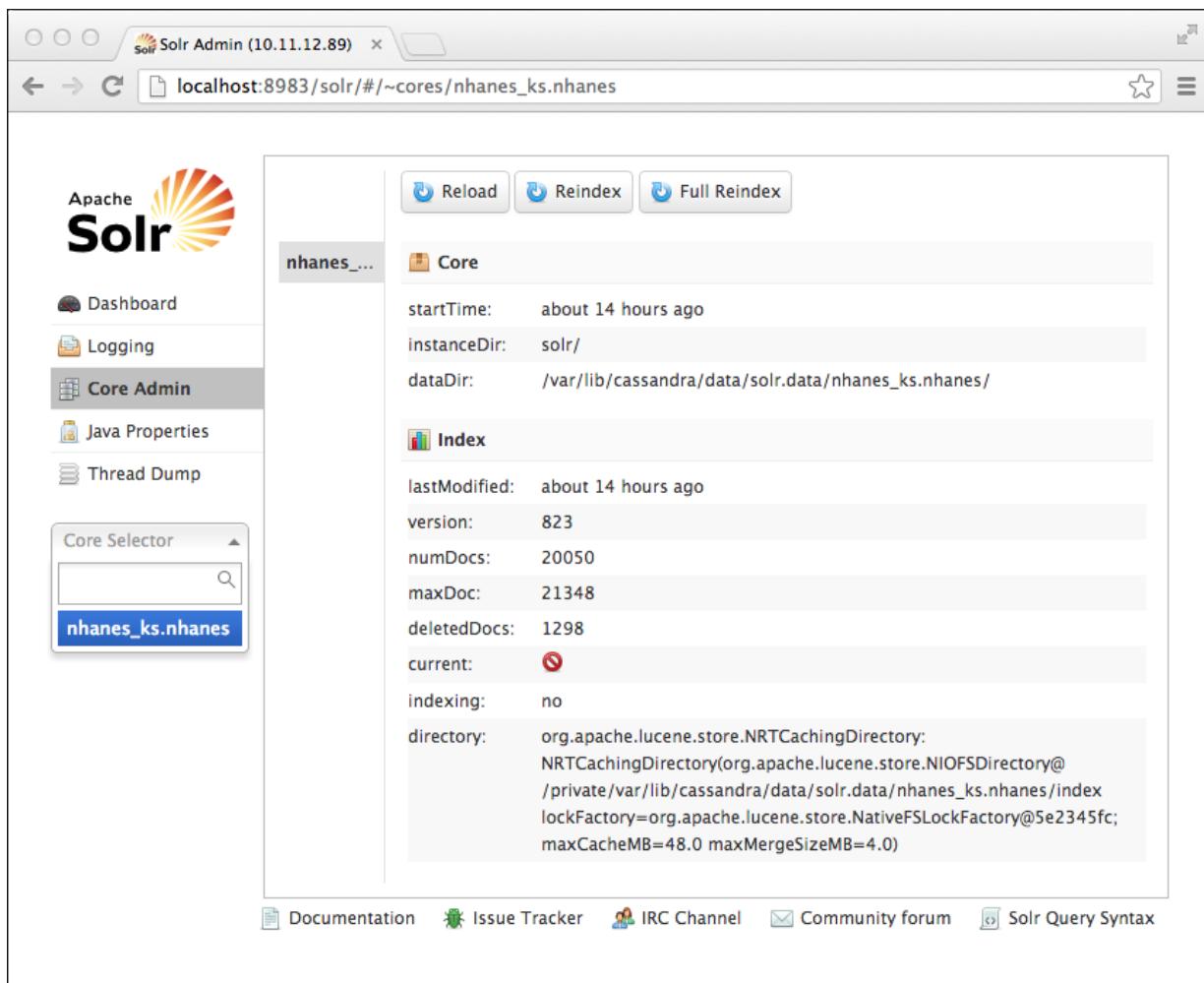
Procedure

To explore the Solr Admin:

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

At the top of the Solr Admin console, the **Reload**, **Reindex**, and **Full Reindex** buttons perform functions that [correspond to RELOAD command options](#). If you modify the schema.xml or solrconfig.xml, you can use these controls to re-index the data or you can use the [classic POST approach](#) used in the [advanced tutorial](#).

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



Search using the Solr Admin

To search the database, you have several choices:

- Run CQL queries in cqlsh or an application.
- Run Solr HTTP API queries in an application, a browser, or on the command line using the curl utility.
- Use the Solr Admin query form.

If you are new to Solr, using the query form has some advantages. The form contains text entry boxes for constructing a query and can provide query debugging information.

Procedure

After generating resources, get started searching the nhanes database by following these steps:

1. In the Solr Admin, click **Query**.
A query form appears.

The screenshot shows the Apache Solr Admin interface for the 'nhanes_ks.nhantes' core. The left sidebar has a 'Query' section selected. The main panel displays a form for constructing a Solr query. The 'Request-Handler (qt)' field is set to '/select'. The 'q' field contains the value '*.*'. Other fields include 'fq', 'sort', 'start, rows' (set to 0 to 10), 'fl', 'df', 'Raw Query' (containing 'key1=val1&key2=val'), 'wt' (set to 'xml'), and 'indent' (checked). The bottom status bar shows the URL 'localhost:8983/solr/#/nhanes_ks.nhantes/query'.

Notice that the form has a number of query defaults set up, including the select URL in **Request-Handler (qt)** and `*.*` in the main query parameter entry box--**q**.

2. Select **xml** from the **wt** drop down.
Output will appear in XML format.
3. Scroll down the form and click **Execute Query**.
The defaults select all the fields in all the documents, starting with row 0 and ending at row 10. The output looks something like this:

The screenshot shows the Apache Solr Admin interface at localhost:8983/solr/nhanes_ks.nhanes/query. The left sidebar has a 'Query' tab selected. The main area shows a 'Request-Handler (qt)' dropdown set to '/select'. The 'q' field contains '**'. Other fields include 'fq', 'sort', 'start, rows' (0, 10), 'fl', 'df', and 'wt' (xml). Parameters include 'key1=val1&key2=val'. The results pane displays an XML response with a 'responseHeader' section and a large 'response' section containing numerous XML entries.

Search using CQL

After generating resources, get started searching the nhanes database by following these steps:

Procedure

1. Start `cqlsh`.
2. Search the `family_size` field to find the ids of families of 6 or more.

```
SELECT id FROM nhanes_ks.nhanes WHERE solr_query='family_size:6' LIMIT 3;
```

```

id
-----
13322
36213
8856

(3 rows)

```

3. Perform a search for the ids of subjects in a Federal Information Processing Standards (fips) region that starts with the letters "Il" and whose ethnicity starts with the letters "Me".

```
SELECT id FROM nhanes_ks.nhanes WHERE solr_query='fips:Il* AND ethnicity:Mex*' LIMIT 5;
```

```
id
-----
48654
11298
36653
35025
35344

(5 rows)
```

4. Perform a fuzzy search for subjects are non-Hispanic.

```
select id, ethnicity FROM nhanes_ks.nhanes WHERE solr_query='ethnicity:"~Hispanic"' LIMIT 10;
```

id	ethnicity
38875	Not Hispanic
7789	Not Hispanic
50309	Not Hispanic
38721	Not Hispanic
48797	Not Hispanic
46146	Not Hispanic
49842	Other Hispanic
47675	Not Hispanic
13861	Not Hispanic
13014	Not Hispanic

```
(10 rows)
```

5. Perform a range search for ids of subjects who are from 551 to 590 months old.

```
SELECT id FROM nhanes_ks.nhanes WHERE solr_query='age_months:[551 TO 590]' LIMIT 3;
```

```
id
-----
50309
40371
32907

(3 rows)
```

6. Perform a JSON-based query that searches for the ids of subjects whose ethnicity is Mexican-American. Sort the results by id in descending order.

```
SELECT id FROM nhanes_ks.nhanes WHERE solr_query='{"q":"ethnicity:Mexi*","sort":"id asc"}' LIMIT 3;
```

```
id
-----
53582
53592
53595
```

(3 rows)

Tutorial: Advanced

Before attempting to step through this tutorial, complete the following prerequisites:

- Install the [curl utility](#) on your computer.
- Perform the “Setup,” “Create a Cassandra table,” and “Import data” sections of the [basic tutorial](#).

Use facets: Solr Admin

Distributed pivot facetting is supported. This tutorial performs a faceted search using the Solr Admin query form to drill down into filter search results that are based on a category of data.

Faceting is the arrangement of search results into categories based on indexed terms. Searchers are presented with the indexed terms, along with numerical counts of how many matching documents were found for each term. Faceting makes it easy to explore search results by narrowing the search results to what you are looking for.

Procedure

The following steps drill down into the health census database that you set up in the [basic tutorial](#). Use the Solr Admin to query the database using the age facet parameter in a query.

1. Open the Solr Admin.

```
http://localhost:8983/solr/
```

2. In **Core Selector**, select the name of the Solr core, nhanes_ks.nhanes.
3. Click Core Admin and then click Query.
4. In the Solr Admin query form, specify a family size of 9 in the main query parameter text entry box--**q**:

```
family_size:9
```

5. In **sort**, specify sorting by age in ascending order, youngest to oldest:

```
age asc
```

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

```
age family_size
```

Results from the main query will include only data about families of 9.

7. Select **xml** from the **wt** drop down.
Output will appear in XML format.
8. Select the **facet** option.
Text entry boxes for entering facet parameter values appear.
9. In **facet.field**, type this value:

```
age
```

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

10. Click **Execute Query**.
The numfound value shows that 186 families having nine members were found. The query results include only results from the fields in the filter list, age and family_size.

The screenshot shows the Apache Solr Admin interface running on port 8983. The left sidebar contains navigation links: Dashboard, Logging, Core Admin, Java Properties, Thread Dump, and a dropdown menu for the collection 'nhanes_ks.nhan...'. The 'Query' link is currently selected. The main area is titled 'Request-Handler (qt)' and shows a form for constructing a Solr query. The fields include:

- Request-Handler (qt):** /select
- common:** q=family_size:9
- fq:** age family_size
- sort:** age asc
- start, rows:** 0, 10
- fl:** age family_size
- df:** Raw Query
- Parameters:** key1=val1&key2=val2
- wt:** xml
- checkboxes:** indent (checked), debugQuery, dismax, edismax, hl, facet (checked)
- facets:**
 - facet.query:** (empty)
 - facet.field:** age
 - facet.prefix:** (empty)

To the right of the form is a large text area displaying the XML response from the Solr server. The response shows the query parameters and the resulting document set.

```

http://localhost:8983/solr/nhanes_ks.nhanes/select?q=family_size%3A9&wt=xml&indent=true&start=0&rows=10&sort=age+asc&fq=age+family_size&q=family_size:9&key1=val1&key2=val2

<?xml version="1.0" encoding="UTF-8"?>
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">160</int>
  </lst>
  <lst name="params">
    <str name="facet">true</str>
    <str name="fl">age family_size</str>
    <str name="sort">age asc</str>
    <str name="indent">true</str>
    <str name="q">family_size:9</str>
    <str name="_id">1380149425522</str>
    <str name="facet.field">age</str>
    <str name="wt">xml</str>
  </lst>
  <result name="response" numFound="186" start="0">
    <doc>
      <int name="family_size">9</int>
      <int name="age">17</int>
    </doc>
    <doc>
      <int name="family_size">9</int>
      <int name="age">17</int>
    </doc>
  </result>
</response>

```

11. Scroll to the end of the query form to see the facet results.
The facet results show 11 people of age 17, 10 of age 34, and so on.

The screenshot shows the Apache Solr Admin interface running on port 8983 at localhost. The URL in the address bar is `localhost:8983/solr/#/nhanes_ks.nhantes/query`. The left sidebar has a tree view with nodes like 'Dashboard', 'Logging', 'Core Admin', 'Java Properties', and 'Thread Dump'. Under 'Core Admin', the 'Query' node is selected and highlighted in grey. The main panel contains a form for a 'facet.query' search. The 'facet.field' dropdown is set to 'age'. The 'Execute Query' button is visible. To the right of the form is the raw XML response from the Solr server, which includes facets for 'family_size' and 'age'.

```

<int name="family_size">9</int>
<int name="age">17</int></doc>
<doc>
<int name="family_size">9</int>
<int name="age">17</int></doc>
<doc>
<int name="family_size">9</int>
<int name="age">17</int></doc>
</result>
<lst name="facet_counts">
<lst name="facet_queries"/>
<lst name="facet_fields">
<lst name="age">
<int name="17">11</int>
<int name="34">10</int>
<int name="19">9</int>
<int name="23">7</int>
<int name="28">7</int>
<int name="31">7</int>
<int name="37">7</int>
<int name="39">7</int>
<int name="18">6</int>
<int name="20">6</int>
<int name="27">6</int>
<int name="41">6</int>

```

Distributed pivot facetting is supported. You can do field, query, and range facetting with a JSON query.

Search the data: Solr HTTP API

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

Procedure

To get started using the Solr HTTP API:

1. Scroll to the top of the form, and click the greyed out URL.

The screenshot shows the Apache Solr Admin interface. On the left, there's a sidebar with links: Dashboard, Logging, Core Admin, Java Properties, and Thread Dump. The main area has a "Request-Handler" section with a dropdown menu set to "http://localhost:8983/solr/nhanes_ks.nhanes/select?q=family_size%3A". Below it is a "common" section with fields for "q" (set to "family_size:9") and "fq". To the right, the XML response to the query is displayed:

```

<?xml version="1.0" encoding="UTF-8"?>
<response>

<lst name="responseHeader">
<int name="status">0</int>
<int name="QTime">3</int>
<lst name="params">
<str name="facet">true</str>
<str name="fl">age family_size</str>
<str name="sort">age asc</str>

```

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

```

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

```

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

FROM:

```

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

```

TO:

```

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

```

The modified URL looks like this:

```

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

```

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

3. Use the modified URL to execute the query. Move to the end of the URL, and press ENTER.

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

```

. . .
</doc>
</result>
<lst name="facet_counts">
<lst name="facet_queries"/>
<lst name="facet_fields">
<lst name="age">

```

```

<int name="23">423</int>
<int name="24">407</int>
<int name="31">403</int>
<int name="30">388</int>
<int name="40">382</int>
<int name="28">381</int>
<int name="27">378</int>
<int name="21">377</int>
<int name="33">377</int>
<int name="22">369</int>
<int name="29">367</int>
<int name="20">365</int>
<int name="32">363</int>
<int name="34">361</int>
<int name="36">361</int>
<int name="25">358</int>
<int name="26">358</int>
<int name="35">358</int>
<int name="38">353</int>
<int name="37">339</int>
<int name="39">291</int>
<int name="17">0</int>
. . .

```

Create a CQL collection

In these advanced DSE Search tutorial steps, you create a Cassandra table having a map collection column. Using [dynamic fields](#), you process multiple Solr fields the same way by using a generic prefix or suffix to reference the field. In this task, you create a Cassandra table having a map collection column. This column will correspond to a dynamic field that you set up in the Solr schema in the next task.

Prerequisites

To use Solr dynamic fields for map and collection type fields:

- Use a prefix as the name of the field itself.
- When inserting data into the field, the map key must contain the prefix that you used to name the field.

See [Using dynamic fields](#) on page 207.

Procedure

1. Create a keyspace.

```

CREATE KEYSPACE mykeyspace
    WITH REPLICATION = {'class':'NetworkTopologyStrategy', 'Solr':1};

USE mykeyspace;

```

2. Create a table having a map collection column. Apply the dynamic field naming convention that you plan to use in the schema to the column name.

```

CREATE TABLE hits (
    song uuid,
    lang_map<text, text>,
    PRIMARY KEY (song)
) ;

```

3. Insert the following data about Italian and Hawaiian songs into the hits table. Use the lang_ to prefix the first component of each map pair.

```
INSERT INTO hits (song, lang_) VALUES
( 62c36092-82a1-3a00-93d1-46196ee77204, { 'lang_i-title' : 'La Vita E La
Felicità', 'lang_i-artist' : 'Michele Bravi' });
INSERT INTO hits (song, lang_) VALUES ( 8a172618-b121-4136-bb10-
f665cfc469eb, { 'lang_h-title' : 'Blew it', 'lang_h-artist' : 'Maoli f/
Fiji' });
INSERT INTO hits (song, lang_) VALUES ( a3e64f8f-bd44-4f28-
b8d9-6938726e34d4, { 'lang_i-title' : 'Dimmi Che Non Passa Felicità',
'lang_i-artist' : 'Violetta' });
```

Create a custom schema

The tutorial files that you downloaded in the basic tutorial include a Solr schema and a solrconfig file. You replace the schema with a custom schema that corresponds to the hits table and defines a dynamic field.

Procedure

1. Open the `schema.xml` in the `solr_tutorial46` directory.
2. Compare the schema with the corresponding hits table that you created.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="topHits" version="1.5">
  <types>
    <fieldType class="org.apache.solr.schema.TextField" name="TextField">
      <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <filter class="solr.LowerCaseFilterFactory"/>
      </analyzer>
    </fieldType>
    <fieldType class="org.apache.solr.schema.UUIDField" name="UUIDField"/>
  </types>
  <fields>
    <dynamicField indexed="true" multiValued="false" name="lang_*"
      stored="true" type="TextField"/>
    <field indexed="true" multiValued="false" name="song" stored="true"
      type="UUIDField"/>
  </fields>
  <uniqueKey>song</uniqueKey>
</schema>
```

The uniqueKey is the name of the CQL primary key. The dynamicField is the name of the CQL lang_ column plus the asterisk wildcard suffix. A tokenizer determines the parsing of the example text. The fields specify the data that Solr indexes and stores. You will be able to query on data using lang_*, as shown later in this tutorial.

Check the request handler

To run CQL Solr queries, the `solrconfig.xml` must include a `solr_query` request handler. An automatically generated `solrconfig` includes this request handler. Verify that the `solrconfig.xml` file includes the request handler.

Procedure

1. In a text editor, open the `solrconfig.xml` file in the `solr_tutorial46` directory that you downloaded.
2. In your editor, search the `solrconfig.xml` file for "SearchHandler".

You see this location:

```
<!-- SearchHandler
     http://wiki.apache.org/solr/SearchHandler

     For processing Search Queries, the primary Request Handler
     provided with Solr is "SearchHandler" It delegates to a sequent
     of SearchComponents (see below) and supports distributed
     queries across multiple shards
-->
```

3. Check that the following request handler appears below the SearchHandler comment.

```
<requestHandler
  class="com.datastax.bdp.search.solr.handler.component.CqlSearchHandler"
  name="solr_query">
```

If the `solr_query` request handler is not in `solrconfig.xml`, add it.

Upload custom resources

Follow these Advanced DSE Search tutorial steps to upload custom resources.

Procedure

On the operating system command line in the `solr_tutorial46` directory:

Upload the custom resource files:

```
$ dsetool create_core mykeyspace.hits solrconfig=/path/my/solrconfig.xml
schema=/path/my/schema.xml
```

The return code 0 indicates success.

Search the dynamic field

To find data about hit songs in Italy, query on either of the prefixed map literals, `lang_i-title` or `lang_i-artist`.

Procedure

1. Open a browser.
2. Enter this Solr HTTP query in the address bar:

```
http://localhost:8983/solr/mykeyspace.hits/select?q=lang_i-title
%3A*&wt=xml&indent=true
```

```
<result name="response" numFound="2" start="0">
<doc>
  <str name="song">62c36092-82a1-3a00-93d1-46196ee77204</str>
  <str name="lang_i-artist">Michele Bravi</str>
  <str name="lang_i-title">La Vita E La Felicita</str.</doc>
```

```

<doc>
  <str name="song">a3e64f8f-bd44-4f28-b8d9-6938726e34d4</str>
  <str name="lang_i-artist">Violetta</str>
  <str name="lang_i-title">Dimmi Che Non Passa Felicita</str></doc>
</result>

```

Running Wikipedia demo using DSE Search

The following instructions describe how to run the Wikipedia demo on a single node. You run scripts that download 3,000 Wikipedia articles, create a CQL table, store the articles, and index the articles in Solr. The demo includes a web interface for querying the articles. You can also use the Solr HTTP API or CQL to query the articles.

The scripts that you run in this demo are written to set up the localhost and fail if the default interface of the node is not 127.0.0.1.

Procedure

1. Start DataStax Enterprise as a Solr node if you haven't already done so.
2. When using cqlsh, pagination is on by default. Queries with small result sets will see increased performance when paging is turned off, so use the [CQL PAGING](#) command to disable pagination:

```
PAGING OFF
```

3. Go to the wikipedia demo directory.
 - Installer-Services and Package installations: \$ cd /usr/share/dse/demos/wikipedia
 - Installer-No Services and Tarball installations: \$ cd *install_location*/demos/wikipedia
4. Upload the schema by running the add schema script. On Linux, for example:

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

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

- `http://localhost:8983/solr/resource/wiki.solr/solrconfig.xml`
- `http://localhost:8983/solr/resource/wiki.solr/schema.xml`

The script also creates the Solr index and core. The `wiki.solr` part of the URL creates the keyspace (`wiki`) and the column family (`solr`) in Cassandra.

5. Index the articles contained in the `wikipedia-sample.bz2` file in the demo directory by running the index script.

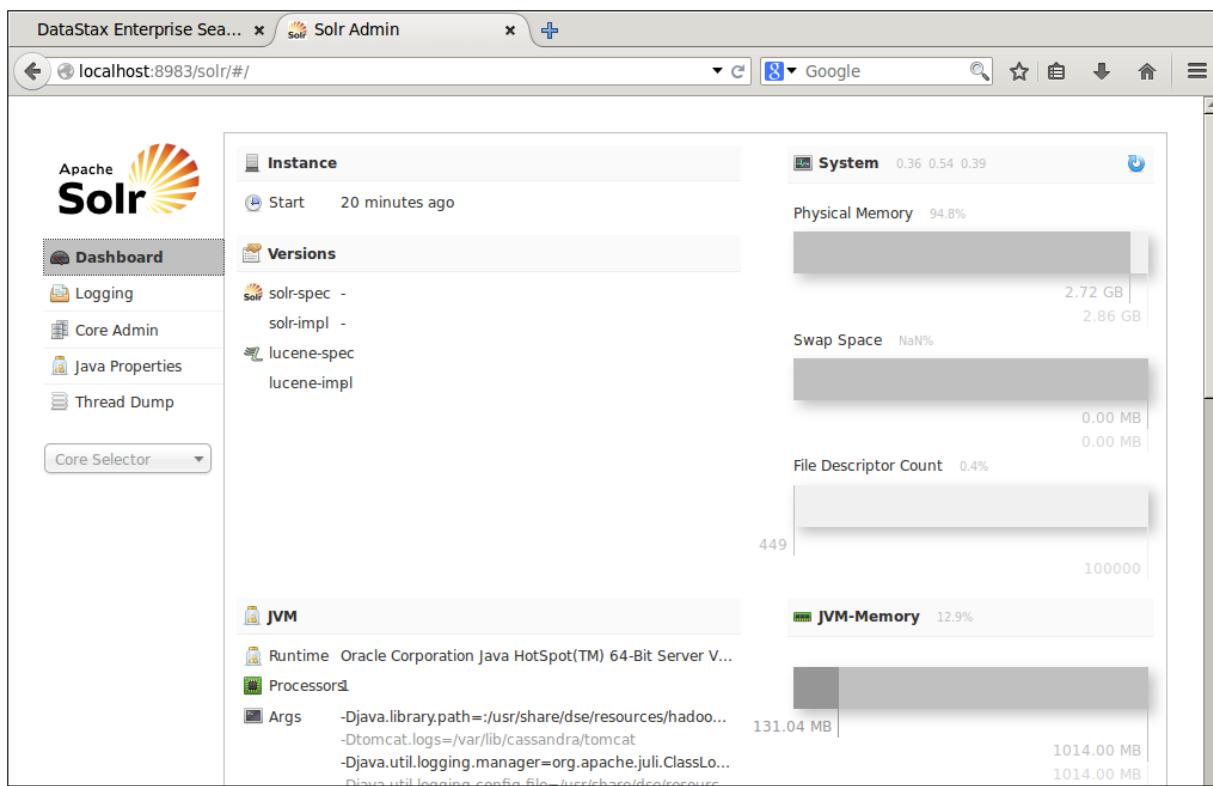
```
$ ./2-index.sh --wikifile wikipedia-sample.bz2
```

Three thousand articles load.

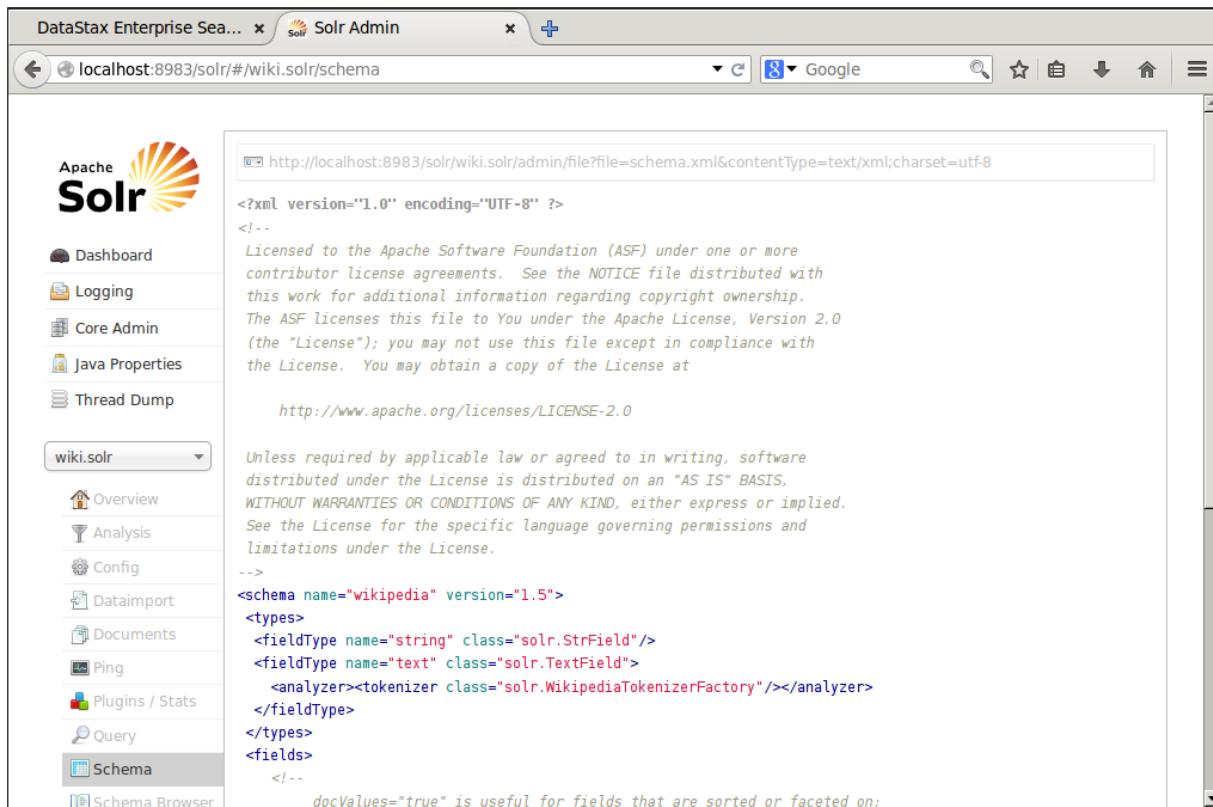
6. Open the **Solr Admin** tool.

Be sure to enter the trailing "/".

```
http://localhost:8983/solr/
```



7. Inspect the schema. In the **Solr Admin**, select wiki.solr from the **Core Selector** drop-down. Click the **Schema** in the vertical navigation bar.



You can use the Solr Admin to query the Wikipedia database in Cassandra. You can also use the [Solr HTTP API](#) or cqlsh to query the database.

8. Start [cqlsh](#), and use the wiki keyspace. Execute a CQL select statement using the [solr_query expression](#) to find the titles in the table named solr that begin with the letters natio:

```
USE wiki;

SELECT title FROM solr WHERE solr_query='title:natio*';
```

The output, sorted in lexical order, appears:

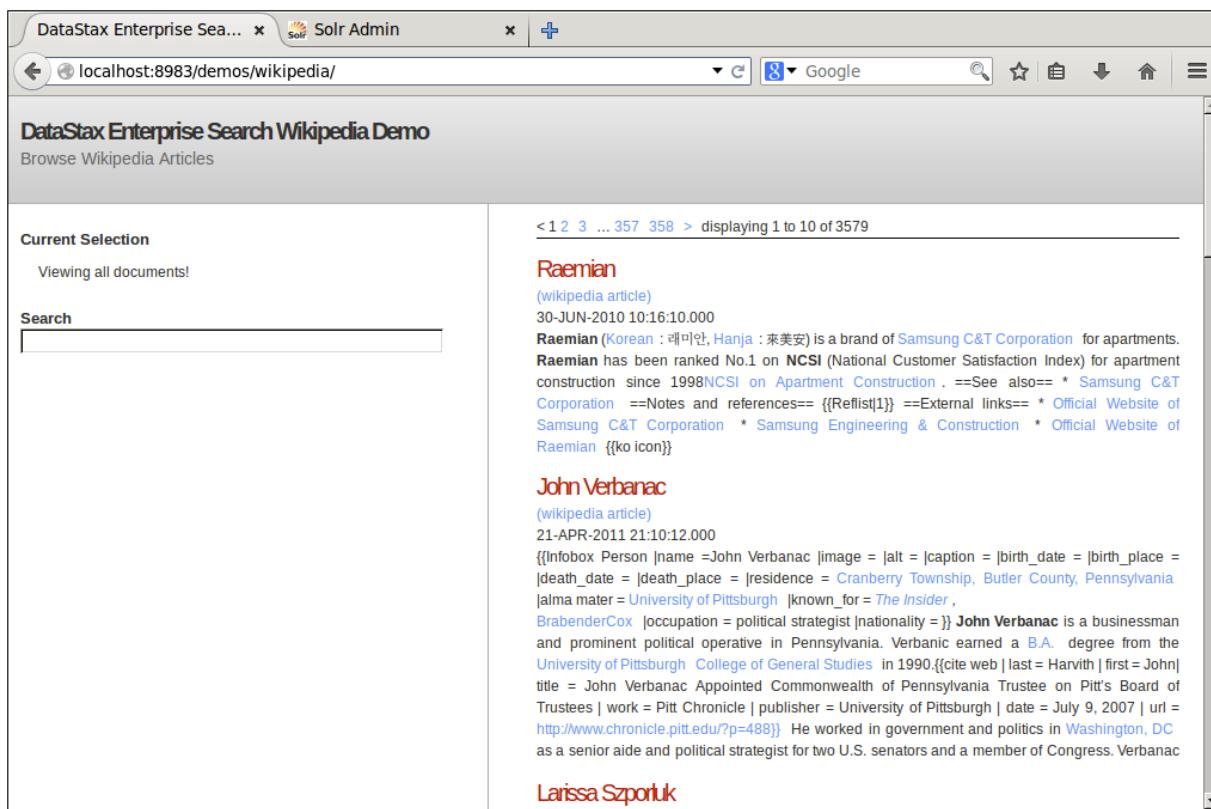
title
Kenya national under-20 football team
Bolivia national football team 2002
Israel men's national inline hockey team
List of French born footballers who have played for other national teams
Bolivia national football team 1999
Bolivia national football team 2001
Bolivia national football team 2000

Using CQL, you can enclose the Solr query string in single quotation marks. For example, after running the [Solr Demo](#), you can use these Solr query strings:

Type of Query	Example	Description
Field search	'title:natio*' AND Kenya	You can use multiple fields defined in the schema: 'title:natio*' AND body:CarlosAragonés'
Wildcard search	'Ken?a'	Use ? or * for single or multi-character searches.
Fuzzy search	'Kenya~'	Use with caution, many hits can occur.
Phrase search	"American football player"	Searches for the phrase enclosed in double quotation marks.
Proximity search	"football Bolivia"~10'	Searches for football and Bolivia within 10 words of each other.
Range searches	'title:[football TO soccer]'	Supports both inclusive and exclusive bounds using square brackets and curly braces, respectively.
Term boosting	"football"^-4 "soccer"	By default, the boost factor is 1. Must be a positive number.
Boolean operator	'+Macedonian football'	AND, +, OR, NOT and - can be used.
Grouping	'(football OR soccer) AND Carlos Aragonés'	Use parentheses to group clauses.
Field grouping	'title:(+football +"Bolivia")'	Use parentheses to group multiple clauses into one field.

9. To see the sample Wikipedia search UI, open your web browser and go to this URL:

```
http://localhost:8983/demos/wikipedia
```



10. To search in the bodies of the articles, enter a word in the Search field, and press Enter.

Running the Wikipedia search demo on a secure cluster

Kerberos Options

- -a enable Kerberos authentication
- -h *hostname* server hostname (not required if server hostname resolution is correctly set up)

HTTP Basic Authentication

Use with Cassandra's PasswordAuthenticator.

- -u username
- -p password

SSL Options

- -e *cert* enable HTTPS for client to node encryption, using *cert* certificate file
- -k disable strict hostname checking for SSL certificates

Related information

[DSE Advanced Security](#) on page 301

Troubleshooting

Handling inconsistencies in query results

DSE Search implements an efficient, highly available distributed search algorithm on top of Cassandra, which tries to select the minimum number of replica nodes required to cover all token ranges, and also

avoid hot spots. Consequently, due to the eventually consistent nature of Cassandra, some replica nodes might not have received or might not have indexed the latest updates yet. This situation might cause DSE Search to return inconsistent results (different numFound counts) between queries due to different replica node selections. This behavior is intrinsic to how highly available distributed systems work, as described in the ACM article, "[Eventually Consistent](#)" by Werner Vogels. Most of the time, [eventual consistency is not an issue](#), yet DSE Search implements *session stickiness* to guarantee that consecutive queries will hit the same set of nodes on a healthy, stable cluster, to provide monotonic results. Session stickiness works by adding a *session seed* to request parameters as follows:

```
shard.shuffling.strategy=SEED
shard.shuffling.seed=session_id
```

In the event of unstable clusters with missed updates due to failures or network partitions, consistent results can be achieved by repairing nodes using the [subrange repair method](#).

Finally, another minor source of inconsistencies is caused by different soft commit points on different replica nodes: A given item might be indexed and committed on a given node, but not yet on its replica. This situation is primarily a function of the load on each node. Implement the following best practices:

- Evenly balance read/write load between nodes
- Properly tune soft commit time and async indexing concurrency
- Configure back pressure in the [dse.yaml](#) file

For information about multi-threaded asynchronous indexing that uses a back pressure mechanism, see [Configuring multi-threaded indexing threads](#) on page 267.

To maximize insert throughput, DSE Search buffers insert requests from Cassandra so that application insert requests can be acknowledged as quickly as possible. However, if too many requests accumulate in the buffer (a configurable setting), DSE Search pauses or blocks incoming requests until DSE Search catches up with the buffered requests. In extreme cases, that pause causes a timeout to the application.

Tracing Solr HTTP requests

For debugging and troubleshooting queries, you can [trace](#) Solr HTTP requests in one of the following ways:

- Enable [probabilistic tracing](#).
- Pass an explicit `cassandra.trace=true` request parameter in the HTTP query.

After running the [example of using a join query](#), you can trace the join query by adding the `cassandra.trace` parameter to the HTTP request:

```
http://localhost:8983/solr/internet.songs/select/?
q={!join+from=song+to=id+fromIndex=internet.lyrics
+force=true}words:love&indent=true&wt=json&cassandra.trace=true
```

The Solr response includes a `cassandra.trace.session` value, the unique session id of the tracing session for the request:

```
{
  "cassandra.trace.session": "3e503490-bdb9-11e3-860f-73ded3cb6170",
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "indent": "true",
      "q": "{!join from=song to=id fromIndex=internet.lyrics
force=true}words:love",
      "wt": "json",
      "cassandra.trace": "true" } },
```

```

"response": {"numFound":2,"start":0,"docs": [
    {
        "id":"8a172618-b121-4136-bb10-f665cf469eb",
        "title":"Internet Love Song",
        "artist":"John Cedrick"},
    {
        "id":"a3e64f8f-bd44-4f28-b8d9-6938726e34d4",
        "title":"Dangerous",
        "artist":"Big Data"}]
}
}

```

To see the information from the trace, query the system_traces.events, using the session id to filter the output.

```

cqlsh> select * from system_traces.events where session_id = 3e503490-
bdb9-11e3-860f-73ded3cb6170;

session_id | activity
            | source_elapsed
-----
+-----+
+-----+
3e503490... |          Parsing SELECT * from "internet"."songs" WHERE "id"
= 8a172618...|          2607
3e503490... |
Preparing statement |          3943
3e503490... |          Executing single-partition
query on songs |          4246
3e503490... |          Acquiring
sstable references |          4261
3e503490... |          Merging
memtable tombstones |          4305
3e503490... |          Key cache hit
for sstable 1 |          4388
3e503490... |          Seeking to partition indexed section
in data file |          4399
3e503490... | Skipped 0/1 non-slice-intersecting sstables, included 0 due
to tombstones |          4873
3e503490... |          Merging data from memtables
and 1 sstables |          4954
3e503490... |          Read 1 live and 0
tombstoned cells |          5162
3e503490... |          Parsing SELECT * from "internet"."songs" WHERE "id" =
a3e64f8f... |          6160
3e503490... |
Preparing statement |          7424
. . .

```

For example purposes, the event_id, node IP address, and thread id have been deleted from this output to fit on the page.

In the case of distributed queries over several nodes, Cassandra uses the same tracing session id on all nodes, which makes it possible to correlate Cassandra operations on all the nodes taking part in the distributed query.

Using Solr MBeans

DataStax Enterprise provides enhanced visibility into native memory allocation through the solr/NativeAllocatorStats MBean, exposing the following information:

- enabled: if native memory is enabled or not.
- debug: if debug mode is enabled or not.

- numAlloc: number of native objects allocations.
- numFree: number of freed native objects.
- activeAllocatedMemoryInBytes: allocated native memory currently in use.
- totalAllocatedMemoryInBytes: total allocated native memory over time.
- totalFreedMemoryInBytes: total freed native memory over time.

The solr/NativeTrackerStats MBean provides information about the tracked native objects and related threads that allocated them:

- registeredThreads: number of threads currently registered and actively tracking (allocating) native objects.
- trackedObjects: number of currently tracked (allocated and not freed) native objects.
- handedOffObjects: number of currently handed off (allocated and stored for later reuse) native objects.

Using the ShardRouter Mbean

The ShardRouter Mbean, not present in open source Solr, provides information about how DSE search routes queries. JMX Mbeans can be accessed by connecting to the (default) JMX port 7199 on any DataStax Enterprise node using a JMX application like JConsole. The following the attributes and operations are available in this Mbean:

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

DSE Advanced Security

About security management

DataStax Enterprise includes advanced data protection for enterprise-grade databases:

- [LDAP authentication support](#) for external LDAP services.
- [Internal authentication](#) using login accounts and passwords
- [Managing object permissions using internal authorization](#) on page 346 based on the [GRANT/REVOKE](#) paradigm
- [Client-to-node encryption](#) on page 318 using SSL for data going from the client to the Cassandra cluster and for [Sqoop-imported and exported data](#)
- [Node to node encryption](#) using SSL for data between nodes
- [Kerberos authentication](#) to allow nodes to communicate over a non-secure network by proving their identity to one another in a secure manner using tickets
- [Configuring and using data auditing](#) on page 334 for creating detailed audit trails of cluster activity
- [Transparent data encryption](#) on page 325 that transparently encodes data flushed from the memtable in system memory to the SSTables on disk (at rest data), making the at rest data unreadable by unauthorized users

The [TCP-communications layer for Solr](#) supports client-to-node and node-to-node encryption using SSL, but does not support Kerberos.

If you use the [bring your own Hadoop \(BYOH\)](#) model and use Kerberos to protect your data, configure external Hadoop security under Kerberos on your cluster. For information about configuring Hadoop security, see "[Using Cloudera Manager to Configure Hadoop Security](#)" or the [Hortonworks documentation](#).

The [DataStax Java Driver](#) and [DataStax C# Driver](#), available on the [DataStax web site](#), enables Kerberos support and also SSL for client/server communication.

Data security by workload type

Assuming you configure security features, this table describes which data is secured based on the workload type: transactional DSE Cassandra, DSE Analytics (Hadoop/Spark), and DSE Search.

Feature	DataStax Enterprise security	DSE Hadoop	DSE Search with Solr	DSE Analytics with Spark
Internal authentication/LDAP	Yes	Yes [1]	Partial [12]	Yes [2]
Object permission management	Yes	Partial [3]	Partial [3]	Partial [3]
Client-to-node encryption	Yes [4]	Yes [5]	Yes [6]	Yes [7]
Kerberos authentication	Yes [8]	Yes	Yes	Yes [2]
Transparent data encryption	Yes [9]	Yes	Partial [10]	No
Data auditing	Yes	Partial [11]	Full	Partial [11]

[1] Password authentication pertains to connecting Hadoop to Cassandra, not authenticating Hadoop components between each other.

[2] Password authentication pertains to connecting Spark to Cassandra, not authenticating Spark components between each other, for internal authentication and Kerberos. The Spark Web UI is not secured and might show the Spark configuration, including username, password, or delegation token when Kerberos is used.

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

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

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

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

[7] SSL client-to-node encryption is for Spark Executor to Cassandra connections only.

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

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

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

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

[12] Search using CQL is supported for Internal authentication/LDAP, but is not supported over HTTP.

Using Kerberos and SSL at the same time

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

- Kerberos - If you enable Kerberos authentication, integrity protection is also enabled. However, you can enable integrity protection without encryption.
- SSL - When using SSL, authentication, integrity protection, and encryption are all enabled or disabled.
- Kerberos and SSL - You can enable both Kerberos authentication and SSL together. However, some overlap occurs because authentication is performed twice by two different schemes: Kerberos authentication and certificates through SSL. DataStax recommends choosing one and using it for both encryption and authentication. These settings are described in the [dse.yaml](#) configuration file.

Security options for sstableloader

The [procedure for securing sstableloader](#) has changed slightly from previous releases.

Authenticating a cluster with Kerberos

Note: Also see [Use Kerberos authentication for DSE Search in production](#) and [Kerberos authentication for Spark](#).

Kerberos guidelines

This section provides information on configuring DataStax Enterprise as a Kerberos client. Kerberos is a computer network authentication protocol that allows nodes communicating over a non-secure network to prove their identity to one another in a secure manner using tickets. For information on installing and setting up Kerberos, see the [MIT Kerberos Consortium](#) documentation.

Note: The [Kerberos Tutorial](#) provides step-by-step instructions on configuring DataStax Enterprise as a Kerberos client. It is intended for anyone interested in enabling [Kerberos](#) authentication in DataStax Enterprise and OpsCenter.

Kerberos guidelines

The following are general guidelines for setting up Kerberos and configuring DataStax Enterprise as a Kerberos client:

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

- Ensure that you are familiar with Kerberos and have reviewed the [MIT Kerberos Consortium](#) documentation. At a minimum understand how to use these commands: kinit, klist, and kdestroy.
- You must have authority to set [cassandra.yaml](#) and [dse.yaml](#) options.
- Before implementing Kerberos on your DataStax Enterprise nodes, set up your Kerberos servers.
- Set up several machines as authentication servers (Key Distribution Center [KDC]). One server is the primary or administration KDC, the other servers will be secondary.

- You must have privileges or have access to KDC administrators who can manage Kerberos principals and export keytab files.
- Do not install the KDC servers on DataStax Enterprise nodes.
- Set up firewalls on each KDC server.
- Physically protect the KDC machines.
- Secure the keytab files that are owned by the user running DataStax Enterprise. The files should be readable and writable only by the owner, without permissions for any other user (`chmod 600`).
- If using Oracle Java 7, you must use at least 1.7.0_25. If using Oracle Java 8, you must use at least 1.8.0_40. In some cases, using JDK 1.8 causes minor performance degradation compared to JDK 1.7.

Using Kerberos with DataStax Enterprise

The following topics provide information on using Kerberos with various DataStax Enterprise features and other software:

- [Using Kerberos and SSL at the same time](#)
- [Enabling dsetool to use Kerberos](#)
- When using [audit logging](#) with Kerberos authentication, the login events take place on Kerberos and are not logged in DataStax Enterprise. Authentication history is available only on Kerberos. When DataStax Enterprise is unable to authenticate a client with Kerberos, a `LOGIN_ERROR` event is logged.
- To configure Kerberos in an external Hadoop system, see [Using Cloudera Manager to Configure Hadoop Security](#) or the [Hortonworks documentation](#).

AES-256 support

Some of the cipher suites in the default set of `server_encryption_options` in `cassandra.yaml` are included only in the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files. To ensure support for all encryption algorithms, install the JCE Unlimited Strength Jurisdiction Policy Files.

By default Kerberos uses the AES-256 cipher. DataStax recommends using AES-256 encryption. OpenJDK includes AES-256. However, Oracle Java does not include the AES-256 cypher due to export restrictions to certain countries. To use AES-256 with Oracle Java, install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy using one of the following methods:

Installing the JCE on RHEL-based systems

Install the EPEL repository:

```
$ sudo yum install epel-release
```

Installing the JCE on Debian-based systems

Install JCE using [webupd8 PPA repository](#):

```
$ sudo apt-get install oracle-java8-unlimited-jce-policy
```

Installing the JCE using the Oracle JAR

1. Download the Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files from [Oracle Java SE download page](#).
2. Unzip the downloaded file.
3. Copy `local_policy.jar` and `US_export_policy.jar` to the `$JAVA_HOME/jre/lib/security` directory to overwrite the existing JARS.

Removing AES-256

If you do not use AES-256, you must remove the AES-256 settings as an allowed cypher for each principal and then regenerate the keys for the krbtgt principal.

Prerequisites

These methods require Kerberos 5-1.2 on the KDC.

Procedure

Remove AES-256 settings in one of the following ways:

- If you have **not** created the principals, use the -e flag to specify encryption:salt type pairs. For example: -e "arcfour-hmac:normal des3-hmac-sha1:normal".
- If you have already created the principals, modify the Kerberos principals using the -e flag as described above and then recreate the keytab file.

Alternately, you can modify the `/etc/krb5kdc/kdc.conf` file by removing any entries containing `aes256` from the `supported_enctypes` variable for the realm in which the DataStax Enterprise nodes are members. Then change the keys for the krbtgt principal.

Note: If the KDC is used by other applications, changing the krbtgt principal's keys invalidates any existing tickets. To prevent this, use the `-keepold` option when executing the `change_password` command. For example:

```
'cpw -randkey krbtgt/krbtgt/REALM@REALM'
```

What to do next

[Setting up the environment for Kerberos](#)

Setting up the environment for Kerberos

Each node in your cluster requires DNS to be working properly, NTP to be enabled and the system time synchronized, and the Kerberos client libraries installed.

Note: Do not upgrade DataStax Enterprise and set up Kerberos at the same time; see [Security Recommendations](#).

Prerequisites

- You have read and implemented the [Kerberos guidelines](#).
- Latest version of [Oracle Java SE Runtime Environment 7 or 8](#) or [OpenJDK 7](#) is recommended.

Note: If using Oracle Java 7, you must use at least 1.7.0_25. If using Oracle Java 8, you must use at least 1.8.0_40. In some cases, using JDK 1.8 causes minor performance degradation compared to JDK 1.7.

- If you are using Oracle Java, make sure the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files are [installed on each node](#).

Procedure

Perform the following steps on every node:

1. On each node, confirm DNS is working:

```
$ hostname
```

```
node1.example.com
```

- On each node, confirm that NTP is configured and running:

```
$ ntpq -p
```

remote	refid	st	t	when	poll	reach	delay	offset
jitter								
*li506-17.member	209.51.161.238	2	u	331	1024	377	80.289	1.384
1.842								
-tock.eoni.com	216.228.192.69	2	u	410	1024	377	53.812	1.706
34.692								
+time01.muskegon	64.113.32.5	2	u	402	1024	377	59.378	-1.635
1.840								
-time-a.nist.gov	.ACTS.	1	u	746	1024	151	132.832	26.931
55.018								
+golem.canonical	131.188.3.220	2	u	994	1024	377	144.080	-1.732
20.072								

- Install the Kerberos client software.

- RHEL-based systems:

```
$ sudo yum install krb5-workstation krb5-libs krb5-pkinit-openssl
```

- Debian-based systems:

```
$ sudo apt-get install krb5-user krb5-config krb5-pkinit
```

- If you are not using the JCE Unlimited Strength Jurisdiction Policy, make sure that your ticket granting principal does not use [AES-256](#).

- If your Kerberos sever is using MIT Kerberos server for Linux, copy the `krb5.conf` from the Kerberos server to each DataStax Enterprise node. If using other Kerberos server solution, copy the REALM section to the `krb5.conf` on each DataStax Enterprise node.

```
$ scp /etc/krb5.conf node1.example.com:/etc/
```

The `krb5.conf` file contains configuration information for your Kerberos domain.

What to do next

[Adding Kerberos service principals for each node in a cluster](#)

Adding Kerberos service principals for each node in a cluster

Prerequisites

- Installed and verified the software as described in [Setting up your environment](#).
- An existing Kerberos domain.
- An existing KDC is running.
- Admin rights to the KDC.

Procedure

- On each node, note the fully qualified domain name (FQDN) of the machine:

```
$ hostname --fqdn
```

```
node1.example.com
```

- On the Kerberos Key Distribution Center (KDC), run the `kadmin` command:

```
kadmin -p user_name/admin
addprinc -randkey dse_user/FQDN
addprinc -randkey HTTP/FQDN
quit
```

where

Parameter	Description
addprinc	The <code>add_principal</code> command requires the <code>add</code> administrative privilege and creates the new principal.
dse_user	This value depends on the type of install: <ul style="list-style-type: none"> Installer-Services and Package installations: usually <code>cassandra</code> Package installations: the name of the UNIX user that starts the service
FQDN	The fully qualified domain name of the host where DataStax Enterprise is running.
-randkey	Sets the key of the principal to a random value.

Example:

```
kadmin -p parzival/admin
addprinc -randkey cassandra/node1.example.com
addprinc -randkey HTTP/node1.example.com
addprinc -randkey cassandra/node2.example.com
addprinc -randkey HTTP/node2.example.com
```

- Optional: Verify that the principals have been added by running the `listprincs` command within `kadmin`:

```
$ listprincs
```

```
HTTP/node1.example.com@EXAMPLE.COM
HTTP/node2.example.com@EXAMPLE.COM
cassandra/node1.example.com@EXAMPLE.COM
cassandra/node2.example.com@EXAMPLE.COM
kadmin/admin@EXAMPLE.COM
```

where `node*.example.com` is the FQDN and EXAMPLE.COM is your Kerberos realm, which must be all uppercase.

- Create a keytab file for each node with the principals keys for that node:

```
kadmin -p user_name/admin
ktadd -k dse.keytab cassandra/FQDN
ktadd -k dse.keytab HTTP/FQDN
quit
```

where `ktadd -k` creates or appends a keytab for the `dse` and `HTTP` principals.

Example:

```
kadmin -p parzival/admin
ktadd -k /tmp/node1.keytab cassandra/node1.example.com
ktadd -k /tmp/node1.keytab HTTP/node1.example.com
ktadd -k /tmp/node2.keytab cassandra/node2.example.com
ktadd -k /tmp/node2.keytab HTTP/node2.example.com
```

5. Optional: Use the `klist` command to view your principals and keytabs:

Node1:

```
$ sudo klist -e -kt /var/tmp/dse.keytab
```

Keytab name: FILE:/tmp/dse.keytab	KVNO	Timestamp	Principal
	- - - -		
	2	14/02/16 22:03	HTTP/node1FQDN@YOUR_REALM (des3-cbc-sha1)
	2	14/02/16 22:03	HTTP/node1FQDN@YOUR_REALM (arcfour-hmac)
	2	14/02/16 22:03	HTTP/node1FQDN@YOUR_REALM (des-hmac-sha1)
	2	14/02/16 22:03	HTTP/node1FQDN@YOUR_REALM (des-cbc-md5)
	2	14/02/16 22:03	cassandra/node1FQDN@YOUR_REALM (des3-cbc-sha1)
	2	14/02/16 22:03	cassandra/node1FQDN@YOUR_REALM (arcfour-hmac)
	2	14/02/16 22:03	cassandra/node1FQDN@YOUR_REALM (des-hmac-sha1)
	2	14/02/16 22:03	cassandra/node1FQDN@YOUR_REALM (des-cbc-md5)

where: `-e` displays the encryption type and `-kt` displays the keytab file and its timestamp.

6. Copy the node-specific keytab files from the KDC machine to the nodes:

```
$ scp /tmp/node1.keytab dse_user@node1.FQDN:/etc/dse/
$ scp /tmp/node2.keytab dse_user@node2.FQDN:/etc/dse/
```

Example:

```
$ scp /tmp/node1.keytab cassandra@node1.example.com:/etc/dse/
$ scp /tmp/node2.keytab cassandra@node2.example.com:/etc/dse/
```

7. On each node, change the name of the keytab file to `dse.keytab`.

Make the file names identical across all the nodes to ensure that the entry in each node's `dse.yaml` is the same.

Example:

```
$ hostname --fqdn
node1.example.com
$ mv /etc/dse/node1.keytab /etc/dse/dse.keytab
```

8. Change the permissions on `dse.keytab` so that only the `dse_user` user can read and write to the keytab file. For example:

```
$ sudo chown cassandra:cassandra /etc/dse/dse.keytab
$ sudo chmod 600 /etc/dse/dse.keytab
```

9. To use a Kerberos non-default REALM with Hadoop, you must specify mapping rules to map the Kerberos principal to the local UNIX user name. Add this configuration key to the `resources/hadoop/conf/dse-core.xml` file:

```
<property>
    <name>hadoop.security.auth_to_local</name>
    <value>
```

```

RULE: [1:$1] (.* )s/.* /\${username} /
DEFAULT
</value>
</property>
```

The default location of the `dse-core.xml` Hadoop configuration file depends on the type of installation:

Installer-Services and Package installations	<code>/etc/dse/hadoop/conf</code>
Installer-No Services and Tarball installations	<code>install_location/resources/hadoop/conf/</code>

What to do next

[Enabling DataStax Enterprise for Kerberos authentication](#)

Configuring DataStax Enterprise for Kerberos authentication

How to add the Kerberos authenticator to `cassandra.yaml` and Kerberos options to `dse.yaml`.

Procedure

1. On each node, edit the `cassandra.yaml` file to set the authenticator to the Kerberos Authenticator.

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

2. Make sure the `rpc_address` and `listen_address` options in `cassandra.yaml` are set to the IP address or hostname that matches the hostname in DNS (which is the same as the FQDN (Fully Qualified Domain Name) portion of the service principals created in the earlier step from `kadmin`). They should not be set to `localhost`.

```
rpc_address: 1.2.3.4
listen_address: 1.2.3.4
```

3. On each node, edit the `dse.yaml` file and enter the correct Kerberos options.

The options are located in the `kerberos_options` section. See the [table](#) below.

```
kerberos_options:
  keytab: path_to_keytab/dse.keytab
  service_principal: dse_user/_HOST@REALM
  http_principal: HTTP/_HOST@REALM
  qop: auth
```

Example:

```
kerberos_options:
  keytab: /etc/dse/dse.keytab
  service_principal: cassandra/_HOST@EXAMPLE.COM
  http_principal: HTTP/_HOST@EXAMPLE.COM
  qop: auth
```

Table: Kerberos options

Option	Description
keytab	The keytab file must contain the credentials for both of the fully resolved principal names, which replace _HOST with the FQDN of the host in the service_principal and http_principal settings. The UNIX user running DataStax Enterprise must also have read permissions on the keytab.
service_principal	Sets the principals that the Cassandra and DSE Search (Solr) processes run under. Use the form dse_user/_HOST@REALM, where <ul style="list-style-type: none"> • Installer-Services and Package installations: cassandra • Package installations: the name of the UNIX user that starts the service Leave _HOST as is. This variable is used in dse.yaml. DataStax Enterprise automatically substitutes the FQDN of the host where it runs. Credentials must exist for this principal in the keytab file and readable by the user that Cassandra runs as, usually cassandra. The service_principal must be consistent everywhere: <ul style="list-style-type: none"> • dse.yaml file • keytab • cqlshrc file (where it is separated into the service/hostname)
http_principal	Set REALM to the name of your Kerberos realm. In the Kerberos principal, REALM must be all uppercase. Leave _HOST as is. This variable is used in dse.yaml. DataStax Enterprise automatically substitutes the FQDN of the host where it runs. Credentials must exist for this principal in the keytab file and readable by the user that Cassandra runs as, usually cassandra. The http_principal is used by the application container, which is tomcat, and used to run Solr. The web server uses GSS-API mechanism (SPNEGO) to negotiate the GSSAPI security mechanism (Kerberos). To set up password authentication for a DSE Search node, see Running the Wikipedia search demo on a secure cluster .
qop	A comma-delimited list of Quality of Protection (QOP) values that clients and servers can use for each connection. The client can have multiple QOP values, while the server can have only a single QOP value. The valid values are: <ul style="list-style-type: none"> • auth - Default: Authentication only. • auth-int - Authentication plus integrity protection for all transmitted data. • auth-conf - Authentication plus integrity protection and encryption of all transmitted data. Encryption using auth-conf is separate and independent of whether encryption is done using SSL. If both auth-conf and SSL are enabled, the transmitted data is encrypted twice. DataStax recommends choosing only one method and using it for both encryption and authentication.

4. If the cluster will run Hadoop in a Kerberos secure environment, change the task-controller file ownership to root and access permissions to 4750. For example:

```
sudo chown root /usr/share/dse/resources/hadoop/native/Linux-amd64-64/bin/
task-controller
sudo chmod 4750 /usr/share/dse/resources/hadoop/native/Linux-amd64-64/bin/
task-controller
```

Package installations only: The default location of the task-controller file should be /usr/share/dse/resources/hadoop/native/Linux-amd64-64/bin/task-controller.

- After the cluster is up and running, change the replication strategy and default replication factor for the system_auth keyspace. See [Configuring system_auth keyspace replication](#).

DataStax recommends configuring system_auth keyspaces for fault tolerance (in case of failure). In a multi-node cluster, if the node storing the user data goes down, the default replication factor of 1 for the system_auth keyspace precludes logging into any secured node.

Creating Kerberos users

Steps to create Kerberos users.

Procedure

- In the cassandra.yaml file, set the password authenticator:

```
authenticator: org.apache.cassandra.auth.PasswordAuthenticator
```

- Start cqlsh and login using the superuser name and password:

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

- Create the other Kerberos users, such as user@REALM. Be sure to create at least one user with superuser privileges.

```
cqlsh> create user 'newuser@YOURREALM' with password 'secret123' SUPERUSER;
```

- (Highly recommended.) Remove the cassandra user. See [DROP USER](#).

- Re-enable Kerberos authorization in the cassandra.yaml file:

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

- When you create new users with Kerberos authentication enabled, you do not specify a password for the new user.

```
cqlsh> create user 'newuser@YOURREALM';
```

Enabling and disabling Kerberos security

After setting up Kerberos users, you can turn Kerberos authorization on and off by adding and removing the Kerberos as an authentication scheme in the cassandra.yaml file:

Enabling cqlsh to use Kerberos

Install required packages to use cqlsh with Kerberos.

Prerequisites

To use cqlsh with a Kerberized cluster, you must install the python-pure-sasl packages and PyKerberos.

- The python-pure-sasl package is a pure Python client-side SASL (Simple Authentication and Security Layer) implementation.
- The PyKerberos package is a high-level wrapper for Kerberos (GSSAPI) operations.

Procedure

To use cqlsh with Kerberos:

1. Set up Kerberos using the guidelines in [Kerberos guidelines](#) on page 303.
2. Ensure that the Kerberos client is installed and configured in your Kerberos realm:

RHEL-based platforms

```
$ yum install krb5-workstation krb5-libs krb5-auth-dialog
```

Debian-based platforms

```
$ sudo apt-get install krb5-user
```

Mac OS X

See the documentation [MIT Kerberos Consortium](#).

3. Install pure-sasl:

```
$ sudo pip install pure-sasl
```

4. Install PyKerberos. Use only the python2[67]-kerberos RPM provided by rpm.datastax.com repository for RedHat.

RHEL 6

```
$ sudo yum install python26-kerberos
```

RHEL 7

```
$ sudo yum install python27-kerberos
```

Note: RedHat EL7 might install a python-kerberos RPM that is distributed on its own official repository, but it is not compatible with cqlsh kerberos. You must use the python27-kerberos RPM from the rpm.datastax.com repository.

Debian-based platforms

```
$ sudo apt-get install python-kerberos
```

Other platforms

```
$ sudo pip install kerberos
```

5. You must [create the Kerberos user](#), such as user@REALM.
6. Create a `cqlshrc` file in `~/.cassandra` or client program `~/.cassandra` directory.
7. Specify the hostname when you login. When using Kerberos, the host is required.

```
$ cqlsh [options] host [port]
```

Enabling dsetool to use Kerberos

To enable dsetool commands to use Kerberos authentication, use one of these options:

Using the `~/.dserc` file

Create or edit the `~/.dserc` file in your DataStax Enterprise home directory and add the following entries:

```
sasl_protocol=service_name
login_config=path_to_login_config
```

Command line options

Specify the service name and JAAS configuration file on the command line:

```
-Ddse.sasl.protocol=service_name
-Djava.security.auth.login.config=path_to_login_config
```

where:

- *service_name* is the service name component of the *service_principal* that is defined in the `dse.yaml` file
- *path_to_login_config* is the JAAS configuration file with the following options declared in it:

```
DseClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useTicketCache=true
    renewTGT=true;
};
```

Using Kerberos authentication with Sqoop

Sqoop can use Kerberos user authentication when connecting to DataStax Enterprise nodes.

Prerequisites

- [Create a Kerberos principal user](#) for the realm.
- Add the principal's user to Cassandra on the node on which Sqoop will run.

Procedure

1. On the machine running Sqoop, create a ticket for the Kerberos principal.

```
$ kinit principal_name
```

Enter the principal's password when prompted.

2. Create a JAAS configuration file to enable Kerberos for DataStax Enterprise.

```
DseClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useTicketCache=true
    renewTGT=true;
};
```

3. In a Sqoop options file, add the Kerberos configuration options that are customized for your environment:

```
--cassandra-host
FQDN_of_Cassandra_host
--cassandra-enable-kerberos
--cassandra-kerberos-config-path
JAAS_configuration_filepath
--cassandra-kerberos-service-principal
principal_name/HOST@REALM
```

4. Run Sqoop with the options file.

```
$ bin/dse sqoop --options-file path_to_Sqoop_options_file
```

Authenticating a cluster with LDAP

The Lightweight Directory Access Protocol (LDAP) is a standard way of authenticating users across applications. DataStax Enterprise supports LDAP authentication for external LDAP services.

When you enable LDAP authentication in DataStax Enterprise, users that are managed by external LDAP servers can be authenticated by DataStax Enterprise. Authenticated users can then be authorized to access Cassandra objects, as described in [Managing object permissions using internal authorization](#) on page 346.

LDAP authentication is supported in the following DataStax Enterprise components:

- CQL
- DSE Analytics
 - Spark
 - Hadoop
 - Hive
 - Pig
- Sqoop

LDAP authentication is supported by the following component:

- DevCenter 1.5.0 and greater

LDAP authentication is not supported in the following components:

- OpsCenter versions earlier than 5.2
- Mahout
- DevCenter earlier than 1.5.0
- Solr using the HTTP interface

Note: Because of the stateless property of HTTP when using LDAP, each issued request verifies credentials and can significantly degrade performance when using the HTTP interface. Use [Kerberos](#) or [search using CQL](#) instead.

Enabling LDAP authentication

LDAP authentication is enabled by configuring DataStax Enterprise to use an external LDAP server.

Prerequisites

You must have a properly configured LDAP v3 server running. The supported LDAP servers are:

- Microsoft Active Directory:
 - Windows 2008
 - Windows 2012
- OpenLDAP 2.4.x
- Oracle Directory Server Enterprise Edition 11.1.1.7.0

Procedure

1. Open the `cassandra.yaml` file in a text editor and set the authenticator to `com.datastax.bdp.cassandra.auth.LdapAuthenticator`.

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

- Open the dse.yaml file in a text editor and set the configuration for your LDAP server. The settings are only used if the authenticator is set to com.datastax.bdp.cassandra.auth.LdapAuthenticator in cassandra.yaml.

Option	Description
server_host	The host name of the LDAP server.
server_port	The port on which the LDAP server listens. The default value is 389.
	The default SSL port for LDAP is 636.
search_dn	The username of the user that is used to search for other users on the LDAP server.
search_password	The password of the search_dn user.
use_ssl	Set to true to enable SSL connections to the LDAP server. If set to true, you might need to change server_port to the SSL port of the LDAP server. The default value is false.
use_tls	Set to true to enable TLS connections to the LDAP server. If set to true, you might need to change the server_port to the TLS port of the LDAP server. The default value is false.
truststore_path	The path to the trust store for SSL certificates.
truststore_password	The password to access the trust store.
truststore_type	The type of trust store. The default value is jks.
user_search_base	The search base for your domain, used to look up users. Set the ou and dc elements for your LDAP domain. Typically this is set to ou=users,dc=domain,dc=top_level_domain. For example, ou=users,dc=example,dc=com. Active Directory uses a different search base, typically CN=search,CN=Users,DC=ActDir_domname,DC=internal. For example, CN=search,CN=Users,DC=example-sales,DC=internal.
user_search_filter	The search filter for looking up user names. The default setting is (uid={0}). When using Active Directory set the filter to (SAMAccountName={0}).
credentials_validity_in_ms	The duration period in milliseconds for the credential cache. To disable the cache, set it to 0. The cache is disabled by default.
	With the cache enabled, DataStax Enterprise stores the user credentials locally during the specified time. Binding to a remote LDAP server takes time and resources, so enabling a credential cache usually results in faster performance following the initial authentication phase. Changes in user credentials on the LDAP server are not

Option	Description
search_validity_in_seconds	reflected in DataStax Enterprise during the cache period.
	The duration period in milliseconds for the search cache. To disable the cache, set it to 0. The cache is disabled by default.
	Enabling a search cache improves performance by reducing the number of requests that are sent to the LDAP server. Changes in user data on the LDAP server are not reflected during the cache period.
connection_pool	The configuration settings for the connection pool for making LDAP requests.
max_active	The maximum number of active connections to the LDAP server. The default value is 8.
max_idle	The maximum number of idle connections in the pool awaiting requests. The default value is 8.

```

ldap_options:
  server_host: localhost
  server_port: 389
  search_dn: cn=Admin
  search_password: secret
  use_ssl: false
  use_tls: false
  truststore_path:
  truststore_password:
  truststore_type: jks
  user_search_base: ou=users,dc=example,dc=com
  user_search_filter: (uid={0})
  credentials_validity_in_ms: 0
  connection_pool:
    max_active: 8
    max_idle: 8

```

3. Repeat these steps on each node in the cluster.

Creating LDAP users

DataStax Enterprise automatically creates a `cassandra` superuser but it is unlikely that this user will be available on the remote LDAP service. Use the following steps to create a superuser and other users in Cassandra.

Procedure

1. In the `cassandra.yaml` file, set the authenticator to `org.apache.cassandra.auth.PasswordAuthenticator`.

```
authenticator: org.apache.cassandra.auth.PasswordAuthenticator
```

2. Start `cqlsh` and login using the superuser name and password.

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

3. [Create the other LDAP users](#) but give them blank passwords. Be sure to create at least one with superuser privileges. These users need to match the available users in the remote LDAP service.
4. Re-enable LDAP authorization in the `cassandra.yaml` file.

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

5. Login as the new superuser and delete the default `cassandra` user.

Note: This step is highly recommended to improve the security DataStax Enterprise.

6. [Enable LDAP authentication](#) on each node in the cluster.

Encryption

DataStax Enterprise supports encryption for in-flight data and at-rest data.

Note: For Spark security, see [Spark SSL encryption](#) on page 320.

Encrypting sensitive property values

DataStax recommends encrypting sensitive properties in the `dse.yaml` and `cassandra.yaml` on-disk configuration files.

Procedure

1. In `dse.yaml`, verify that the `config_encryption_active` property is false.

```
config_encryption_active: false
```

2. In the `dse.yaml` file, define where the system keys are stored on disk. Verify or set the `system_key_directory` property. The default value is `/etc/dse/conf`.

3. Generate a system key.

On-server:

```
$ dsetool createsystemkey cipher strength system_key_file
```

Off-server

```
$ dsetool createsystemkey cipher strength system_key_file -kmip=kmip_groupname
```

For example:

```
$ dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 system_key_file
```

where:

- `system_key_file` is a file with a unique file name
- `cipher` is a valid `cipher_algorithm`
- `strength` is the secret key strength

See [Encryption/compression options and sub-options](#)

You can create a global encryption key in the location that is specified by `system_key_directory` in the `dse.yaml` file. This default global encryption key is used when the `system_key_file` subproperty is not specified.

4. Copy the returned value of the `dsetool createsystemkey` command.

- In the `dse.yaml` file, paste the value that was returned to set the `config_encryption_key_name` property:

```
config_encryption_key_name: Sa9xOVaym7bddjXUT/eeOQ==
```

- Use the `dsetool encryptconfigvalue` command for each property that you want to encrypt. This command takes no arguments and prompts for the value to encrypt.

One at a time, enter the encrypted value that is output for each property into the `dse.yaml` or `cassandra.yaml` configuration file. Ensure that each property is encrypted or commented out.

<code>dse.yaml</code>	<ul style="list-style-type: none"> <code>ldap_options.search_password</code> <code>ldap_options.truststore_password</code>
<code>cassandra.yaml</code>	<ul style="list-style-type: none"> <code>server_encryption_options.keystore_password</code> <code>server_encryption_options.truststore_password</code> <code>client_encryption_options.keystore_password</code> <code>client_encryption_options.truststore_password</code> <code>ldap_options.truststore_password</code>

- In `dse.yaml`, set the `config_encryption_active` property to true:

```
config_encryption_active: true
```

When the `config_encryption_active` property is true, the configuration values must be encrypted or commented out.

- [Start dse](#).

Client-to-node encryption

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

DSE Search

When you enable SSL, the authentication/authorization filters are automatically enabled in the Solr `web.xml` file and an SSL connector in Tomcat is configured. You do not have to change your `web.xml` or `server.xml` files.

Note: If the TomcatSolrRunner doesn't find a connector in `resources/tomcat/conf/server.xml` it creates a default connector. The default connector binds to the `rpc_address` in `cassandra.yaml`.

To satisfy specific security requirements with SSL, you can [change the IP address for client connections to DSE Search](#). For example, to isolate a subnet.

Procedure

Perform these steps on each node.

- In the `cassandra.yaml` file, in the `client_encryption_options` section:

- To enable encryption, set `enabled` to true.
- Set the paths to your `.keystore` and `.truststore` files.
- Provide the passwords that were used when generating the keystore and truststore.
- To enable client certificate authentication, set `require_client_auth` to true.

- For DSE Search nodes and DSE Analytics nodes with Spark: set the truststore entries.

```
client_encryption_options:
  enabled: true
  keystore: resources/dse/conf/.keystore ## Path to your .keystore file
  keystore_password: keystore password ## Password that you used to
  generate the keystore
  store_type: JKS
  truststore: resources/dse/conf/.truststore ## Path to
  your .truststore
  truststore_password: truststore password ## Password that you used to
  generate the truststore
  protocol: ssl
  require_client_auth: true
  cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA,
  TLS_RSA_WITH_AES_256_CBC_SHA]
```

For information about using Kerberos with SSL, see [Using Kerberos and SSL at the same time](#). To encrypt the truststore and keystore passwords with KMIP, see [Configuring encryption using off-server encryption keys](#) on page 327.

2. If you use strong cipher suites to ensure secure client-to-node communication, you must [install JCE](#) to ensure support for all encryption algorithms. Some of the cipher suites in the default set of `server_encryption_options` in `cassandra.yaml` are included only in the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files. To ensure support for all encryption algorithms, install the JCE Unlimited Strength Jurisdiction Policy Files.
3. If you do not use the JCE Unlimited Strength Jurisdiction Policy, make sure that your ticket granting principal does not use [AES-256](#).
If your ticket granting principle uses AES-256, you might see a warning like this in the logs:

```
WARN [StreamConnectionEstablisher:18] 2015-06-22 14:12:18,589
  SSLFactory.java (line 162) Filtering out
  TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_
  as it isn't supported by the socket
```

Node-to-node encryption

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

SSL settings for node-to-node encryption

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

On each node, under `encryption_options`:

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

The available inter-node options are:

- `all`
- `none`
- `dc` - Cassandra encrypts the traffic between the data centers.

- rack - Cassandra encrypts the traffic between the racks.

```
encryption_options:
  internode_encryption: internode_option
  keystore: resources/dse/conf/.keystore
  keystore_password: keystore password
  truststore: resources/dse/conf/.truststore
  truststore_password: truststore password
  require_client_auth: true or false
```

To encrypt the truststore and keystore passwords with KMIP, see [Configuring encryption using off-server encryption keys](#) on page 327.

Spark SSL encryption

Communication between Spark clients and clusters as well as communication between Spark nodes can be encrypted using SSL. You must configure encryption on each node in your cluster.

Configure Spark SSL encryption on the server-side by editing `dse.yaml`, and for Spark clients by editing `spark-defaults.conf` in the Spark configuration directory.

The default location of the Spark configuration files depends on the type of installation:

Installer-Services and Package installations	<code>/etc/dse/spark/</code>
Installer-No Services and Tarball installations	<code>install_location/resources/spark/conf/</code>

Spark SSL encryption is limited to Akka control messages and file sharing. It does not encrypt RDD data exchanges or the web user interface.

Note: Using Spark with SSL encryption might result in a slight drop in Spark job performance because of the extra time to encrypt and decrypt the data and perform the extra garbage collection (GC).

Procedure

1. Open `dse.yaml` in a text editor.
2. In the `spark_encryption_options` section set the options as described below.

Option	Description
enabled	Enables or disables server-side encryption. The default is false.
keystore	The path to the keystore file, relative to the Spark configuration directory. The default keystore is a file named <code>.keystore</code> located in the Spark configuration directory.
keystore_password	The password used to access the keystore. The default password is <code>cassandra</code> .
truststore	The path to the truststore file, relative to the Spark configuration directory. The default truststore is a file named <code>.truststore</code> located in the Spark configuration directory.
truststore_password	The password used to access the truststore. The default password is <code>cassandra</code> .

Option	Description
protocol	The SSL protocol used when encrypting communications. The default is TLS.
cipher_suites	The cipher suites used with the protocol, enclosed in square brackets ([]) and separated by commas. The default suites are [TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_GCM_SHA384].
3. In each client, set the client encryption options in the <code>spark-defaults.conf</code> file in the Spark configuration directory.	
Option	Description
spark.ssl.enabled	Enables or disables client-side encryption. The default is false.
spark.ssl.keyStore	The path to the keystore file, relative to the Spark configuration directory. The default keystore is a file named <code>.keystore</code> located in the Spark configuration directory.
spark.ssl.keyStorePassword	The password used to access the keystore. The default password is <code>cassandra</code> .
spark.ssl.keyPassword	The password for the private key. The default password is <code>cassandra</code> .
spark.ssl.trustStore	The path to the truststore file, relative to the Spark configuration directory. The default truststore is a file named <code>.truststore</code> located in the Spark configuration directory.
spark.ssl.trustStorePassword	The password used to access the truststore. The default password is <code>cassandra</code> .
spark.ssl.protocol	The SSL protocol used when encrypting communications. The default is TLS.
spark.ssl.enabledAlgorithms	The cipher suites used with the protocol, separated by commas. The default suites are [TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_GCM_SHA384].
spark.ssl.useNodeLocalConf	Sets whether the Spark executors inherit the SSL configuration from the Spark Workers. The default is true.

Preparing server certificates

All nodes require relevant SSL certificates on all nodes. Generate SSL certificates for [client-to-node encryption](#) or [node-to-node encryption](#). When you generate the certificates for one type of encryption, you do not need to generate them again for the other: the same certificates are used for both.

When using SSL [client-to-node encryption](#), the common name (CN) in the certificate must be the hostname or IP address of the node that issues the command.

A keystore contains private keys. The truststore contains SSL certificates for each node and does not require signing by a trusted and recognized public certification authority.

Procedure

To prepare server certificates:

1. Generate the private and public key pair for the nodes of the cluster leaving the key password the same as the keystore password:

```
$ keytool -genkey -alias dse_node0 -keyalg RSA -keystore .keystore
```

For this prompt What is your first and last name?, enter the hostname or the fully qualified domain name (FQDN) of the Cassandra node for which you are generating the keys. The values that you enter for the first and last name are used to set the certificate CN (Common Name) that is used for server authentication by the client.

2. Repeat the previous step on each node using a different alias for each node.
3. Export the public part of the certificate to a separate file and copy these certificates to all other nodes.

```
$ keytool -export -alias dse_node0 -file dse_node0.cer -keystore .keystore
```

4. Add the certificate of each node to the truststore of each node, so the nodes can verify the identity of other nodes.

A prompt for setting a password for the newly created truststore appears.

```
$ keytool -import -v -trustcacerts -alias dse_node0 -file dse_node0.cer -keystore .truststore
$ keytool -import -v -trustcacerts -alias dse_node1 -file dse_node1.cer -keystore .truststore
. .
$ keytool -import -v -trustcacerts -alias dse_nodeN -file dse_nodeN.cer -keystore .truststore
```

5. Make sure .keystore is readable only by the DataStax Enterprise daemon and not by any user of the system.

Spark security

DataStax Enterprise supports Password and LDAP authentication, Kerberos, client-to-node encryption through SSL security in Spark, and Spark SSL encryption. To use internal authentication, see [Running spark-submit job with internal authentication](#).

Password and LDAP authentication

You can pass Cassandra credentials to Spark by setting the following properties in the [Spark configuration](#) object `SparkConf` before creating `Spark Context`:

- `cassandra.username`
- `cassandra.password`

For DataStax Enterprise Spark applications and tools, you can setup a [set up a .dserc file](#) or use the Spark authentication commands to provide the login credentials.

The following examples show how to include Cassandra credentials in your applications:

Example: Passing hard-wired Cassandra credentials

```
import com.datastax.bdp.spark.DseSparkConfHelper._
import org.apache.spark.{SparkConf, SparkContext}

object AuthenticationExample extends App {

    def createSparkContext() = {
        val myJar =
            getClass.getProtectionDomain.getCodeSource.getLocation.getPath
    }
}
```

```

val conf = new SparkConf()
    .setAppName("Authentication example")
    .setMaster("local")
    .setJars(Array(myJar))
    .set("cassandra.username", "cassandra")
    .set("cassandra.password", "cassandra")
    .forDse

    new SparkContext(conf)
}

val sc = createSparkContext()

// ...

sc.stop()
}

```

Example: Prompting for Cassandra credentials

```

import com.datastax.bdp.spark.DseSparkConfHelper._
import org.apache.spark.{SparkConf, SparkContext}

object AuthenticationExample extends App {

def createSparkContext() = {
/*
  -Dcassandra.username=... and -Dcassandra.password=... arguments will be
  copied to system properties and removed
  from the args list
*/

  val args = setSystemPropertiesFromArgs(this.args)
  val myJar =
getClass.getProtectionDomain.getCodeSource.getLocation.getPath

  val conf = new SparkConf()
    .setAppName("Authentication example")
    .setMaster("local")
    .setJars(Array(myJar))
    .forDse

    new SparkContext(conf)
}

val sc = createSparkContext()

// ...

sc.stop()
}

```

You can [configure a number of parameters](#) to run your own Spark applications with DataStax Enterprise.

Providing credentials for Cassandra in a Spark application

This procedure describes how to write a Spark application that uses password authentication. The `SparkContext` is not authenticated. The authentication pertains to connecting Spark to Cassandra, not authenticating Spark components between each other.

1. Include the instruction in your application to import the `DseSparkConfHelper` package.

```
import com.datastax.bdp.spark.DseSparkConfHelper._
```

2. Set authentication properties.

```
System.setProperty("cassandra.username", "xxx")
System.setProperty("cassandra.password", "yyy")
```

3. Create a new `SparkContext`, passing `SparkConf.forDSE` as an argument. The `.forDSE` method extends the `SparkConf` object for DataStax Enterprise.

```
new SparkContext(args(0), "PortfolioDemo",
new SparkConf().setJars(Array(myJar)).forDse)
```

If the `~/.dserc` file is not configured, use the `DseSparkConfHelper` method to find properties in the format `Dprop=value` and pass them to the `System` properties automatically. You call `setSystemPropertiesFromArgs(args)` where `args` are command line arguments passed to the main method.

Kerberos authentication

[Kerberos authentication](#) applies to connecting Spark to Cassandra, not authenticating Spark components between each other. The Spark Web UI is not secured and might show the Spark configuration, including delegation token, when using Kerberos.

Spark to Cassandra SSL encryption

Client-to-node encryption protects data in flight for the Spark Executor to Cassandra connections by establishing a secure channel between the client and the coordinator node. SSL is fully distributed and does not require setting up a shared authentication service. You need to [prepare server certificates](#) and [enable client-to-node SSL](#).

Spark SSL encryption

Spark internode and client-to-cluster communication can also be encrypted using SSL by enabling it server-side in `dse.yaml` and client-side in the Spark configuration file `spark-defaults.conf`. See [Spark SSL encryption](#) on page 320 for details.

Security limitations

DataStax Enterprise is limited in securing Spark data:

- Client-to-node encryption using SSL is supported for Spark Executor to Cassandra connections only.
- Spark executors run under the same user account as DataStax Enterprise.
- The Spark Web UI is not secured and might show the Spark configuration, including username, password, or delegation token when Kerberos is used.
- DataStax Enterprise provides internal authentication support for some Hadoop tools and for connecting Spark to Cassandra, not authenticating Spark components between each other.

DataStax recommends the following security practices:

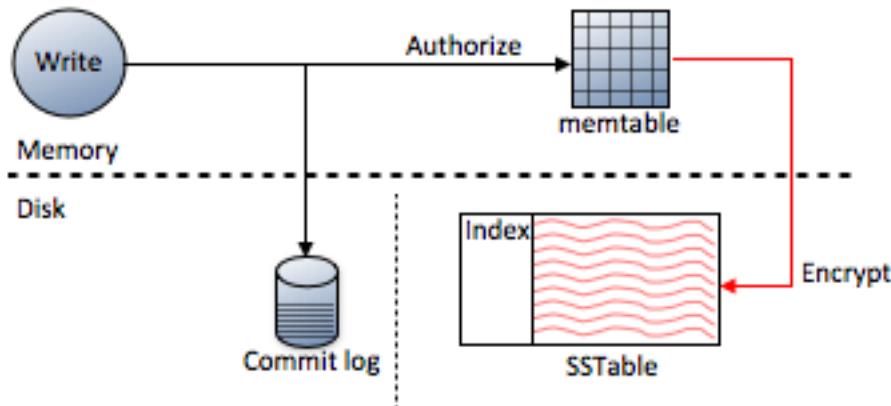
- Expose Spark components to trusted users only.
- Allow only trusted users to access the file system.

Because Spark executors run under the same user account as DataStax Enterprise, an unapproved user can execute a potentially malicious Spark program that can access the file system on the nodes. System files as well as Cassandra SSTables are vulnerable. Users who cannot access Cassandra files on the node, but who you entrust with your file system, can access temporary directories where RDD fragments are stored temporarily. Having sufficient privileges, a user can also execute malicious system commands.

Using password authentication, LDAP, or Kerberos to secure Cassandra makes no sense unless you restrict direct access to the file system.

Transparent data encryption

Transparent data encryption (TDE) protects at rest data. At rest data is data that has been stored on disk.



As shown in the diagram, data stored in the commit log is not encrypted. If you need commit log encryption, store the commit log on an OS-level encrypted file system using a security product such as Vormetric. Data can be encrypted using different algorithms, or not at all. SSTable data files are immutable after they have been flushed to disk and encrypted only once when they are written to disk.

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

Requirements

TDE requires a secure local file system to be effective. Encryption certificates are stored [off-server with KMIP encryption](#) or locally with [on-server encryption](#).

TDE limitations and recommendations

Data is not directly protected by TDE when you access the data using the following utilities.

Utility	Reason utility is not encrypted
json2sstable	Operates directly on the SSTables.
nodetool	Uses only JMX, so data is not accessed.
sstable2json	Operates directly on the SSTables.
sstablekeys	Operates directly on the SSTables.
sstableloader	Operates directly on the SSTables.
sstablescrub	Operates directly on the SSTables.

Compression and encryption introduce performance overhead.

TDE options

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

Configuring encryption using local encryption keys

To encrypt data using encryption keys that are stored locally, use the dse command to create a system key for encryption. Next, copy the system key to the other nodes in the cluster. The entire cluster uses the system key to decrypt SSTables for operations such as repair. You also use the system key during upgrading and restoring SSTables that might have been corrupted.

Procedure

1. Back up SSTables.
2. Set the system_key_directory.
 - On a packaged installation, accept the default system_key_directory /etc/dse/conf. Go to the next step to set permissions on the directory.
 - On a tarball installation, optionally change the directory on each node in the cluster from /etc/dse/conf to another directory, or skip this step and adjust permissions as described in the next step. You must configure the path to the system key to relocate the key to a directory that you have permission to access.
 - Navigate to *install-directory/resources/dse/conf*.
 - Open the dse.yaml file for editing.
 - Change the path of the system_key_directory to the path of a directory that you have permission to access.
3. Set permissions on the system_key_directory to give rights to change the keytab file only to the user/group running DataStax Enterprise. JNA takes care of setting these permissions.
4. Ensure that the user who encrypts data has been granted **ALTER permission** on the table that contains the data to be encrypted. You can use **LIST PERMISSIONS** to view the permissions that are granted to a user.
5. Create a system key using the `dsetool createsystemkey` command.

For example:

```
$ dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 system_key
```

6. Restart the cluster.
7. Copy the created key to the system_key_directory on each node in the cluster.
8. **Set encryption options** when you create a table or alter an existing table.
Tables are encrypted when Cassandra stores the tables on disk as SSTables.
9. Rewrite all SSTables using `nodetool upgradessstables --include-all-sstables` to immediately store the tables on disk.
10. After encrypted SSTables are flushed to disk, you can verify that the `dse_system` keyspace and `encrypted_keys` table exist:

```
cqlsh:mykeyspace> DESCRIBE KEYSPACES;
system  dse_system  mykeyspace  system_traces
```

On all nodes, the system key appears when selected from the `dse_system.encrypted_keys` table:

```
cqlsh:mykeyspace> SELECT * FROM dse_system.encrypted_keys;
key_file  | cipher | strength | key_id          | key
-----+-----+-----+-----+-----+
system_key | AES   | 128    | 2e4ea4a0-... | uyBEGhX...
```

Configuring encryption using off-server encryption keys

Configure KMIP (Key Management Interoperability Protocol) encryption to use encryption keys that are stored on another server. In addition to encrypting table data, you can optionally encrypt passwords in configuration files and sensitive information in system tables. Use [OpsCenter](#) to configure an alert to monitor KMIP server status.

Procedure

1. Perform host configuration for one or more KMIP key server groups.
 - a) Configure the KMIP key manager and authorize each DataStax Enterprise node to the KMIP key server group. Consult the KMIP key server documentation.
 - b) On each DataStax Enterprise node, open the `dse.yaml` file in a text editor and configure the KMIP key server group or key server groups in the `kmip_hosts` section. Configure options for a `kmip_groupname` section for each KMIP key server or group of KMIP key servers. Using separate key server configuration settings allows use of different key servers to encrypt table data, and eliminates the need to enter key server configuration information in DDL statements and other configurations.

Option	Description
<code>hosts</code>	A comma-separated list of hosts[:port] for the KMIP key server. There is no load balancing. In failover scenarios, failover occurs in the same order that servers are listed. For example: <code>hosts: kmip1.yourdomain.com, kmip2.yourdomain.com</code>
<code>keystore_path</code>	The path to a java keystore that identifies the DSE node to the KMIP key server. For example: <code>/path/to/keystore.jks</code>
<code>keystore_type</code>	The type of key store. The default value is <code>jks</code> .
<code>keystore_password</code>	The password to access the key store.
<code>truststore_path</code>	The path to a java truststore that identifies the KMIP key server to the DataStax Enterprise node. For example: <code>/path/to/truststore.jks</code>
<code>truststore_type</code>	The type of trust store. The default value is <code>jks</code> .
<code>truststore_password</code>	The password to access the trust store.
<code>key_cache_millis</code>	Milliseconds to locally cache the encryption keys that are read from the KMIP hosts. The longer the encryption keys are cached, the fewer requests are made to the KMIP key server, but the longer it takes for changes, like revocation, to propagate to the DSE node. Default: 300000.
<code>timeout</code>	Socket timeout in milliseconds. Default: 1000.

This example shows configuration settings for Vormetric and Thales key servers:

```
kmip_hosts:
  vormetricgroup:
    hosts: vormetric1.mydomain.com, vormetric2.mydomain.com,
           vormetric3.mydomain.com
    keystore_path: path/to/kmip/keystore.jks
    keystore_type: jks
    keystore_password: password
    truststore_path: path/to/kmip/truststore.jks
    truststore_type: jks
    truststore_password: password
```

```

thalesgroup:
  hosts: thales1.mydomain.com, thales2.mydomain.com
  keystore_path: path/to/kmip/keystore.jks
  keystore_type: jks
  keystore_password: password
  truststore_path: path/to/kmip/truststore.jks
  truststore_type: jks
  truststore_password: password

```

2. On each DataStax Enterprise node, confirm communication with the KMIP key server and restart the node.
- a) Use the dsetool utility to confirm communication.

```
$ dsetool managekmip list kmip_groupname
```

- b) After communication between the DataStax Enterprise node and the KMIP key server or servers is verified, restart the node. Repeat this step on each node.
The DataStax Enterprise node will not start if it is unable to connect to the configured KMIP key server.

3. Set and use KMIP as the encryption key provider.

- a) Set [KMIP encryption options](#) when you create a table or alter an existing table.
- b) Optional: Configure password encryption to encrypt stored passwords in the configuration files. Use dsetool to generate the required URL:

```
$ dsetool createsystemkey 'AES/ECB/PKCS5' 128 -k kmip_hosts_value
```

Edit the dse.yaml file in a text editor. For the config_encryption_key_name property, paste the URL that is returned from the dsetool createsystemkey utility. See [Encrypting sensitive property values](#) on page 317.

- c) Optional: Configure system table encryption to encrypt system tables that contain sensitive information. Edit the dse.yaml file in a text editor. In the system_info_encryption section, comment out key_name, and uncomment or add key_provider and kmip_host:

```

system_info_encryption:
  enabled: false
  cipher_algorithm: AES
  secret_key_strength: 128
  chunk_length_kb: 64
  #key_name: system_table_keytab
  key_provider: KmipKeyProviderFactory
  kmip_host: <kmip_groupname>

```

Use key_provider: KmipKeyProviderFactory only to specify a KMIP key server.

Encrypting table data with KMIP encryption keys

Designate transparent data encryption (TDE) on a per table or cluster-wide basis. Using encryption, your application can read and write to SSTables that use different encryption algorithms or use no encryption at all. You must login as a superuser to encrypt data. For example:

```
$ cqlsh -u cassandra -p cassandra
```

To encrypt table data using keys that are provided by a KMIP key server, without compression:

```

CREATE TABLE customers
  ...
  WITH COMPRESSION =
  { 'sstable_compression': 'Encryptor',

```

```
'key_provider': 'KmipKeyProviderFactory',
'kmip_host': 'kmip_group1',
'cipher_algorithm': 'AES/ECB/PKCS5Padding',
'secret_key_strength': 128 };
```

- 'key_provider': 'KmipKeyProviderFactory' tells the encryptor to use a KMIP key server to manage its encryption keys. Include the 'key_provider' entry only to specify to use a KMIP key server, otherwise omit this entry.
- 'kmip_host': 'kmip_group1' specifies the user-defined the KMIP key server group named kmip_group1 that is set in the kmip_hosts section in dse.yaml.

To encrypt table data using keys that are provided by a KMIP key server, and use compression, specify a compression algorithm such as the EncryptingDeflateCompressor compressor:

```
ALTER TABLE customers
...
WITH COMPRESSION =
{ 'sstable_compression': 'EncryptingDeflateCompressor',
'key_provider': 'KmipKeyProviderFactory',
'kmip_host': 'kmip_group2',
'cipher_algorithm': 'AES/ECB/PKCS5Padding',
'secret_key_strength': 128 };
```

Configuring encryption per table (TDE)

Designate transparent data encryption (TDE) on a per table or cluster-wide basis. Using encryption, your application can read and write to SSTables that use different encryption algorithms or use no encryption at all. You must login as a superuser to encrypt data. For example:

```
$ cqlsh -u cassandra -p cassandra
```

System key overview

- You can create any number of system keys using the `dsetool createsystemkey` command.
- Different tables can use different system keys.
- All of the system keys must be consistent cluster-wide at the location set by the `system_key_directory` property in `dse.yaml`.
- The system keys must have read write permissions for the user.
- To use a specific system key for any table, specify the `'system_key_file': 'system_key_filename'` subproperty in the `CREATE TABLE` or `ALTER TABLE` statement.
- A default global system key for the DataStax Enterprise cluster is used when the `'system_key_file': 'system_key_filename'` subproperty is not specified.
 - The global system key is an encryption key that you create in the location that is specified for the `system_key_directory` property in the `dse.yaml` file.
 - You can use any name for the global key file. With the `dsetool createsystemkey` command, use the `file` option to specify a key name. For example: `dsetool createsystemkey file globalsystemkey` If you do not specify the key name, the default key name is `system_key`. The default key name is not configurable.

The default global system key file is inserted in the `dse_system.encrypted_keys` table that is used across the cluster.

Encrypting table data with encryption and compression

You can use a single CREATE TABLE or ALTER TABLE statement to set encryption and compression. The single CQL statement is:

```
CREATE TABLE users
...
WITH compression =
{ 'sstable_compression' : 'EncryptingSnappyCompressor',
  'cipher_algorithm' : 'AES/ECB/PKCS5Padding',
  'secret_key_strength' : 128,
  'chunk_length_kb' : 128 };
```

Note: Designating data for encryption using ALTER TABLE does not encrypt existing SSTables, just new SSTables that are generated.

Encryption/compression options and sub-options

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

The high-level container options for encryption and/or compression that can be used in the CREATE TABLE and ALTER TABLE statements are:

Encryptor	Encrypts table data
EncryptingDeflateCompressor	Encrypts table data and uses Deflate compression algorithm
EncryptingSnappyCompressor	Encrypts table data and uses Snappy compression algorithm
DeflateCompressor	Does not encrypt table data, uses Deflate compression algorithm
SnappyCompressor	Does not encrypt table data, uses Snappy compression algorithm
LZ4Compressor (default)	Does not encrypt table data, uses LZ4 compression algorithm

If defining a table with the Encryptor encryptor, set the young generation heap (-Xmn) parameter to a larger space to improve garbage collection (GC).

For example if running [cassandra-stress](#), set : -Xmn1600M. The encryption and compression sub-options are:

cipher-algorithm sub-option

When Java Cryptography Extension ([JCE](#)) is installed, the cipher_algorithm options and acceptable secret_key_strength values for the algorithms are:

cipher_algorithm	secret_key_strength
AES/CBC/PKCS5Padding	128, 192, or 256
AES/ECB/PKCS5Padding	128, 192, or 256
DES/CBC/PKCS5Padding	56
DESe/CBC/PKCS5Padding	112 or 168

cipher_algorithm	secret_key_strength
Blowfish/CBC/PKCS5Padding	32-448
RC2/CBC/PKCS5Padding	40-128

When JCE is installed, the following encryption options are valid:

- sstable_compression = EncryptingDeflateCompressor
- cipher_algorithm = 'AES/CBC/PKCS5Padding'
- secret_key_strength = 256
- chunk_length_kb = 128
- key_provider = KmipKeyProviderFactory
- kmip_host = kmip_group2

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

key_provider

Specify KmipKeyProviderFactory to use the KMIP key server for encryption.

kmip_host

The name of the KMIP key server group set in the kmip_hosts section in dse.yaml.

The key location sub-option

[Create global encryption keys](#) using the `dsetool createsystemkey` command. You can create a global encryption key in the location that is specified by `system_key_directory` in the `dse.yaml` file. This default global encryption key is used when the `system_key_file` subproperty is not specified.

To use a specific encryption key, use:

```
'system_key_file': 'name_of_system_key_file'
```

The chunk_length_kb sub-option

On disk, SSTables are encrypted and compressed by block (to allow random reads). This subproperty of compression defines the size (in KB) of the block and is a power of 2. Values larger than the default value might improve the compression rate, but increases the minimum size of data to be read from disk when a read occurs. While the default value (64) is a good middle-ground for compressing tables, the maximum key size for Data Encryption Standard (DES) is 64 and the maximum key size for all other encryption algorithms is 128. For stronger encryption, install Java Cryptography Extension ([JCE](#)).

Using just encryption and no compression, the size of SSTables are larger than they would be if you combined compression. During creation of the table, DataStax Enterprise looks for the system key as specified in `dse.yaml`. You do not need to specify the location of keytab file that contains the system key.

The iv_length sub-option

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

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

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

Using SolrJ Auth to implement encryption

To use the SolrJ-Auth libraries to implement encryption, follow instructions in the solrj-auth-README.md file.

The SolrJ-Auth dependencies are publicly available on the [datastax-public-releases-local](#) repository. The [SolrJ-Auth code](#) is public.

Migrating encrypted tables

Steps to migrate encrypted tables from earlier versions to DataStax Enterprise.

Procedure

1. Back up the entire keyspace that has a dse_system.encrypted_keys table.
2. Back up all system keys.
3. Upgrade the cluster to DataStax Enterprise 4.8, following instructions in the ["DataStax Upgrade Guide."](#)
4. Restart the cluster as described in the [Upgrade Guide](#).
5. Check that the dse_system.encrypted_keys table was created using the cqlsh **DESCRIBE KEYSPACES** command.
If you need to restore the dse_system.encrypted_keys table, load the table. Do not truncate or delete anything.
6. If the dse_system.encrypted_keys table was created, go to the next step; otherwise, create the table manually:

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

USE dse_system;

CREATE TABLE encrypted_keys (
    key_file text,
    cipher text,
    strength int,
    key_id timeuuid,
    key text,
    PRIMARY KEY (key_file, cipher, strength, key_id)
);
```

7. Rewrite all SSTables.

```
$ nodetool upgradesstables --include-all-sstables
```

Using cqlsh with Kerberos/SSL

To run cqlsh with Kerberos, SSL, or Kerberos and SSL, use the sample files and make changes as appropriate for your environment.

DataStax Enterprise provides a sample cqlshrc.sample.kerberos file that you can use as a starting point.

The default location of the cqlshrc.sample.kerberos file depends on the type of installation:

Installer-Services and Package installations	/usr/share/doc/dse-libcassandra/ cqlshrc.sample.kerberos
--	---

Installer-No Services and Tarball installations

install_location/resources/cassandra/conf/cqlshrc.sample.kerberos

Kerberos example

Required settings for Kerberos authentication:

```
[connection]
hostname = 192.168.1.2
port = 9042
factory = cqlshlib.kerberos.kerberos_transport_factory ;; Mandatory

[kerberos]
hostname = cassandra01.example.com ;; Mandatory
service = cassandra ;; Mandatory
principal = bill/cassandra-admin@example.com ;; Optional.
qops = auth-conf ;; Optional, see the paragraph below.
```

The `kerberos`, `hostname`, and `service` are mandatory settings and must match the values in the `dse.yaml` configuration file or in environment variables.

- In the `kerberos_options` section of the `dse.yaml` file, set `service_principal`. The `service_principal` must be consistent everywhere: in the `dse.yaml` file, present in the keytab, and in the `cqlshrc` file (where `service_principal` is separated into `service/hostname`).
- The environment variables (`KRB_HOST`, `KRB_SERVICE`, and `KRB_PRINCIPAL`) override the options that are set in `dse.yaml`.

The default (auth) is used when `qops` is not specified. On the client side, the `qops` option is a comma-delimited list of the quality of protection (QOP) values that are allowed by the client for the connection.

- The client (cqlsh) value list must contain at least one of the QOP values that are specified on the server.
- The client can have multiple QOP values, while the server can only have a single QOP value that is specified in the `dse.yaml` file.

SSL example

DataStax Enterprise provides a sample `cqlshrc.sample.ssl` file that you can use as a starting point.

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

[connection]
hostname = 127.0.0.1
port = 9042

[ssl]
certfile = ~/keys/cassandra.cert
validate = false ;; Optional, true by default. See the paragraph below.

[certfiles] ;; Optional section, overrides the default certfile in the [ssl]
section.
10.209.182.160 = /etc/dse/cassandra/conf/dsenode0.cer
10.68.65.199 = /etc/dse/cassandra/conf/dsenode1.cer
```

- When `validate = false` there is no server authentication, only data encryption.
- When `validate = true` cqlsh will validate the server's certificate against the certfile

When `validate` is enabled, you must create a pem key which is used in the `cqlshrc` file. For example:

```
$ keytool -importkeystore -srckeystore .keystore -destkeystore user.p12 -
deststoretype PKCS12
```

```
openssl pkcs12 -in user.p12 -out user.pem -nodes
```

Note: When generating the certificate, be sure to set the CN to the hostname of the node.

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

Kerberos and SSL

For information about using Kerberos with SSL, see [Using Kerberos and SSL at the same time](#).

The settings for using both Kerberos and SSL are a combination of the Kerberos and SSL sections in these examples.

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

The default location of the sample files depends on the type of installation:

Package installations	/etc/dse/cassandra
Installer-Services installations	/usr/share/dse/resources/cassandra/conf
Installer-No Services and Tarball installations	<i>install_location/resources/cassandra/conf</i>

Configuring and using data auditing

The audit logger logs information only on nodes set up for logging. For example, node 0 has audit turned on, node 1 does not. This means issuing updates and other commands on node 1 does not affect the node 0 audit log. For maximum information from data auditing, turn on data auditing on every node.

Audit logs can be written to filesystem log files using [logback](#), or to a Cassandra table. When you turn on audit logging, the default is to write to logback filesystem log files.

For simple installations, logging to logback files is typically easier than logging audit data to a Cassandra tables. The log files can be read from a terminal for troubleshooting queries or managing security. However, larger clusters can make logback audit logs cumbersome. Because the log files grow extremely large, it's difficult to analyze all the messages. Additionally, the format of the logback files are not flexible. Moreover, because the node's log files are local, it is difficult to find out what is happening across the cluster.

As your cluster scales up, logging audit data to a Cassandra table is more useful. The data can be queried like any other table, making analysis easier and custom audit reports possible.

Audit logging of queries and prepared statements submitted to the DataStax drivers, which use the CQL binary protocol, is supported.

When using audit logging with [Kerberos](#) authentication, the login events take place on Kerberos and are not logged in DataStax Enterprise. Authentication history is available only on Kerberos. When DataStax Enterprise is unable to authenticate a client with Kerberos, a LOGIN_ERROR event is logged.

When using audit logging with Cassandra native protocol authentication, the login events take place as part of connection negotiation and are not logged in DataStax Enterprise.

Procedure

1. Open the dse.yaml file in a text editor.
2. In the `audit_logging_options` section, set `enabled` to true.

```
# Audit logging options
audit_logging_options:
    enabled: true
```

3. Set the logger option to either:
 - `CassandraAuditWriter`
Logs to a Cassandra table.
 - `SLF4JAuditWriter`
Logs to the SLF4J logger.
4. Optional: To include or exclude event categories from being logged, add the event types `include_categories` or `exclude_categories` and specify the categories in a comma separated list. You can set either event type, but not both.

Setting	Logging
<code>ADMIN</code>	Logs describe schema versions, cluster name, version, ring, and other administration events.
<code>AUTH</code>	Logs login events.
<code>DML</code>	Logs insert, update, delete and other DML events.
<code>DDL</code>	Logs object and user create, alter, drop, and other DDL events.
<code>DCL</code>	Logs grant, revoke, create user, drop user, and list users events.
<code>QUERY</code>	Logs all queries.

5. Optional: To include or exclude Cassandra keyspaces from being logged, add a comma separated list of keyspaces to the `included_keyspaces` or `excluded_keyspaces` options. You can set either one, but not both.
6. If you are logging to a Cassandra table, set the retention time for logged events by setting the `retention_time` option to the number of hours the events should be retained. The default value is `0`, which retains all event data indefinitely.
7. Configure the audit logging writer.
 - `SLF4JAuditWriter`
 - `Cassandra table`
8. Optional: To enable verbose logging with Sqoop, add these switches to the `Cassandra logback-tools.xml` file:

```
<root level="WARN">
<appender-ref ref="STDERR" />
</root>
<root level="INFO">
<appender-ref ref="STDERR" />
</root>
<root level="DEBUG">
<appender-ref ref="STDERR" />
</root>
```

Example

The following example sets the audit logger to log to a Cassandra table.

```
# Audit logging options
audit_logging_options:
  enabled: true
  logger: CassandraAuditWriter
```

Configuring audit logging to a logback log file

If you've enabled audit logging and set the logger to output to the SLF4JAuditWriter as described in [Configuring and using data auditing](#) on page 334, you can configure the logger by setting options in `logback.xml`. DataStax Enterprise places the audit log in the directory defined in the `logback.xml` configuration file. After the log file reaches the configured size threshold, it rolls over, and the log file name is changed. The file names include a numerical suffix that is determined by the `maxBackupIndex` property.

Because auditing is configured through a text file in the file system, the file is vulnerable to OS-level security breaches. You can address this issue by changing DataStax Enterprise's umask setting to change the permissions to 600 on the audit files by default. Be aware that if other tools look at the data, changing this setting can cause read problems. Alternately, you can store the audit file on an OS-level encrypted file system such as Vormetric.

Configuring data auditing

You can configure which categories of audit events to log, and whether to omit operations against specific keyspaces from audit logging.

Procedure

1. Open the `logback.xml` file in a text editor.
2. Accept the default settings or change the properties in the `logback.xml` file to configure data auditing:

```
<!--audit log-->
<appender name="SLF4JAuditWriterAppender"
  class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${cassandra.logdir}/audit/audit.log</file> <!-- logfile location
  -->
  <encoder>
    <pattern>%-5level [%thread] %date{ISO8601} %F:%L - %msg%n</pattern>
  <!-- the layout pattern used to format log entries -->
  <immediateFlush>true</immediateFlush>
  </encoder>
  <rollingPolicy
    class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
    <fileNamePattern>${cassandra.logdir}/audit/audit.log.%i.zip</
    fileNamePattern>
    <minIndex>1</minIndex>
    <maxIndex>5</maxIndex> <!-- max number of archived logs that are
    kept -->
  </rollingPolicy>
  <triggeringPolicy
    class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
    <maxFileSize>200MB</maxFileSize> <!-- The size of the logfile that
    triggers a switch to a new logfile, and the current one archived -->
  </triggeringPolicy>
```

```

</appender>
<logger name="SLF4JAuditWriter" level="INFO" additivity="false">
    <appender-ref ref="SLF4JAuditWriterAppender"/>
</logger>

```

The audit logger logs at INFO level, so the `DataAudit` logger must be configured at INFO (or lower) level in `logback.xml`. Setting the logger to a higher level, such as WARN, prevents any log events from being recorded, but it does not completely disable the data auditing. Some overhead occurs beyond overhead that is caused by regular processing.

3. Restart the node to see changes in the log.

Formats of logs

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

Field Label	Field Value	Optional
host	dse node address	no
source	client address	no
user	authenticated user	no
timestamp	system time of log event	no
category	DML/DDL/QUERY for example	no
type	API level operation	no
batch	batch id	yes
ks	keyspace	yes
cf	column family	yes
operation	textual description	yes

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

Auditing is completely separate from authorization, although the data points logged include the client address and authenticated user, which may be a generic user if the default authenticator is not overridden. Logging of requests can be activated for any or all of the list of categories described in [Configuring and using data auditing](#) on page 334.

CQL logging examples

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

CQL USE

```

USE dsp904;

host:/192.168.56.1|source:/192.168.56.101|user:#User allow_all groups=[]

```

```
|timestamp:1351003707937|category:DML|type:SET_KS|ks:dsp904|operation:use
dsp904;
```

CLI USE

```
USE dsp904;

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

CQL query

```
SELECT * FROM t0;

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

CQL BATCH

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

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

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

host:192.168.56.1|source:/192.168.56.101|user:#User allow_all groups=[]
|timestamp:1351005482413|category:DML|type:CQL_DELETE
|batch:fc386364-245a-44c0-a5ab-12f165374a89|ks:dsp904|cf:t1
|operation:DELETE FROM t1 WHERE id = 2
```

CQL DROP KEYSPACE

```
DROP KEYSPACE dsp904;

host:/192.168.56.1|source:/192.168.56.101|user:#User allow_all groups=[]
|timestamp:1351004777354|category:DDL|type:DROP_KS
|ks:dsp904|operation:drop keyspace dsp904;
```

CQL prepared statement

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

Thrift batch_mutate

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

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

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

DataStax Java Driver queries

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

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

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

host:ip-10-85-22-245.ec2.internal/10.85.22.245|source:/127.0.0.1|
user:anonymous
|timestamp:1370537704885|category:QUERY|type:CQL_SELECT
|ks:test|cf:new_cf|operation:select * from test.new_cf;
```

Batch updates

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

Configuring audit logging to a Cassandra table

If you've enabled audit logging and set the logger to output to a Cassandra table as described in [Configuring and using data auditing](#) on page 334, you can configure the logger by setting options in `dse.yaml`.

Audit events are written to the `dse_audit.audit_log` table. The default compaction strategy for the `dse_audit.audit_log` table is DateTieredCompactionStrategy (DTCS). DataStax recommends changing the compaction strategy for tables that were created before DataStax Enterprise 4.8.0:

```
ALTER TABLE dse_audit.audit_log WITH
COMPACTATION={'class':'DateTieredCompactionStrategy'};
```

The logger can be run synchronously or asynchronously. By default, the logger runs synchronously. The permissions for accessing `dse_audit.audit_log` can be managed using the `GRANT` or `REVOKE` CQL commands.

When run synchronously, an event will not complete until the event has been written to the table. If there is a failure after the event has been written to the table but before the event completed, the log may

contain events that were never completed. For example, a query may be logged in the table but it did not successfully complete.

When run asynchronously, audit events are queued for writing to the table, but may not be logged before the event is completed. For example, when logging a query, the query may execute before the audit event is written to the table. A pool of writer threads handles logging audit events from the queue, writing to the table in batch queries. The advantage of writing audit events asynchronously is better performance under load, however if there is a failure before an audit event is written to the table, the audit event may not be logged even though the event has completed.

Procedure

1. Open dse.yaml in a text editor.
2. Set the options in the `audit_logging_options` section.

Option	Description
<code>cassandra_batch_size</code>	The maximum number of events the writer will dequeue before writing them to the audit table. The default value is 50. Set this option to less than 1 to log events synchronously. If you see warnings about the batches being too large, set this number to a lower number or increase the setting of <code>batch_size_warn_threshold_in_kb</code> in <code>cassandra.yaml</code> .
<code>cassandra_flush_time</code>	The maximum amount of time in milliseconds an event will be dequeued by a writer before being written out. The default value is 500. Set this option to less than 1 to log events synchronously. This option prevents events from waiting too long before being written to the table when there are few audit events occurring.
<code>cassandra_num_writers</code>	The number of worker threads asynchronously logging events to the table. The default value is 0. Set this value to less than 1 to log events synchronously. To log events asynchronously, setting this option to 10 is a good starting value.
<code>cassandra_queue_size</code>	The size of the queue feeding the asynchronous audit log writer threads. The default value is 10,000. When there are more audit events than the queue can handle, new events will be blocked until there is space in the queue. If this option is set to less than 1, the queue size will be unbounded, which can lead to resource exhaustion under heavy loads.
<code>cassandra_dropped_event_log</code>	When running asynchronously, failures may prevent the events in the queue from being written to the table. If this occurs, the events are logged to this file. The default setting is <code>/var/log/cassandra/dropped_audit_events.log</code> .
<code>cassandra_keyspace_replication</code>	This section is used to configure how the audit logging table is replicated, has two suboptions: <code>class</code> and <code>replication_factor</code> . By default, <code>class</code> is set to <code>SimpleStrategy</code> , and <code>replication_factor</code> is set to 3.

Option	Description
<code>cassandra_table_compression</code>	This section configures the audit logging table's compression, has one suboption: <code>sstable_compression</code> . By default, <code>sstable_compression</code> is set to <code>SnappyCompressor</code> .
<code>cassandra_table_compaction</code>	This section configures the audit logging table's compaction strategy, and has one suboption: <code>class</code> . By default <code>class</code> is set to <code>SizeTieredCompactionStrategy</code> .

- Save the file and restart DataStax Enterprise.

CassandraAuditWriter table columns

When logging audit data to a Cassandra table using the `CassandraAuditWriter` logger, the audit data is stored in the `dse_audit.audit_log` table. This table has the following columns.

Table: Audit log table columns

Column	Description
<code>date</code>	Date of the event.
<code>node</code>	DSE node address.
<code>day_partition</code>	
<code>event_time</code>	The system timestamp of the event.
<code>batch_id</code>	The UUID of the batch query the event was grouped with when written to Cassandra.
<code>category</code>	The event category.
<code>keyspace_name</code>	The keyspace of the event.
<code>operation</code>	The query or event description.
<code>source</code>	The IP address of the client.
<code>table_name</code>	The table affected by the event.
<code>type</code>	The type of the event.
<code>username</code>	The authenticated user triggering the event. If authentication isn't enabled, the user is anonymous.

Configuring auditing for a DSE Search cluster

If auditing is enabled, DSE Search nodes do not require additional configuration. If the `filter-mapping` element in the Solr web.xml file is commented out, the auditor cannot log anything from Solr.

Procedure

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

```
<filter-mapping>
```

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

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

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

Configuring and using internal authentication

Like [object permission management](#) (which uses internal authorization), internal authentication is based on Cassandra-controlled login accounts and passwords and is supported for use with [dse commands](#) and the [dsetool utility](#). Internal authentication is supported on the following clients when you provide a user name and password to start up the client:

- Astyanax
- cassandra-cli
- cqlsh
- [Drivers](#)
- Hector
- pycassa

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

Spark component limitations

DataStax Enterprise provides internal authentication support for some Hadoop tools and for connecting Spark to Cassandra, not authenticating Spark components between each other.

Hadoop tool authentication limitations

The following authentication limitations apply when using Hadoop tools:

- Internal authentication is not supported for Mahout.
- Using internal authentication to run the `hadoop jar` command is not supported.

The `hadoop jar` command accepts only the JAR file name as an option, and rejects other options such as username and password. The main class in the jar is responsible for making sure that the credentials are applied to the job configuration.

- In Pig scripts that use the custom storage handlers `CqlNativeStorage` and `CassandraStorage`, provide credentials in the URL of the [URL-encoded prepared statement](#):

```
cql://username:password@keyspace/columnfamily
cassandra://username:password@keyspace/columnfamily
```

Use this method of providing authentication for Pig commands regardless of the mechanism that is used to pass credentials to Pig.

- To use Hadoop tools, such as Hive, a user who is not a superuser needs *all* privileges to `HiveMetaStore` and `cfs` keyspaces. To configure a user account named `jdoe`, for example, to use Hadoop tools, use these `cqlsh` commands:

```
cqlsh> GRANT ALL PERMISSIONS ON KEYSPACE "HiveMetaStore" TO jdoe;
```

```
cqlsh> GRANT ALL PERMISSIONS ON KEYSPACE cfs TO jdoe;
```

Configuring internal authentication and authorization

You must set internal authentication and authorization at the same time. After setting the Authorizer and the Authenticator in the `cassandra.yaml` file, set object permissions, as described in [Managing object permissions using internal authorization](#) on page 346.

Procedure

Perform the first three steps on every node.

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

```
authenticator: org.apache.cassandra.auth.PasswordAuthenticator
```

You can use any authenticator except `AllowAll`.

2. Change the authorizer option by commenting the `AllowAllAuthorizer` and adding the `CassandraAuthorizer`:

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

3. Restart the node.

Note: You can [enable internal authorization on existing clusters with no downtime](#).

4. On one node, configure the `system_auth` keyspace replication factor.

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

5. Run a [full repair](#) of the `system_auth` keyspace.
6. Start `cqlsh` using the same superuser name and password (`cassandra`) that you use to start the supported client. For example, to start `cqlsh` on Linux:

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

The `cassandra` user name is provided only as an example.

Note: Use the default `cassandra` user only to assist with initial setup of new users and superusers, and then disable it.

- Logins for the `cassandra` user are performed with QUORUM consistency.

Do not use the default `cassandra` user in production, because QUORUM consistency has significant performance degradation for multiple data centers.

- Logins for all other users are performed with LOCAL_ONE consistency.

Best practices for security and performance:

- Restrict rights of users as appropriate for security. For example, do not allow access to other keyspaces.
- Follow these steps to change the default superuser.

7. Change the [superuser's user name and password](#).

Providing credentials for authentication

Authentication works with any combination of internal Cassandra password authentication, LDAP pass-through authentication, and Kerberos authentication.

Authentication is supported for use with [dse commands](#) and the [dsetool utility](#).

Providing credentials

You can provide the credentials in several ways:

Command line

Provide login credentials on the command line:

```
$ dse [-u username -p password] [-f config_file] [-a jmx_username -b jmx_password] subcommand
$ dsetool dsetool [-l username -p password] [-f config_file] [-a jmx_username -b jmx_password] subcommand
```

where:

- `-f config_file` is the path to a configuration file that stores credentials. If not specified, then use `~/.dserc` if it exists.

The configuration file can contain Cassandra and JMX login credentials. For example:

```
username=cassandra
password=cassandra
jmx_username=cassandra
jmx_password=jmx
```

The credentials in the configuration file are stored in clear text. DataStax recommends restricting access to this file only to the specific user.

- `dse -u username` is the user name to authenticate against the configured Cassandra user.
- `dsetool -l username` is the user name to authenticate against the configured Cassandra user.
- `-p password` is the password to authenticate against the configured Cassandra user. If you do not provide a password on the command line, you are prompted to enter one.
- `-a jmx_username` is the user name for authenticating with secure JMX.
- `-b jmx_password` is the password for authenticating with secure JMX. If you do not provide a password on the command line, you are prompted to enter one.

`~/.dserc` file

Create a file named `.dserc` in your home directory. The `~/.dserc` file contains the Cassandra user name and password:

```
username=username
password=password
```

When you launch a password-protected tool and authentication is not provided on the command line, the credentials in the `~/.dserc` file are used. The `~/.dserc` is ignored when a configuration file is specified with `-f`.

Changing the default superuser

By default, each installation of Cassandra includes a superuser account named `cassandra` whose password is also `cassandra`. Superuser permissions allows creation and deletion of other users and the ability to grant or revoke permissions.

Note: Use the default `cassandra` user only to assist with initial setup of new users and superusers, and then disable it.

- Logins for the `cassandra` user are performed with QUORUM consistency.
Do not use the default `cassandra` user in production, because QUORUM consistency has significant performance degradation for multiple data centers.
- Logins for all other users are performed with LOCAL_ONE consistency.

Best practices for security and performance:

- Restrict rights of users as appropriate for security. For example, do not allow access to other keyspaces.
- Follow these steps to change the default superuser.

Procedure

1. [Configure internal authentication](#) if you have not already done so.
2. Create another superuser, not named `cassandra`, using the [CREATE USER](#) command.
3. Log in as that new superuser.
4. Change the `cassandra` user password to something long and incomprehensible, and then forget about it. It won't be used again.
5. Take away the `cassandra` user's superuser status.
6. Now, that the superuser password is secure, set up user accounts and authorize users to access the database objects by using CQL to [grant them permissions](#) on those objects.

CQL supports the following authentication statements:

- [alter-user](#)
- [create-user](#)
- [drop-user](#)
- [list-users](#)

Enable internal security without downtime

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

Procedure

1. On each node, in the `cassandra.yaml` file:
 - Set the `authenticator` to `com.datastax.bdp.cassandra.auth.TransitionalAuthenticator`.
 - Set the `authorizer` to `com.datastax.bdp.cassandra.auth.TransitionalAuthorizer`.
2. Perform a rolling restart.
3. Run a [full repair](#) of the `system_auth` keyspace

4. After the restarts are complete, use cqlsh with the default superuser login to setup the users, credentials, and permissions.
5. After the setup is complete, edit the `cassandra.yaml` file again and perform another rolling restart:
 - Change the authenticator to `org.apache.cassandra.auth.PasswordAuthenticator`.
 - Change the authorizer to `org.apache.cassandra.auth.CassandraAuthorizer`.
6. After the restarts have completed, remove the default superuser and [create at least one new superuser](#).

Logging in with cqlsh

To avoid having to pass credentials for every login using cqlsh, you can [create a `cqlshrc` file](#) in your `~/.cassandra` directory. When present, it passes default login information to cqlsh. For example:

Procedure

Create the `cqlshrc` file with the following information:

```
[authentication]
username = username
password = password
```

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

Note: Sample `cqlshrc` files are available.

The default location of the sample files depends on the type of installation:

Package installations	<code>/etc/dse/cassandra</code>
Installer-Services installations	<code>/usr/share/dse/resources/cassandra/conf</code>
Installer-No Services and Tarball installations	<code>install_location/resources/cassandra/conf</code>

Managing object permissions using internal authorization

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

CQL supports the following authorization statements:

- [GRANT](#)
- [LIST PERMISSIONS](#)
- [REVOKE](#)

Accessing system resources

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

- `system.schema_keyspace`

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

Configuration

CassandraAuthorizer is one of many possible `IAuthorizer` implementations, and the one that stores permissions in the `system_auth.permissions` table to support all authorization-related CQL statements. Configuration consists mainly of changing the authorizer option in `cassandra.yaml` as described in [Configuring internal authentication and authorization](#).

Note: You must set internal authentication and authorization at the same time.

Configuring `system_auth` and `dse_security` keyspace replication

Cassandra uses the `system_auth` and `dse_security` keyspaces for storing security authentication and authorization information.

- DataStax Enterprise uses the `system_auth` keyspace when you enable any kind of authentication.
- DataStax Enterprise uses the `dse_security` keyspace only on analytics nodes (CFS, Hadoop, Spark) when you enable Kerberos authentication.

Increase the replication factor of these keyspaces depending on your failure tolerance. Data is queried at a consistency level LOCAL_ONE or QUORUM. See [About write consistency](#). The data can be queried frequently. In small clusters, such as those with fewer than 10 nodes, you can set the replication strategy to EverywhereStrategy. However, for larger clusters, choose Simple or Network replication strategy with the replication factor based on your specific requirements.

Attention: To prevent a potential problem logging into a secure cluster, set the replication factor of the `system_auth` and `dse_security` keyspaces to a value that is greater than 1. In a multi-node cluster, using the default of 1 prevents logging into any node when the node that stores the user data is down.

Use a keyspace command such as `ALTER KEYSPACE` to change the replication factor.

Setting the replication factor

Follow this procedure to increase the default replication factor of 1 of the `system_auth` and `dse_security` keyspaces.

Procedure

1. Set the replication factor based on one of the following examples depending on your environment:

- **SimpleStrategy example:**

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

ALTER KEYSPACE "dse_security"
    WITH REPLICATION = { 'class' : 'SimpleStrategy',
    'replication_factor' : 3 };
```

- NetworkTopologyStrategy example:

```
ALTER KEYSPACE "system_auth"
    WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'dc1' : 3,
    'dc2' : 2};

ALTER KEYSPACE "dse_security"
    WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'dc1' : 3,
    'dc2' : 2};
```

2. Run the `nodetool repair` command on the system_auth and dse_security keyspaces. (`nodetool repair system_auth; nodetool repair dse_security`)

```
$ nodetool repair system_auth
$ nodetool repair dse_security
```

Configuring firewall port access

DataStax strongly recommends to run a firewall on all nodes in your Cassandra or DataStax Enterprise cluster.

Begin with a restrictive configuration that blocks all traffic except SSH. Then, open up the following ports to allow communication between the nodes, including certain Cassandra ports. If these ports are not opened, the node acts as a standalone database server rather than joining the database cluster when you start Cassandra (or Hadoop in DataStax Enterprise) on a node.

Procedure

Open the following ports:

Port	Description	Configurable in
Public Facing Ports		
22	SSH (default)	See your OS documentation on sshd.
<i>DataStax Enterprise public ports</i>		
4040	Spark application web site port.	
7080	Spark Master web site port.	spark-env.sh
7081	Spark Worker web site port.	spark-env.sh
8012	Hadoop Job Tracker client port. The Job Tracker listens on this port for job submissions and communications from Task Trackers; allows traffic from each analytics node in a cluster.	cassandra.yaml See Setting the Job Tracker node .
8983	Solr port and Demo applications web site port (Portfolio, Search, Search log, Weather Sensors)	
8090	Spark Jobserver REST API port.	See Spark Jobserver .
9999	Spark Jobserver JMX port. Only required if Spark Jobserver is running and remote access to JMX is required.	

Port	Description	Configurable in
18080	Spark application history server web site port. Only required if Spark application history server is running. Can be changed with the spark.history.ui.port setting.	See Spark history server .
50030	Hadoop Job Tracker web site port. The Job Tracker listens on this port for HTTP requests. If initiated from the OpsCenter, these requests are proxied through the opscenterd daemon; otherwise, they come directly from the browser. [1]	mapred-site.xml using the mapred.job.tracker.http.address property.
50060	Hadoop Task Tracker web site port. Each Task Tracker listens on this port for HTTP requests coming directly from the browser and not proxied by the opscenterd daemon. [1]	mapred-site.xml using the mapred.task.tracker.http.address property.
<i>OpsCenter public ports</i>		
8888	OpsCenter web site port. The opscenterd daemon listens on this port for HTTP requests coming directly from the browser. [1]	opscenterd.conf
Inter-node Ports		
<i>Cassandra inter-node ports</i>		
1024 - 65355	JMX reconnection/loopback ports. Please read the description for port 7199.	
7000	Cassandra inter-node cluster communication port.	cassandra.yaml See storage_port .
7001	Cassandra SSL inter-node cluster communication port.	cassandra.yaml See ssl_storage_port .
7199	Cassandra JMX monitoring port.	cassandra-env.sh See JMX options in Tuning Java resources .
9160	Cassandra client port (Thrift) port. OpsCenter agents makes Thrift requests to their local node on this port. Additionally, the port can be used by the opscenterd daemon to make Thrift requests to each node in the cluster.	cassandra.yaml See rpc_port .
<i>DataStax Enterprise inter-node ports</i>		
7077	Spark Master inter-node communication port.	dse.yaml
8984	Solr inter-node communication port.	dse.yaml See Shard transport options for DSE Search communications on page 240.
9042	CQL native clients port.	cassandra.yaml

Port	Description	Configurable in
		See native_transport_port .
9290	Hadoop Job Tracker Thrift port. The Job Tracker listens on this port for Thrift requests coming from the opscenterd daemon.	
10000	Hive server port.	Set with the -p option in the <code>dse hive --service hiveserver -p port</code> command or configure in <code>hive-site.xml</code> .
10000	Spark SQL Thrift server port. Only required if Spark SQL Thrift server is running.	Set with the -p option with the Spark SQL Thrift server .
<i>OpsCenter specific inter-node</i>		
50031	OpsCenter HTTP proxy for Job Tracker port. The opscenterd daemon listens on this port for incoming HTTP requests from the browser when viewing the Hadoop Job Tracker page directly. [1]	
61620	OpsCenter monitoring port. The opscenterd daemon listens on this port for TCP traffic coming from the agent. [1]	
61621	OpsCenter agent port. The agents listen on this port for SSL traffic initiated by OpsCenter. [1]	

[1] See [OpsCenter and DataStax agent ports](#).

Making /tmp non-executable

Many high security environments require that the `/tmp` directory be mounted with the `noexec` flag set to prevent executables from executing from `/tmp`. However, when `/tmp` is non-executable, the Cassandra database fails to start because JNA cannot start when it does not have a file system location to keep temporary files.

If the `/tmp` directory is not available, the Cassandra `system.log` might have an error. To view the error:

```
$ tail -3 /var/log/cassandra/system.log
```

The error is similar to:

```
ERROR main 2015-12-18 09:57:00,879 CassandraDaemon.java:213 - JNA failing
to initialize properly. Use -Dcassandra.boot_without_jna=true to bootstrap
even so.
INFO Thread-2 2015-12-18 09:57:00,880 DseDaemon.java:418 - DSE shutting
down...
INFO Thread-2 2015-12-18 09:57:00,881 PluginManager.java:103 - All plugins
are stopped.
```

Procedure

To make `/tmp` non-executable and provide access:

- Verify that DataStax Enterprise is running:

```
$ service dse status
```

dse is running

- Verify how the `/tmp` directory is mounted.

```
$ mount | grep /tmp
```

Result:

```
tmpfs on /tmp type tmpfs (rw,nosuid,nodev,relatime,size=3668992k)
```

- Stop DataStax Enterprise.

```
$ service dse stop
```

- Verify that DataStax Enterprise is not running.

```
$ service dse status
```

Result:

```
dse is not running
```

- Remount `/tmp` as non-executable.

```
$ mount -o remount,noexec /tmp
```

- Verify that the `/tmp` directory is mounted as non-executable.

```
$ mount | grep /tmp
```

Result:

```
tmpfs on /tmp type tmpfs (rw,nosuid,nodev,noexec,relatime,size=3668992k)
```

- In the `cassandra-env.sh` file, add this line for the `/tmp` directory that you have access to:

```
JVM_OPTS="$JVM_OPTS -Djna.tmpdir=/tmp/directory/you/have/exec/access/to
```

- Restart DataStax Enterprise.

DSE Management Services

DSE Management Services are a set of services in DataStax Enterprise and OpsCenter that are designed to automatically handle various administration and maintenance tasks and assist with overall database cluster management.

Performance Service

The DataStax Enterprise Performance Service automatically collects and organizes performance diagnostic information into a set of data dictionary tables that can be queried with CQL.

About the Performance Service

The DataStax Enterprise Performance Service automatically collects and organizes performance diagnostic information from Cassandra, [DSE Search](#), and DSE Analytics into a set of data dictionary tables. These tables are stored in the `dse_perf` keyspace and can be queried with CQL using any CQL-based utility, such as `cqlsh`, [DataStax DevCenter](#), or any application using a Cassandra CQL driver.

Use this service to obtain database metrics and optimize Cassandra performance and fine-tune DSE Search. Examples include:

- Identify slow queries on a cluster to easily find and tune poorly performing queries.
- View latency metrics for tables on all user (non-system) keyspaces.
- Collect per node and cluster wide lifetime metrics by table and keyspace.
- Obtain recent and lifetime statistics about tables, such as the number of SSTables, read/write latency, and partition (row) size.
- Track read/write activity on a per-client, per-node level for both recent and long-lived activity to identify problematic user and table interactions.
- Detect bottlenecks in DSE Search.
- Monitor the resources used in a DSE Analytics cluster.
- Monitor particular DSE Analytics applications.

See the following for a complete listing of the available diagnostic tables:

- [Cassandra diagnostic table reference](#)
- [Solr diagnostic table reference](#)

The following is sample output from querying thread pool statistics:

```
cqlsh:dse_perf> select * from thread_pool;
```

Result:

node_ip	pool_name	active	all_time_blocked	blocked	completed	pending
127.0.0.1	AntiEntropyStage	0	0	0	0	0
127.0.0.1	CacheCleanupExecutor	0	0	0	0	0
127.0.0.1	CompactionExecutor	0	0	0	819	0
127.0.0.1	FlushWriter	0	0	0	935	0
127.0.0.1	GossipStage	0	0	0	0	0
127.0.0.1	HintedHandoff	0	0	0	0	0
127.0.0.1	InternalResponseStage	0	0	0	0	0
127.0.0.1	MemoryMeter	0	0	0	1673	0
127.0.0.1	MemtablePostFlusher	0	0	0	1041	0
127.0.0.1	MigrationStage	0	0	0	26	0
127.0.0.1	MiscStage	0	0	0	0	0
127.0.0.1	MutationStage	0	0	0	8654	0
127.0.0.1	PendingRangeCalculator	0	0	0	1	0
127.0.0.1	ReadRepairStage	0	0	0	0	0
127.0.0.1	ReadStage	0	0	0	2681	0
127.0.0.1	ReplicateOnWriteStage	0	0	0	0	0
127.0.0.1	RequestResponseStage	0	0	0	2589	0
127.0.0.1	ValidationExecutor	0	0	0	0	0
127.0.0.1	commitlog_archiver	0	0	0	0	0

(19 rows)

Configuring Performance Service replication strategy

To configure the Performance Service replication strategy, adjust the dse_perf keyspace that stores performance metrics data. Depending on the specific requirements, adjust the replication factor with a keyspace command, such as [ALTER KEYSPACE](#), to prevent potential unavailability of metrics data when nodes are down.

Enabling security

Tables in the dse_perf keyspace that stores performance metrics data do not require special handling for user reads and writes. Because DataStax Enterprise uses internal system APIs to write data to these tables, you do not have to create a system user to perform the writes when security is enabled.

1. To enforce restrictions, enable [internal authentication and authorization](#) and specify appropriate permissions on the tables.
2. To prevent users from viewing sensitive information like keyspace, table, and user names that are recorded in the performance tables, restrict users from reading the tables.

Setting the replication factor

By default DataStax Enterprise writes performance metrics data with consistency level ONE and writes are performed asynchronously. If you need to increase the replication factor of performance metrics data, use [ALTER KEYSPACE](#). See [Configuring data consistency](#).

Procedure

Set the replication factor based depending on your environment:

- **SimpleStrategy example:**

```
ALTER KEYSPACE "dse_perf"
    WITH REPLICATION = { 'class' : 'SimpleStrategy',
        'replication_factor' : 3 };
```

- **NetworkTopologyStrategy example:**

```
ALTER KEYSPACE "dse_perf"
    WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy', 'dc1' : 3,
        'dc2' : 2 };
```

Enabling the collection of Cassandra data

Collecting slow queries

The [node_slow_log table](#) collects information about slow queries on a node and retains query information of long-running CQL statements to help you identify slow queries on a cluster to find and tune poorly performing queries. Collecting slow query logs is enabled by default.

Pending inserts into the node_slow_log might still be processed after the service has been disabled. You can enable and disable the service. After the service is disabled, the logging of queries that take longer than the specified threshold is stopped. However, disabling the logging does not flush the pending write queue, a background thread eventually processes everything.

Procedure

1. By default, collection is enabled for statements that are issued when the query exceeds a specified time threshold.

- To permanently enable collecting information on slow queries, edit the `dse.yaml` file and uncomment the `cql_slow_log_options` parameters, and define values for the CQL slow log settings:

```
# CQL slow log settings
cql_slow_log_options:
  enabled: true
  threshold_ms: 2000
  ttl_seconds: 259200
  async_writers: 1
```

- To temporarily disable collecting information on slow queries that exceeded the threshold of 2000 milliseconds:

```
$ dsetool perf cqslowlog disable
```

- To temporarily change the threshold for slow queries to 1000ms:

```
$ dsetool perf cqslowlog 1000
```

After you collect information using this temporarily set threshold, you can run a script to view queries that took longer with this threshold than the previously set threshold. For example:

```
$ cqlsh
cqlsh> use dse_perf;
cqlsh:dse_perf> select * from node_slow_log;
...
```

2. You can export slow queries using the CQL `copy` command:

```
cqlsh:dse_perf> COPY node_slow_log ( date, commands, duration ) TO
  'slow_queries.csv' WITH HEADER = true;
```

Collecting system level diagnostics

The following system level diagnostic tables collect system-wide performance information about a cluster:

- `key_cache`
Per node key cache metrics. Equivalent to [nodetool info](#).
- `net_stats`
Per node network information. Equivalent to [nodetool netstats](#).
- `thread_pool`
Per node thread pool active/blocked/pending/completed statistics by pool. Equivalent to [nodetool tpstats](#).
- `thread_pool_messages`
Per node counts of dropped messages by message type. Equivalent to [nodetool tpstats](#).

Procedure

To collect system level data:

1. Edit the `dse.yaml` file.

- In the dse.yaml file, set the enabled option for cql_system_info_options to true.

```
# CQL system info tables settings
cql_system_info_options:
    enabled: true
    refresh_rate_ms: 10000
```

- (Optional) To control how often the statistics are refreshed, increase or decrease the refresh_rate_ms parameter.

The refresh_rate_ms specifies the length of the sampling period, that is, the frequency with which this data is updated.

Collecting object I/O level diagnostics

The following object I/O level diagnostic tables collect data on object I/O statistics:

- object_io
Per node recent latency metrics by keyspace and table.
- object_read_io_snapshot
Per node recent latency metrics, broken down by keyspace and table and orders data by mean read latency.
- object_write_io_snapshot
Per node recent latency metrics, broken down by keyspace and table and orders data mean write latency.

Procedure

To enable the collection of this data:

- Edit the dse.yaml file.
- In the dse.yaml file, set the enabled option for resource_level_latency_tracking_options to true.

```
# Data Resource latency tracking settings
resource_level_latency_tracking_options:
    enabled: true
    refresh_rate_ms: 10000
```

- (Optional) To control how often the statistics are refreshed, increase or decrease the refresh_rate_ms parameter.

The refresh_rate_ms specifies the length of the sampling period, that is, the frequency with which this data is updated.

Statistics gathered for objects

To identify which objects (keyspace, table, or client) are currently experiencing the highest average latencies, the Performance Service maintains two latency-ordered tables, which record the mean read/write latencies and total read/write operations on a per-node, per-table basis:

- object_read_io_snapshot
- object_write_io_snapshot

The two tables are essentially views of the same data, but are ordered differently. Using these tables, you can identify which data objects on the node currently cause the most write and read latency to users. Because this is time-sensitive data, if a data object sees no activity for a period, no data will be recorded for them in these tables.

In addition to these two tables, the Performance Service also keeps per-object latency information with a longer retention policy in the object_io table. Again, this table holds mean latency and total count values for both read and write operations, but it can be queried for statistics on specific data objects (either at the keyspace or table level). Using this table enables you to pull back statistics for all tables on a particular node, with the option of restricting results to a given keyspace or specific table.

Table activity broken down by user is retained in the object_user_read_io_snapshot, object_user_write_io_snapshot and object_user_io tables. The first two tables are ordered according to their mean latency values, making it easy for you to quickly identify which clients are currently experiencing the highest latency on specific data objects. Having identified the hot tables on a node, you can drill down and see a breakdown of the users accessing those objects. These tables are refreshed periodically to provide the most up to date view of activity, whereas the user_object_io table retains data for a longer period, enabling it to be queried by node and user with the option of restricting further by keyspace or even table.

Collecting database summary diagnostics

The following database summary diagnostic tables collect statistics at a database level:

- node_table_snapshot

Per node lifetime table metrics broken down by keyspace and table.
- table_snapshot

Cluster wide lifetime table metrics broken down by keyspace and table (aggregates node_table_snapshot from each node in the cluster).
- keyspace_snapshot

Cluster wide lifetime table metrics, aggregated at the keyspace level (rolls up the data in table_snapshot).

Procedure

To enable the collection of database-level statistics data:

1. Edit the dse.yaml file.
2. In the dse.yaml file, set the enabled option for db_summary_stats_options to true.

```
# Database summary stats options
db_summary_stats_options:
    enabled: true
    refresh_rate_ms: 10000
```

3. (Optional) To control how often the statistics are refreshed, increase or decrease the refresh_rate_ms parameter.

The refresh_rate_ms specifies the length of the sampling period, that is, the frequency with which this data is updated.

Collecting cluster summary diagnostics

The following cluster summary diagnostic tables collect statistics at a cluster-wide level:

- cluster_snapshot

Per node system metrics.
- dc_snapshot

- Aggregates node_snapshot data at the datacenter level.
 - node_snapshot
- Aggregates node_snapshot data for the whole cluster.

Procedure

1. Edit the dse.yaml file.
2. In the dse.yaml file, set the enabled option for cluster_summary_stats_options to true.

```
# Cluster summary stats options
cluster_summary_stats_options:
    enabled: true
    refresh_rate_ms: 10000
```

3. (Optional) To control how often the statistics are refreshed, increase or decrease the refresh_rate_ms parameter.

The refresh_rate_ms specifies the length of the sampling period, that is, the frequency with which this data is updated.

Collecting table histogram diagnostics

The following histogram diagnostics tables collect histogram data at a table level:

- cell_count_histograms
Cell count per partition.
- partition_size_histograms
Partition size.
- read_latency_histograms
Read latency.
- sstables_per_read_histograms
SSTables per read.
- write_latency_histograms
Write latency.

Note: These tables somewhat duplicate the information obtained by the [nodetool cfhistograms](#) utility. The major difference is that cfhistograms output is recent data, whereas the diagnostic tables contain lifetime data. Additionally, each time nodetool cfhistograms is run for a column family, the histogram values are reset; whereas the data in the diagnostic histogram tables are not.

Procedure

To enable the collection of table histogram data:

1. Edit the dse.yaml file.
2. In the dse.yaml file, set the enabled option for histogram_data_options to true.

```
# Column Family Histogram data tables options
histogram_data_options:
    enabled: true
    refresh_rate_ms: 10000
    retention_count: 3
```

3. (Optional) To control how often the statistics are refreshed, increase or decrease the refresh_rate_ms parameter.
The refresh_rate_ms specifies the length of the sampling period, that is, the frequency with which this data is updated.
4. Optional: To control the number of complete histograms kept in the tables at any one time, change the retention_count parameter.

Collecting user activity diagnostics

The following diagnostics tables collect user activity:

- object_user_io

Per node, long-lived read/write metrics broken down by keyspace, table, and client connection. Each row contains mean read/write latencies and operation counts for interactions with a specific table by a specific client connection during the last sampling period in which it was active. This data has a 10 minute TTL.

Note: A client connection is uniquely identified by a host and port.

- object_user_read_io_snapshot

Per node recent read/write metrics by client, keyspace, and table. This table contains only data relating to clients that were active during the most recent sampling period. Ordered by mean read latency.

- object_user_write_io_snapshot

Per node recent read/write metrics by client, keyspace, and table. This table contains only data relating to clients that were active during the most recent sampling period. Ordered by mean write latency.

- user_io

Per node, long-lived read/write metrics broken down by client connection and aggregated for all keyspaces and tables. Each row contains mean read/write latencies and operation counts for a specific connection during the last sampling period in which it was active. This data has a 10 minute TTL.

- user_object_io

Per node, long-lived read/write metrics broken down by client connection, keyspace, and table. Each row contains mean read/write latencies and operation counts for interactions with a specific table by a specific client connection during the last sampling period in which it was active. This data has a 10 minute TTL.

Note: [object_user_io](#) and [user_object_io](#) represent two different views of the same underlying data. The former is structured to enable querying by user, the latter for querying by table.

- user_object_read_io_snapshot

Per node recent read/write metrics by keyspace, table, and client. This table contains only data relating to clients that were active during the most recent sampling period. Ordered by mean read latency.

- user_object_write_io_snapshot

Per node recent read/write metrics by keyspace, table, and client. This table contains only data relating to clients that were active during the most recent sampling period. Ordered by mean read latency.

- user_read_io_snapshot

Per node recent read/write metrics by client. This table contains only data relating to clients that were active during the most recent sampling period. Ordered by mean read latency.

- user_write_io_snapshot

Per node recent read/write metrics by client. This table contains only data relating to clients that were active during the most recent sampling period. Ordered by mean write latency.

Procedure

1. Edit the `dse.yaml` file.
2. In the `dse.yaml` file, set the `enabled` option for `user_level_latency_tracking_options` to true.

```
# User/Resource latency tracking settings
user_level_latency_tracking_options:
    enabled: true
    refresh_rate_ms: 10000
    top_stats_limit: 100
```

3. (Optional) To control how often the statistics are refreshed, increase or decrease the `refresh_rate_ms` parameter.

The `refresh_rate_ms` specifies the length of the sampling period, that is, the frequency with which this data is updated.

4. Optional: To limit the number of individual metrics, change the `top_stats_limit` parameter.

Keeping this limit fairly low reduces the level of system resources required to process the metrics.

Statistics gathered for user activity

User activity data is stored in two main ways: Latency-ordered for quickly identifying the hot spots in the system and by user to retrieve statistics for a particular client connection.

To identify which users are currently experiencing highest average latencies on a given node, you can query these tables:

- `user_read_io_snapshot`
- `user_write_io_snapshot`

These tables record mean the read/write latencies and total read/write counts per-user on each node. They are ordered by their mean latency values, so you can quickly see which users are the experiencing the highest average latencies on a given node. Having identified the users experiencing the highest latency on a node, you can then can drill down to find the hot spots for those clients.

To do this, query the `user_object_read_io_snapshot` and `user_object_write_io_snapshot` tables. These tables store mean read/write latency and total read/write count by table for the specified user. They are ordered according to the mean latency values, and therefore able to quickly show for a given user which tables are contributing most to the experienced latencies.

The data in these tables is refreshed periodically (by default every 10 seconds), so querying them always provides an up-to-date view of the data objects with the highest mean latencies on a given node. Because this is time-sensitive data, if a user performs no activity for a period, no data is recorded for them in these tables.

The `user_object_io` table also reports per-node user activity broken down by keyspace/table and retains it over a longer period (4 hours by default). This allows the Performance Service to query by node and user to see latency metrics from all tables or restricted to a single keyspace or table. The data in this table is updated periodically (again every 10 seconds by default).

The `user_io` table reports aggregate latency metrics for users on a single node. Using this table, you can query by node and user to see high-level latency statistics across all keyspaces.

Enabling the collection of Solr data

Collecting slow Solr queries

The `solr_slow_sub_query_log_options` performance object reports distributed sub-queries (query executions on individual shards) that take longer than a specified period of time.

All objects are disabled by default.

Procedure

- To permanently enable collecting information on slow Solr queries, edit the dse.yaml file and uncomment the solr_slow_sub_query_log_options parameters, and define values for the CQL slow log settings:

```
solr_slow_sub_query_log_options:
  enabled:true
  ttl_seconds: 604800
  async_writers: 1
  threshold_ms: 100
```

The default parameter values minimize resource usage.

Options	Determines
enabled	Whether the object is enabled at start up.
ttl_seconds	How many seconds a record survives before it is expired from the performance object .
async_writers	For event-driven objects, such as the slow log, determines the number of possible concurrent slow query recordings. Objects like solr_result_cache_stats are updated in the background.
threshold_ms	For the slow log, the level (in milliseconds) at which a sub-query slow enough to be reported.

- To temporarily enable collecting information on slow Solr queries that exceed the specified threshold:

```
$ dsetool perf solrslowlog disable
```

- You can export slow Solr queries using the CQL [copy](#) command:

```
cqlsh:dse_perf> COPY solr_slow_sub_query_log ( date, commands, duration )
  TO 'slow_solr_queries.csv' WITH HEADER = true;
```

Collecting indexing errors

The [solr_indexing_error_log_options](#) object records errors that occur during document indexing.

All objects are disabled by default.

Procedure

- Edit the dse.yaml file.
- In the dse.yaml file, under the solr_indexing_error_log_options parameter, change enabled to *true* and set the other options as required.

```
# Solr indexing error log options
solr_indexing_error_log_options:
  enabled: true
  ttl_seconds: 604800
  async_writers: 1
```

All objects are disabled by default.

Table: Options

Options	Determines
enabled	Whether the object is enabled at start up.
ttl_seconds	How many seconds a record survives before it is expired from the performance object.
async_writers	For event-driven objects, such as the slow log, determines the number of possible concurrent slow query recordings. Objects like solr_result_cache_stats are updated in the background.

Collecting Solr performance statistics

When solr_latency_snapshot_options is enabled, the performance service creates the required tables and schedules the job to periodically update the relevant snapshot from the specified data source.

The following snapshots collect performance statistics:

- [Query latency snapshot](#) on page 387
Record phase-level cumulative percentile latency statistics for queries over time.
- [Update latency snapshot](#) on page 388
Record phase-level cumulative percentile latency statistics for updates over time.
- [Commit latency snapshot](#) on page 390
Record phase-level cumulative percentile latency statistics for commits over time.
- [Merge latency snapshot](#) on page 391
Record phase-level cumulative percentile latency statistics for index merges over time.

All objects are disabled by default.

Procedure

1. Edit the dse.yaml file.
2. In the dse.yaml file, under the solr_latency_snapshot_options parameter, change enabled to *true* and set the other options as required.

```
# Solr latency snapshot options
solr_latency_snapshot_options:
    enabled: true
    ttl_seconds: 604800
    refresh_rate_ms: 60000
```

All objects are disabled by default.

Table: Options

Options	Determines
enabled	Whether the object is enabled at start up.
ttl_seconds	How many seconds a record survives before it is expired from the performance object.

Options	Determines
refresh_rate_ms	Period (in milliseconds) between sample recordings for periodically updating statistics like the solr_result_cache_stats.

Collecting cache statistics

The `solr_cache_stats_options` object records current and cumulative cache statistics.

The following diagnostic tables collect cache statistics:

- [Filter cache statistics](#) on page 392
Record core-specific query result cache statistics over time.
- [Query result cache statistics](#) on page 394
Record core-specific query result cache statistics over time.

All objects are disabled by default.

Procedure

1. Edit the `dse.yaml` file.
2. In the `dse.yaml` file, under the `solr_cache_stats_options` parameter, change `enabled` to `true` and set the other options as required.

```
# Solr cache statistics options
solr_cache_stats_options:
    enabled: true
    ttl_seconds: 604800
    refresh_rate_ms: 60000
```

All objects are disabled by default.

Table: Options

Options	Determines
enabled	Whether the object is enabled at start up.
ttl_seconds	How many seconds a record survives before it is expired from the performance object .
refresh_rate_ms	Period (in milliseconds) between sample recordings for periodically updating statistics like the <code>solr_result_cache_stats</code> .

Collecting index statistics

The `solr_index_stats_options` object records core-specific index overview statistics over time.

All objects are disabled by default.

Procedure

1. Edit the `dse.yaml` file.

2. In the `dse.yaml` file, for the `solr_index_stats_options` parameter, change `enabled` to `true` and set the other options as required.

```
# Solr index statistics options
solr_index_stats_options:
    enabled: false
    ttl_seconds: 604800
    refresh_rate_ms: 60000
```

All objects are disabled by default.

Table: Options

Options	Determines
<code>enabled</code>	Whether the object is enabled at start up.
<code>ttl_seconds</code>	How many seconds a record survives before it is expired from the performance object .
<code>refresh_rate_ms</code>	Period (in milliseconds) between sample recordings for periodically updating statistics like the <code>solr_result_cache_stats</code> .

Collecting handler statistics

The [Update handler statistics](#) on page 397 records core-specific direct update handler statistics over time.

All objects are disabled by default.

Procedure

1. Edit the `dse.yaml` file.
2. In the `dse.yaml` file, uncomment the `solr_update_handler_metrics_options` parameter and set the options as required.

```
# Solr UpdateHandler metrics options
solr_update_handler_metrics_options:
    enabled: true
    ttl_seconds: 604800
    refresh_rate_ms: 60000
```

All objects are disabled by default.

Table: Options

Options	Determines
<code>enabled</code>	Whether the object is enabled at start up.
<code>ttl_seconds</code>	How many seconds a record survives before it is expired from the performance object .
<code>refresh_rate_ms</code>	Period (in milliseconds) between sample recordings for periodically updating statistics like the <code>solr_result_cache_stats</code> .

Collecting request handler metrics

The `solr_request_handler_metrics_options` object records core-specific direct and request update handler statistics over time.

The following diagnostic tables collect handler metrics:

- [Update handler statistics](#) on page 397
Record core-specific direct update handler statistics over time.
- [Update request handler statistics](#) on page 399
Record core-specific update request handler statistics over time.

All objects are disabled by default.

Procedure

1. Edit the `dse.yaml` file.
2. In the `dse.yaml` file, under the `solr_request_handler_metrics_options` parameter, change `enabled` to `true` and set the other options as required.

```
# Solr request handler metrics options
solr_request_handler_metrics_options:
    enabled: true
    ttl_seconds: 604800
    refresh_rate_ms: 60000
```

All objects are disabled by default.

Table: Options

Options	Determines
<code>enabled</code>	Whether the object is enabled at start up.
<code>ttl_seconds</code>	How many seconds a record survives before it is expired from the performance object .
<code>refresh_rate_ms</code>	Period (in milliseconds) between sample recordings for periodically updating statistics like the <code>solr_result_cache_stats</code> .

Monitoring Spark with Spark Performance Objects

The Performance Service can collect data associated with Spark cluster and Spark applications and save it to a table in Cassandra. This allows monitoring the metrics for DSE Analytics applications for performance tuning and bottlenecks.

If authorization is enabled in your cluster, you must grant the user who is running the Spark application `SELECT` permissions to the `dse_system.spark_metrics_config` table, and `MODIFY` permissions to the `dse_perf.spark_apps_snapshot`.

Monitoring Spark cluster information

The Performance Service stores information about DSE Analytics clusters in the `dse_perf.spark_cluster_snapshot` table. The cluster performance objects store the available and used resources in the cluster, including cores, memory, and workers, as well as overall information about all registered Spark applications, drivers and executors, including the number of applications, the state of each application, and the host on which the application is running.

To enable collecting Spark cluster information, configure the options in the spark_cluster_info_options section of dse.yaml.

Table: Spark cluster info options

Option	Default value	Description
enabled	false	Enables or disables Spark cluster information collection.
refresh_rate_ms	10,000	The time in milliseconds in which the data will be collected and stored.

The `dse_perf.spark_cluster_snapshot` table has the following columns:

name

The cluster name.

active_apps

The number of applications active in the cluster.

active_drivers

The number of active drivers in the cluster.

completed_apps

The number of completed applications in the cluster.

completed_drivers

The number of completed drivers in the cluster.

executors

The number of Spark executors in the cluster.

master_address

The host name and port number of the Spark Master node.

master_recovery_state

The state of the master node.

nodes

The number of nodes in the cluster.

total_cores

The total number of cores available on all the nodes in the cluster.

total_memory_mb

The total amount of memory in MB available to the cluster.

used_cores

The total number of cores currently used by the cluster.

used_memory_mb

The total amount of memory in MB used by the cluster.

workers

The total number of Spark Workers in the cluster.

Monitoring Spark application information

Spark application performance information is stored per application and updated whenever a task is finished. It is stored in the `dse_perf.spark_apps_snapshot` table.

To enable collecting Spark application information, configure the options in the spark_application_info_options section of dse.yaml.

Table: Spark application information options

Option	Default	Description
enabled	false	Enables or disables collecting Spark application information.
refresh_rate_ms	10,000	The time in milliseconds in which the data will be collected and stored.

The driver subsection of spark_application_info_options controls the metrics collected by the Spark Driver.

Table: Spark Driver information options

Option	Default	Description
sink	false	Enables or disables collecting metrics from the Spark Driver.
connectorSource	false	Enables or disables collecting Spark Cassandra Connector metrics.
jvmSource	false	Enables or disables collecting JVM heap and garbage collection metrics from the Spark Driver.
stateSource	false	Enables or disables collecting application state metrics.

The executor subsection of spark_application_info_options controls the metrics collected by the Spark executors.

Table: Spark executor information options

Option	Default	Description
sink	false	Enables or disables collecting Spark executor metrics.
connectorSource	false	Enables or disables collecting Spark Cassandra Connector metrics from the Spark executors.
jvmSource	false	Enables or disables collecting JVM heap or garbage collection metrics from the Spark executors.

The dse_perf.spark_apps_snapshot table has the following columns:

application_id

component_id

metric_id

count

metric_type

rate_15_min

rate_1_min

```

rate_5_min
rate_mean
snapshot_75th_percentile
snapshot_95th_percentile
snapshot_98th_percentile
snapshot_999th_percentile
snapshot_99th_percentile
snapshot_max
snapshot_mean
snapshot_median
snapshot_min
snapshot_stddev
value

```

Cassandra Performance Service diagnostic table reference

The following types of Cassandra performance service diagnostic tables are available:

- [CQL slow log table](#)
- [CQL system info tables](#)
- [Data Resource latency tracking tables](#)
- [Database summary statistics tables](#)
- [Cluster summary statistics tables](#)
- [Histogram tables](#)
- [User and resource latency tracking tables](#)

Note: Table names that contain `_snapshot` are not related to Cassandra [nodetool snapshots](#); they are snapshots of the data in the last few seconds of activity in the system.

CQL slow log table

Table: `node_slow_log table`

Queries on a node exceeding the `threshold_ms` parameter.

Column Name	Data type	Description
<code>node_ip</code>	<code>inet</code>	Node address.
<code>date</code>	<code>timestamp</code>	Date of entry (MM/DD/YYYY granularity).
<code>start_time</code>	<code>timeuuid</code>	Start timestamp of query execution.
<code>commands</code>	<code>list<text></code>	CQL statements being executed.
<code>duration</code>	<code>bigint</code>	Execution time in milliseconds.
<code>parameters</code>	<code>map<text></code>	Not used at this time.
<code>source_ip</code>	<code>inet</code>	Client address.
<code>table_names</code>	<code>set<text></code>	CQL tables touched.

Column Name	Data type	Description
username	text	User executing query, if authentication is enabled.

CQL system info tables

Table: key_cache table

Key cache performance statistics.

Column Name	Data type	Description
node_ip	inet	Node address.
cache_capacity	bigint	Key cache capacity in bytes.
cache_hits	bigint	Total number of cache hits since startup.
cache_requests	bigint	Total number of cache requests since startup.
cache_size	bigint	Current key cache size in bytes.
hit_rate	double	Ratio of hits to requests since startup.

Table: net_stats table

Data flow operations repair tasks and more.

Column Name	Data type	Description
node_ip	inet	Node address.
commands_completed	bigint	Total read repair commands completed since startup.
commands_pending	int	Current number of read repair commands pending.
read_repair_attempted	bigint	Read repairs attempted since startup.
read_repaired_background	bigint	Number of read repairs performed asynchronously since startup.
read_repaired_blocking	bigint	Number of read repairs performed synchronously since startup.
responses_completed	bigint	Current read repairs completed count.
responses_pending	int	Current read repair responses pending count.

Table: thread_pool table

Information on thread pool activity.

Column Name	Data type	Description
node_ip	inet	Node address.
pool_name	text	Thread pool name.
active	bigint	Currently active tasks.
all_time_blocked	bigint	Total blocked tasks since startup.
blocked	bigint	Currently blocked tasks.
completed	bigint	Total completed tasks since startup.

Column Name	Data type	Description
pending	bigint	Currently pending tasks.

Table: thread_pool_messages table

Information about thread pool messages.

Column Name	Data type	Description
node_ip	inet	Node address.
message_type	text	Inter-node message type.
dropped_count	int	Total count of dropped messages since startup.

Data Resource latency tracking tables

Table: object_io table

Per node recent latency metrics by keyspace and table.

Column Name	Data type	Description
node_ip	inet	Node address.
keyspace_name	text	Keyspace name.
table_name	text	Table name.
last_activity	timestamp	End of sampling period in which this object was last active.
memory_only	boolean	DSE memory only table.
read_latency	double	Mean value in microseconds for all reads during the last active sampling period for this object.
total_reads	bigint	Count during the last active sampling period for this object.
total_writes	bigint	Count during the last active sampling period for this object.
write_latency	double	Mean value in microseconds for all writes during the last active sampling period for this object.

Table: object_read_io_snapshot table

Per node recent latency metrics by keyspace and table. Ordered by mean read latency.

Column Name	Data type	Description
node_ip	inet	Node address.
latency_index	int	Ranking by mean read latency during the last sampling period.
keyspace_name	text	Keyspace name.
memory_only	boolean	DSE memory only table.
read_latency	double	In microseconds during the last sampling period.
table_name	text	Table name.
total_reads	bigint	Count during the last sampling period.

Column Name	Data type	Description
total_writes	bigint	Count during the last sampling period.
write_latency	double	In microseconds during the last sampling period.

Table: object_write_io_snapshot table

Per node recent latency metrics by keyspace and table. Ordered by mean write latency. Scale of 0 to 99 (0 is worst).

Column Name	Data type	Description
node_ip	inet	Node address.
latency_index	int	Ranking by mean write latency during the last sampling period.
keyspace_name	text	Keyspace name.
memory_only	boolean	DSE memory only table.
read_latency	double	Mean value in microseconds during the active sampling period.
table_name	text	Table name.
total_reads	bigint	Count during the last sampling period.
total_writes	bigint	Count during the last sampling period.
write_latency	double	Mean value in microseconds during the last sampling period.

Database summary statistics tables

Table: node_table_snapshot table

Per node table metrics by keyspace and table.

Column Name	Data type	Description
node_ip	inet	Node address.
keyspace_name	text	Keyspace name.
table_name	text	Table name.
bf_false_positive_ratio	double	Bloom filter false positive ratio since startup.
bf_false_positives	bigint	Bloom filter false positive count since startup.
compression_ratio	double	Current compression ratio of SSTables.
droppable_tombstone_ratio	double	Ratio of tombstones older than gc_grace_seconds against total column count in all SSTables.
key_cache_hit_rate	double	Current key cache hit rate.
live_sstable_count	bigint	Current SSTable count.
max_row_size	bigint	Maximum partition size in bytes.
mean_read_latency	double	In microseconds for this table since startup.
mean_row_size	bigint	Average partition size in bytes.

Column Name	Data type	Description
mean_write_latency	double	In microseconds for this table since startup.
memtable_columns_count	bigint	Approximate number of cells for this table currently resident in memtables.
memtable_size	bigint	Total size in bytes of memtable data.
memtable_switch_count	bigint	Number of times memtables have been flushed since startup.
min_row_size	bigint	Minimum partition size in bytes.
total_data_size	bigint	Data size on disk in bytes.
total_reads	bigint	Number of reads since startup.
total_writes	bigint	Number of writes since startup.
unleveled_sstables	bigint	Current count of SSTables in level 0 (if using leveled compaction).

Table: table_snapshot table

Cluster wide lifetime table metrics by keyspace and table. This table aggregates node_table_snapshot from each node in the cluster.

Column Name	Data type	Description
keyspace_name	text	Keyspace name.
table_name	text	Table name.
bf_false_positive_ratio	double	Bloom filter false positive ratio since startup.
bf_false_positives	bigint	Bloom filter false positive count since startup.
compression_ratio	double	Current compression ratio of SSTables.
droppable_tombstone_ratio	double	Ratio of tombstones older than gc_grace_seconds against total column count in all SSTables.
key_cache_hit_rate	double	Current key cache hit rate.
live_sstable_count	bigint	Current SStable count.
max_row_size	bigint	Maximum partition size in bytes.
mean_read_latency	double	In microseconds for this table since startup.
mean_row_size	bigint	Average partition size in bytes.
mean_write_latency	double	In microseconds for this table since startup.
memtable_columns_count	bigint	Approximate number of cells for this table currently resident in memtables.
memtable_size	bigint	Total size in bytes of memtable data.
memtable_switch_count	bigint	Number of times memtables have been flushed since startup.
min_row_size	bigint	Minimum partition size in bytes.
total_data_size	bigint	Data size on disk in bytes.

Column Name	Data type	Description
total_reads	bigint	Number of reads since startup.
total_writes	bigint	Number of writes since startup.
unleveled_sstables	bigint	Current count of SSTables in level 0 (if using leveled compaction).

Table: keyspace_snapshot table

Cluster wide lifetime table metrics, aggregated at the keyspace level (aggregates the data in table_snapshot).

Column Name	Data type	Description
keyspace_name	text	Keyspace name.
index_count	int	Number of secondary indexes.
mean_read_latency	double	For all tables in the keyspace and all nodes in the cluster since startup.
mean_write_latency	double	For all tables in the keyspace and all nodes in the cluster since startup.
table_count	int	Number of tables in the keyspace.
total_data_size	bigint	Total size in bytes of SSTables for all tables and indexes across all nodes in the cluster.
total_reads	bigint	For all tables, across all nodes.
total_writes	bigint	For all tables, across all nodes.

Cluster summary statistics tables

Table: node_snapshot table

Per node system metrics.

Column Name	Data type	Description
node_ip	inet	Node address.
cms_collection_count	bigint	CMS garbage collections since startup.
cms_collection_time	bigint	Total time spent in CMS garbage collection since startup.
commitlog_pending_tasks	bigint	Current commit log tasks pending.
commitlog_size	bigint	Total commit log size in bytes.
compactions_completed	bigint	Number of compactions completed since startup.
compactions_pending	int	Number of pending compactions.
completed_mutations	bigint	Total number of mutations performed since startup.
data_owned	float	Percentage of total data owned by this node.
datacenter	text	Datacenter name.
dropped_mutation_ratio	double	Ratio of dropped to completed mutations since startup.
dropped_mutations	bigint	Total number of dropped mutations since startup.

Column Name	Data type	Description
flush_sorter_tasks_pending	bigint	Current number of memtable flush sort tasks pending.
free_space	bigint	Total free disk space in bytes.
gossip_tasks_pending	bigint	Current number of gossip tasks pending.
heap_total	bigint	Total available heap memory in bytes.
heap_used	bigint	Current heap usage in bytes.
hinted_handoff_pending	bigint	Current number of hinted handoff tasks pending.
index_data_size	bigint	Total size in bytes of index column families.
internal_responses_pending	bigint	Current number of internal response tasks pending.
key_cache_capacity	bigint	Key cache capacity in bytes.
key_cache_entries	bigint	Current number of key cache entries.
key_cache_size	bigint	Current key cache size in bytes.
manual_repair_tasks_pending	bigint	Current number of manual repair tasks pending.
mean_range_slice_latency	double	Mean latency in microseconds for range slice operations since startup.
mean_read_latency	double	Mean latency in microseconds for reads since startup.
mean_write_latency	double	Mean latency in microseconds for writes since startup.
memtable_post_flushers_pending	bigint	Current number of memtable post flush tasks pending.
migrations_pending	bigint	Current number of migration tasks pending.
misc_tasks_pending	bigint	Current number of misc tasks pending.
parnew_collection_count	bigint	ParNew garbage collections since startup.
parnew_collection_time	bigint	Total time spent in ParNew garbage collection since startup.
process_cpu_load	double	Current CPU load for the DSE process (Linux only).
rack	text	Rack identifier.
range_slice_timeouts	bigint	Number of timed out range slice requests since startup.
read_repair_tasks_pending	bigint	Current number of read repair tasks pending.
read_requests_pending	bigint	Current read requests pending.
read_timeouts	bigint	Number of timed out range slice requests since startup.
replicate_on_write_tasks_pending	bigint	Current.
request_responses_pending	bigint	Current.
row_cache_capacity	bigint	Row cache capacity in bytes.
row_cache_entries	bigint	Current number of row cache entries.
row_cache_size	bigint	Current row cache size in bytes.
state	text	Node State (JOINING/LEAVING/MOVING/NORMAL).
storage_capacity	bigint	Total disk space in bytes.

Column Name	Data type	Description
streams_pending	int	Current number of pending streams.
table_data_size	bigint	Total size in bytes of non-index column families.
tokens	settext	Tokens owned by the this node.
total_batches_replayed	bigint	Total number of batchlog entries replayed since startup
total_node_memory	bigint	Total available RAM (Linux only).
total_range_slices	bigint	Total number of range slice operations performed since startup.
total_reads	bigint	Total number of reads performed since startup.
total_writes	bigint	Total number of writes performed since startup.
uptime	bigint	Node uptime in seconds.
write_requests_pending	bigint	Total number of write tasks pending.
write_timeouts	bigint	Number of timed out range slice requests since startup.

Table: dc_snapshot table

Aggregates node_snapshot data at the datacenter level.

Column Name	Data type	Description
name	text	Datacenter name
compactions_completed	bigint	Total number of compactions completed since startup by all nodes in the data center.
compactions_pending	int	Total number of pending compactions on all nodes in the datacenter.
completed_mutations	bigint	Total number of mutations performed since startup by all nodes in the data center.
dropped_mutation_ratio	double	Ratio of dropped to completed mutations since startup across all nodes in the datacenter.
dropped_mutations	bigint	Total number of dropped mutations since startup by all nodes in the data center.
flush_sorter_tasks_pending	bigint	Total number of memtable flush sort tasks pending across all nodes in the data center.
free_space	bigint	Total free disk space in bytes across all nodes in the datacenter.
gossip_tasks_pending	bigint	Total number of gossip tasks pending across all nodes in the data center.
hinted_handoff_pending	bigint	Total number of hinted handoff tasks pending across all nodes in the data center.
index_data_size	bigint	Total size in bytes of index column families across all nodes in the data center.
internal_responses_pending	bigint	number of internal response tasks pending across all nodes in the data center.

Column Name	Data type	Description
key_cache_capacity	bigint	Total capacity in bytes of key caches across all nodes in the data center.
key_cache_entries	bigint	Total number of entries in key caches across all nodes in the data center.
key_cache_size	bigint	Total consumed size in bytes of key caches across all nodes in the data center.
manual_repair_tasks_pending	bigint	Total number of manual repair tasks pending across all nodes in the data center.
mean_range_slice_latency	double	Mean latency in microseconds for range slice operations, averaged across all nodes in the datacenter.
mean_read_latency	double	Mean latency in microseconds for read operations, averaged across all nodes in the datacenter.
mean_write_latency	double	Mean latency in microseconds for write operations, averaged across all nodes in the datacenter.
memtable_post_flushers_pending	bigint	Total number of memtable post flush tasks pending across all nodes in the data center.
migrations_pending	bigint	Total number of migration tasks pending across all nodes in the data center.
misc_tasks_pending	bigint	Total number of misc tasks pending across all nodes in the datacenter.
node_count	int	Total number of live nodes in the datacenter.
read_repair_tasks_pending	bigint	Total number of read repair tasks pending across all nodes in the data center.
read_requests_pending	bigint	Total read requests pending across all nodes in the datacenter.
replicate_on_write_tasks_pending	bigint	Total number of counter replicate on write tasks pending across all nodes in the datacenter.
request_responses_pending	bigint	Total number of request response tasks pending across all nodes in the data center.
row_cache_capacity	bigint	Total capacity in bytes of partition caches across all nodes in the data center.
row_cache_entries	bigint	Total number of row cache entries all nodes in the datacenter.
row_cache_size	bigint	Total consumed size in bytes of row caches across all nodes in the data center.
storage_capacity	bigint	Total disk space in bytes across all nodes in the datacenter.
streams_pending	int	number of pending streams across all nodes in the datacenter.
table_data_size	bigint	Total size in bytes of non-index column families across all nodes in the data center.

Column Name	Data type	Description
total_batches_replayed	bigint	Total number of batchlog entries replayed since startup by all nodes in the datacenter.
total_range_slices	bigint	Total number of range slice operations performed since startup by all nodes in the datacenter.
total_reads	bigint	Total number of read operations performed since startup by all nodes in the datacenter.
total_writes	bigint	Total number of write operations performed since startup by all nodes in the datacenter.
write_requests_pending	bigint	Total number of write tasks pending across all nodes in the data center.

Table: cluster_snapshot table

Aggregates node_snapshot data for the whole cluster.

Column Name	Data type	Description
name	text	Cluster name.
compactions_completed	bigint	Total number of compactions completed since startup by all nodes in the cluster.
completed_mutations	bigint	Total number of mutations performed since startup by all nodes in the cluster.
compactions_pending	int	Total number of pending compactions on all nodes in the cluster.
datacenters	settext	Datacenter names.
dropped_mutation_ratio	double	Ratio of dropped to completed mutations since startup across all nodes in the cluster.
dropped_mutations	bigint	Total number of dropped mutations since startup by all nodes in the cluster.
flush_sorter_tasks_pending	bigint	Total number of memtable flush sort tasks pending across all nodes in the cluster.
free_space	bigint	Total free disk space in bytes across all nodes in the cluster.
gossip_tasks_pending	bigint	Total number of gossip tasks pending across all nodes in the cluster.
hinted_handoff_pending	bigint	Total number of hinted handoff tasks pending across all nodes in the cluster.
index_data_size	bigint	Total size in bytes of index column families across all nodes in the cluster.
internal_responses_pending	bigint	Number of internal response tasks pending across all nodes in the cluster.
key_cache_capacity	bigint	Total capacity in bytes of key caches across all nodes in the cluster.

Column Name	Data type	Description
key_cache_entries	bigint	Total number of entries in key caches across all nodes in the cluster.
key_cache_size	bigint	Total consumed size in bytes of key caches across all nodes in the cluster.
keyspace_count	int	Total number of keyspaces defined in schema.
manual_repair_tasks_pending	bigint	Total number of manual repair tasks pending across all nodes in the cluster.
mean_range_slice_latency	double	Mean latency in microseconds for range slice operations, averaged across all nodes in the cluster.
mean_read_latency	double	Mean latency in microseconds for read operations, averaged across all nodes in the cluster.
mean_write_latency	double	Mean latency in microseconds for write operations, averaged across all nodes in the cluster.
memtable_post_flushers_pending	bigint	Total number of memtable post flush tasks pending across all nodes in the cluster.
migrations_pending	bigint	Total number of migration tasks pending across all nodes in the cluster.
misc_tasks_pending	bigint	Total number of misc tasks pending across all nodes in the cluster.
node_count	int	Total number of live nodes in the cluster.
read_repair_tasks_pending	bigint	Total number of read repair tasks pending across all nodes in the cluster.
read_requests_pending	bigint	Total read requests pending across all nodes in the cluster.
replicate_on_write_tasks_pending	bigint	Total number of counter replicate on write tasks pending across all nodes in the cluster.
request_responses_pending	bigint	Total number of request response tasks pending across all nodes in the cluster
row_cache_capacity	bigint	Total capacity in bytes of partition caches across all nodes in the cluster.
row_cache_entries	bigint	Total number of row cache entries all nodes in the cluster.
row_cache_size	bigint	Total consumed size in bytes of row caches across all nodes in the cluster
storage_capacity	bigint	Total disk space in bytes across all nodes in the cluster.
streams_pending	int	Number of pending streams across all nodes in the cluster.
table_count	int	Total number of tables defined in schema.
table_data_size	bigint	Total size in bytes of non-index column families across all nodes in the cluster.
total_batches_replayed	bigint	Total number of batchlog entries replayed since startup by all nodes in the cluster.

Column Name	Data type	Description
total_range_slices	bigint	Total number of read operations performed since startup by all nodes in the cluster.
total_reads	bigint	Total number of write operations performed since startup by all nodes in the cluster.
total_writes	bigint	Total number of write tasks pending across all nodes in the cluster.
write_requests_pending	bigint	Total number of write tasks pending across all nodes in the cluster.

Histogram tables

Table: read_latency_histograms table

Read latency histogram data.

Column Name	Data type	Description
node_ip	inet	Node address
keyspace_name	text	Keyspace name
table_name	text	Table name
histogram_id	timestamp	Groups rows by the specific histogram they belong to. Rows for the same node, keyspace & table are ordered by this field, to enable date-based filtering
bucket_offset	bigint	Read latency in microseconds
bucket_count	bigint	Count of reads where the latency falls in the corresponding bucket

Table: write_latency_histograms table

Write latency histogram data.

Column Name	Data type	Description
node_ip	inet	Node address
keyspace_name	text	Keyspace name
table_name	text	Table name
histogram_id	timestamp	Groups rows by the specific histogram they belong to. Rows for the same node, keyspace & table are ordered by this field, to enable date-based filtering
bucket_offset	bigint	Write latency in microseconds
bucket_count	bigint	Count of writes where the latency falls in the corresponding bucket

Table: sstables_per_read_histograms table

SStables per read histogram data.

Column Name	Data type	Description
node_ip	inet	Node address
keyspace_name	text	Keyspace name
table_name	text	Table name
histogram_id	timestamp	Groups rows by the specific histogram they belong to. Rows for the same node, keyspace & table are ordered by this field, to enable date-based filtering
bucket_offset	bigint	Number of SSTables required to satisfy a read request
bucket_count	bigint	Count of reads where the number of SSTables read falls in the corresponding bucket

Table: partition_size_histograms table

Partition size histogram data.

Column Name	Data type	Description
node_ip	inet	Node address.
keyspace_name	text	Keyspace name.
table_name	text	Table name.
histogram_id	timestamp	Groups rows by the specific histogram they belong to. Rows for the same node, keyspace & table are ordered by this field, to enable date-based filtering.
bucket_offset	bigint	Partition size in bytes.
bucket_count	bigint	Number of partitions where the size falls in the corresponding bucket.

Table: cell_count_histograms table

Cell count per partition histogram data.

Column Name	Data type	Description
node_ip	inet	Node address.
keyspace_name	text	Keyspace name.
table_name	text	Table name.
histogram_id	timestamp	Groups rows by the specific histogram they belong to. Rows for the same node, keyspace, and table are ordered by this field, to enable date-based filtering.
bucket_offset	bigint	Number of cells in a partition.
bucket_count	bigint	Number of partitions where the cell count falls in the corresponding bucket.

User and resource latency tracking tables

Table: user_io table

Per node, long-lived read/write metrics by client connection and aggregated for all keyspaces and tables.

Column Name	Data type	Description
node_ip	inet	Node address.
conn_id	text	Unique client connection ID.
last_activity	timestamp	End of sampling period in which this client was last active.
read_latency	double	In microseconds for the last active sampling period.
total_reads	bigint	Count during the last active sampling period for this client.
total_writes	bigint	Count during the last active sampling period for this client.
user_ip	inet	Client origin address.
username	text	Present if authentication is enabled.
write_latency	double	In microseconds for the last active sampling period.

Table: user_read_io_snapshot table

Per node recent read/write metrics by keyspace, table, and client during the most recent sampling period.

Column Name	Data type	Description
node_ip	inet	Node address.
latency_index	int	Ranking by mean read latency during the last sampling period.
conn_id	text	Unique client connection ID.
read_latency	double	Mean value in microseconds during the last sampling period.
total_reads	bigint	During the last sampling period.
total_writes	bigint	During the last sampling period.
user_ip	inet	Client origin address.
username	text	Present if authentication is enabled.
write_latency	double	Mean value in microseconds during the last sampling period.

Table: user_write_io_snapshot table

Per node recent read/write metrics by keyspace, table, and client during the most recent sampling period.

Column Name	Data type	Description
node_ip	inet	Node address.
latency_index	int	Ranking by mean write latency during the last sampling period.

Column Name	Data type	Description
conn_id	text	Unique client connection ID.
read_latency	double	Mean value in microseconds during the last sampling period.
total_reads	bigint	During the last sampling period.
total_writes	bigint	During the last sampling period.
user_ip	inet	Client origin address.
username	text	Present if authentication is enabled.
write_latency	double	Mean value in microseconds during the last sampling period.

Table: user_object_io table

Per node, long-lived read/write metrics by client connection, keyspace and table.

Column Name	Data type	Description
node_ip	inet	Node address.
conn_id	text	Unique client connection ID.
keyspace_name	text	Keyspace name.
table_name	text	Table name.
last_activity	timestamp	End of sampling period in which this client was last active against this object.
read_latency	double	Mean value in microseconds during the last active sampling period for this object/client.
total_reads	bigint	During the last active sampling period for this object/client.
total_writes	bigint	During the last active sampling period for this object/client.
user_ip	inet	Client origin address.
username	text	Present if authentication is enabled.
write_latency	double	Mean value in microseconds during the last active sampling period for this object/client.

Table: user_object_write_io_snapshot table

Per node recent read/write metrics by client, keyspace, and table during the most recent sampling period.

Column Name	Data type	Description
node_ip	inet	Node address.
latency_index	int	Ranking by mean write latency during the last sampling period.
conn_id	text	Unique client connection ID.
keyspace_name	text	Keyspace name.

Column Name	Data type	Description
read_latency	double	Mean value in microseconds during the last sampling period.
table_name	text	Table name.
total_reads	bigint	During the last sampling period.
total_writes	bigint	During the last sampling period.
user_ip	inet	Client origin address.
username	text	Present if authentication is enabled.
write_latency	double	Mean value in microseconds during the last sampling period.

Table: user_object_read_io_snapshot table

Per node read/write metrics by client, keyspace, and table during the most recent sampling period.

Tracks best-worst latency on a scale of 0 to 99 (0 is worst).

Column Name	Data type	Description
node_ip	inet	Node address.
latency_index	int	Ranking by mean read latency during the last sampling period.
conn_id	text	Unique client connection ID.
keyspace_name	text	Keyspace name.
read_latency	double	Mean value in microseconds during the last sampling period.
table_name	text	Table name.
total_reads	bigint	During the last sampling period.
total_writes	bigint	During the last sampling period.
user_ip	inet	Client origin address.
username	text	Present if authentication is enabled.
write_latency	double	Mean value in microseconds during the last sampling period.

Table: object_user_io table

Overview of the I/O activity by user for each table.

Column Name	Data type	Description
node_ip	inet	Node address.
keyspace_name	text	Keyspace name.
table_name	text	Table name.
conn_id	text	Unique client connection ID.
last_activity	timestamp	End of sampling period in which this client connection was last active against this object.

Column Name	Data type	Description
read_latency	double	Mean value in microseconds during the last active sampling period for this object/client.
total_reads	bigint	Count during the last active sampling period for this object/client.
total_writes	bigint	Count during the last active sampling period for this object/client.
user_ip	inet	Client origin address.
username	text	Present if authentication is enabled.
write_latency	double	Mean value in microseconds during the last active sampling period for this object/client.

Table: object_user_read_io_snapshottable

Per node recent read/write metrics by client, keyspace, and table during the most recent sampling period. Tracks best-worst latency on a scale of 0 to 99 (0 is worst).

Column Name	Data type	Description
node_ip	inet	Node address.
latency_index	int	Ranking by mean read latency during the last sampling period.
conn_id	text	Unique client connection ID.
keyspace_name	text	Keyspace name.
read_latency	double	Mean value in microseconds during the last active sampling period for this object/client.
table_name	text	Table name.
total_reads	bigint	Count during the last active sampling period for this object/client.
total_writes	bigint	Count during the last active sampling period for this object/client.
user_ip	inet	Client origin address.
username	text	Present if authentication is enabled.
write_latency	double	Mean value in microseconds during the last active sampling period for this object/client.

Table: object_user_write_io_snapshot table

Per node recent read/write metrics by client, keyspace, and table during the most recent sampling period. Tracks best-worst latency on a scale of 0 to 99 (0 is worst).

Column Name	Data type	Description
node_ip	inet	Node address.
latency_index	int	Ranking by mean write latency during the last sampling period.
conn_id	text	Unique client connection ID.

Column Name	Data type	Description
keyspace_name	text	Keyspace name.
read_latency	double	Mean value in microseconds during the last active sampling period for this object/client.
table_name	text	Table name.
total_reads	bigint	Count during the last active sampling period for this object/client.
total_writes	bigint	Count during the last active sampling period for this object/client.
user_ip	inet	Client origin address.
username	text	Present if authentication is enabled.
write_latency	double	Mean value in microseconds during the last active sampling period for this object/client.

DSE Search Performance Service diagnostic table reference

Frequently asked questions about the Solr Performance Service

Question: Is it a good idea to leave the Solr performance objects enabled 24/7?

Answer: It depends on your use cases. If you're attempting to collect data pertaining to a problem that occurs sporadically, and you've chosen configuration values that don't introduce a painful amount of performance overhead, there's no reason you can't keep the objects enabled on an ongoing basis.

Question: What kind of performance impact will enabling the Solr performance objects have?

Answer: Performance overhead, in terms of CPU and memory usage, should be negligible when using the DataStax Enterprise's default configuration values. However, the overhead introduced by enabling the objects varies as the configuration is modified (described in the following sections). For instance, setting longer TTLs and shorter refresh intervals leads to higher memory and disk consumption.

Question: Should I enable the Solr performance objects on every node in my cluster?

Answer: The Solr performance objects should only be enabled on search nodes, that is, nodes where indexes reside that can observe search operations. While it is perfectly acceptable to enable the objects across an entire cluster, enabling them on a single node for observation first is a good way to mitigate risk.

Question: Can I use existing Cassandra CF with secondary indexes on some columns, and create Solr indexes on other columns in the same CF?

Answer: Do not mix Solr indexes with Cassandra secondary indexes. Attempting to use both indexes on the same table is not supported.

Slow sub-query log for Solr

Report distributed sub-queries for Solr (query executions on individual shards) that take longer than a specified period of time.

JMX analog

None.

Schema

When [slow Solr query logging is enabled](#), this table is created automatically.

```
CREATE TABLE IF NOT EXISTS dse_perf.solr_slow_sub_query_log (
    core text,
    date timestamp,
    coordinator_ip inet,
    query_id timeuuid,
    node_ip inet,
    start_time timeuuid,
    parameters map<text, text>,
    elapsed_millis bigint,
    component_prepare_millis map<text, bigint>,
    component_process_millis map<text, bigint>,
    num_docs_found bigint,
    PRIMARY KEY ((core, date), coordinator_ip, query_id, node_ip)
)
```

Field	Type	Purpose
core	text	Name of the Solr core (keyspace.table) where the slow sub-query was executed.
date	timestamp	Midnight on the mm/dd/yyyy the slow sub-query started.
coordinator_ip	inet	Distributed query coordinator IP address.
query_id	timeuuid	ID of distributed query to which the slow sub-query belongs.
node_ip	inet	Node IP address.
start_time	timestamp	Timestamp at the start of the slow sub-query.
parameters	map<text, text>	Solr query parameters.
elapsed_millis	bigint	How long the slow sub-query took.
component_prepare_millis	map<text, bigint>	Map of (component name -> time spent in prepare phase).
component_process_millis	map<text, bigint>	Map of (component name -> time spent in process phase).
num_docs_found	bigint	Number of documents found by the slow sub-query.

Slow Solr sub-queries recorded on 10/17/2015 for core keyspace.table for coordinator at 127.0.0.1:

```
SELECT *
FROM solr_slow_sub_query_log
WHERE core = 'keyspace.table' AND date = '2015-10-17' AND coordinator_ip =
'127.0.0.1';
```

Slow Solr sub-queries recorded on 10/17/2015 for core keyspace.table for coordinator at 127.0.0.1 for a particular distributed query with an ID of 33e56d33-4e63-11e4-9ce5-335a04d08bd4 :

```
SELECT *
FROM solr_slow_sub_query_log
WHERE core = 'keyspace.table'
  AND date = '2015-10-17'
  AND coordinator_ip = '127.0.0.1'
  AND query_id = 33e56d33-4e63-11e4-9ce5-335a04d08bd4;
```

Related tasks

[Collecting slow Solr queries](#) on page 359

Indexing error log

Record errors that occur during document indexing.

Specifically, this log records errors that occur during document validation. A common scenario is where a non-stored copy field is copied into a field with an incompatible type.

JMX Analog

None.

Schema

```
CREATE TABLE IF NOT EXISTS dse_perf.solr_indexing_errors (
    node_ip inet,
    core text,
    date timestamp,
    time timeuuid,
    document text,
    field_name text,
    field_type text,
    message text,
    PRIMARY KEY ((node_ip, core, date), time)
)
WITH CLUSTERING ORDER BY (time DESC)
```

Field	Type	Purpose
node_ip	inet	Node address.
core	text	Solr Core name, such as keyspace.table.
date	timestamp	Midnight on the mm/dd/yyyy the error occurred.
time	timeuuid	Timestamp for the time the error occurred.
document	text	The primary key for the Cassandra row corresponding to the document. For example: [foo, bar, baz] for a complex PK, or foo for a single element PK.
field_name	text	Name of the field that caused the validation error.

Field	Type	Purpose
field_type	text	Name of the field that caused the validation error, such as solr.StrField.
message	text	Error message.

Indexing validation errors recorded on 10/17/2014 for core keyspace.table for at node 127.0.0.1:

```
SELECT *
FROM solr_indexing_errors
WHERE core = 'keyspace.table' AND date = '2014-10-17' AND node_ip =
'127.0.0.1';
```

Most recent 5 indexing validation errors recorded on 10/17/2014 for core keyspace.table for at node 127.0.0.1:

```
SELECT *
FROM solr_indexing_errors
WHERE core = 'keyspace.table'
    AND date = '2014-10-17'
    AND node_ip = '127.0.0.1'
ORDER BY time DESC
LIMIT 5;
```

Related tasks

[Collecting indexing errors](#) on page 360

Query latency snapshot

Record phase-level cumulative percentile latency statistics for queries over time.

Note: All statistics reset upon node restart.

This table is configured with [gc_grace_seconds 0](#) to avoid issues with persistent tombstones as rows expire; tombstones are removed during compaction no matter how recently they were created.

JMX Analog

```
com.datastax.bdp/search/core/QueryMetrics
```

See [Query metrics MBean](#) on page 261.

Schema

```
CREATE TABLE dse_perf.solr_query_latency_snapshot (
    node_ip inet,
    core text,
    date timestamp,
    time timestamp,
    phase text,
    count bigint,
    latency_percentiles_micros maptext, bigint PRIMARY KEY ((node_ip, core),
    phase, time)
)
WITH CLUSTERING ORDER BY (phase ASC, time DESC)
AND gc_grace_seconds=0
```

Field	Type	Purpose
node_ip	inet	Node IP address.
core	text	Solr Core name, such as keyspace.table.
date	timestamp	Midnight on the mm/dd/yyyy the snapshot was recorded.
time	timestamp	Time the snapshot was recorded.
phase	text	EXECUTE, COORDINATE, RETRIEVE
count	bigint	Cumulative number of queries recorded.
latency_percentiles_micros	map<text, bigint>	Cumulative latency percentiles of query: 25%, 50%, 75%, 95%, 99% and 99.9%

Snapshots recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_query_latency_snapshot
WHERE node_ip = '127.0.0.1' AND core = 'keyspace.table' AND date =
'2014-10-17';
```

Most recent 5 snapshots for the EXECUTE phase recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_query_latency_snapshot
WHERE node_ip = '127.0.0.1'
    AND core = 'keyspace.table'
    AND date = '2014-10-17'
    AND phase = 'EXECUTE'
LIMIT 5;
```

Related tasks

[Collecting Solr performance statistics](#) on page 361

Update latency snapshot

Record phase-level cumulative percentile latency statistics for updates over time.

Note: All statistics reset upon node restart.

This table is configured with [gc_grace_seconds 0](#) to avoid issues with persistent tombstones as rows expire; tombstones are removed during compaction no matter how recently they were created.

JMX analog

```
com.datastax.bdp/search/core/UpdateMetrics
```

See [Query metrics MBean](#) on page 261.

Schema

```
CREATE TABLE dse_perf.solr_update_latency_snapshot (
    node_ip inet,
    core text,
    date timestamp,
    time timestamp,
    phase text,
    count bigint,
    latency_percentiles_micros map<text, bigint>
    PRIMARY KEY ((node_ip, core), phase, time)
)
WITH CLUSTERING ORDER BY (phase ASC, time DESC)
AND gc_grace_seconds=0
```

Field	Type	Purpose
node_ip	inet	Node IP address.
core	text	Solr Core name, such as keyspace.table.
date	timestamp	Midnight on the mm/dd/yyyy the snapshot was recorded.
time	timestamp	Time the snapshot was recorded.
phase	text	WRITE, QUEUE, PREPARE, EXECUTE
count	bigint	Cumulative number of queries recorded.
latency_percentiles_micros	map<text,bigint>	Cumulative latency percentiles of query: 25%, 50%, 75%, 95%, 99% and 99.9%.

Snapshots recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_update_latency_snapshot
WHERE node_ip = '127.0.0.1' AND core = 'keyspace.table' AND date =
'2014-10-17';
```

Most recent 5 snapshots for the EXECUTE phase recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_update_latency_snapshot
WHERE node_ip = '127.0.0.1'
    AND core = 'keyspace.table'
    AND date = '2014-10-17'
    AND phase = 'EXECUTE'
LIMIT 5;
```

Related tasks

[Collecting Solr performance statistics](#) on page 361

Commit latency snapshot

Record phase-level cumulative percentile latency statistics for commits over time.

Note: All statistics reset upon node restart.

This table is configured with [gc_grace_seconds 0](#) to avoid issues with persistent tombstones as rows expire; tombstones are removed during compaction no matter how recently they were created.

JMX Analog

```
com.datastax.bdp/search/core/CommitMetrics
```

See [Commit metrics MBean](#) on page 259.

Schema

```
CREATE TABLE dse_perf.solr_commit_latency_snapshot (
    node_ip inet,
    core text,
    date timestamp,
    time timestamp,
    phase text,
    count bigint,
    latency_percentiles_micros maptext, bigint
    PRIMARY KEY ((node_ip, core, date), phase, time)
)
WITH CLUSTERING ORDER BY (phase ASC, time DESC)
AND gc_grace_seconds=0
```

Field	Type	Purpose
node_ip	inet	Node IP address.
core	text	Solr Core name, such as keyspace.table.
date	timestamp	Midnight on the mm/dd/yyyy the snapshot was recorded.
time	timestamp	Time the snapshot was recorded.
phase	text	FLUSH, EXECUTE
count	bigint	Cumulative number of queries recorded.
latency_percentiles_micros	maptext, bigint	Cumulative latency percentiles of query: 25%, 50%, 75%, 95%, 99% and 99.9%.

Snapshots recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_commit_latency_snapshot
WHERE node_ip = '127.0.0.1' AND core = 'keyspace.table' AND date =
'2014-10-17';
```

Most recent 5 snapshots for the EXECUTE phase recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_commit_latency_snapshot
WHERE node_ip = '127.0.0.1'
    AND core = 'keyspace.table'
    AND date = '2014-10-17'
    AND phase = 'EXECUTE'
LIMIT 5;
```

Related tasks

[Collecting Solr performance statistics](#) on page 361

Merge latency snapshot

Record phase-level cumulative percentile latency statistics for index merges over time.

Note: All statistics reset upon node restart.

This table is configured with [gc_grace_seconds 0](#) to avoid issues with persistent tombstones as rows expire; tombstones are removed during compaction no matter how recently they were created.

JMX analog

```
com.datastax.bdp/search/core/MergeMetrics
```

See [Merge metrics MBean](#) on page 260.

Schema

```
CREATE TABLE dse_perf.solr_merge_latency_snapshot (
    node_ip inet,
    core text,
    date timestamp,
    time timestamp,
    phase text,
    count bigint,
    latency_percentiles_micros maptext, bigint
    PRIMARY KEY ((node_ip, core, date), phase, time)
)
WITH CLUSTERING ORDER BY (phase ASC, time DESC)
AND gc_grace_seconds=0
```

Field	Type	Purpose
node_ip	inet	Node IP address.
core	text	Solr Core name, such as keyspace.table.
date	timestamp	Midnight on the mm/dd/yyyy the snapshot was recorded.
time	timestamp	Time the snapshot was recorded.
phase	text	INIT, WARM, EXECUTE
count	bigint	Cumulative number of queries recorded.

Field	Type	Purpose
latency_percentiles_micros	map<text, bigint>	Cumulative latency percentiles of query: 25%, 50%, 75%, 95%, 99% and 99.9%.

Snapshots recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_merge_latency_snapshot
WHERE node_ip = '127.0.0.1' AND core = 'keyspace.table' AND date =
  '2014-10-17';
```

Most recent 5 snapshots for the EXECUTE phase recorded on 10/17/2014 for core keyspace.table" on the node 127.0.0.1:

```
SELECT *
FROM solr_merge_latency_snapshot
WHERE node_ip = '127.0.0.1'
  AND core = 'keyspace.table'
  AND date = '2014-10-17'
  AND phase = 'EXECUTE'
LIMIT 5;
```

Related tasks

[Collecting Solr performance statistics](#) on page 361

Filter cache statistics

Record core-specific filter cache statistics over time.

Note: All statistics reset upon node restart.

This table is configured with [gc_grace_seconds 0](#) to avoid issues with persistent tombstones as rows expire; tombstones are removed during compaction no matter how recently they were created.

Solr exposes a core's filter cache statistics through its registered index searcher, but the core may have many index searchers over its lifetime. To reflect this, it provides statistics for the currently registered searcher as well as cumulative/lifetime statistics.

JMX analog

```
solr/core/dseFilterCache/com.datastax.bdp.search.solr.FilterCacheMBean
```

Schema

```
CREATE TABLE dse_perf.solr_filter_cache_stats (
  node_ip inet,
  core text,
  date timestamp,
  time timestamp,
  hits bigint,
  inserts bigint,
  evictions bigint,
  hit_ratio float,
  lookups bigint,
  num_entries bigint,
  cumulative_lookups bigint,
```

```

        cumulative_hits bigint,
        cumulative_hitratio float,
        cumulative_inserts bigint,
        cumulative_evictions bigint,
        warmup_time bigint,
        PRIMARY KEY ((node_ip, core, date), time)
    )
WITH gc_grace_seconds=0

```

Field	Type	Purpose
node_ip	inet	Node IP address.
core	text	Solr Core name, such as keyspace.table.
date	timestamp	Midnight on the mm/dd/yyyy the statistics were recorded.
time	timestamp	The exact time the statistics were recorded.
hits	bigint	Cache hits for the registered index searcher.
inserts	bigint	Cache insertions for the registered index searcher.
evictions	bigint	Cache evictions for the registered index searcher.
hit_ratio	float	The ratio of cache hits/lookups for the registered index searcher.
lookups	bigint	Cache lookups for the registered index searcher.
num_entries	bigint	Number of cache entries for the registered index searcher.
cumulative_lookups	bigint	Cumulative cache lookups for the core.
cumulative_hits	bigint	Cumulative cache hits for the core.
cumulative_hitratio	float	Cumulative ratio of cache hits/lookups for the core.
cumulative_inserts	bigint	Cumulative cache inserts for the core.
cumulative_evictions	bigint	Cumulative cache evictions for the core.
warmup_time	bigint	Warm-up time for the registered index searcher.

Snapshots for cumulative statistics recorded on 10/17/2014 for core “keyspace.table” on the node 127.0.0.1:

```
SELECT cumulative_lookups, cumulative_hits, cumulative_hitratio,
       cumulative_inserts
  FROM solr_filter_cache_stats
 WHERE node_ip = '127.0.0.1' AND core = 'keyspace.table' AND date =
   '2014-10-17';
```

Most recent 5 snapshots recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
  FROM solr_filter_cache_stats
 WHERE node_ip = '127.0.0.1'
       AND core = 'keyspace.table'
       AND date = '2014-10-17'
 ORDER BY time DESC
 LIMIT 5;
```

Related tasks

[Collecting cache statistics](#) on page 362

Query result cache statistics

Record core-specific query result cache statistics over time.

Solr exposes a core’s result cache statistics through its registered index searcher, but the core may have many index searchers over its lifetime. To reflect this, it provides statistics for the currently registered searcher as well as cumulative/lifetime statistics.

JMX analog

```
solr/core/queryResultCache/*
```

Schema

```
CREATE TABLE dse_perf.solr_result_cache_stats (
    node_ip inet,
    core text,
    date timestamp,
    time timestamp,
    hits bigint,
    inserts bigint,
    evictions bigint,
    hit_ratio float,
    lookups bigint,
    num_entries bigint,
    cumulative_lookups bigint,
    cumulative_hits bigint,
    cumulative_hitratio float,
    cumulative_inserts bigint,
    cumulative_evictions bigint,
    warmup_time bigint,
    PRIMARY KEY ((node_ip, core, date), time)
)
WITH gc_grace_seconds=0
```

Field	Type	Purpose
node_ip	inet	Node IP address.
core	text	Solr Core name, such as keyspace.table.
date	timestamp	Midnight on the mm/dd/yyyy the statistics were recorded.
time	timestamp	The exact time the statistics were recorded.
hits	bigint	Cache hits for the registered index searcher.
inserts	bigint	Cache insertions for the registered index searcher.
evictions	bigint	Cache evictions for the registered index searcher.
hit_ratio	float	The ratio of cache hits / lookups for the registered index searcher.
lookups	bigint	Cache lookups for the registered index searcher.
num_entries	bigint	Number of cache entries for the registered index searcher.
cumulative_lookups	bigint	Cumulative cache lookups for the core.
cumulative_hits	bigint	Cumulative cache hits for the core.
cumulative_hitratio	float	Cumulative ratio of cache hits/ lookups for the core.
cumulative_inserts	bigint	Cumulative cache inserts for the core.
cumulative_evictions	bigint	Cumulative cache evictions for the core.
warmup_time	bigint	Warm-up time for the registered index searcher.

Snapshots for cumulative statistics recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT cumulative_lookups, cumulative_hits, cumulative_hitratio,
       cumulative_inserts
  FROM solr_result_cache_stats
 WHERE node_ip = '127.0.0.1' AND core = 'keyspace.table' AND date =
   '2014-10-17';
```

Most recent 5 snapshots recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
  FROM solr_result_cache_stats
 WHERE node_ip = '127.0.0.1'
```

```

AND core = 'keyspace.table'
AND date = '2014-10-17'
ORDER BY time DESC
LIMIT 5;

```

Related tasks[Collecting cache statistics](#) on page 362

Index statistics

Record core-specific index overview statistics over time.

JMX analog

```
solr/core_name/core/core_name & solr/core_name/Searcher*
```

Schema

```

CREATE TABLE dse_perf.solr_index_stats (
    node_ip inet,
    core text,
    date timestamp,
    time timestamp,
    size_in_bytes bigint,
    num_docs int,
    max_doc int,
    docs_pending_deletion int,
    PRIMARY KEY ((node_ip, core, date), time)
)
WITH gc_grace_seconds=0

```

Field	Type	Purpose
node_ip	inet	Node IP address.
core	text	Solr Core name, such as keyspace.table.
date	text	Midnight on the mm/dd/yyyy the statistics were recorded.
time	bigint	The exact time the statistics were recorded.
size_in_bytes	bigint	Index size on file system.
num_docs	int	The number of documents inserted into index.
max_docs	int	The number of documents inserted into index, plus those marked as removed, but not yet physically removed.
docs_pending_deletion	int	max_docs - num_docs

Snapshots recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_index_stats
WHERE node_ip = '127.0.0.1' AND core = 'keyspace.table' AND date =
'2014-10-17';
```

Most recent 5 snapshots recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_index_stats
WHERE node_ip = '127.0.0.1'
    AND core = 'keyspace.table'
    AND date = '2014-10-17'
ORDER BY time DESC
LIMIT 5;
```

Update handler statistics

Record core-specific direct update handler statistics over time.

Note: Do not confuse this with [Update request handler statistics](#) on page 399.

A few fields in this table have both cumulative and non-cumulative versions. The non-cumulative statistics are zeroed out following rollback or commit, while the cumulative versions persist through those events. The exception is errors, which is actually cumulative and takes into account a few failure cases that cumulative_errors does not.

JMX analog

```
solr/core/updateHandler
```

Schema

```
CREATE TABLE dse_perf.solr_update_handler_metrics (
    node_ip inet,
    core text,
    date timestamp,
    time timestamp,
    adds bigint,
    cumulative_adds bigint,
    commits bigint,
    autocommits int,
    autocommit_max_time text,
    autocommit_max_docs int,
    soft_autocommits int,
    soft_autocommit_max_docs int,
    soft_autocommit_max_time text,
    deletes_by_id bigint,
    deletes_by_query bigint,
    cumulative_deletes_by_id bigint,
    cumulative_deletes_by_query bigint,
    expunge_deletes bigint,
    errors bigint,
    cumulative_errors bigint,
    docs_pending bigint,
    optimizes bigint,
    rollbacks bigint,
```

```

    PRIMARY KEY ((node_ip, core, date), time)
)
WITH gc_grace_seconds=0

```

Field	Type	Purpose
node_ip	inet	Node IP address.
core	text	Solr Core name, such as keyspace.table.
date	timestamp	Midnight on the mm/dd/yyyy the statistics were recorded.
time	timestamp	Exact time the statistics were recorded.
adds	bigint	Document add commands since last commit/rollback.
cumulative_adds	bigint	Cumulative document additions.
commits	long	Number of explicit commit commands issued.
autocommits	int	Number of auto-commits executed.
autocommit_max_time	text	Maximum time between auto-commits.
autocommit_max_docs	int	Maximum document adds between auto-commits.
soft_autocommits	int	Number of soft auto-commits executed.
soft_autocommit_max_docs	int	Maximum time between soft auto-commits.
soft_autocommit_max_time	int	Maximum document adds between soft auto-commits.
deletes_by_id	long	Currently uncommitted deletions by ID.
deletes_by_query	bigint	Currently uncommitted deletions by query.
cumulative_deletes_by_id	bigint	Cumulative document deletions by ID.
cumulative_deletes_by_query	bigint	Cumulative document deletions by ID.
expunge_deletes	bigint	Number of commit commands issued with expunge deletes.
errors	bigint	Cumulative errors for add/delete/commit/rollback commands.
cumulative_errors	bigint	Cumulative errors for add/delete commands.

Field	Type	Purpose
docs_pending	bigint	Number of documents pending commit.
optimizes	bigint	Number of explicit optimize commands issued.
rollbacks	bigint	Number of rollbacks executed.

Snapshots recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_update_handler_metrics
WHERE node_ip = '127.0.0.1' AND core = 'keyspace.table' AND date =
'2014-10-17';
```

Most recent 5 snapshots recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_update_handler_metrics
WHERE node_ip = '127.0.0.1'
    AND core = 'keyspace.table'
    AND date = '2014-10-17'
ORDER BY time DESC
LIMIT 5;
```

Related tasks

[Collecting request handler metrics](#) on page 364

Update request handler statistics

Record core-specific update request handler statistics over time.

Note: Do not to confuse this with [Update handler statistics](#) on page 397.

JMX analog

```
solr/core/update[ / | /<csv | /json]
```

Schema

```
CREATE TABLE dse_perf.solr_update_request_handler_metrics (
    node_ip inet,
    core text,
    date timestamp,
    handler_name text,
    time timestamp,
    requests bigint,
    errors bigint,
    timeouts bigint,
    total_time_seconds double,
    avg_requests_per_second double,
    five_min_rate_reqs_per_second double,
    fifteen_min_rate_reqs_per_second double,
    PRIMARY KEY ((node_ip, core, date), handler_name, time)
)
WITH CLUSTERING ORDER BY (handler_name ASC, time DESC)
```

```
AND gc_grace_seconds=0
```

Field	Type	Purpose
node_ip	inet	Node IP address.
core	text	Solr Core name, such as keyspace.table.
date	timestamp	Midnight on the mm/dd/yyyy the statistics were recorded.
handler_name	text	A handler name specified in the solrconfig.xml file.
time	timestamp	Exact time the statistics were recorded.
requests	bigint	Number of requests processed by the handler.
errors	bigint	Number of errors encountered by the handler.
timeouts	bigint	Number of responses received with partial results.
total_time	double	The sum of all request processing times.
avg_requests_per_second	double	Average number of requests per second.
five_min_rate_reqs_per_second	double	Requests per second over that past 5 minutes.
fifteen_min_rate_reqs_per_second	double	Requests per second over that past 15 minutes.

Snapshots recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_update_request_handler_metrics
WHERE node_ip = '127.0.0.1' AND core = 'keyspace.table' AND date =
'2014-10-17';
```

Most recent 5 snapshots for handler “search” recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_search_request_handler_metrics
WHERE node_ip = '127.0.0.1'
    AND core = 'keyspace.table'
    AND date = '2014-10-17'
    AND handler_name = 'search'
LIMIT 5;
```

Related tasks

[Collecting request handler metrics](#) on page 364

Search request handler statistics

Record core-specific search request handler statistics over time.

JMX analog

```
solr/core/search
```

Schema

```
CREATE TABLE dse_perf.solr_search_request_handler_metrics (
    node_ip inet,
    core text,
    date timestamp,
    handler_name text,
    time timestamp,
    requests bigint,
    errors bigint,
    timeouts bigint,
    total_time_seconds double,
    avg_requests_per_second double,
    five_min_rate_reqs_per_second double,
    fifteen_min_rate_reqs_per_second double,
    PRIMARY KEY ((node_ip, core, date), handler_name, time)
)
WITH CLUSTERING ORDER BY (handler_name ASC, time DESC)
AND gc_grace_seconds=0
```

Field	Type	Purpose
node_ip	inet	Node IP address.
core	text	Solr Core name, such as keyspace.table.
date	timestamp	Midnight on the mm/dd/yyyy the statistics were recorded.
handler_name	text	A handler name specified in the solrconfig.xml file.
time	timestamp	Exact time the statistics were recorded.
requests	bigint	Number of requests processed by the handler.
errors	bigint	Number of errors encountered by the handler.
timeouts	bigint	Number of responses received with partial results.
total_time_seconds	double	The sum of all request processing times.
avg_requests_per_second	double	Average number of requests per second.

Field	Type	Purpose
five_min_rate_reqs_per_second	double	Requests per second over that past 5 minutes.
fifteen_min_rate_reqs_per_second	double	Requests per second over that past 15 minutes.

Snapshots recorded for all update handlers on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_search_request_handler_metrics
WHERE node_ip = '127.0.0.1' AND core = 'keyspace.table' AND date =
'2014-10-17';
```

Most recent 5 snapshots for handler “/update/json” recorded on 10/17/2014 for core keyspace.table on the node 127.0.0.1:

```
SELECT *
FROM solr_search_request_handler_metrics
WHERE node_ip = '127.0.0.1' AND
      core = 'keyspace.table'
      AND date = '2014-10-17'
      AND handler_name = '/update/json'
LIMIT 5;
```

Related tasks

[Collecting request handler metrics](#) on page 364

Capacity Service

The Capacity Service automatically collects data about a cluster’s operations and provides for the ability to do historical trend analysis and forecasting of future trends.

For more details, see [Capacity Service](#) in the *OpsCenter User Guide*.

Repair Service

The Repair Service is designed to automatically keep data synchronized across a cluster and can be managed either visually through OpsCenter [Repair Service](#) or by using the command line.

DSE In-Memory

DSE In-Memory delivers in-memory computing capabilities to Cassandra. DSE In-Memory allows developers, architects, and administrators to easily choose what parts (some or all) of a database reside fully in RAM. DSE In-Memory is designed for use cases that lend themselves to in-memory computing, while allowing disk-based workloads to be serviced by Cassandra’s traditional storage model. This design allows applications with in-memory and disk-based requirements to be supported by one database platform.

DSE In-Memory delivers lightning-fast read performance for use cases that include primarily read-only workloads with slowly changing data and/or semi-static datasets. For example, a product catalog that is refreshed nightly, but read constantly during the day.

DSE In-Memory is not suitable for workloads with heavily changing data or monotonically growing datasets that can exceed the RAM capacity on the nodes/cluster.

DataStax recommends using [OpsCenter](#) to check performance metrics before and after configuring DSE In-Memory.

Creating or altering tables to use DSE In-Memory

Use CQL directives to create and alter tables to use DSE In-Memory and `dse.yaml` to limit the size of tables.

Creating a table to use DSE In-Memory

To create a table that uses DSE In-Memory, add a CQL directive to the `CREATE TABLE` statement. Use the compaction directive in the statement to specify the `MemoryOnlyStrategy` class, disable compression, and disable the key and row caches.

```
CREATE TABLE customers (
    uid text,
    fname text,
    lname text,
    PRIMARY KEY (uid)
) WITH compaction= { 'class': 'MemoryOnlyStrategy' }
    AND compression = {'sstable_compression' : ''}
    AND caching = {'keys':'NONE', 'rows_per_partition':'NONE'};
```

Altering an existing table to use DSE In-Memory

Use the `ALTER TABLE` statement to change a traditional table to use in-memory, or to change an in-memory table to a traditional table. For example, use the `DESCRIBE` command for a table named `employee`. Verify that `employee` is a traditional table because the output of the `DESCRIBE` command does not include a line that looks something like:

```
compaction={'class': 'MemoryOnlyStrategy'} >
```

Alter the `employee` table to use DSE In-Memory and, as a best practice, [disable caching](#):

```
ALTER TABLE employee WITH compaction= { 'class': 'MemoryOnlyStrategy' }
    AND compression = {'sstable_compression' : ''}
    AND caching = {'keys':'NONE', 'rows_per_partition':'NONE'};
```

Limiting the size of tables

Use the `max_memory_to_lock_fraction` or `max_memory_to_lock_mb` configuration option in the `dse.yaml` file to specify how much system memory to use for all in-memory tables.

<code>max_memory_to_lock_fraction</code>	Specify a fraction of the system memory. The default value of 0.20 specifies to use up to 20% of system memory.
<code>max_memory_to_lock_mb</code>	Specify a maximum amount of memory in MB.

Disabling caching on tables

DataStax recommends disabling caching on tables that use the DSE In-Memory option. If caching is not disabled, a warning is logged. Set the table caching property to disable both types of caching:

```
ALTER TABLE customers WITH caching = {'keys':'NONE',
  'rows_per_partition':'NONE'};
```

Verifying table properties

In cqlsh, use the DESCRIBE command to view table properties:

```
cqlsh> DESCRIBE TABLE employee;
```

This output shows that the table uses DSE In-Memory:

```
CREATE TABLE employee (
  uid text PRIMARY KEY,
  fname text,
  lname text
) WITH bloom_filter_fp_chance = 0.01
AND caching = '{"keys":"NONE", "rows_per_partition":"NONE"}'
AND comment = ''
AND compaction = {'min_threshold': '4', 'class':
'org.apache.cassandra.db.compaction.MemoryOnlyStrategy',
'max_threshold': '32'}
AND compression = {}
AND dclocal_read_repair_chance = 0.1
AND default_time_to_live = 0
AND gc_grace_seconds = 864000
AND max_index_interval = 2048
AND memtable_flush_period_in_ms = 0
AND min_index_interval = 128
AND read_repair_chance = 0.0
AND speculative_retry = '99.0PERCENTILE';
```

Managing memory

Because DataStax Enterprise runs in a distributed environment, you can inadvertently add excessive data that exceeds the available memory.

When using DSE In-Memory, you must monitor and carefully manage available memory. You can use OpsCenter to [monitor in-memory usage](#).

DSE In-Memory retains the [durability guarantees of Cassandra](#).

Recommended limits

To prevent exceeding the RAM capacity, DataStax recommends that in-memory objects consume no more than 45% of a node's free memory.

Managing available memory

If the maximum memory capacity is exceeded, locking some of the data into memory is stopped, and read performance will degrade and a warning message is displayed.

The warning message looks something like this:

```
WARN [main] 2015-03-27 09:34:00,050 MemoryOnlyStrategy.java:252 - File
MmappedSegmentedFile(path='/data/ks/test-f590c150b95911e4b66d85e0b6fd73a5/
ks-test-ka-94-Data.db',
length=43629650) buffer address: 140394485092352 length: 43629650 could not
be locked.
Sizelimit (1048576) reached. After locking size would be: 43630592
```

Checking available memory

Use the `dsetool inmemorystatus` command to check the amount of data that is currently in memory. When the data size exceeds the specified **Max Memory to Lock** value, or some other problem exists, the **Couldn't Lock** column displays its value. The `system.log` file provides useful information for problem resolution.

		Size	Couldn't Lock
Keyspace	ColumnFamily		
Max Memory to Lock:		1MB	
Current Total Memory Locked:		0MB	
Current Total Memory Not Able To Lock:		46MB	
mos_ks	testmemory	0MB	46MB
mos_ks	testmemory2	0MB	0MB
mos_ks	testmemory4	0MB	0MB
mos_ks	testmemory3	0MB	0MB

Backing up and restoring data

The procedures for backing up and restoring data are the same for DSE In-Memory data and on-disk data. Use the OpsCenter [Backup Service](#) or use the Cassandra [snapshot process](#) to manage backups and restores.

Deploying

Production deployment planning

For guidance in planning a DataStax Enterprise cluster, see [Planning a cluster deployment](#) in the Cassandra documentation. The Cassandra documentation includes information about:

- Spinning disks versus SSDs
- Memory and CPU recommendations
- Disk space
- Data size
- Network requirements

Deploying

- RAID
- Data size
- Anti-patterns

The following resources and guidelines are also recommended:

- [DataStax Enterprise Reference Architecture](#) white paper.
- For EC2 deployments, see:
 - [Amazon EC2 documentation](#)
 - [EC2 clusters spanning multiple regions and availability zones](#)
 - [What is the story with AWS storage](#)
 - [Get in the Ring with Cassandra and EC2](#)
- DataStax Enterprise requires a solid network layer. Although not required, jumbo frames are recommended to improve streaming performance during processes such as bootstrapping and repair.
- DSE Analytics and DSE Search nodes require their own nodes/disks and have specific hardware requirements. See [Capacity Planning](#) in the *DataStax Enterprise Reference Architecture* and the [Hadoop](#) and [Solr](#) documentation.
- DataStax does not support or recommend using Network Attached Storage (NAS) because of performances issues, such as network saturation, I/O overload, pending-task swamp, excessive memory usage, and disk contention.
- If using a firewall, make sure that nodes within a cluster can reach each other. See [Configuring firewall port access](#).

Configuring replication

Cassandra can store multiple copies of data on multiple nodes for reliability and fault tolerance. To configure replication, you must:

- [Configure gossip](#).

Nodes communicate with each other about replication and other things using the gossip protocol.

- Choose whether to [use vnodes](#).

Vnodes provide many tokens per node and simplify many tasks in Cassandra.

Attention: DataStax Enterprise turns off virtual nodes (vnodes) by default. DataStax does not recommend turning on vnodes for DSE Hadoop or BYOH nodes. Before turning vnodes on for Hadoop, understand the [implications](#). DataStax Enterprise does support turning on vnodes for Spark nodes.

- Choose a [data partitioner](#).

Data partitioning determines how data is placed across the nodes in the cluster.

- Choose a [snitch](#).

A snitch determines which datacenters and racks are written to and read from.

- Choose [replica placement strategy](#).

A replication strategy determines the nodes where replicas are placed.

For information about how these components work, see [Data distribution and replication](#).

Partitioner settings

You can use either Murmur3Partitioner or RandomPartitioner with virtual nodes.

The [Murmur3Partitioner](#) (`org.apache.cassandra.dht.Murmur3Partitioner`) is the default partitioning strategy for Cassandra clusters. The Murmur3Partitioner is the right choice for new clusters in

almost all cases. You can use Murmur3Partitioner for new clusters; you cannot change the partitioner in existing clusters.

The [RandomPartitioner](#) (`org.apache.cassandra.dht.RandomPartitioner`) was the default partitioner in Cassandra 1.2 and earlier. You can continue to use this partitioner when migrating to virtual nodes.

Snitch settings

A snitch determines which datacenters and racks are written to and read from. It informs Cassandra about the network topology so that requests are routed efficiently and allows Cassandra to distribute replicas by grouping machines into datacenters and racks. All nodes must have exactly the same snitch configuration. You set the snitch in the [endpoint_snitch](#) property in the `cassandra.yaml` file.

The following sections describe commonly used snitches. All available snitches are described in the [Cassandra documentation](#).

DseSimpleSnitch

Use DseSimpleSnitch only for development in DataStax Enterprise deployments. This snitch logically configures each type of node in separate datacenters to segregate the analytics, real-time, and search workloads.

When defining your keyspace, use Analytics, Cassandra, or Search for your datacenter names.

Note: Do not use SimpleSnitch with DataStax Enterprise nodes.

GossipingPropertyFile Snitch

The GossipingPropertyFileSnitch defines a local node's datacenter and rack; it uses gossip for propagating this information to other nodes. The `cassandra-rackdc.properties` file defines the default datacenter and rack that are used by this snitch:

```
dc=DC1
rack=RAC1
```

PropertyFileSnitch

The PropertyFileSnitch property allows you to define your datacenter and rack names to be whatever you want. Using this snitch requires that you define network details for each node in the cluster in the `cassandra-topology.properties` configuration file.

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

For example, suppose you had non-uniform IPs and two physical datacenters with two racks in each, and a third logical datacenter for replicating analytics data, you would specify them as follows:

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

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

# Data Center Two
110.56.12.120=DC2:RAC1
```

```
110.50.13.201=DC2:RAC1
110.54.35.184=DC2:RAC1

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

# Analytics Replication Group

172.106.12.120=DC3:RAC1
172.106.12.121=DC3:RAC1
172.106.12.122=DC3:RAC1

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

Make sure the datacenter names defined in the `cassandra-topology.properties` file correlates to the data centers names in your [keyspace definition](#).

Choosing keyspace replication options

When you create a keyspace, you must define the [replica placement strategy class](#) and the number of replicas. DataStax recommends choosing NetworkTopologyStrategy for single and multiple datacenter clusters. This strategy is as easy to use as the SimpleStrategy and allows for expansion to multiple datacenters in the future. It is easier to configure the most flexible replication strategy when you create a keyspace, than to reconfigure replication after data is loaded into your cluster.

NetworkTopologyStrategy takes as options the number of replicas you want per data center. Even for single datacenter clusters, you can use this replica placement strategy and just define the number of replicas for one datacenter. For example:

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

For a single node cluster, use the default datacenter name, Cassandra, Solr, or Analytics.

```
CREATE KEYSPACE test
    WITH REPLICATION= { 'class' : 'NetworkTopologyStrategy',
'Analytics' : 1 } ;
```

To define the number of replicas for a multiple datacenter cluster:

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

When [creating the keyspace](#), what you name your datacenters depends on the [snitch](#) you have chosen for your cluster. The datacenter 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 are rejected, but reads are served as long as the desired [consistency level](#) can be met. The default consistency level is QUORUM.

To avoid DSE Hadoop operational problems, change the replication factor of these system keyspaces:

- cfs
- cfs_archive
- HiveMetaStore

Mixing workloads in a cluster

A common question is how to use the following types of nodes in the same cluster:

- Real-time Cassandra,
- DSE Hadoop, which is integrated Hadoop
- External Hadoop in the bring your own Hadoop (BYOH) model
- DSE Analytics (Spark)
- DSE Search nodes

The answer is to organize the nodes running different workloads into virtual datacenters: analytics workloads (either DSE Hadoop, Spark, or BYOH) nodes in one datacenter, search nodes in another, and Cassandra real-time nodes in another datacenter.

DataStax supports a datacenter that contains one or more nodes running in dual Spark/DSE Hadoop mode. Dual Spark/DSE Hadoop mode means you started the node using the -k and -t options on tarball or Installer-No Services installations, or set the startup options HADOOP_ENABLED=1 and SPARK_ENABLED=1 on package or Installer-Services installations.

DSE Analytics (Spark) workloads

The recommend approach for running Spark and Cassandra is to run all analytics (OLAP) in a datacenter that is separate from transactional (OLTP) Cassandra workload. This workload segregation avoids contention for Cassandra resources, and allows you to scale resources for OLTP and OLAP separately. If there is not a large demand for analytics, then you can run DataStax Enterprise in a single datacenter that serves both OLTP and OLAP requests. However, this combined transactional (OLTP) and analytics (OLAP) workload results in decreased performance.

DSE SearchAnalytics workloads (experimental)

DSE SearchAnalytics clusters can use DSE Search queries within DSE Analytics jobs. An integrated DSE SearchAnalytics cluster allows analytics jobs to be performed using [search queries](#).

BYOH workloads

BYOH nodes must be isolated from Cloudera or Hortonworks masters.

DSE Search workloads

The batch needs of Hadoop and the interactive needs of DSE Search are incompatible from a performance perspective, so these workloads need to be segregated.

Cassandra workloads

To keep Cassandra write throughout at the maximum performance, segregate Cassandra workloads separate from other workload types like DSE Search workloads.

Creating a virtual datacenter

When you [create a keyspace](#) using CQL, Cassandra creates a virtual datacenter for a cluster, even a one-node cluster, automatically. You assign nodes that run the same type of workload to the same datacenter. The separate, virtual datacenters for different types of nodes segregate workloads that run DSE Search from those nodes that run other workload types. Segregating workloads ensures that only one type of workload is active per datacenter.

Workload segregation

DataStax recommends separating nodes that run a sequential data load from nodes that run any other type of workload. In the following diagram, nodes in separate datacenters run a mix of:

Deploying

- Real-time queries (Cassandra and no other services)
- DSE Analytics (either DSE Hadoop, Spark, or dual mode DSE Hadoop/Spark)
- DSE Search
- External Hadoop system (BYOH)



In a cluster that has BYOH and DSE Hadoop nodes, the DSE Hadoop nodes would have priority with regard to start up. Start up seed nodes in the BYOH datacenter after starting up DSE Hadoop datacenters.

Occasionally, there is a use case for keeping DSE Hadoop and Cassandra nodes in the same datacenter. You do not have to have one or more additional replication factors when these nodes are in the same datacenter.

To deploy a mixed workload cluster, see "[Multiple datacenter deployment](#)."

In this diagram, nodes in datacenters 1 and 2 (DC 1 and DC 2) run a mix of:

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

Datacenters 3 and 4 (DC 3 and DC 4) are dedicated to search.



This diagram shows DSE Hadoop analytics, Cassandra, and DSE Search nodes in separate datacenters. In separate datacenters, some DSE nodes handle search while others handle MapReduce, or just act as real-time Cassandra nodes. Cassandra ingests the data, Solr indexes the data, and you run MapReduce against that data in one cluster without performing manual extract, transform, and load (ETL) operations. Cassandra handles the replication and isolation of resources. The DSE Search nodes run HTTP and hold the indexes for the Cassandra table data. If a DSE Search node goes down, the commit log replays the Cassandra inserts, which correspond to Solr inserts, and the node is restored automatically.

Restrictions

- Do not create the keyspace using SimpleStrategy for production use or for use with mixed workloads.
- Within the same datacenter, do not run Solr workloads on some nodes and other types of workloads on other nodes.
- Do not run DSE Search and DSE Hadoop on the same node in either production or development environments.

- Do not run some nodes in DSE Hadoop mode and some in Spark mode in the same datacenter.
You can run all the nodes in Spark mode, all the nodes in Hadoop mode or all the nodes in Spark/DSE Hadoop mode.

Recommendations

Run the CQL or Thrift inserts on a DSE Search node in its own datacenter.

NetworkTopologyStrategy is highly recommended for most deployments because it is much easier to expand to multiple datacenters when required by future expansion.

Getting cluster workload information

You can query the Cassandra [system.peers](#) table to get the types of workloads running on cluster nodes except the coordinator. The different workloads are:

- Analytics
- Cassandra
- Search

An Analytics workload is either DSE Hadoop, Spark, or dual mode DSE Hadoop/Spark. A Cassandra workload is Cassandra and no other services. A Search workload is DSE Search.

You can also query the system.local table to get the type of workload running on any local node. This table has a column of workload data that Cassandra does not include in the output when you select all the data. You must explicitly query the workload column.

```
SELECT workload FROM system.local;
```

The output looks something like this:

```
workload
-----
Analytics
```

Using the [DESCRIBE FULL schema](#) command reveals the definitions of all the columns. For example:

```
DESCRIBE FULL schema
```

The output shows the system and other table schemas. For example, the peers table schema is:

```
CREATE TABLE peers (
    peer inet,
    data_center text,
    host_id uuid,
    preferred_ip inet,
    rack text,
    release_version text,
    rpc_address inet,
    schema_version uuid,
    tokens settext,
    workload text,
    PRIMARY KEY ((peer))
) WITH
    . . .;
```

Replicating data across datacenters

You set up replication by [creating a keyspace](#). You can [change the replication](#) of a keyspace after creating it.

Single datacenter deployment per workload type

In this scenario, a mixed workload cluster has only one datacenter for each type of workload. For example, if the cluster has 3 analytics nodes, 3 Cassandra nodes, and 2 DSE Search nodes, the cluster has 3 datacenters, one for each type of workload. In contrast, a [multiple data-center cluster](#) has more than one datacenter for each type of workload.

In Cassandra, a datacenter can be a physical datacenter or virtual datacenter. Different workloads must always use separate datacenters, either physical or virtual. In a single datacenter deployment, data is replicated within its datacenter. For more information about replication:

- [Choosing keyspace replication options](#)
- [Configuring replication](#) on page 406
- [Single-token architecture deployment](#) on page 418
- [Data replication](#) (Applies only to the single-token-per-node architecture.)

Prerequisites

- A good understanding of how Cassandra works. Be sure to read at least [Understanding the architecture](#), [Data Replication](#), and [Cassandra's rack feature](#).
- Ensure DataStax Enterprise is installed on each node.
- Choose a name for the cluster.
- For a mixed-workload cluster, determine the purpose of each node.
- Determine the [snitch](#) and [replication strategy](#). The [GossipingPropertyFileSnitch](#) and [NetworkTopologyStrategy](#) are recommended for production environments.
- Get the IP address of each node.
- Determine which nodes are seed nodes. **Do not make all nodes seed nodes.** Seed nodes are not required for [DSE Search](#) datacenters. Read [Internode communications \(gossip\)](#).
- Use the [yaml_diff tool](#) to review and make appropriate changes to the `cassandra.yaml` configuration file.
- Review and make appropriate changes to other property files, such as `cassandra-rackdc.properties`.
- Set virtual nodes correctly for the type of datacenter. DataStax does not recommend using virtual nodes on datacenters running BYOH or DSE Hadoop. See [Virtual nodes](#).

Procedure

This configuration example describes installing an 8 node cluster spanning 2 racks in a single datacenter. The default consistency level is QUORUM.

1. Suppose the nodes have the following IPs and one node per rack will serve as a seed:
 - node0 110.82.155.0 (Cassandra seed)
 - node1 110.82.155.1 (Cassandra)
 - node2 110.82.155.2 (Cassandra)
 - node3 110.82.155.3 (Analytics seed)
 - node4 110.82.155.4 (Analytics)
 - node5 110.82.155.5 (Analytics)
 - node6 110.82.155.6 (Search)
 - node7 110.82.155.7 (Search)
2. If the nodes are behind a firewall, open the required ports for internal/external communication. See [Configuring firewall port access](#).
3. If DataStax Enterprise is running, stop the nodes and clear the data:

- Installer-Services and Package installations:

```
$ sudo service dse stop
$ sudo rm -rf /var/lib/cassandra/* # Clears the data from the default
directories
```

- Installer-No Services and Tarball installations:

From the install directory:

```
$ sudo bin/dse cassandra-stop
$ sudo rm -rf /var/lib/cassandra/* # Clears the data from the default
directories
```

Note: If you are clearing data from an AMI installation for restart, you need to [preserve the log files](#).

4. Set the properties in the `cassandra.yaml` file for each node, located in:

Important: After making any changes in `cassandra.yaml`, you must restart the node for the changes to take effect.

If the nodes in the cluster are identical in terms of disk layout, shared libraries, and so on, you can use the same copy of the `cassandra.yaml` file on all of the nodes.

Properties to set:

- `num_tokens: 256` for Cassandra nodes
- `num_tokens: 1` for Hadoop and DSE Search nodes
- `num_tokens: 64` to `256` for DSE Search nodes when using vnodes; otherwise `num_tokens: 1`
- `-seeds: internal_IP_address` of each seed node
- `listen_address: empty`

If not set, Cassandra asks the system for the local address, the one associated with its host name. In some cases Cassandra doesn't produce the correct address and you must specify the `listen_address`.

- `endpoint_snitch: snitch` See [endpoint_snitch](#). If you are changing snitches, see [Switching snitches](#).
- `auto_bootstrap: false`

Add the `bootstrap` setting only when initializing a fresh cluster with no data.

- `endpoint_snitch: snitch`

For more information, see [endpoint_snitch](#) and [About Snitches](#).

- If you are using a `cassandra.yaml` file from a previous version, remove the following options, as they are no longer supported by DataStax Enterprise:

```
## Replication strategy to use for the auth keyspace.
auth_rePLICATION_strategy: org.apache.cassandra.locator.SimpleStrategy

auth_rePLICATION_options:
    replication_factor: 1
```

Example:

```
cluster_name: 'MyDemoCluster'
num_tokens: 256
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      - seeds: "110.82.155.0,110.82.155.3"
listen_address:
endpoint_snitch: GossipingPropertyFileSnitch
```

Deploying

- In the `cassandra-rackdc.properties` (`GossipingPropertyFileSnitch`) or `cassandra-topology.properties` (`PropertyFileSnitch`) file, use your naming convention to assign datacenter and rack names to the IP addresses of each node, and assign a default datacenter name and rack name for unknown nodes.

Example:

```
# Cassandra Node IP=Data Center:Rack
110.82.155.0=DC_Cassandra:RAC1
110.82.155.1=DC_Cassandra:RAC1
110.82.155.2=DC_Cassandra:RAC1
110.82.155.3=DC_Analytics:RAC1
110.82.155.4=DC_Analytics:RAC1
110.82.155.5=DC_Analytics:RAC1
110.82.155.6=DC_Solr:RAC1
110.82.155.7=DC_Solr:RAC1

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

- After you have installed and configured DataStax Enterprise on all nodes, start the seed nodes one at a time, and then start the rest of the nodes:
 - Packages/Services: See [Starting DataStax Enterprise as a service](#).
 - Tarball/No Services: See [Starting DataStax Enterprise as a stand-alone process](#).

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

- Check that your cluster is up and running:

- Installer-Services and Package installations: `$ nodetool status`
- Installer-No Services and Tarball installations: `$ install_location/bin/nodetool status`

Results

```
Datacenter: Cassandra
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address          Load      Tokens  Owns      Host ID      Rack
UN 110.82.155.0    21.33 KB    256     33.3%    a9fa31c7-f3c0-...  RAC1
UN 110.82.155.1    21.33 KB    256     33.3%    f5bb416c-db51-...  RAC1
UN 110.82.155.2    21.33 KB    256     16.7%    b836748f-c94f-...  RAC1

Datacenter: Analytics
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address          Load      Owns      Host ID      Tokens
   Rack
UN 110.82.155.3    28.44 KB    13.0.%  e2451cdf-f070-...  -922337...
   RAC1
UN 110.82.155.4    44.47 KB    16.7%   f9fa427c-a2c5-...  30745512...
   RAC1
UN 110.82.155.5    54.33 KB    23.6%   b9fc31c7-3bc0-...  45674488...
   RAC1

Datacenter: Solr
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address          Load      Owns      Host ID      Tokens
   Rack
UN 110.82.155.6    15.44 KB    50.2.%  e2451cdf-f070-...  9243578...
```

UN 110.82.155.7	18.78 KB	49.8.%	e2451cdf-f070-...	10000
RAC1				

Related concepts[DSE Search](#) on page 192**Related information**[DSE Analytics](#) on page 73

Multiple datacenter deployment per workload type

In this scenario, a mixed workload cluster has more than one datacenter for each type of node. For example, if the cluster has 4 analytics nodes, 4 Cassandra nodes, and 2 DSE Search nodes, the cluster could have 5 datacenters: 2 datacenters for analytics nodes, 2 datacenters for Cassandra nodes, and 1 datacenter for the DSE Search node. A [single datacenter cluster](#) has only one datacenter for each type of node.

In Cassandra, a datacenter can be a physical datacenter or virtual datacenter. Different workloads must always use separate datacenters, either physical or virtual.

Uses for multiple datacenter deployments include:

- Isolating replicas from external infrastructure failures, such as networking between datacenters and power outages.
- Distributed data replication across multiple, geographically dispersed nodes.
- Between different physical racks in a physical datacenter.
- Between public cloud providers and on-premise managed datacenters.
- To prevent the slow down of a real-time analytics cluster by a development cluster running analytics jobs on live data.
- To ensure your reads from a specific datacenter is local to the requests, especially when using a consistency level greater than ONE, use virtual datacenters in the physical datacenter. This strategy ensures lower latency because it avoids reads from one node in New York and another read from a node in Los Angeles.

For more information about replication:

- [Choosing keyspace replication options](#)
- [Configuring replication](#) on page 406
- [Single-token architecture deployment](#) on page 418
- [Data replication](#) (Applies only to the single-token-per-node architecture.)

Prerequisites

To configure a multi-node cluster with multiple datacenters:

- A good understanding of how Cassandra works. Be sure to read at least [Understanding the architecture](#), [Data Replication](#), and [Cassandra's rack feature](#).
- Ensure DataStax Enterprise is installed on each node.
- Choose a name for the cluster.
- For a mixed-workload cluster, determine the purpose of each node.
- Determine the [snitch](#) and [replication strategy](#). The [GossipingPropertyFileSnitch](#) and [NetworkTopologyStrategy](#) are recommended for production environments.
- Get the IP address of each node.
- Determine which nodes are seed nodes. **Do not make all nodes seed nodes.** Seed nodes are not required for [DSE Search](#) datacenters. Read [Internode communications \(gossip\)](#).

Deploying

- Develop a naming convention for each datacenter and rack, for example: DC1, DC2 or 100, 200 and RAC1, RAC2 or R101, R102.
- Use the [yaml_diff tool](#) to review and make appropriate changes to the `cassandra.yaml` configuration file.
- Set virtual nodes correctly for the type of datacenter. DataStax does not recommend using virtual nodes on datacenters running BYOH or DSE Hadoop. See [Virtual nodes](#).

Procedure

This configuration example describes installing a 6 node cluster spanning 2 data centers. The default consistency level is QUORUM.

1. Suppose you install DataStax Enterprise on these nodes:

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

2. If the nodes are behind a firewall, open the required ports for internal/external communication. See [Configuring firewall port access](#).

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

- Installer-Services and Package installations:

```
$ sudo service dse stop  
$ sudo rm -rf /var/lib/cassandra/* # Clears the data from the default  
directories
```

- Installer-No Services and Tarball installations:

From the install directory:

```
$ sudo bin/dse cassandra-stop  
$ sudo rm -rf /var/lib/cassandra/* # Clears the data from the default  
directories
```

Note: If you are clearing data from an AMI installation for restart, you need to [preserve the log files](#).

4. Set the properties in the `cassandra.yaml` file for each node:

Important: After making any changes in `cassandra.yaml`, you must restart the node for the changes to take effect.

Properties to set:

Note: If the nodes in the cluster are identical in terms of disk layout, shared libraries, and so on, you can use the same copy of the `cassandra.yaml` file on all of them.

- `num_tokens: 256` for Cassandra nodes
- `num_tokens: 1` for Hadoop and DSE Search nodes
- `num_tokens: 64 to 256` for DSE Search nodes when using vnodes; otherwise `num_tokens: 1`
- `-seeds: internal_IP_address` of each seed node
- `listen_address: empty`

If not set, Cassandra asks the system for the local address, the one associated with its host name. In some cases Cassandra doesn't produce the correct address and you must specify the `listen_address`.

- `endpoint_snitch: snitch` See [endpoint_snitch](#). If you are changing snitches, see [Switching snitches](#).
- `auto_bootstrap: false`

Add the `bootstrap` setting only when initializing a fresh cluster with no data.

- endpoint_snitch: *snitch*
For more information, see [endpoint_snitch](#) and [About Snitches](#).
- If you are using a `cassandra.yaml` file from a previous version, remove the following options, as they are no longer supported by DataStax Enterprise:

```
## Replication strategy to use for the auth keyspace.
auth_rePLICATION_strategy: org.apache.cassandra.locator.SimpleStrategy

auth_rePLICATION_options:
    replication_factor: 1
```

Example:

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

```
cluster_name: 'MyDemoCluster'
num_tokens: 256
seed_provider:
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
        parameters:
            - seeds: "10.168.66.41,10.176.170.59"
listen_address:
endpoint_snitch: GossipingPropertyFileSnitch
```

5. In the `cassandra-rackdc.properties` (GossipingPropertyFileSnitch) or `cassandra-topology.properties` (PropertyFileSnitch) file, use your naming convention to assign datacenter and rack names to the IP addresses of each node, and assign a default datacenter name and rack name for unknown nodes.

Example:

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

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

6. After you have installed and configured DataStax Enterprise on all nodes, start the seed nodes one at a time, and then start the rest of the nodes:
 - Packages/Services: See [Starting DataStax Enterprise as a service](#).
 - Tarball/No Services: See [Starting DataStax Enterprise as a stand-alone process](#).

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

7. Check that your cluster is up and running:

- Installer-Services and Package installations: `$ nodetool status`
- Installer-No Services and Tarball installations: `$ install_location/bin/nodetool status`

Results

```
Datacenter: DC1
=====
Status=Up/Down
```

```
| / State=Normal/Leaving/Joining/Moving
-- Address Load Tokens Owns Host ID Rack
UN 10.168.66.41 45.96 KB 256 27.4% c885aac7-f2c0-... RAC1
UN 10.168.247.41 66.34 KB 256 36.6% fa31416c-db22-... RAC1
UN 10.169.61.170 55.72 KB 256 33.0% f488367f-c14f-... RAC1
Datacenter: DC2
=====
Status=Up/Down
| / State=Normal/Leaving/Joining/Moving
-- Address Load Tokens Owns Host ID Rack
UN 10.176.43.66 45.96 KB 256 27.4% f9fa31c7-f3c0-... RAC1
UN 10.176.170.59 66.34 KB 256 36.6% a5bb526c-db51-... RAC1
UN 10.169.30.138 55.72 KB 256 33.0% b836478f-c49f-... RAC1
```

What to do next

- [Configuring replication](#) on page 406
- [Configuring system_auth and dse_security keyspace replication](#) on page 347
- [Configuring replication](#) on page 406

Related concepts

[DSE Search](#) on page 192

Related information

[DSE Analytics](#) on page 73

Single-token architecture deployment

Follow these steps only when not using [virtual nodes](#) (vnodes).

Prerequisites

- Have a basic understanding of [tokens](#) and [Database internals](#).
- Ensure DataStax Enterprise is installed on each node.
- Choose a name for the cluster.
- Take note of the total number of nodes in the cluster.
- For a mixed-workload cluster, determine the purpose of each node.
- Determine which nodes are seed nodes. **Do not make all nodes seed nodes.** Seed nodes are not required for [DSE Search](#) datacenters. Read [Internode communications \(gossip\)](#).
- If using multiple datacenters, develop a naming convention for each datacenter and rack, for example: DC1, DC2 or 100, 200 and RAC1, RAC2 or R101, R102.
- Use the [yaml_diff tool](#) to review and make appropriate changes to the cassandra.yaml configuration file.

Procedure

1. Suppose you install DataStax Enterprise on these nodes:
 - node0 10.168.66.41 (seed1)
 - node1 10.176.43.66
 - node2 10.168.247.41
 - node3 10.176.170.59 (seed2)
 - node4 10.169.61.170
 - node5 10.169.30.138
2. Calculate the token assignments using the information on [Calculating tokens](#) on page 421.

Table: Single Data Center

Node	Token
node0	0
node1	21267647932558653966460912964485513216
node2	42535295865117307932921825928971026432
node3	63802943797675961899382738893456539648
node4	85070591730234615865843651857942052864
node5	106338239662793269832304564822427566080

Table: Multiple Data Centers

Node	Token	Offset	Data Center
node0	0	NA	DC1
node1	567137278201564105772291012386280 05 242	05	DC1
node2	113427455640312821154458202477256 07 0485	07	DC1
node3	100	100	DC2
node4	567137278201564105772291012386280 36 042	36	DC2
node5	113427455640312821154458202477256 00 0585	00	DC2

3. If the nodes are behind a firewall, open the required ports for internal/external communication. See [Configuring firewall port access](#).
4. If DataStax Enterprise is running, stop the nodes and clear the data:

- Installer-Services and Package installations:

```
$ sudo service dse stop
$ sudo rm -rf /var/lib/cassandra/* # Clears the data from the default
                                directories
```

- Installer-No Services and Tarball installations:

From the install directory:

```
$ sudo bin/dse cassandra-stop
$ sudo rm -rf /var/lib/cassandra/* # Clears the data from the default
                                directories
```

Note: If you are clearing data from an AMI installation for restart, you need to [preserve the log files](#).

5. Set the properties in the `cassandra.yaml` file for each node.

Important: After making any changes in `cassandra.yaml`, you must restart the node for the changes to take effect.

Properties to set:

- `initial_token: token`
- `num_tokens: 1`
- `-seeds: internal_IP_address` of each seed node
- `listen_address: empty`

If not set, Cassandra asks the system for the local address, the one associated with its hostname. In some cases Cassandra doesn't produce the correct address and you must specify the `list_address`.

- endpoint_snitch: *snitch*
For more information, see [About Snitches](#).
- auto_bootstrap: *false*
Add the `bootstrap` setting only when initializing a fresh cluster with no data.
- If you are using a `cassandra.yaml` from a previous version, remove the following options, as they are no longer supported by DataStax Enterprise:

```
## Replication strategy to use for the auth keyspace.  
auth_rePLICATION_strategy: org.apache.cassandra.locator.SimpleStrategy  
  
auth_rePLICATION_options:  
    replication_factor: 1
```

Example:

If using more than one datacenter, include at least one seed node from each datacenter. It is a best practice to have more than one seed node per data center.

```
cluster_name: 'MyDemoCluster'  
num_tokens: 256  
seed_provider:  
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider  
        parameters:  
            - seeds: "10.168.66.41,10.176.170.59"  
listen_address:
```

6. In the `cassandra-topology.properties` file, use your naming convention to assign datacenter and rack names to the IP addresses of each node, and assign a default datacenter name and rack name for unknown nodes.

Example:

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

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

- Packages/Services: See [Starting DataStax Enterprise as a service](#).
- Tarball/No Services: See [Starting DataStax Enterprise as a stand-alone process](#).

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

8. Check that your cluster is up and running:

- Installer-Services and Package installations: `$ nodetool status`
- Installer-No Services and Tarball installations: `$ install_location/bin/nodetool status`

Calculating tokens

When you are not using vnodes, follow use this information to calculate tokens when using single-token architecture. You do not need calculate tokens when using virtual nodes (vnodes).

When you start a DataStax Enterprise cluster, you must choose how the data is divided across the nodes in the cluster. A partitioner determines what each node stores by row. The token generator can generate tokens for the Murmur3Partitioner or the RandomPartitioner.

A token is a partitioner-dependent element of the cluster. Each node in a cluster is assigned a token and that token determines the node's position in the ring and what data the node is responsible for in the cluster. Distribute the tokens assigned to your nodes throughout the entire possible range of tokens. Each node is responsible for the region of the ring between itself (inclusive) and its predecessor (exclusive). As a simple example, if the range of possible tokens is 0 to 100 and there are 4 nodes, the tokens for the nodes would be: 0, 25, 50, 75. This approach ensures that each node is responsible for an equal range of data.

Note:

For multiple datacenter deployments, calculate the tokens for each datacenter so that the hash range is evenly divided for the nodes in each datacenter. Partition the datacenter as if it were its own distinct ring.

For single datacenter deployments, calculate tokens by dividing the hash range by the number of nodes in the cluster.

Before the node is started for the first time, each node in the cluster must be assigned a token. Set the token with the [initial_token](#) property in the `cassandra.yaml` configuration file. Be sure to comment out the `num_tokens` property.

See [Generating tokens](#) for steps to generate tokens.

For more information on the Cassandra database, see [Database internals](#).

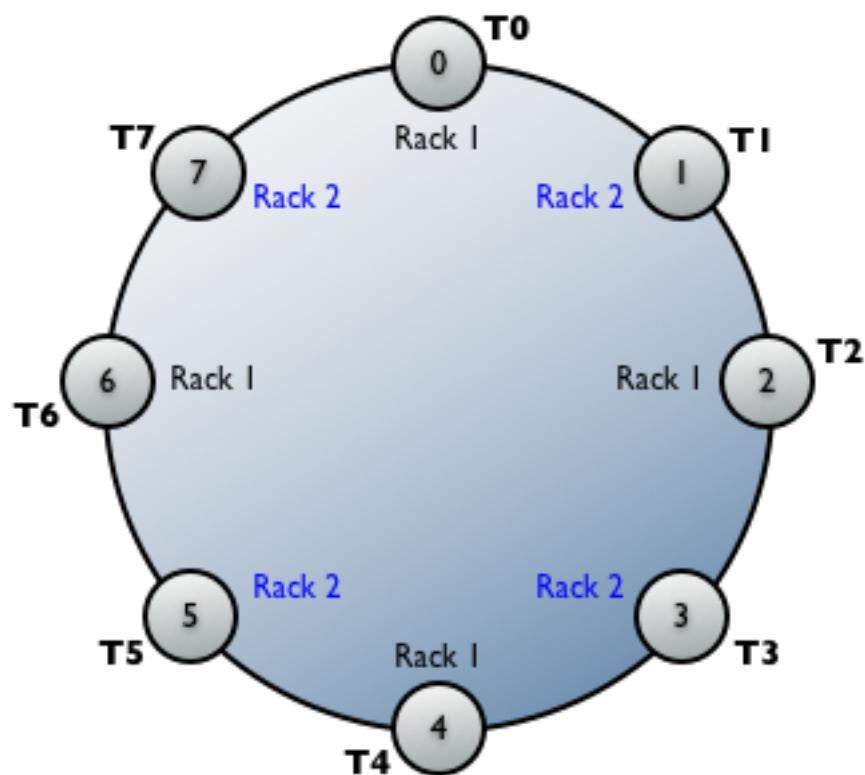
Calculating tokens for a single datacenter

For example, for 6 nodes in a single datacenter, the results for calculating tokens using the [Murmur3Partitioner](#) are:

```
[ '-9223372036854775808', '-6148914691236517206', '-3074457345618258604',
  '-2', '3074457345618258600', '6148914691236517202' ]
```

Calculating tokens for multiple racks in a single datacenter

If you have multiple racks in single datacenter, calculate the tokens for the number of nodes and then assign the tokens to nodes on alternating racks. For example: rack1, rack2, rack1, rack2, and so on. The image shows the rack assignments:



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

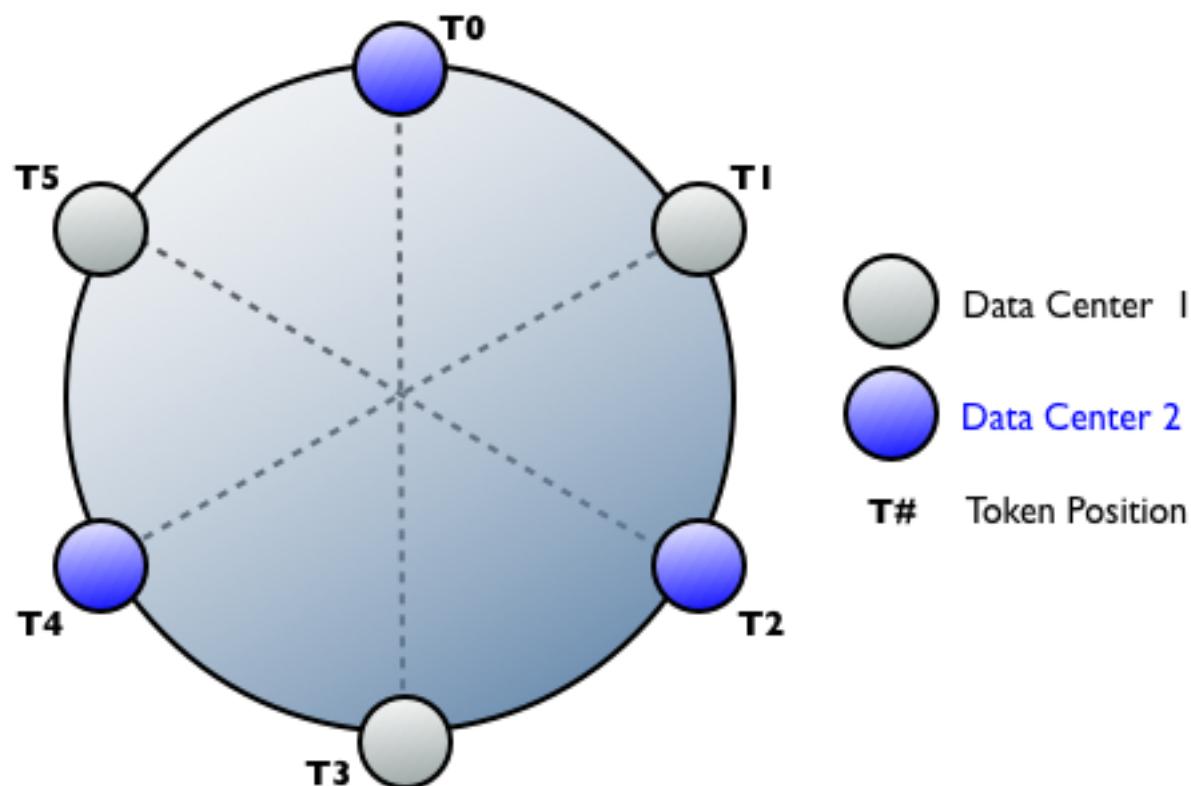
Calculating tokens for a multiple datacenter cluster

In multiple datacenter deployments using the NetworkTopologyStrategy, calculate the replica placement for your custom keyspaces per datacenter. The NetworkTopologyStrategy determines replica placement independently within each datacenter. The first replica is placed according to the partitioner. Additional replicas in the same datacenter are determined by walking the ring clockwise until a node in a different rack from the previous replica is found. If no such node exists, additional replicas are placed in the same rack.

Do not use SimpleStrategy for this type of cluster. There are different methods you can use when calculating multiple datacenter clusters. The important point is that the nodes within each datacenter manage an equal amount of data; the distribution of the nodes within the cluster is not as important. DataStax recommends using DataStax Enterprise OpsCenter to re-balance a cluster.

Alternating token assignments

Calculate the tokens for each datacenter and then alternate the token assignments so that the nodes for each datacenter are evenly dispersed around the ring. The following image shows the token position and datacenter assignments:



Avoiding token collisions

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

The following shows the tokens for a cluster with two 3 node datacenters and one 2 node datacenter.

Tokens for 3 nodes:

```
[ '-9223372036854775808',
  '-3074457345618258603',
  '3074457345618258602']
```

Tokens for 2 nodes:

```
[ '-9223372036854775808',
  '0']
```

Using an offset value of 100:

- Data Center 1

```
[ '-9223372036854775808',
  '-3074457345618258603',
  '3074457345618258602']
```

- Data Center 2

```
[ '-9223372036854775708',
  '-3074457345618258503',
  '3074457345618258702']
```

- Data Center 3

```
[ '-9223372036854775608']
```

'200']

Expanding a DataStax AMI cluster

The best way to expand your EC2 implementations is to use OpsCenter:

- [Provisioning a new cluster](#)
- [Adding an existing cluster](#)
- [Adding nodes to a cluster](#)

DataStax Enterprise data migration

Migrating data using Sqoop

About Sqoop

DSE Hadoop supports Sqoop, 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 supports the following operations:

- Import and export data to and from CQL tables and any JDBC-compliant data source.
- Import SQL files into a CQL collection set, list, and map.
- Import data into CQL using a re-useable, file-based import command.
- Import legacy data using the `thrift-import` tool that supports backward compatibility with previous DataStax Enterprise versions.
- Use conventional Sqoop commands to import data into the Cassandra File System (CFS), the counterpart to HDFS, instead of a CQL table.

You can import and export MySQL, PostgreSQL, and Oracle data types that are listed in the [Sqoop reference](#). An analytics node runs the MapReduce job that imports and exports data from a data source using Sqoop. You need a JDBC driver for the RDBMS or other type of data source.

Importing data

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

- DB2
- MySQL
- Oracle
- SQL Server
- Sybase

Securing Sqoop

DataStax Enterprise supports password authentication for Sqoop operations. Configure password authentication using [Cassandra-specific properties](#). Kerberos is also supported. [Client-to-node encryption](#) (SSL) is supported for Sqoop-imported and exported data.

Running the Sqoop demo

The Sqoop 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. The demo runs SQL commands to put the data from a CSV file into a MySQL table in a MySQL database. You then import the SQL data from MySQL to a CQL table in Cassandra. The following steps show running the SQL commands from the mysql command line. Alternatively, you can run the commands on the operating system command line described below. The demo exports the data from MySQL, and then uses a subset of Sqoop commands to import the data into a CQL table.

Prerequisites

To run the demo, you need:

- Latest version of Oracle Java SE Development Kit (JDK) 7. The JRE alone does not work.
- An installation of MySQL
- Sufficient MySQL database privileges to create database objects
- A JDBC driver for MySQL in the directory specified by the following demo procedure
- The connection string that is appropriate for the JDBC driver
- A DataStax Enterprise Analytics node
- A PATH environment variable that includes the `bin` directory of the DSE installation

To import data to CQL, the keyspace and CQL table must exist prior to the importation. If the CQL table contains data prior to the importation, `cql-import` updates the data.

Procedure

1. Install MySQL and download the JDBC driver for MySQL from the MySQL site.
2. Copy the JDBC driver for MySQL to the Sqoop library.

The default location of the Sqoop library depends on the type of installation:

Installer-Services	<code>/usr/share/dse/resources/sqoop/lib</code>
Package installations	<code>/usr/share/dse/sqoop/lib</code>
Installer-No Services and Tarball installations	<code>install_location/resources/sqoop/lib</code>

3. Start DataStax Enterprise as an analytics node. For example:

- **Installer-Services and Package installations:**

1. Set `HADOOP_ENABLED=1` in `/etc/default/dse`.
2. Start an analytics node:

```
$ sudo service dse start
```

- **Installer-No Services and Tarball installations:**

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

4. Start MySQL and create the demo database:

```
mysql> CREATE DATABASE npa_nxx_demo ;
```

5. Connect to the database and create the table:

```
mysql> CONNECT npa_nxx_demo;
```

```
mysql> CREATE TABLE npa_nxx (
    npa_nxx_key int(11) NOT NULL,
    npa          int(11) DEFAULT NULL,
    nxx          int(11) DEFAULT NULL,
    lat          float  DEFAULT NULL,
    lon          float  DEFAULT NULL,
    linetype     char(1) DEFAULT NULL,
    state        varchar(2) DEFAULT NULL,
    city         varchar(36) DEFAULT NULL,
    PRIMARY KEY (npa_nxx_key)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

- Locate the `npa_nxx_csv` file of the DataStax Enterprise installation.

The default location of the Sqoop demo depends on the type of installation:

Installer-Services and Package installations	/usr/share/dse/demos/sqoop
Installer-No Services and Tarball installations	<i>install_location/demos/sqoop</i>

- Populate the table by loading the CSV file:

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

Output is:

```
Query OK, 105291 rows affected (1.01 sec)
Records: 105291 Deleted: 0 Skipped: 0 Warnings: 0
```

- On the analytics node you started in [step 3](#), create a CQL keyspace and table that maps to the SQL table. Use [compatible data types](#). For example, start cqlsh and run these commands:

```
cqlsh> CREATE KEYSPACE npa_nxx WITH REPLICATION =
           {'class':'NetworkTopologyStrategy', 'Analytics':1};

cqlsh> CREATE TABLE npa_nxx.npa_nxx_data (npa int, nxx int,
                                             latitude float, longitude float, state text, city text,
                                             PRIMARY KEY(npa, nxx));
```

Alternatively, you can run the commands on the operating system command line from a cql script in the `demos/sqoop` directory.

The default location of the Sqoop demo depends on the type of installation:

Installer-Services and Package installations	/usr/share/dse/demos/sqoop
Installer-No Services and Tarball installations	<i>install_location/demos/sqoop</i>

- In a text editor, open the `import.options` file in the `demos/sqoop` directory.

The default location of the Sqoop demo depends on the type of installation:

Installer-Services and Package installations	/usr/share/dse/demos/sqoop
Installer-No Services and Tarball installations	<i>install_location/demos/sqoop</i>

The `import.options` file contains these options:

Table: import.options file

Contents	Description
cql-import	Perform an import operation.
--table	A SQL table name follows this option.
npa_nxx	SQL table name for the demo.
--cassandra-keyspace	A keyspace name follows this option.
npa_nxx	The keyspace name for the demo.
--cassandra-table	A Cassandra table name follows this option.
npa_nxx_data	The Cassandra table name for the demo.
--cassandra-column-mapping	A CQL:SQL column mapping follows this option.
npa:npa,nxx:nxx,latitude:lat,longitude:lon,state:state	FROM: Cassandra column names:corresponding MySQL column names, cql1:sql1,cql2:sql2, . . .
--connect	The JDBC connection string follows this option.
jdbc:mysql://<mysql_host>/npa_nxx_demo	The JDBC connection string.
--username	A MySQL user name follows this option.
<mysql_user>	The user name you configured as the MySQL admin.
--password	A MySQL password follows this option.
<mysql_password>	The MySQL administrative password you configured.
--cassandra-host	The IP address of the MySQL host node follows this option.
<cassandra_host>	The IP address of the host node. For example, 127.0.0.1. A fully-qualified domain name if using Kerberos.

Alternatively, you can enter these commands as options to the `dse sqoop` command to import the data from the SQL to the CQL table as shown in the example of exporting data.

10. Modify the `import.options` file for your environment. For example, assuming you plan to run the demo on a single-node cluster, modify the options as follows:

- --connect
FROM: `jdbc:mysql://<mysql_host>/npa_nxx_demo`
TO: `jdbc:mysql://127.0.0.1/npa_nxx_demo`
- --username
FROM: `mysql_user`
TO: your MySQL user name
- --password
FROM: `mysql_password`
TO: your MySQL password
- --cassandra-host

```
FROM: cassandra_host
```

```
TO: 127.0.0.1
```

As described in the Sqoop reference, you can list multiple IP addresses.

- Import the SQL data into Cassandra using the file you edited. Use the `dse import` command to import the data from the MySQL table to the table in Cassandra. On Linux, for example:

```
$ bin/dse sqoop --options-file fully_qualified_path/demos/sqoop/import.options
```

The MapReduce job runs and the end of the output looks like this:

```
14/05/23 14:41:17 INFO mapreduce.ImportJobBase: Transferred 0 bytes in  
50.5956 seconds (0 bytes/sec)  
14/05/23 14:41:17 INFO mapreduce.ImportJobBase: Retrieved 105291 records.
```

- In `cqlsh`, verify that the data import succeeded.

```
cqlsh> SELECT * FROM npa_nxx.npa_nxx_data LIMIT 5;
```

npa	nxx	city	latitude	longitude	state
660	200	Braymer	39.59	93.8	MO
660	202	Sedalia	38.7	93.22	MO
660	213	La Belle	40.11	91.91	MO
660	214	Chillicothe	39.79	93.55	MO
660	215	Maryville	40.34	94.87	MO

Importing SQL to a CQL table or CFS

To import data to CQL, the keyspace and CQL table must exist prior to the importation. If the CQL table contains data prior to the importation, `cql-import` updates the data. The Sqoop [demo](#) shows how to import SQL data into a CQL table.

In addition to importing data to a CQL table, you can also import data to the Cassandra File System (CFS). The CFS is the Cassandra counterpart to the Hadoop Distributed File System (HDFS). The example in this section shows how to import SQL data to CFS. Using Hive and other utilities, you can access the CFS data.

Procedure

- Follow the steps in the [Sqoop demo](#) to create the SQL database and table and the CQL keyspace and table.
- Use the `dse sqoop import` command to migrate the data from the MySQL table to text files in the directory `npa_nxx` in the CFS. Use the database username and password. If the database account is not password-protected, just omit the password option. On Linux, for example:

```
$ bin/dse sqoop import --connect  
jdbc:mysql://127.0.0.1/npa_nxx_demo  
--username mysql  
--password password  
--table npa_nxx
```

```
--target-dir /npa_nxx
```

DataStax Enterprise returns this message:

```
INFO mapreduce.ImportJobBase: Retrieved 105291 records.
```

3. Use the command to view the results in the CFS. On Linux, for example:

```
$ bin/dse hadoop fs -ls /npa_nxx
```

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

To view the contents of these files, use this [hadoop fs](#) command:

```
$ bin/dse hadoop fs -cat /npa_nxx/part-m-00000
```

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

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

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

Importing data into a CQL list or set

DataStax Enterprise supports importing data into a CQL collection using the cql-import tool. You can use the cql-import tool to map SQL columns to items in a collection set, list, or map.

The cql-import tool supports two distinct mechanisms for importing data into list and set data types. Both mechanisms use the --cassandra-column-mapping parameter.

Mapping multiple SQL columns in a single row to a CQL list or set

The cql-import command supports the following cassandra-column-mapping parameter for mapping multiple SQL columns in a single row to a CQL list or set.

```
CQLCOL: [SQLCOL1,SQLCOL2,SQLCOL3]
```

This form of mapping adds the SQL columns SQLCOL1,SQLCOL2, and SQLCOL3 to the list or set CQLCOL.

The following example shows how to map a MySQL table having multiple SQL columns in a single row to a CQL list.

Suppose you have created and populated a MySQL table using the following commands:

```
mysql> CREATE TABLE sql_table (sqlid INTEGER PRIMARY KEY, a VARCHAR(25), b VARCHAR(25), c VARCHAR(25));
mysql> INSERT INTO sql_table (sqlid, a, b, c) values (1, 'valuea', 'valueb', 'valuec');
mysql> INSERT INTO sql_table (sqlid, a, b, c) values (2, 'valued', 'valuee', 'valuef');
```

Using cqlsh, suppose you create the following table in Cassandra that corresponds to the MySQL table:

```
cqlsh> CREATE TABLE cql_table (cqlid int PRIMARY KEY, mylist listtext);
```

The following map along with other options imports the data into CQL:

```
--cassandra-column-mapping cqlid:sqlid,mylist:[a,b,c]
```

Note that there are no spaces after the comma (,) or the colon (:) in cqlid:sqlid,mylist:[a,b,c].

Querying Cassandra to select the table produces the following output:

id	mylist
1	{'valuea', 'valueb', 'valuec'}
2	{'valued', 'valuee', 'valuef'}

Mapping a single SQL column from multiple SQL rows to a CQL list or set

The cql-import command also supports the following cassandra-column-mapping parameter to map a single SQL column from multiple SQL rows to a CQL list or set.

```
CQLCOL:SQLCOL
```

This form of mapping appends SQL column values from multiple SQL rows that share a common key to a CQL list or set.

The following example shows how to map a MySQL table having a single SQL column from multiple SQL rows to a CQL list.

Suppose you have created and populated a MySQL table using the following commands:

```
mysql> CREATE TABLE sql_table (sqlid INTEGER PRIMARY KEY, id INTEGER, a VARCHAR(25));
mysql> INSERT INTO sql_table (sqlid, id, a) values (1, 1, 'valuea');
mysql> INSERT INTO sql_table (sqlid, id, a) values (2, 1, 'valueb');
mysql> INSERT INTO sql_table (sqlid, id, a) values (3, 1, 'valuec');
mysql> INSERT INTO sql_table (sqlid, id, a) values (4, 2, 'valued');
mysql> INSERT INTO sql_table (sqlid, id, a) values (5, 2, 'valuee');
```

Using cqlsh, suppose you create the following table in Cassandra that corresponds to the MySQL table:

```
cqlsh> CREATE TABLE cql_table (cqlid int PRIMARY KEY, mylist listtext);
```

The following map along with other options imports the data into CQL:

```
--cassandra-column-mapping cqlid:id,mylist:a
```

Note that there are no spaces after the comma (,) or the colon (:) in cqlid:id,mylist:a.

Querying Cassandra to select the table produces the following output:

id	mylist
1	{'valuea', 'valueb', 'valuec'}
2	{'valued', 'valuee'}

Importing data into a CQL map

You can use the cql-import tool to map SQL columns to items in a map, similar to importing data into list and set. You use the following cassandra-column-mapping parameter to import data into a map.

```
CQLCOL: [KEY1:SQLCOL1,KEY2:SQLCOL2,KEY3:SQLCOL3]
```

Note that there are no spaces after the comma (,) or the colon (:) in [KEY1:SQLCOL1,KEY2:SQLCOL2,KEY3:SQLCOL3].

This form of mapping maps SQL column data to map entries using the key name specified in the mapping. The SQL column names can be used as the key names by omitting the key from the mapping.

The mapping mechanism supports a mixed key name mapping.

```
CQLCOL: [KEY1:SQLCOL1,SQLCOL2,KEY3:SQLCOL3]
```

The following example shows how to import a MySQL table into a CQL map collection.

Suppose you have created and populated a MySQL table using the following commands:

```
mysql> CREATE TABLE sql_table (sqlid INTEGER PRIMARY KEY, a VARCHAR(25), b VARCHAR(25), c VARCHAR(25));
mysql> INSERT INTO sql_table (sqlid, a, b, c) values (1, 'valuea', 'valueb', 'valuec');
mysql> INSERT INTO sql_table (sqlid, a, b, c) values (2, 'valued', 'valuee', 'valuef');
```

Using cqlsh, create the following table in Cassandra that corresponds to the MySQL table:

```
cqlsh> CREATE TABLE cql_table (cqlid int PRIMARY KEY, mymap map<text>,text);
```

The following map along with other options imports the data into CQL:

```
--cassandra-column-mapping cqlid:sqlid,mymap:[key1:a,b,key3:c]
```

Note that there are no spaces after the comma (,) or the colon (:) in cqlid:sqlid,mymap:[key1:a,b,key3:c].

Querying Cassandra to select the table produces the following output:

cqlid	mymap
1	{'key1':'valuea', 'b':'valueb', 'key3':'valuec'}
2	{'key1':'valued', 'b':'valuee', 'key3':'valuef'}

Importing joined tables

A common use case is to import multiple tables, which are joined in SQL, to Cassandra. This example shows how to import two tables from MySQL. In MySQL, you use query joins to get famous quotations from one table and the author of the quotation from another. For example:

```
mysql> SELECT * FROM person INNER JOIN mysql_quotations ON
person.id=mysql_quotations.speaker;
```

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| id | name | title | id | speaker | quote
|     |       |       |   |
+-----+-----+-----+-----+
| 123 | Christopher Morley | Life | 1 | 123 | Life is a foreign
language; all men mispronounce it.
| 123 | Christopher Morley | Life | 2 | 123 | There are three
ingredients in the good life: learning . . .
| 124 | Daniel Akst | Life | 3 | 124 | In matters of self-
control as we shall see again and . . .
| 124 | Daniel Akst | Life | 4 | 124 | We Have Met the Enemy:
Self-Control in an Age of Exc. . .
| 125 | Abraham Lincoln | Success | 5 | 125 | Always bear in mind
that your own resolution to . . .
| 125 | Abraham Lincoln | Success | 6 | 125 | Better to remain
silent and be thought a fool than . . .
| 126 | Albert Einstein | Success | 7 | 126 | If A is success in
life, then A equals x plus y plus . . .
+-----+-----+-----+-----+
+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

This example assumes an analytics node is running.

Procedure

To import SQL tables into CQL using a collection set for the quotations:

1. Download the [import_quotations.zip](#) file.
2. Create the mysql_quotations and person tables in MySQL. You can copy/paste commands from the downloaded file to produce these tables.
3. Create the famous_words keyspace and quotations table in cqlsh. You can copy/paste the commands from the downloaded file.

```
cqlsh> CREATE KEYSPACE famous_words WITH REPLICATION =
      {'class':'NetworkTopologyStrategy', 'Analytics':1};
cqlsh> USE famous_words;
cqlsh:famous_words> CREATE TABLE quotations (
      id text PRIMARY KEY,
      name text,
      title text,
      quotes set text
    );
```

4. Insert the data from the downloaded file into the person and mysql_quotations tables.
5. Create an import options file named import_persons.options having the following contents.

```
cql-import
--table
person
--cassandra-keyspace
famous_words
--cassandra-table
quotations
--cassandra-column-mapping
id:id,name:name,title:title
--connect
jdbc:mysql://127.0.0.1/famous_words
--username
root
```

```
--password
root
--cassandra-host
127.0.0.1
```

- Import the person table into the CQL table. On Linux, for example:

```
$ sudo bin/dse sqoop --options-file path/import_person.options
```

The MapReduce job runs and at then end, looks something like this:

```
...
14/06/16 20:26:43 INFO mapreduce.ImportJobBase: Transferred 0 bytes in
35.1743 seconds (0 bytes/sec)
14/06/16 20:26:43 INFO mapreduce.ImportJobBase: Retrieved 4 records.
```

- Check that the CQL quotations table now contains the data from the MySQL person table.

```
cqlsh:famous_words> SELECT * FROM quotations;
```

id	name	quotes	title
123	Christopher Morley	null	Life
125	Abraham Lincoln	null	Success
126	Albert Einstein	null	Success
124	Daniel Akst	null	Life

(4 rows)

- Create another import options file to import the mysql_quotations table. Use a free form query to import the quotations into the CQL table. The literal string \$CONDITIONS needs to appear in the WHERE clause of the query. Sqoop replaces the string with its refined constraints. For example:

```
cql-import
--query
select person.id, person.name, person.title, mysql_quotations.quote from
person INNER JOIN mysql_quotations ON person.id=mysql_quotations.speaker
WHERE $CONDITIONS
--target-dir
/sqoop
--split-by
person.id
--cassandra-keyspace
famous_words
--cassandra-table
quotations
--cassandra-column-mapping
id:id,name:name,title:title,quotes:quote
--connect
jdbc:mysql://127.0.0.1/famous_words
--username
root
--password
root
--cassandra-host
127.0.0.1
```

>

9. Import the resultset of quotations into the CQL table. On Linux, assuming you named the options file import_quotes.options:

```
$ sudo bin/dse sqoop --options-file path/import_quotes.options
```

10. Check that the CQL quotations table now contains a collection set of quotations as well as the name of the speaker and other information in the MySQL table.

```
cqlsh:famous_words> SELECT * FROM quotations;
```

Exporting CQL data to SQL

Using Sqoop, you can export data of different data types from CQL to MySQL. You can import SQL into CQL collections, and exporting CQL collections to multiple rows in SQL. This example creates a CQL table of columns of different data types, inserts values into the table, and exports the data to SQL.

Procedure

1. Create a keyspace using the default datacenter name Analytics, and use the keyspace.

```
cqlsh> CREATE KEYSPACE tosql WITH REPLICATION =
      {'class':'NetworkTopologyStrategy', 'Analytics':1};
cqlsh> USE tosql;
```

2. Create a table in CQL, and then, insert some data.

```
cqlsh:tosql> CREATE TABLE cql_table (
    id int PRIMARY KEY,
    a timestamp,
    b float,
    c boolean,
    d blob,
    e inet,
    f uuid);
cqlsh:tosql> INSERT INTO cql_table ( id, a, b, c, d, e, f ) VALUES ( 123,
    '1974-07-17 22:18:32', 3.14159265, true, 0x1afb, '127.0.0.1', 69d5c4fd-
    a7b7-4269-9cb5-c6f7d5fc076e );
cqlsh:tosql> INSERT INTO cql_table ( id, b ) VALUES ( 789, 11.001001000 ) ;
```

Observe the [range limitation](#) of MySQL timestamps.

3. Create a database and table in MySQL that corresponds to the CQL table. Use compatible data types, which are listed in [Export data types](#) on page 444.

```
mysql> CREATE DATABASE fromcql;
mysql> USE fromcql;
mysql> CREATE TABLE sql_table (
    id INTEGER PRIMARY KEY,
    a TIMESTAMP,
    b VARCHAR(25),
    c BOOLEAN,
    d BLOB,
    e VARCHAR(15),
    f VARCHAR(40)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

4. Export the CQL data to MySQL. This example shows the export command entered on the command line instead of using an options file.

```
$ dse sqoop cql-export --connect jdbc:mysql://127.0.0.1/fromcql --username root --password root --table sql_table --cassandra-host 127.0.0.1 --cassandra-keyspace tosql --cassandra-table cql_table
```

Alternatively, you can adapt the export.options file to your environment in the manner shown earlier for modifying the import.options file, and then use this command:

```
$ bin/dse sqoop --options-file path_to_export.options
```

The MapReduce job runs and the end of the output looks like this:

```
. .
14/05/29 08:08:33 INFO mapreduce.ExportJobBase: Transferred 0 bytes in
52.2312 seconds (0 bytes/sec)
14/05/29 08:08:33 INFO mapreduce.ExportJobBase: Exported 2 records.
```

5. Check that the data was exported into the MySQL table.

```
mysql> SELECT * FROM sql_table;

+----+-----+-----+-----+-----+-----+
| id | a      | b      | c      | d      | e      | f      |
+----+-----+-----+-----+-----+-----+
| 123 | 1974-07-17... | 3.1415927 | 1|?    |/127.0.0.1|69d5c4fd...
| 789 | 2014-05-29... | 11.001001 | NULL |NULL|NULL    |NULL
+----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Exporting selected CQL data to SQL

You can use export options to select certain columns for export and limit the page size of the export. You can also conditionally filter the data to select for export using the --cassandra-where-clause option *clause*. You enclose the CQL WHERE clause in double quotation marks.

This example creates a CQL table of columns of different data types, inserts values into the table, and exports the data to SQL.

Procedure

1. Use the keyspace created in the previous example.

```
cqlsh> USE tosql;
```

2. Create a table in CQL, and then, insert some data.

```
cqlsh:tosql> CREATE TABLE ruling_stewards (
    steward_name text,
    king text,
    reign_start int,
    event text,
    PRIMARY KEY (steward_name, king, reign_start)
);

cqlsh:tosql> INSERT INTO ruling_stewards (steward_name, king, reign_start,
event) VALUES ('Hador', 'none', 2278, 'Last long-lived Duedian');
```

```
cqlsh:tosql> INSERT INTO ruling_stewards (steward_name, king, reign_start, event) VALUES ('Denethor', 'Brego', 2435, 'Watchful Peace broken');

cqlsh:tosql> INSERT INTO ruling_stewards (steward_name, king, reign_start, event) VALUES ('Boromir', 'Brego', 2477, 'Attacks continue');

cqlsh:tosql> INSERT INTO ruling_stewards (steward_name, king, reign_start, event) VALUES ('Cirion', 'Brego', 2489, 'Defeat of Balchoth');
```

3. Test a WHERE clause to use to filter data for export. Select only the data from King Brego's reign from 2450 up to, but not including, 2500.

```
cqlsh:tosql> SELECT * FROM ruling_stewards WHERE king = 'Brego' AND reign_start >= 2450 AND reign_start < 2500 ALLOW FILTERING;

  steward_name | king    | reign_start | event
  -----+-----+-----+-----+
  Boromir   | Brego  |      2477 | Attacks continue
  Cirion    | Brego  |      2489 | Defeat of Balchoth

(2 rows)
```

4. Use the fromcql database from the previous example and create a MySQL table to accommodate the CQL ruling_stewards data.

```
mysql> USE fromcql;
mysql> create table sql_rulers (
    steward_name varchar(15) PRIMARY KEY,
    king varchar(15),
    reign_start INTEGER,
    event varchar (40)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

5. Export the only the CQL data from King Brego's reign from 2450 up to, but not including, 2500. This example shows the export command entered on the command line.

```
$ dse sqoop cql-export
--connect jdbc:mysql://127.0.0.1/fromcql
--username root --password root
--table sql_rulers
--cassandra-host 127.0.0.1
--cassandra-keyspace tosql
--cassandra-table ruling_stewards
--cassandra-select-columns steward_name,king,reign_start
--cassandra-where-clause "king='Brego' AND reign_start >=2450 AND reign_start < 2500"
```

The MapReduce job runs and the end of the output indicates success exporting two records:

```
...
14/09/17 20:25:37 INFO mapreduce.ExportJobBase: Exported 2 records.
```

6. Verify that the data was exported into the MySQL table.

```
mysql> select * from sql_rulers;
+-----+-----+-----+-----+
| steward_name | king    | reign_start | event
+-----+-----+-----+-----+
| Boromir   | Brego  |      2477 | NULL
| Cirion    | Brego  |      2489 | NULL
+-----+-----+-----+-----+
```

```
2 rows in set (0.02 sec)
```

Exporting data from CQL collections

The cql-export tool supports the export of data from list, set, and map collection types.

Exporting a set or list

You can export a CQL list and set to multiple SQL rows. You map each element in the list to an SQL row, and then use the `cql-export` command to export the data. In the SQL database, multiple rows store the collection.

The cql-export tool supports exporting list and set data as multiple SQL rows using the following mapping:

```
CQLCOL:SQLCOL
```

The following example shows how to map a list of multiple SQL rows.

Suppose you have created and populated a CQL table using the following commands:

```
cqlsh> CREATE TABLE cql_table (cqlid int PRIMARY KEY, mylist listtext);
cqlsh> INSERT INTO cql_table (cqlid, mylist) VALUES (1,
['value1','value2','value3']);
```

Using MySQL, you create the following table that corresponds to the CQL table:

```
mysql> CREATE TABLE sql_table(sqlid INTEGER GENERATED BY DEFAULT AS IDENTITY
PRIMARY KEY, id INTEGER, value VARCHAR(20));
```

The following map along with other options exports the data into MySQL:

```
--cassandra-column-mapping cqlid:id,mylist:value
```

Querying MySQL to select the table produces the following output:

sqlid	id	value
1	1	value1
2	1	value2
3	1	value3

Sqoop does not export sqlid from CQL. MySQL auto-generates the sqlid to give the table a unique id.

Exporting a map

You can export a CQL map collection to a single SQL row. You map each map key to SQL column names. You can only map one collection per Sqoop statement.

You use the following `cassandra-column-mapping` parameter to export CQL map entries to SQL columns where the key maps to a SQL column name. Where the map key is the same as the SQL column name, you can omit the key from the mapping:

```
CQLCOL: [SQLCOL1,SQLCOL2,SQLCOL3]
```

Like the importing mechanism, the mapping mechanism for exporting supports a mix of key name mapping.

```
CQLCOL: [KEY1:SQLCOL1,SQLCOL2,KEY3:SQLCOL3]
```

The following example shows how to map a CQL map to an SQL table.

Create and populate a CQL table using the following commands:

```
cqlsh> CREATE TABLE cql_table (cqlid int PRIMARY KEY, mymap map<text>,text);
cqlsh> INSERT INTO cql_table (cql, mymap) values (1,
{'key1':'value1','col2':'value2','key3':'value3'});
```

Using MySQL, create the following table that corresponds to the CQL table:

```
mysql> CREATE TABLE sql_table(sqlid INTEGER PRIMARY KEY, col1 VARCHAR(20),
col2 VARCHAR(20), col3 VARCHAR(20));
```

The following map along with other options exports the data to MySQL:

```
--cassandra-column-mapping cqlid:sqlid,mymap:[key1:col1,col2,key3:col3]
```

Querying MySQL to select the table produces the following output:

```
+-----+-----+-----+
| sqlid | col1   | col2   | col3   |
+-----+-----+-----+
| 1     | value1 | value2 | value3 |
+-----+-----+-----+
```

Automating a Sqoop operation

DataStax Enterprise supports a native Cassandra implementation of the Sqoop metastore. You use the Sqoop metastore to store jobs, which are operations that you can run directly from Sqoop, such as an import or export. The native implementation saves the jobs in the `sqoop_meta_store` table in the `dse_system` keyspace.

You can save configuration information for an import or export as a job and run the job from the metastore repeatedly. You typically run the job from the metastore to incrementally import data. Sqoop imports only the newest rows.

Configuring the metastore

You use the `sqoop-site.xml` file that is [installed with](#) DataStax Enterprise to configure the metastore. The default configuration sets up the native Cassandra metastore for use in a development environment. You need to make configuration changes to the following properties to use the metastore correctly in a working cluster:

Table: Cassandra metastore properties

Property	Description	Default
<code>sqoop.cassandra.host</code>	A comma-separated list of nodes that the metastore can use to connect to Cassandra	127.0.0.1
<code>sqoop.cassandra.port</code>	The native protocol port number that the metastore uses to connect to Cassandra	9042

Property	Description	Default
sqoop.job.storage.write.consistency	The consistency level for metastore writes	LOCAL_ONE
sqoop.job.storage.read.consistency	The consistency level for metastore reads	LOCAL_ONE
sqoop.metastore.client.record.password	Send passwords with the job	true

Job command syntax

To create and manage a job, use the job tool. The syntax of the job command is:

```
$ dse sqoop job option [jobId] -- sqoop_commands
```

The following list describes Sqoop job options:

- `dse sqoop job --create jobId -- sqoop_commands`
Creates a new job using the commands given after the '--'.
- `dse sqoop job --list`
Lists available jobs.
- `dse sqoop job --show jobId`
Displays information about a job.
- `dse sqoop job --delete jobId`
Deletes an existing job.
- `dse sqoop job --exec jobId`
Executes a saved job.

Creating a job

This example creates a job named myjob that imports the Sqoop demo data from the MySQL npa_nxx_demo database into a CQL table named npa_nxx in Cassandra:

```
$ dse sqoop job --create myjob -- cql-import --table npa_nxx --cassandra-keyspace npa_nxx --cassandra-table npa_nxx_data --cassandra-column-mapping npa:npa,nxx:nxx,latitude:lat,longitude:lon,state:state,city:city --connect jdbc:mysql://127.0.0.1/npa_nxx_demo
```

The following output indicates success. A job named myjob is saved in the DseMetaStore for execution later.

```
14/09/10 16:58:22 INFO policies.DCAwareRoundRobinPolicy: Using data-center name 'Analytics' for DCAwareRoundRobinPolicy (if this is incorrect, please provide the correct datacenter name with DCAwareRoundRobinPolicy constructor)
14/09/10 16:58:22 INFO core.Cluster: New Cassandra host /127.0.0.1:9042 added
```

Listing a job

This example shows how to list the jobs saved in the DseMetaStore:

```
$ dse sqoop job --list
```

```
Available jobs:
```

myjob

Viewing the job configuration

This example shows how to view the configuration of a job:

```
$ dse sqoop job --show myjob

Job: myjob
Tool: cql-import
Options:
-----
verbose = false
db.connect.string = jdbc:mysql://127.0.0.1/npa_nxx_demo
codegen.output.delimiters.escape = 0
codegen.output.delimiters.enclose.required = false
codegen.input.delimiters.field = 0
hbase.create.table = false
db.require.password = false
hdfs.append.dir = false
db.table = npa_nxx
import.fetch.size = null
codegen.input.delimiters.escape = 0
codegen.input.delimiters.enclose.required = false
codegen.output.delimiters.record = 10
import.max.inline.lob.size = 16777216
hcatalog.create.table = false
db.clear.staging.table = false
codegen.input.delimiters.record = 0
enable.compression = false
hive.overwrite.table = false
hive.import = false
codegen.input.delimiters.enclose = 0
hive.drop.delims = false
codegen.output.delimiters.enclose = 0
hdfs.delete-target.dir = false
codegen.output.dir = .
codegen.auto.compile.dir = true
mapreduce.num.mappers = 4
import.direct.split.size = 0
export.new.update = UpdateOnly
codegen.output.delimiters.field = 44
incremental.mode = None
hdfs.file.format = TextFile
codegen.compile.dir = /tmp/sqoop-root/
compile/498dc667d886a4c710b70c0062493de
direct.import = false
hive.fail.table.exists = false
db.batch = false
mapred.used.genericoptionsparser = true
sqoop.cassandra.keyspace = npa_nxx
sqoop.cassandra.column.family = npa_nxx_data
sqoop.cassandra.column.mapping =
  npa:npa,nxx:nxx,latitude:lat,longitude:lon,state:state,city:city
sqoop.cassandra.tool = cql-import
```

Running a job

This example assumes that you have [truncated](#) the npa_nxx.npa_nxx_data table using cqlsh. The following command runs the saved job.

```
$ dse sqoop job --exec myjob -- --username someuser -P
```

```
Enter password: ...
```

MapReduce runs the saved job.

Saved jobs and passwords

DataStax recommends using the --username and -P options on the command line as shown in the example of running a job. Because multiple users can access the DseMetaStore, it does not store passwords. You can set the sqoop.metastore.client.record.password option to true in the sqoop-site.xml to make the password prompt appear each time you create a job that requires a password. No prompting occurs when you run show or exec.

For security reasons, configuring these parameters in the sqoop-site.xml is not recommended:

- sqoop.metastore.client.autoconnect.username
- sqoop.metastore.client.autoconnect.password

Importing data incrementally

To import data in increments, you use the --incremental argument with the import command. Sqoop compares the values in a check column against a reference value for the most recent import. These arguments import all rows having an id greater than 100.

- --incremental
- --check-column id
- --last-value 100

If you run an incremental import from the command line, Sqoop prints the *last value* in a subsequent incremental import. If you run an incremental import from a saved job, Sqoop retains the last value in the saved job. To import only newer rows than those previously imported, use the --exec *row_id* option. Sqoop imports only rows that have an id greater than the specified row id.

Sqoop command

For use with Sqoop, use the DataStax Enterprise dse sqoop command to import SQL data to CQL, export SQL data to CQL, and import Thrift/CLI data to CQL.

All of the command options can be used in the `import.options` file.

```
$ dse sqoop action --connect jdbc_url --cassandra-keyspace ks --cassandra-table cf --cassandra-host host --table sql_table
```

where *action* is one of the following command options:

- Import SQL data into CQL:

```
cql-import
```

- Export SQL data to CQL.

```
cql-export
```

- Import Thrift/CLI data to CQL.

```
thrift-import
```

The following cql-import and cql-export command options are supported.

- [Cluster options](#)
- [Data options](#)

- Connection options
- Security options
- Conversion data types
- Export data types

Cluster options

These options define Cassandra cluster settings.

Table: cql-import and cql-export cluster options

Command	Description
--cassandra-consistency-level <i>consistencylevel</i>	The Cassandra consistency level, which is LOCAL_ONE by default.
--cassandra-host <i>host</i>	A comma separated list of Cassandra hosts.
--cassandra-partitioner <i>partitioner</i>	The Cassandra partitioner, which is Murmur3Partitioner by default.
--cassandra-port <i>port</i>	The Cassandra port.

Data options

These data options work with the cql-import and cql-export tools, except where noted.

Table: cql-import and cql-export data options

Command	Description
--query <i>sql_query</i>	Supports importing SQL joins.
--cassandra-column-mapping <i>map</i> ... where <i>map</i> = <i>cql1:sql1,cql2:sql2</i> ... CQLLISTSET:[SQLCOL,SQLCOL,SQLCOL] ... CQLMAP:[SQLCOL:VALCOL,SQLCOL:VALCOL]	Supports mapping ambiguous columns for import/export. Maps <i>cql</i> and <i>sql</i> columns (not collections) for import/export. Maps a list or set type for import/export. Handles importing/exporting of a map type.
--cassandra-page-size	cql-export only. Limits the page size of columns selected for export.
--cassandra-select-columns	cql-export only. Select the named columns to export.
--cassandra-where-clause	cql-export only. Filter the data selected for export based on the where condition.

Connection options

Sqoop, by default, identifies the JDBC driver to use from the JDBC URL. For drivers that are not supported directly by Sqoop, you must specify the driver class name using the --driver option. To use the DataStax Enterprise specific tools with a custom driver, you must tell Sqoop which connection manager to use.

The --connection-manager parameter is required when the --driver option is specified for cql-import and cql-export actions. For example:

```
dse sqoop cql-import --driver driver --connection-manager
com.datastax.bdp.sqoop.DseConnectionManager
```

```
dse sqoop cql-export --driver driver --connection-manager
com.datastax.bdp.sqoop.DseConnectionManager
```

Table: cql-import and cql-export connection options

Command	Description
--driver	For drivers that are not directly supported by Sqoop, specify the driver class name. For example: com.informix.jdbc.IfxDriver or com.teradata.jdbc.TeraDriver
--connection-manager	Specifies the connection manager for import or export to DataStax Enterprise. Required when the --driver option is used. The required value is: com.datastax.bdp.sqoop.DseConnectionManager

Security options

These security options are supported for the cql-import and cql-export actions.

Table: cql-import and cql-export security options

Command	Description
--cassandra-enable-kerberos	Enables kerberos authentication
--cassandra-kerberos-config-path <i>jaas.config_path</i>	Path to the users <i>jaas.config</i> file
--cassandra-enable-ssl	Enables SSL transport
--cassandra-ssl-protocol <i>protocol</i>	Configures the SSL protocol
--cassandra-truststore-algo <i>algo</i>	Configures the SSL trust store algorithm
--cassandra-truststore-ciphers <i>ciphers</i>	Configures the SSL trust store ciphers
--cassandra-truststore-location <i>location</i>	Path to the SSL trust store
--cassandra-truststore-password <i>tspassword</i>	Configures the SSL trust store password
--cassandra-truststore-type <i>type</i>	Configures the SSL trust store type
--cassandra-username <i>username</i>	Authenticates password, works only with the local Job Tracker
--cassandra-password <i>password</i>	Authenticates password
--cassandra-kerberos-service-principal <i>service_principal</i>	The Kerberos principal for which you have created a ticket using <i>kinit</i>

Conversion data types

These data types are supported for conversion from SQL to CQL.

Table: Allowable data type conversions for importing SQL to CQL

SQL data type	CQL data type
VARCHAR	text, ascii, varchar
BIT	boolean, text, ascii, varchar
BIT(1)	boolean, text, ascii, varchar
BIT(>1)	blob
TINYINT	int, bigint, varint, float, double, decimal, text, ascii, varchar
SMALLINT	int, bigint, varint, float, double, decimal, text, ascii, varchar
INTEGER	int, bigint, varint, float, double, decimal, text, ascii, varchar
BIGINT	bigint, varint, float, double, decimal, text, ascii, varchar
FLOAT	float, double, decimal, text, ascii, varchar
DOUBLE	double, decimal, text, ascii, varchar
DECIMAL	decimal, text, ascii, varchar
NUMERIC	decimal, text, ascii, varchar
BLOB	blob
CLOB	blob, text, ascii, varchar
BINARY(n)	blob, text, ascii, varchar
VARBINARY(n)	blob, text, ascii, varchar
DATE	timestamp, text, ascii, varchar
TIME	timestamp, text, ascii, varchar
TIMESTAMP	timestamp, text, ascii, varchar

Export data types

Use the following data type map for exporting from CQL to SQL.

Table: Export data types

CQL Type	SQL Type
int	TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, DECIMAL, NUMERIC, VARCHAR
bigint	BIGINT, FLOAT, DOUBLE, DECIMAL, NUMERIC, VARCHAR
varint	DECIMAL, NUMERIC, VARCHAR
float	FLOAT, DOUBLE, DECIMAL, NUMERIC, VARCHAR
double	DOUBLE, DECIMAL, NUMERIC, VARCHAR

CQL Type	SQL Type
decimal	DECIMAL, NUMERIC, VARCHAR
ascii	VARCHAR, CLOB, BLOB, VARBINARY
text	VARCHAR, CLOB, BLOB, VARBINARY
varchar	VARCHAR, CLOB, BLOB, VARBINARY
timestamp	DATE, TIME, TIMESTAMP, VARCHAR
boolean	BOOLEAN, BIT, BIT(1), VARCHAR
blob	BLOB, VARBINARY
inet	VARCHAR
uuid	VARCHAR
timeuuid	VARCHAR

About the generated Sqoop JAR file

After running the `dse sqoop import` command, a Java class named `npa_nxx.java` appears in the DSE installation `bin` directory. This file 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. See [Sqoop documentation](#).

Getting information about the `sqoop` command

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

```
$ cd install_location/bin
$ ./dse sqoop import --help
```

Migrating data using other methods

[Apache Sqoop](#) transfers data between an RDBMS and Hadoop. DataStax Enterprise modified Sqoop so you can move data directly into Cassandra as well as transfer data from an RDBMS to the Cassandra File System (CFS). DataStax offers several solutions in addition to Sqoop for migrating from other databases:

- [The COPY command](#), which mirrors what the PostgreSQL RDBMS uses for file/export/import
- [The DSE Search Data Import Handler](#), which is a configuration-driven method for importing data to be indexed for searching
- The [Cassandra bulk loader](#) that provides the ability to bulk load external data into a cluster
- [Bulk saving RDD to Cassandra](#) to save RDD data to Cassandra.

About the COPY command

You can use COPY in Cassandra's CQL shell to load flat file data into Cassandra as well as write data out to OS files. Typically, an RDBMS has unload utilities for writing table data to OS files.

ETL Tools

If you need more sophistication applied to a data movement situation than just extract-load, you can use any number of extract-transform-load (ETL) solutions that now support Cassandra. These tools provide excellent transformation routines for manipulating source data to suit your needs and then loading the data into a Cassandra target. The tools offer many other features such as visual, point-and-click interfaces, scheduling engines, and more.

Many ETL vendors who support Cassandra supply community editions of their products that are free and able to solve many different use cases. Enterprise editions are also available that have useful features for serious enterprise data users.

You can freely download and try ETL tools from Jaspersoft, Pentaho, and Talend that work with DataStax Enterprise and Cassandra.

Bulk saving data from Spark RDD to Cassandra

Bulk saving data from a Spark resilient distributed dataset (RDD) to the Cassandra database writes rows directly to SSTables that are created in a local temporary directory on each Spark Executor. This bulk savings from Spark RDD improves performance by bypassing the standard Cassandra write-path.

The standard `saveToCassandra` method sends rows through the Java driver to a Cassandra node, which then orders the rows and flushes into SSTables. Using the standard Cassandra write-path, much of the work is pushed onto the Cassandra cluster.

The bulk saving from Spark RDD to Cassandra uses the `bulkSaveToCassandra` method with the same semantics, but writes rows directly to SSTables that are created in a local temporary directory on each Spark Executor. The `bulkSaveToCassandra` method then streams the SSTables to Cassandra nodes in a DataStax Enterprise cluster. Performance is improved because the data bypasses a number of stages in the Cassandra write path, which results in a reduced load on the server side.

Example of bulk saving from Spark RDD

You must use the `SparkContext` to load data from an RDD. The following example shows how to bulk save data from a Spark RDD to a Cassandra database.

```
val rdd: RDD[SomeType] = ... // create some RDD to save
import com.datastax.bdp.spark.writer.BulkTableWriter._
rdd.bulkSaveToCassandra(keyspace, table)
```

Performance tuning

`BulkTableWriter` generates at least one local SSTable per Spark partition. Use these guidelines to tune performance:

- Ensure that the partitions of the RDD are at least several tens of megabytes large to minimize the cost of compacting the partitions on the server side.
- Increase the buffer size for each task to generate larger SSTables.

By default, a 64 MB buffer is reserved for each task. Bulk writing requires a significant amount of memory on the client. To control the buffer size, pass a custom `writeConf` object with the `bufferSizeInMB` property value with the `bulkSaveToCassandra` method call. When the RDD partitions are large enough, increase the size of this buffer to generate larger SSTables.

Note: If your keyspaces or table names use mixed case, specify the keyspace and table name in all lower case when calling `bulkSaveToCassandra`. For example, if your keyspace name is `myKeyspace` and your table name is `myTable`, call `bulkSaveToCassandra("mykeyspace", "mytable")`.

There is no point in setting this parameter to a value much larger than RDD partition size.

DataStax Enterprise tools

dse commands

The dse commands provide additional controls for starting and using DataStax Enterprise.

dse commands

Synopsis

```
$ dse [-f config_file] [-u username -p password] [-a jmx_username -b jmx_password] subcommand [command-arguments]
```

You can provide user credentials in several ways, see [Providing credentials for authentication](#).

This table describes the authentication command arguments that can be used with all subcommands.

Command arguments	Description
-f	Path to configuration file that stores credentials. If not specified, then use <code>~/.dserc</code> if it exists.
-u	User name to authenticate against the configured Cassandra authentication schema.
-p	Password to authenticate against the configured Cassandra authentication schema.
-a	User name to authenticate with secure JMX.
-b	Password to authenticate with secure JMX.

This table describes the `dse version` command that can be used without authentication:

Command argument	Description
-v	Send the DSE version number to standard output.

dse subcommands

This table describes the dse subcommands that use authentication.

Subcommand	Command arguments	Description
beeline		Start the Beeline shell.
cassandra		Start up a real-time Cassandra node in the background. See Starting DataStax Enterprise .
cassandra	-c	Enable the Cassandra File System (CFS) but not the integrated DSE Job Trackers and Task Trackers. Use to start nodes for running an external Hadoop system.

Subcommand	Command arguments	Description
cassandra	-f	Start up a real-time Cassandra node in the foreground. Can be used with -k, -t, or -s options.
cassandra	-k	Start up an analytics node in Spark mode in the background. See Starting Spark .
cassandra	-k -t	Start up an analytics node in Spark and DSE Hadoop mode. See Starting Spark .
cassandra	-s	Start up a DSE Search node in the background. See Starting DataStax Enterprise .
cassandra	-t	Start up an analytics node in DSE Hadoop mode in the background. See Starting DataStax Enterprise .
cassandra-stop	-p <i>pid</i>	Stop the DataStax Enterprise process number <i>pid</i> . See Stopping a node .
cassandra	-s -Ddse.solr.data.dir= <i>path</i>	Use <i>path</i> to store DSE Search data. See Moving solr.data
cassandra	-Dcassandra.replace_address	After replacing a node, replace the IP address in the table. See Replacing a dead node . All -D options in Cassandra start up commands are supported.
client-tool		See dse client-tool .
esri-import	ESRI import tool options	The DataStax Enterprise custom ESRI import tool supports the Enclosed JSON format. See Spatial analytics support on page 158.
hadoop	version	Sends the version of the Hadoop component to standard output.
hadoop	fs <i>options</i>	Invoke the Hadoop FileSystem shell. See the Hadoop tutorial .
hadoop	fs -help	Send Apache Hadoop fs command descriptions to standard output. See the Hadoop tutorial .
hive		Start a Hive client .
hive	--service <i>name</i>	Start a Hive server by connecting through the JDBC driver.
hive-schema		Create a hive schema representing the Cassandra table when Using Hive with BYOH .
hive-metastore-migrate	Hive-metastore-migrate tool options	Map custom external tables to the new release format after upgrading. See dse hive-metastore-migrate -to dest_release_num .
mahout	<i>mahout_command options</i>	Run Mahout commands .
mahout hadoop	<i>hadoop_command options</i>	Add Mahout classes to classpath and execute the hadoop command. See Mahout commands .
pig		Start Pig .
pyspark		PySpark .
spark		Accessing Cassandra from the Spark shell .

Subcommand	Command arguments	Description
spark-history-server	start	Start Spark history server.
spark-history-server	stop	Stop Spark history server.
spark-sql-thriftserver	start	Start Spark SQL Thrift server.
spark-sql-thriftserver	stop	Stop Spark SQL Thrift server.
spark-jobserver	start submit options	Launch applications on a cluster and use Spark Jobserver .
spark-jobserver	stop	Stop the Spark Jobserver .
spark-sql		Spark SQL command line
spark-submit	options	Launch applications on a cluster and use Spark cluster managers. See dse spark-submit .
spark-beeline		Use the Beeline client with Spark SQL Thrift Server.
sqoop	-help	Send Apache Sqoop command line help to standard output. See the Sqoop reference and the Sqoop demo .

Note: The directory in which you run the `dse` Spark commands must be writable by the current user.

Hadoop, hive, mahout, and pig commands must be issued from an analytics node. The `hadoop fs` options, which DSE Hadoop supports with one exception (`-moveToLocal`), are described in the *HDFS File System Shell Guide* on the [Apache Hadoop](#) web site. DSE Hadoop does not support the `-moveToLocal` option; use the `-copyToLocal` option instead.

The default location of the `dse` tool depends on the type of installation:

Package installations	/usr/bin/dse
Installer-Services installations	/usr/bin/dse
Installer-No Services and Tarball installations	<code>install_location/bin/dse</code>

dse client-tool

The `dse client-tool` subcommands use Cassandra authentication, not JMX authentication like the `dsetool` command.

Note: To show the command line help for `dse client-tool`

```
$ dse client-tool --help
```

For the `dse client-tool` subcommands, provide the optional user authentication, optional port and host arguments, and subcommand:

```
$ $ dse [-f config_file] [-u username -p password] [-a jmx_username -b jmx_password] client-tool [--port port] [--host address] subcommand
```

[Kerberos authentication](#) with `dse client-tool` is supported with the `kinit` tool.

client-tool subcommand	Command arguments	Description
	[--port <i>port</i>] [--host <i>address</i>]	The general arguments for <code>dse client-tool</code> apply to all subcommands: <ul style="list-style-type: none">• <code>--port <i>port</i></code> is the Cassandra RPC connection port (Thrift)• <code>--host <i>address</i></code> is the Cassandra host RPC broadcast address
hadoop	[<i>job-tracker-address</i>] [<i>version</i>]	Run the <code>dse client-tool hadoop</code> command for Hadoop-related operations. The arguments are: <ul style="list-style-type: none">• <i>job-tracker-address</i> Returns the address of the Hadoop Job Tracker in the datacenter that you are connecting to as determined by the host address.• <i>version</i> Returns the Hadoop version that is bundled with DataStax Enterprise.
help		Send client-tool command descriptions to standard output.
spark	[<i>master-address</i>] [<i>version</i>]	Runs the <code>dse client-tool spark</code> command for Spark-related operations. The arguments are: <ul style="list-style-type: none">• <i>master-address</i> Returns the current address of the Spark Master in the datacenter that you are connecting to as determined by the host address. Replaces the <code>dsetool sparkmaster</code> command.• <i>version</i> Returns the Spark version that is bundled with DataStax Enterprise.

dsetool utility

Use the `dsetool` utility for creating system keys, encrypting sensitive configuration, and performing Cassandra File System (CFS) and Hadoop-related tasks, such as checking the CFS, and listing node subranges of data in a keyspace.

Synopsis and dsetool command arguments

Synopsis

```
$ dsetool [-f config_file] [-l username -p password] [-a jmx_username -b jmx_password] [-h=hostname] [-s=Solr_port] [-j=jmx_port] command args
```

You can provide user credentials in several ways, see [Providing credentials for authentication](#).

This table describes the `dsetool` arguments that are supported for all `dsetool` commands:

Short form	Long form	Description
-l	--username	User name for authenticating with the configured Cassandra user.
-p	--password	Password to authenticate with the configured Cassandra user.
-f		Path to configuration file that stores credentials. The credentials in this configuration file override the <code>~/.dserv</code> credentials.
-a	--jmxusername <i>arg</i>	User name for authenticating with secure JMX.

Short form	Long form	Description
-b	--jmxpathword arg	Password for authenticating with secure JMX.
-c	--cassandra_port	Cassandra port number.
-h	--host arg	Node hostname or IP address.
-j	--jmlexport arg	Remote JMX agent port number.
-s	--port	Solr port number.
-u	--use_hadoop_config	Get Cassandra host from Hadoop configuration files.

dsetool commands

autojt

Elects Job Trackers for a specified datacenter, or for all datacenters when none is specified. Automatically manage Job Tracker selection and remove manual selections. If the current manually selected tracker is up, the manually selected Job Tracker continues to be used.

checkcfs cfs:///|filepath|

Checks a single Cassandra File System (CFS) file or the whole CFS using these options:

- *cfs:///* - Scan the entire Cassandra File System (CFS) for corrupted files
- *filepath* - Get details about a particular file that has been corrupted.

See [Checking the CFS using dsetool](#).

cleanup_leases

Deletes stale leaders and assigned candidates.

core_indexing_status keyspace.table [-all]

Retrieves the dynamic indexing status (INDEXING, FINISHED, or FAILED) of the specified core in a DSE Search node. Specify `--all` to retrieve the dynamic indexing status of all Solr cores.

```
$ dsetool -h IP_address core_indexing_status core_name
```

where `IP_address` is the IP address of the host that is output by the `dsetool ring` command.

If you do not specify the IP address, the default is the local DataStax Enterprise node. For example:

```
$ dsetool core_indexing_status wiki.solr
wiki.solr: INDEXING
```

create_core keyspace.table [option ...]

Supports Cassandra password authentication with `[-l username -p password]`.

Creates the Solr core and optionally generates resources automatically. The Solr core is created with the specified keyspace and table name. This command preserves the case of keyspace and table names. You must use the correct case for the keyspace and table names. The core is created with the following options:

Option	Settings	Default	Description
schema=	<i>filepath</i>	n/a	Path of the schema file. Cannot be specified when <code>generateResources=true</code> .
solrconfig=	<i>filepath</i>	n/a	Path of the <code>solrconfig.xml</code> file. Cannot be specified when <code>generateResources=true</code> .
distributed=	true or false	true	<ul style="list-style-type: none"> • true distributes and applies the operation to all nodes in the local DC.

Option	Settings	Default	Description
			<ul style="list-style-type: none"> • false applies the operation only to the node it was sent to.
deleteAll=	true or false	false	<ul style="list-style-type: none"> • true deletes the already existing index before re-indexing; search results will return either no or partial data while the index is rebuilding. • false does not delete the existing index, causing the re-index to happen in-place; search results will return partially incorrect results while the index is updating.
recovery	true or false	false	<ul style="list-style-type: none"> • true if the Solr core is unable to load due to corrupted index, recovers it by deleting and recreating the index. The deleteAll flag is set based on the recovery flag unless deleteAll is specifically set. • false no recovery.
reindex=	true or false	false	<p>Observed only on auto-core creation (generateResources=true); otherwise, always re-indexes on core creation.</p> <ul style="list-style-type: none"> • true re-indexes the data. • false does not re-index the data.
generateResources=	true or false	false	Cannot be used with schema= and solrconfig=.
coreOptions	n/a	n/a	Path to the options file when generateResources=true. See Customizing automatic resource generation on page 231.

createsystemkey *algorithm[/mode/padding]* *secret_key_strength* [*file*] [-k=*kmip_groupname*] [-t *kmip_template*] [-n *namespace*]]

Creates a global encryption key, called a system key, for SSTable encryption using the following options:

- *algorithm[/mode/padding]* *secret_key_strength* - When Java Cryptography Extension ([JCE](#)) is installed, the cipher_algorithm options and acceptable secret_key_strength values for the algorithms are:

cipher_algorithm	secret_key_strength
AES/CBC/PKCS5Padding	128, 192, or 256
AES/ECB/PKCS5Padding	128, 192, or 256
DES/CBC/PKCS5Padding	56
DESede/CBC/PKCS5Padding	112 or 168
Blowfish/CBC/PKCS5Padding	32-448
RC2/CBC/PKCS5Padding	40-128

Key strength is not required for HMAC algorithms.

- *file* - Specify the name of the system key file to create. If no name is specified, the default system key file name is `system_key`. The default system key file name is not configurable.

- `-k=kmip_groupname` - Use the KMIP connection information to create a remote system key for the KMIP key server group that is defined in the `kmip_hosts` section in the `dse.yaml` file. The following options are available only for the specified KMIP key server group:
 - `-t kmip_template` - Uses the specified KMIP server key template.
 - `-n namespace` - Specifies the namespace to create the system key with.

See [Encryption/compression options and algorithm sub-options](#) and [Encrypting sensitive property values](#).

encryptconfigvalue

Encrypts sensitive configuration information. This command takes no arguments and prompts for the value to encrypt.

get_core_config keyspace.table [current=true|false]

Outputs the latest uploaded `solrconfig.xml` resource file for the specified core. If current is set to true, returns the current live solrconfig.

get_core_schema keyspace.table [current=true|false]

Supports Cassandra password authentication with `[-l username -p password]`.

Outputs the latest uploaded Solr schema. If current is set to true, returns the current live schema.

infer_solr_schema keyspace.table [coreOptions path_to_options_file]

Supports Cassandra password authentication with `[-l username -p password]`.

Automatically infers and proposes a schema that is based on the specified keyspace and table. Solr cores are not modified. The Solr schema is inferred with the specified coreOptions YAML file.

inmemorystatus [keyspace.table]

Provides the memory size, capacity, and percentage for this node and the amount of memory each table is using. To get information about a single table, specify the keyspace and table. The unit of measurement is MB. Bytes are truncated.

jobtracker

Returns the Job Tracker hostname and port. Returns the Job Tracker that is local to the data center from which you are running the command. See [managing the Job Tracker using dsetool commands](#) for examples of using dsetool commands for managing the Job Tracker.

listjt

Lists all Job Tracker nodes grouped by the datacenter that is local to them.

list_subranges keyspace.table keys_per_range start_token, end_token

Divides a token range for a given keyspace/table into a number of smaller subranges of approximately `keys_per_range`. To be useful, the specified range should be contained by the target node's primary range. See [Listing sub-ranges using dsetool](#).

managekmip subcommand kmip_groupname [command_arguments]

Verifies communication with the specified KMIP key server and lists the [KMIP encryption](#) keys on that key server. The follow subcommands are supported:

list kmip_groupname [namespace=key_namespace]

Lists the encryption keys on the specified KMIP host. You can optionally specify the namespace.

expirekey kmip_groupname key_id [datetime]

Specifies an expiration date and time for the specified encryption key. After the specified `datetime`, no new data will be encrypted with the key. Data can be decrypted with the key after this expire date/time. If an expire date/time is not specified, the key is expired immediately.

Format of `datetime` is YYYY-MM-DD HH:MM:SS:T. For example, use 2016-04-13 20:05:00:0 to expire the encryption key at 8:05 p.m. on 13 April 2016.

revoke kmip_groupname key_id

Revokes the specified encryption key. After a key is revoked, the key cannot be used to decrypt data.

destroy kmip_groupname key_id

Destroys the specified encryption key. After a key is destroyed, the key cannot be used to decrypt data.

movejt

Moves the Job Tracker and notifies the Task Tracker nodes. This option has been deprecated. Use `setjt` and `setrjt`.

node_health

Retrieves a dynamic score between 0 and 1 that describes the health of a DataStax Enterprise node. If you do not specify the IP address, the default is the local DataStax Enterprise node. A higher score indicates better node health. Nodes that have a large number of dropped mutations and nodes that are just started have a lower health score.

```
$ dsetool -h IP_address node_health
```

where *IP_address* is the IP address that is output by the `dsetool ring` command.

For example:

```
$ dsetool -h 200.192.10.11 node_health
Node Health: 0.7
```

Specify `-all` to retrieve the **node health scores** for all nodes:

```
$ dsetool node_health -all
```

partitioner

Returns the fully qualified classname of the IPartitioner that is in use by the cluster.

perf *subcommand*

Modifies performance object settings as described in the **subcommand section**.

read_resource *keyspace.table* name=*resfilename*

Supports Cassandra password authentication with `[-l username -p password]`.

Reads the specified DSE Search resource file.

rebuild_indexes *keyspace.table* [*idx1, idx2, ...*]

Rebuilds specified secondary indexes for specified keyspace/table. To rebuild all indexes, do not specify indexes and use only `rebuild_indexes keyspace.table`.

reload_core *keyspace.table* [*option ...*]

Supports Cassandra password authentication with `[-l username -p password]`.

Reloads a Solr core with the specified keyspace and table name. This command preserves the case of keyspace and table names. You must use the correct case for the keyspace and table names. The core is reloaded with following options:

Option	Settings	Default	Description
<code>schema=</code>	<i>filepath</i>	n/a	Path of the schema file
<code>solrconfig=</code>	<i>filepath</i>	n/a	Path of the solrconfig.xml file
<code>distributed=</code>	true or false	true	<ul style="list-style-type: none"> • true distributes and applies the reload operation to all nodes in the local DC. • false applies the reload operation only to the node it was sent to.
<code>reindex=</code>	true or false	false	<ul style="list-style-type: none"> • true re-indexes the data. • false does not re-index the data.

Option	Settings	Default	Description
deleteAll=	true or false	false	<ul style="list-style-type: none"> • true deletes the already existing index before re-indexing; search results will return either no or partial data while the index is rebuilding. • false does not delete the existing index, causing the re-index to happen in-place; search results will return partially incorrect results while the index is updating.

repaircfs

Repairs the CFS from orphan blocks.

ring

Lists the nodes in the ring, including their node type.

setjt *IP_address_of_JobTracker_node*

Moves the JobTracker to the specified node and notifies the TaskTracker nodes of the change.

setrjt *IP_address_of_reserve_JobTracker_node*

Moves the reserve JobTracker node to the specified IP address and notifies the TaskTracker nodes of the change.

sparkmaster [*subcommand*]

Deprecated, use the [dse client-tool spark-master](#) command instead. Unless a subcommand is provided, this command returns the address of Spark Master running in a datacenter. Otherwise, this command executes a subcommand related to Spark Master.

- cleanup - Drops and recreates the Spark Master recovery table.
- cleanup *data_center_name* - Removes recovery data for the specified datacenter.

sparkworker restart

Manually restarts the Spark Worker on the selected node, without restarting the node.

status

Lists the nodes in their ring, including the node type. Same as the [ring](#) command.

write_resource

Supports Cassandra password authentication with `[-l username -p password]`.

Writes the specified DSE Search resource file.

```
$ dsetool write_resource keyspace.table name=ResourceFile.xml
file=schemaFile.xml
```

You can specify a path for the resource file.

```
$ dsetool write_resource keyspace.table name=ResourceFile.xml file=myPath1/
myPath2/schemaFile.xml
```

Resource files are stored in the Cassandra database. To view the resources, use the Solr Admin interface.

unload_core *keyspace.table* [*option ...*]

Supports Cassandra password authentication with `[-l username -p password]`.

Removes a [Solr core](#) with the specified keyspace and table name. This command preserves the case of keyspace and table names. You must use the correct case for the keyspace and table names. The core is unloaded with the following options:

Option	Settings	Description
deleteDataDir=	true or false	If true, deletes the underlying Cassandra data.

Option	Settings	Description
deleteResources=	true or false	If true, deletes the resources associated with the core, the Solrconfig.xml and schema.
distributed=	true or false	If true and deleteDataDir=true, deletes the index data directory on all nodes.

Checking the CFS using dsetool

Use the `dsetool checkcfs` command to scan the Cassandra File System (CFS) for corrupted files. For example:

```
$ dsetool checkcfs cfs:///
```

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

```
$ dsetool checkcfs /tmp/myhadoop/mapred/system/jobtracker.info
```

Listing sub-ranges using dsetool

The `dsetool` command syntax for listing subranges of data in a keyspace is:

```
$ dsetool [-h hostname] list_subranges keyspace table rows_per_subrange start_token end_token
```

- *rows_per_subrange* - The approximate number of rows per subrange.
- *start_partition_range* - The start range of the node.
- *end_partition_range* - The end range of the node.

Note: Run `nodetool repair` on a single node using the output of `list_subranges`. The output must be partition ranges that are used on that node.

Example

```
$ dsetool list_subranges Keyspace1 Standard1 10000
113427455640312821154458202477256070485 0
```

Output

The output lists the subranges to use as input to the `nodetool repair` command. For example:

Start Token Estimated Size	End Token
113427455640312821154458202477256070485	
132425442795624521227151664615147681247	11264
132425442795624521227151664615147681247	
151409576048389227347257997936583470460	11136
151409576048389227347257997936583470460	0
11264	

Nodetool repair command options

You must use the `nodetool` utility to work with sub-ranges. The `start partition range (-st)` and `end partition range (-et)` options specify the portion of the node that needs repair. You get values for the start and end tokens from the output of `dsetool list_subranges` command. The `nodetool repair` syntax for using these options is:

```
$ nodetool repair keyspace table -st start_token -et end_token
```

Example

```
$ nodetool repair Keyspace1 Standard1 -st
113427455640312821154458202477256070485 -et
132425442795624521227151664615147681247
$ nodetool repair Keyspace1 Standard1 -st
132425442795624521227151664615147681247 -et
151409576048389227347257997936583470460
$ nodetool repair Keyspace1 Standard1 -st
151409576048389227347257997936583470460 -et 0
```

These commands begins an anti-entropy node repair from the start partition range to the end partition range.

Performance object subcommands

The self-explanatory `dsetool perf` command subcommands are:

Note: Enabling or disabling with the performance object subcommands does not persist between reboots and is useful only for short-term diagnostics. To make these settings permanent, see [CQL Performance Service options](#).

Subcommand name	Possible values	Description
clustersummary	- enable disable	Toggle cluster summary statistics. See Collecting database summary diagnostics on page 356.
cqlslowlog	- <i>threshold</i> - enable disable	Set the CQL slow log threshold. Toggle the CQL slow log. See Collecting slow queries on page 353.
cqlsysteminfo	- enable disable	Toggle CQL system information statistics. See Collecting system level diagnostics on page 354.
dbsummary	- enable disable	Toggle database summary statistics. See Collecting database summary diagnostics on page 356.
histograms	- enable disable	Toggle table histograms. See Collecting table histogram diagnostics on page 357.
resourcelatencytracking	- enable disable	Toggle resource latency tracking. See Collecting system level diagnostics on page 354.
solrcachestats	- enable disable	Toggle Solr cache statistics.
solrindexingerrorlog	- enable disable	Toggle Solr indexing error log.
solrindexstats	- enable disable	Toggle Solr index statistics.
solrlatencysnapshots	- enable disable	Toggle Solr latency snapshots.
solrrequesthandlerstats	- enable disable	Toggle Solr request handler statistics.
solrslowlog	- enable disable	Toggle Solr slow sub-query log. See Collecting slow Solr queries on page 359.
solrupdatehandlerstats	- enable disable	Toggle Solr update handler statistics.
userlatencytracking	- enable disable	Toggle user latency tracking. See Collecting user activity diagnostics on page 358.

The cfs-stress tool

Usage:

```
$ cfs-stress [options] cfs_directory
```

where *cfs_directory* sets where to store test files.

Note: The tool uses the [listen_address](#) property in the `cassandra.yaml` file. If not using localhost, add the correct IP as an additional argument:

```
$ stress-cfs [options] cfs_directory listen_address
```

Table: Options

Short form	Long form	Description
-d	--data-generator <i>class</i>	Data generator to create files. Available generators: RandomDataGenerator, TextDataGenerator, ZeroDataGenerator. The RandomDataGenerator is a fast pseudo-random data generator that delivers about 1.5 GB of data per second on a single core of Core i7 @ 2.4 GHz.
-h	-help	Display help.
-n	--count <i>number</i>	Total number of files read/written. Default: 100.
-o	--operation R W WR WRD	Operation: <i>R</i> read, <i>W</i> write, <i>WR</i> write and read, <i>WRD</i> write and read and delete. Default: <i>W</i>
--r	--streams <i>number</i>	Maximum number of streams kept open per thread. Default 2.
-s	--size <i>number</i>	Size of each file in KB. Default 1024.
	--shared-cfs	Causes all threads to share the same CFS object.
-t	--threads <i>number</i>	Number of threads. Default 8.

The `cfs-stress` tool is located in the `tools` directory.

The default location of the `tools` directory depends on the type of installation:

Installer-Services and Package installations	/usr/share/dse/tools
Installer-No Services and Tarball installations	<i>install_location</i> /dse/tools

Example

From the `tools` directory:

```
$ ./cfs-stress cfs_directory
```

The output looks like:

```
Writing 104 MB to cfs://localhost:9160/user/pt/cfs_directory in 100 files.
progress      bytes      curr rate      avg rate      max latency
 0.0%        0.0 MB      0.0 MB/s      0.0 MB/s      -----
 0.0%        0.0 MB      0.0 MB/s      0.0 MB/s      -----
```

0.0%	0.0 MB	0.0 MB/s	0.0 MB/s	-----
0.0%	0.0 MB	0.0 MB/s	0.0 MB/s	-----
0.0%	0.0 MB	0.0 MB/s	0.0 MB/s	-----
32.0%	33.6 MB	2.6 MB/s	5.5 MB/s	129.554 ms
80.0%	83.9 MB	31.4 MB/s	11.7 MB/s	10.303 ms
82.0%	86.0 MB	48.5 MB/s	10.5 MB/s	-----
100.0%	104.9 MB	14.5 MB/s	12.4 MB/s	0.012 ms

Data	Description
progress	Total progress of the stress operation.
bytes	Total bytes written/read.
curr rate	Current rate of bytes being written/read per second.
avg rate	Average rate of bytes being written/read per second.
max latency	Maximum latency in milliseconds during the current reporting window.

Pre-flight check and yaml_diff tools

The pre-flight check tool, located in `/usr/share/dse/tools` of packaged installations, is a collection of tests that can be run on a node to detect and fix a configuration. The tool can detect and fix many invalid or suboptimal configuration settings. The tool is not available in tarball installations.

The `yaml_diff` tool in the `tools` directory filters differences between two `cassandra.yaml` files, which is useful during upgrades.

Using the Cassandra bulk loader in a secure environment

The [Cassandra bulk loader](#) is the `sstableloader` tool. The command-line options for configuring secure `sstableloader` operations using Kerberos have changed slightly. If you run `sstableloader` from a DataStax Enterprise node that has been configured for Kerberos or client-to-node/node-to-node encryption using SSL, no additional configuration is needed for securing `sstableloader` operations. The `sstableloader` tool will pick up all required options from the configured node automatically, so no further configuration is needed. On an unconfigured developer machine, however, configure Kerberos or SSL as follows:

Kerberos

If you have not configured Kerberos on a DataStax Enterprise node, but you want to run `sstableloader` in a secure Kerberos environment, set the options on the command line as follows:

- To use credentials from default ticket cache, no extra options are necessary. `sstableloader` will do the right thing.
- To set the keytab location through system properties, use this example as a guide to setting the options:

```
JVM_OPTS="-Dkerberos.use.keytab=true \
-Dkerberos.keytab=/home/dse/cassandra.keytab \
-Dkerberos.client.principal=cassandra@LOCAL.DEV" \
resources/cassandra/bin/sstableloader -d 192.168.56.102 /var/lib/
cassandra/data/Keyspace1/Standard1
```

Troubleshooting

- To set Kerberos options using the JAAS config, use this example as a guide to setting the options:

```
JVM_OPTS="-Dkerberos.use.config.file=true \
-Djava.security.auth.login.config=/home/dse/keytab-basic-jaas.conf" \
resources/cassandra/bin/sstableloader -d 192.168.56.102 /var/lib/
cassandra/data/Keyspace1/Standard1
```

- In the JAAS config, /home/dse/keytab-basic-jaas.conf, set these options:

```
Client {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    keyTab="/home/dse/cassandra.keytab"
    principal="cassandra@LOCAL.DEV";
};
```

Client- and node-to-node encryption using SSL

If you have not configured SSL on a DataStax Enterprise node, but you want to run sstableloader in a secure SSL environment, you can use the sstableloader script from Apache Cassandra to load SSTables into a cluster with client-to-node/node-to-node SSL encryption enabled. Use the following basic options:

```
resources/cassandra/bin/sstableloader -d 192.168.56.102 /var/lib/cassandra/
data/Keyspace1/Standard1 \
-tf org.apache.cassandra.thrift.SSLTransportFactory \
-ts /path/to/truststore \
-tspw truststore_password
```

If you want to configure require_client_auth=true on the target, set these additional options:

```
resources/cassandra/bin/sstableloader -d 192.168.56.102 /var/lib/cassandra/
data/Keyspace1/Standard1 \
-tf org.apache.cassandra.thrift.SSLTransportFactory \
-ts /path/to/truststore \
-tspw truststore_password \
-ks /path/to/keystore \
-kspw keystore_password
```

Troubleshooting

The following common problems, solutions, or workarounds have been reported about using DataStax Enterprise. Be sure to also check the [Cassandra troubleshooting documentation](#).

Unable to write to the standard tmp directory for JNA

Set JNA temporary path to be executable. In `cassandra-env.sh`, add the line:

```
JVM_OPTS="$JVM_OPTS -Djna.tmpdir=/var/tmp"
```

Mahout Jobs that Use Lucene Not Supported

DataStax does not currently support Mahout jobs, such as [built-in support for creating vectors from Lucene indexes](#), that use Lucene features. Attempting to run Mahout jobs that use Lucene features results in this type of error message:

```
Error: class org.apache.mahout.vectorizer.
```

```
DefaultAnalyzer overrides final method
tokenStream.
```

MX4J warning message during installation

When Cassandra loads, you may notice a message that MX4J will not load and that mx4j-tools.jar is not in the classpath.

You can ignore this message. MX4j provides an HTML and HTTP interface to JMX and is not necessary to run Cassandra. DataStax recommends using OpsCenter. It has more monitoring capabilities than MX4J.

DSE Search cannot find custom files

Solr supports relative paths that are set by the <lib> property in the solrconfig.xml, but DSE Search does not. [Configuring the Solr library](#) path describes a workaround for this issue.

Subprocesses not killed when DataStax Enterprise is shut down improperly

Attention: To prevent this problem, avoid using `kill -9` and shut down DataStax Enterprise as described in [Stopping a node](#).

If DataStax Enterprise is shut down with `kill -9`, you must reboot the node or manually kill any remaining sub-processes:

- **Installer-Services and Package installations:**

For example, if DataStax Enterprise was started using `sudo services dse start` or `sudo /etc/init.d/dse start` and the main process was killed using `$ kill -9 `cat /var/run/dse/dse.pid``:

1. To view the subprocesses left behind (all DSE processes run under user "cassandra"):

```
$ pgrep -c -ucassandra >/dev/null && ps -o pid,ppid,user,args `pgrep -ucassandra`
```

2. To shut down the subprocesses:

```
$ sudo pkill -ucassandra
```

- **Installer-No Services and Tarball installations:**

For example, if DataStax Enterprise 4.8 was started using `sudo dse cassandra -k -t` and the main process was killed using `$ sudo kill -9 `cat /var/run/dse/dse.pid`` or `$ sudo pkill -9 -f jmxremote.port=7199`:

1. To view the subprocesses left behind:

```
$ pgrep -c -f dse-4.8 >/dev/null && ps -o pid,ppid,user,args `pgrep -f dse-4.8`
```

All DSE processes run under user `cassandra`.

2. To shut down the subprocesses:

```
$ sudo pkill -f dse-4.8
```

Note: The `kill` command (SIGTERM) shuts down the subprocesses.

DataStax Enterprise 4.8 release notes

DataStax Enterprise release notes cover components, changes and enhancements, issues, and resolved issues for DataStax Enterprise 4.8 releases.

Included in this document are release notes for:

- [DataStax Enterprise 4.8.9](#)
- [DataStax Enterprise 4.8.8](#)
- [DataStax Enterprise 4.8.7](#)
- [DataStax Enterprise 4.8.6](#)
- [DataStax Enterprise 4.8.5](#)
- [DataStax Enterprise 4.8.4](#)
- [DataStax Enterprise 4.8.3](#)
- **Warning:** DataStax does not recommend 4.8.1 or 4.8.2 versions for production, see warning. Use 4.8.3 instead.
 - [DataStax Enterprise 4.8.2](#)
 - [DataStax Enterprise 4.8.1](#)
- [DataStax Enterprise 4.8](#)

4.8.9 Release notes for DataStax Enterprise

1 July 2016

4.8.9 Components

The follow components are changed from the previous version of DataStax Enterprise:

- Apache Cassandra 2.1.15.1403
- Apache Solr 4.10.3.1.1170

4.8.9 Changes and enhancements

- Fix Solr Data Import Handler Deprecation Message. (DSP-10121)
- DataStax Enterprise fails to start after upgrading Hadoop with Kerberos enabled. (DSP-8912)
- Additional logging for analyzing core loading performance. (DSP-6560)
- Support for CentOs and RHEL 7.2 (DSP-10276)

4.8.9 Resolved issues

- sstablemetadata estimated droppable tombstones does not take partition level tombstones into account (DSP-10335)
- Duplicate documents caused by Tuple/UDT field updates. (DSP-10177)
- SparkWorker error logs during shutdown. (DSP-10128)
- Add a dse.yaml configuration option to avoid waiting for [post-bootstrap reindexing](#) so that the node is not marked down. (DSP-10067)
- CVE 2015-5262 - Update http-client used in Spark streaming demo to 4.5.2. (DSP-10041)
- Spark options in dse.yaml are not ignored even though SPARK_ENABLED=0. (DSP-10023)
- Fixed CVE-2016-2175 - Apache PDFBox before 1.8.12 and 2.x before 2.0.1 does not properly initialize the XML parsers. (DSP-9975)
- CVE-2014-4715 and CVE 2014-4611 - Driver vulnerability due to LZ4 and xxHash 1.2.0. (DSP-9910)
- CVE 2015-3250 - Transitive dependency on Apache directory LDAP API Model 1.0.0-M24. (DSP-9909)

- solr_stress does not work if the schema does not have default field. (DSP-9897)
- Upgrade unsafe Apache Tika Version Solr pulls into DataStax Enterprise. (DSP-9811)
- CVE-2014-0114 - Apache Commons BeanUtils. (DSP-9628)
- Missing index updates after core failure. (DSP-8502)
- Weather Sensor Demo throws SASL errors on CentOS. (DSP-8309)

4.8.9 Cassandra changes

DataStax Enterprise 4.8.9 certifies Cassandra 2.1.15 with additional production-certified [Cassandra changes](#).

4.8.8 Release notes for DataStax Enterprise

31 May 2016

4.8.8 Components

The follow components are changed from the previous version of DataStax Enterprise:

- Apache Cassandra 2.1.14.1346
- Apache Solr 4.10.3.1.1060
- Apache Spark Connector 1.4.4

4.8.8 Changes and enhancements

- DSE Analytics: Add `dsetool cleanup_leases` to cleanup old LeaderManager entries. (DSP-9647)
- Improve multi-threaded processing of long running queries for search. (DSP-9685)
- Improve security: CVE-2014-0114 (commons-beanutils-core). (DSP-9628)

4.8.8 Resolved issues

- Fixed search core closing exceptions on node shutdown. (DSP-9738)
- Rows with non-indexed UDT fields are not searchable. (DSP-9711)
- Properly quote secondary index names for Solr cores. (DSP-9705)
- Bound multithreaded deletes only with live indexing. (DSP-9620)
- dsetool create_core should accept generateResources=false. (DSP-9555)
- Error out on multiple 'q' for solr_queries. (DSP-9517)
- Properly quote CQL statements when unloading search cores. (DSP-9345)
- Change the log level of the slow query log. (DSP-9308)
- Co-located Spark Master and cluster mode driver cannot fail over reliably. (DSP-9158)
- Sqoop SSL encryption is broken for CQL imports. (DSP-8061)
- dsetool create_core generateResources=false should not allow coreOptions. (DSP-7918)
- Empty options file for resource generation causes NPE and better error handling. (DSP-7606)

4.8.8 Cassandra changes

DataStax Enterprise 4.8.8 certifies Cassandra 2.1.14 with additional production-certified [Cassandra changes](#).

4.8.7 Release notes for DataStax Enterprise

6 May 2016

4.8.7 Components

The follow components are changed from the previous version of DataStax Enterprise:

- Apache Cassandra 2.1.14.1272
- Apache Solr 4.10.3.1.1021
- Hive Connector 0.2.14
- Spark Cassandra Connector 1.4.3

4.8.7 Changes and enhancements

- Add version of Spark job server and Spark connector to system.log/output.log. (DSP-9355)
- Improved Solr stress capabilities and metrics reporting. Merge geo names loader into Solr stress with improvements to geonames. (DSP-8602)
- Clarify resource generation when static resources are specified in `dsetool`. If generateResources=true and resources already exist, then fail gracefully. (DSP-7799)

4.8.7 Resolved issues

- Overly granular [node health scores](#) isolate request submission to too few replicas. (DSP-9512)
- TTL collector shouldn't read whole rows from Cassandra. (DSP-9446)
- Improve the dynamic field warning for incorrectly stored map data in Cql3CassandraRowReader. (DSP-9394)
- Cannot geodist() sort with RPT fields on multi-node clusters. (DSP-9392)
- [Spark history server](#) creates extra system.log. (DSP-9364)
- Hive create external table location keyword location does not work. (DSP-9359)
- CFS keyspace warning about phantom datacenter DC1 or UNKNOWN-DC. (DSP-9246)
- Copy field returns no result if the source field is non-indexed. (DSP-9157)
- NPE When spark_master_recovery table does not exist. (DSP-9140)
- Spark History Server can't start when auth is enabled. (DSP-9134)
- Spark workers will hang if can't reach consistency level for Cassandra read/write. (DSP-9061)
- NullPointerException in Spark SQL Thriftserver. (DSP-8647)
- Spark Master/Worker can shut down the entire node. (DSP-8324)
- Multiple CQL Solr query fq clauses should work. (DSP-7975)
- The install-jts.sh script doesn't work. You must manually [install the JTS Topology Suite](#). (DSP-7403)
- Node reaches NORMAL status before internode transport is ready. (DSP-7331)
- Error out on unrecognized facet parameters. (DSP-7212)

4.8.7 Cassandra changes

DataStax Enterprise 4.8.7 certifies Cassandra 2.1.14 with additional production-certified [Cassandra changes](#).

4.8.6 Release notes for DataStax Enterprise

1 April 2016

4.8.6 Components

The follow components are changed from the previous version of DataStax Enterprise:

- Apache Cassandra 2.1.13.1218 with additional production-certified [Cassandra changes](#)
- Hive Connector 0.2.13
- Apache Solr 4.10.3.1.415
- Apache Spark 1.4.2.4
- Apache Tomcat 6.0.45

4.8.6 Changes and enhancements

- Support for the Solr time zone (TZ) parameter for [search queries with JSON](#). (DSP-8377)
- TomcatSolrRunner should get the cipher suites for SSL connections from `cassandra.yaml`. (DSP-8370)
- Make EverywhereStrategy return the correct primary endpoint from `calculateNaturalEndpoints`. (DSP-8507)

4.8.6 Resolved issues

- Remove DataStax Enterprise dependency on OpenJDK. (DSP-4934)
- `bulkCassandraWriter` failed in retry tasks when directory already exists. (DSP-5234)
- Wrong timeout value logged following flush timeout exception. (DSP-6971)
- Netty EventExecutorGroup shutdown can occur before channel shutdown. (DSP-8074)
- Support all inter-node (node-to-node) [encryption options](#) in DSE Search legacy netty transport. Use `server_encryption_options` properties in `cassandra.yaml`. (DSP-8374)
- Slow start of spark-sql-thriftserver due to reconnection in SchemaManagerService. (DSP-8470)
- Named parameters in simple statements don't work. (DSP-8495)
- Allow core admin to direct user to use `dsetool create core`. (DSP-8518)
- LeaderManager should ignore decommissioned nodes. (DSP-8532)
- Node should exit when plugin fails to start (deadlock in `PluginManager.activate/deactivate`). (DSP-8648)
- Do not log passwords in log files or show in Spark Web UI. [Environment variables](#) `DSE_USERNAME` and `DSE_PASSWORD` are supported for all Spark commands. Only DataStax Enterprise user has permissions (600) to the log file created by Spark SQL Thrift Server. (DSP-8674)
- Services Only and Services Start doesn't work in Mac standalone installer. (DSP-8710)
- Can't set `SPARK_DAEMON_MEMORY` that is used by Spark-Thriftserver and Spark history server. (DSP-8760)
- [SolrJ-Auth dependencies](#) are now publicly available on the `datastax-public-releases-local` repository. (DSP-8798)
- `dsetool search` commands work with hostname and IP. (DSP-8832)
- Avoid reading from Cassandra during first phase of CQL Solr Query without lazy field loading. (DSP-8833)
- Solr rounds the timestamp key to second if used as primary key when distributed. (DSP-8879)
- Document deletion fails due to incorrect escaping of Solr special characters in unique key. (DSP-8884)
- Make `dsetool search` commands work with SSL + Kerberos. (DSP-8896)
- Fix duplicate results from Solr query. (DSP-8936)
- NoSuchElementException key not found due to concurrency bug in Spark Thriftserver. (DSP-9031)

4.8.5 Release notes for DataStax Enterprise

29 February 2016

4.8.5 Components

The follow components are changed from the previous version of DataStax Enterprise:

- Apache Cassandra 2.1.13.1131 with additional production-certified [Cassandra changes](#)
- Netty 4.0.34.Final
- Spark Cassandra Connector 1.4.2
- Apache Solr 4.10.3.1.358

4.8.5 Changes and enhancements

- [Configure the uptime](#) required for a node to reach full health. (DSP-6883)
- Certified with Centos 6.7. (DSP-7852)
- Provide [user credentials in a configuration file](#) for use with `dse` and `dsetool`. (DSP-8128)

- Enable some [dsetool commands](#) to work with Cassandra password authentication. User credentials can be provided in several ways, see [Providing credentials for authentication](#) on page 344. (DSP-8251)

4.8.5 Resolved issues

- Throw error if Solr schema unique key has a tokenized field. (DSP-5431)
- dse.in.sh should set CASSANDRA_LOG_DIR only if it isn't already set. (DSP-5514)
- Solr slow sub-query request modification consumes POST content. (DSP-6609)
- Can't select inet/varint type in spark-beeline when there is a null inet/varint collection in the table. (DSP-7002)
- Print out warning or error message when kerberos keytab file is not readable. (DSP-7296)
- Solr and demo apps should be served from CATALINA_HOME. (DSP-7414)
- Tarball installs: resources/cassandra/bin/nodetool fails if DSE_HOME is set. (DSP-7445)
- Add a warning after `dsetool create_core` if reindex=true is not specified. (DSP-7631)
- TDseClientTransportFactory is hard coded for `cassandra.thrift.framed.size_mb`. Respect the `thrift_framed_transport_size_in_mb` value in [Cassandra.yaml](#). (DSP-7719)
- No milliseconds in stdout log appender for Cassandra. (DSP-7827)
- Bootstrapping a new data center reverts `dse_perf` keyspace to SimpleStrategy. (DSP-7845)
- The [dsetool utility](#) does not have argument for non-default Cassandra port. (DSP-7856)
- `dsetool core_indexing_status` does not accept hostname (accepts only IP) (DSP-7985)
- Fix live indexing sorted terms seek and iteration. (DSP-8009)
- `dsetool ring` reports clusters as using vnodes when they aren't. (DSP-8148)
- EverywhereStrategy should calculate natural endpoints in sorted order. (DSP-8023)
- Spark job has retries infinitely, RDD partitions out of order, related to CassandraRDDPartitioner violates Partition Contract [SPARKC-323](#). (DSP-8033)
- Fix example JMX remote password file location in `cassandra-env.sh`. (DSP-8245)
- Cannot query non-stored copy field with tuple sub-field source. (DSP-8471)
- FuzzySet assertion for large segments. (DSP-8479)
- Stop optimizing in the commit that occurs at the end of re-indexing a core. (DSP-8613)

4.8.5 Cassandra changes

DataStax Enterprise 4.8.5 certifies Cassandra 2.1.13 with additional production-certified [Cassandra changes](#).

4.8.4 Release notes for DataStax Enterprise

13 January 2016

4.8.4 Components

The follow components are changed from the previous version of DataStax Enterprise:

- Apache Cassandra 2.1.12.1046
- Apache Solr 4.10.3.1.287
- Hive Connector 0.2.11
- Netty 4.0.33.dse

4.8.4 Changes and enhancements

- Enable the [Spark history server](#). (DSP-4532)
- Support for [CentOS 7.1](#). (DSP-7548)

4.8.4 Resolved issues

- dse.in.sh silently fails if not all debian packages are installed. (DSP-5322)
- Incremental repair fails on large LCS column families. (DSP-7124)
- Solr/Lucene versions information is missing in Solr Admin UI. (DSP-7234)
- Re-index doesn't correctly page through wide partitions. (DSP-7591)
- Fix passing [security credentials](#) for new style of start/stop Spark SQL Thrift Server. (DSP-7644)

4.8.4 Cassandra changes

DataStax Enterprise 4.8.4 certifies Cassandra 2.1.12 with additional production-certified [Cassandra changes](#).

4.8.3 Release notes for DataStax Enterprise

9 December 2015

4.8.3 Components

The follow components are changed from the previous version of DataStax Enterprise:

- Apache Cassandra 2.1.11.969
- Apache Spark 1.4.2.2
- Apache Solr 4.10.3.1.263
- Spark Connector 1.4.1

4.8.3 Changes and enhancements

- New dsetool [write_resource](#) and [read_resource](#) commands support uploading extra DSE Search resource files. (DSP-7011)
- Global scheduler for running Solr TTLs. (DSP-7275)

To manage system resource consumption and prevent many Solr cores from executing simultaneous TTL deletes, a new [thread_pool_size](#) option in `dse.yaml` sets the maximum number of threads that concurrently execute TTL deletes.

4.8.3 Resolved issues

- Search Solr NPE in Lucene TermHandler. (DSP-6908)
- Fix consistency for start/stop spark-sql-thriftserver dse commands. (DSP-6922)
- Removed byoh-env.sh: PIG_PARTITIONER='dsetool partitioner' call to speed up BYOH start. (DSP-6962)
- After creating Thrift table for data import handler, unable to create valid Solr core. (DSP-6982)
- Conversion of unordered docids into bitsets for a Solr join query. (DSP-7021)
- Spark Jobserver does not start on package installs. (DSP-7052)
- Upgrade Apache HttpComponents library to 4.4.1. (DSP-7194)
- Gossip race condition causing unavailable ranges during queries. (DSP-7230)
- Cannot write UDT types from Pyspark using DFs. (DSP-7235)
- User latency tracking no longer performs a reverse DNS lookup. (DSP-7252)
- Exclude hadoop-client lib from Spark. (DSP-7253)
- Various streaming errors with out-of-order keys. (DSP-7424)

If you upgraded to 4.8.1 or 4.8.2 and started receiving streaming errors similar to the following errors, **you must scrub the affected table after upgrading**. These errors are triggered when range

tombstones are streamed incorrectly, and can prevent critical operations like repair and bootstrapping from completing.

```
WARN [STREAM-IN-/1.2.3.4] 2015-11-26 10:51:35,930 StreamSession.java:625
- [Stream #481a5680-9423-11e5-bd1f-75f3cea48b3b] Retrying for following
error java.lang.RuntimeException: Last written key DecoratedKey(XXX, YYY)
=> current key DecoratedKey(-ZZZ, QQQ) writing into /var/lib/cassandra/
data/ks/cf/ks-cf-tmp-ka-4-Data.db
```

4.8.3 Cassandra changes

DataStax Enterprise 4.8.3 certifies Cassandra 2.1.11 with additional production-certified [Cassandra changes](#).

4.8.2 Release notes for DataStax Enterprise

Warning: DataStax does not recommend 4.8.1 or 4.8.2 versions for production, see warning. Use 4.8.3 instead. Various streaming errors with out-of-order keys. (DSP-7424)

If you upgraded to 4.8.1 or 4.8.2 and started receiving streaming errors similar to the following errors, **you must scrub the affected table after upgrading**. These errors are triggered when range tombstones are streamed incorrectly, and can prevent critical operations like repair and bootstrapping from completing.

```
WARN [STREAM-IN-/1.2.3.4] 2015-11-26 10:51:35,930 StreamSession.java:625
- [Stream #481a5680-9423-11e5-bd1f-75f3cea48b3b] Retrying for following
error java.lang.RuntimeException: Last written key DecoratedKey(XXX, YYY)
=> current key DecoratedKey(-ZZZ, QQQ) writing into /var/lib/cassandra/
data/ks/cf/ks-cf-tmp-ka-4-Data.db
```

10 November 2015

4.8.2 Components

The follow components are changed from the previous version of DataStax Enterprise:

- Apache Cassandra 2.1.11.908 with production-certified [Cassandra changes](#)
- Apache Thrift 0.9.3
- Hive Connector 0.2.9

4.8.2 Resolved issues

- Error loading class 'solr.FastLRUCache' in search. (DSP-4702)
- Dynamic field marker does not have per document ttl applied to it in Thrift tables. (DSP-6886)
- Reload core using dse tool does not update the solr_config file with coreOptions. (DSP-6899)
- Better dsetool core name error handling. (DSP-6953)
- Possible out of memory error from malformed Thrift protocol messages. (DSP-7019)
- CQLSH COPY command could fail while reading timestamp options. (DSP-7045)

4.8.2 Cassandra changes

DataStax Enterprise 4.8.2 certifies Cassandra 2.1.11 with additional production-certified [Cassandra changes](#).

4.8.1 Release notes for DataStax Enterprise

Warning: DataStax does not recommend 4.8.1 or 4.8.2 versions for production, see warning. Use 4.8.3 instead. Various streaming errors with out-of-order keys. (DSP-7424)

If you upgraded to 4.8.1 or 4.8.2 and started receiving streaming errors similar to the following errors, **you must scrub the affected table after upgrading**. These errors are triggered when range tombstones are streamed incorrectly, and can prevent critical operations like repair and bootstrapping from completing.

```
WARN [STREAM-IN-/1.2.3.4] 2015-11-26 10:51:35,930 StreamSession.java:625
  - [Stream #481a5680-9423-11e5-bd1f-75f3cea48b3b] Retrying for following
    error java.lang.RuntimeException: Last written key DecoratedKey(XXX, YYY)
      >= current key DecoratedKey(-ZZZ, QQQ) writing into /var/lib/cassandra/
        data/ks(cf/ks-cf-tmp-ka-4>Data.db
```

23 October 2015

4.8.1 Components

The follow components are changed from the previous version of DataStax Enterprise:

- Apache Cassandra 2.1.11.872 with production-certified [Cassandra changes](#)
- Apache Solr 4.10.3.1.204
- Apache Spark 1.4.1.3
- Apache Tomcat 6.0.44

4.8.1 Changes and enhancements

- Sqoop import and export jobs support alternate [connection managers](#). (DSP-5974)
- Support [interval facetting](#) via CQL. (DSP-6804)

4.8.1 Resolved issues

- SOLR created fields like _docBoost do not honor row TTLs - blocking TTL of row. (DSP-4872)
- dse script hard codes ulimits during startup. (DSP-5713)
- NullPointerException resulting from standard CQL query on Solr field. (DSP-6396)
- Add package dependencies for 'which'. (DSP-6459)
- Sqoop cq-import fails with StackOverflowError when too many columns requested. (DSP-6528)
- Portfolio demo: HistLoss table created by Hive is wrong (Spark okay). (DSP-6621)
- FuzzySet integer min value causes assertion bug. (DSP-6658)
- ConcurrentModificationException when using live indexing. (DSP-6717)
- Misleading error message when attempting to unload non-existing core. (DSP-6722)
- Solr slow sub-query log doesn't report prepare and process times for CQL Solr queries. (DSP-6724)
- Ensure hard commit is executed immediately after re-indexing. (DSP-6735)
- Indentation error in the Pyspark Streaming Distribution. (DSP-6739)
- Disabling CqlSlowQueryLog service is not properly [logged](#). (DSP-6760)
- Unable to use [multiple fq](#) in a CQL Solr query. (DSP-6763)
- Spark Streaming [checkpointing](#) doesn't work with internal authentication. (DSP-6776)
- Exceptions in leader manager are swallowed and silently disable leader manager. (DSP-6786)
- Support interval facetting via CQL. (DSP-6804)
- Tuple docs are deleted twice. (DSP-6806)
- Solr CQL queries fail with non-compound timestamp keys. (DSP-6810)
- Installer needs to symlink /usr/bin/dse-client-tool. (DSP-6851)
- Xmn should not be set when using G1GC in cassandra-env.sh. (DSP-6860)
- Disallow setting [Solr concurrency level](#) at 1 via JMX. (DSP-6868)
- SSL + Kerberos qop=auth-int or auth-conf not reported as warning. (DSP-6939)

4.8.1 Cassandra changes

DataStax Enterprise 4.8.1 certifies Cassandra 2.1.11 with additional production-certified [Cassandra changes](#).

4.8 Release notes for DataStax Enterprise

23 September 2015

4.8 Components

- Apache Cassandra 2.1.9.791
- Apache Hadoop 1.0.4.18
- Apache Hive 0.12.0.11
- Apache Mahout 0.8
- Apache Pig 0.10.1
- Apache Solr 4.10.3.1.172
- Apache Spark 1.4.1.1
- Spark Jobserver 0.5.2
- Apache Sqoop 1.4.5.15.1
- Apache Tomcat 6.0.39
- Java Driver for Apache Cassandra 2.1.7.1
- Hive Connector 0.2.8
- Netty 4.0.23.Final
- Spark Connector 1.4.0

4.8 Changes and enhancements

DataStax Enterprise 4.8 introduces the following changes in addition to the [new features](#).

- [Delete \(unload_core\) a Solr index](#) without deleting/dropping the underlying keyspace and table. (DSP-1533)
- DSE Search supports indexing and querying of Cassandra [tuples and user defined types \(UDTs\)](#). (DSP-3748)
- Support [SELECT count\(\)](#) in CQL Solr queries. (DSP-4843)
- Solr [Data Import Handler](#) is deprecated. Use parquet or JSON instead. For Spark earlier than 1.4, use the DataReader API. (DSP-4887)
- Support [dynamic field Solr LatLonType](#) with spatial subfield prefix naming conventions. (DSP-5021)
- Enable [slow query log](#) collection by default, change [dse.yaml options](#) and default values. (DSP-5211)
- DSE Search supports [advanced spatial queries](#) for polygon shapes. Use [installation script](#) for JTS. (DSP-5320)
- [Audit filter](#) in dse.yaml can filter on keyspaces using a regular expression. (DSP-5407)
- Restrict [HTTP and Solr Admin access to DSE Search](#). Create a Tomcat connector or use rpc_address in cassandra.yaml. (DSP-5440)
- Audit log tables use DateTieredCompactionStrategy (DTCS). (DSP-5462)

DataStax recommends changing tables that were created in earlier releases to use DTCS:

```
ALTER TABLE dse_audit.audit_log WITH
    COMPACTON={ 'class': 'DateTieredCompactionStrategy' };
```

- Implement updateDocuments for block join in [live indexing](#). (DSP-5506)
- Node health change and new commands to inspect node health and indexing status. (DSP-5508)
 - Node health options are always enabled. You can customize the refresh rate for [node_health_options](#) in dse.yaml.

- Use the [dsetool node_health](#) command to retrieve a dynamic score between 0 and 1 that describes the health of a DataStax Enterprise node.
- Use the [dsetool core_indexing_status](#) command to retrieve the dynamic indexing status (INDEXING, FINISHED, or FAILED) of the specified core in a DSE Search node. Specify `--all` to retrieve the dynamic indexing status of all Solr cores.
- The CassandraStorage() handler is deprecated. Use [CqlNativeStorage\(\)](#) instead. Thrift-based Hadoop classes that are not Java-driver based are deprecated. (DSP-5750)
- Support for Shark is removed. Use [Spark SQL](#) instead. (DSP-5824)
- Automatically [insert CQLSearchHandler](#) into `solrconfig.xml`. (DSP-5831)
- Integrate [Spark Jobserver](#). (DSP-5827)
- The [configuration of credentials](#) for the Spark SQL Thrift Server is controlled by the `hive-site.xml` file in the Spark directory. (DSP-5927)
- DseSparkConfHelper, DseSparkContext, DseStreamingContext are deprecated. DataStax Enterprise configuration is applied with scripts at startup. Using Spark requires no modifications. (DSP-5987)
- Reduce search index size by providing a way to disable [DSE from adding _partitionKey field](#) to Solr documents. (DSP-6021)
- The [dse client-tool](#) supports basic operations that are required to run client applications. (DSP-6025)
 - In the authenticated cluster, provide Cassandra credentials instead of JMX username and password.
 - The `dsetool sparkmaster` command is deprecated, and is replaced by the [dse client-tool spark master-address](#) command.
 - Replaces the `dsetool jobtracker` command.
- The multi-threaded indexing back pressure algorithm provides improved performance, reliability, and visibility. (DSP-6103)
- DSE PySpark Scala wrappers are deprecated. PySpark and DSE PySpark are still supported using the more efficient [DataFrames API](#). (DSP-6203).
- [BYOH Hive](#) is deprecated and will be removed in a future release. (DSP-6337)
- Integrate Spark 1.4. (DSP-6356)
- You can use an ODBC or JDBC client to interact with the Spark SQL Thrift server. When reading and writing large amounts of data, DataStax recommends using the [Cassandra-backed DataFrames](#). (DSP-6385)
- Parameter name change for the [Spark Connector property](#): `spark.cassandra.input.page.size` is renamed to `spark.cassandra.input.fetch.size_in_rows`.
- G1GC Java Garbage Collector is enabled by default when DataStax Enterprise runs under Java 8. (DSP-6392)
- In DSE Search, Solr deep paging is off by default and is [configurable](#). (DSP-6400)

Set the [cql_solr_query_paging](#) option in the `dse.yaml` file as appropriate for the workload.

- For development clusters in mixed workloads, DataStax recommends using the default `cql_solr_query_paging: off` setting to turn paging off. You can dynamically enable paging in a [JSON query](#) by using the `paging:driver` parameter. This `cql_solr_query_paging: off` setting is helpful for a mixed workload cluster where normal search queries are run.
- For [SearchAnalytics](#) nodes, DataStax recommends using the `cql_solr_query_paging: driver` setting. This `cql_solr_query_paging: driver` setting is helpful when analytics nodes leverage search.
- Verify all unique key elements are indexed in Solr schema. (DSP-6460)
- [HiveServer2 authentication](#) changes in a secured cluster. (DSP-6508)
- Deleted doc might show in a DSE Search query when using facet tagging and excluding filters or query result cache is on. (DSP-6529)
- Improve indexing performance by using multi-thread indexing for wide rows. (DSP-6592)
- The default log directory is changed to `$HOME/spark-thrift-server` for starting the spark-thrift-server. (DSP-6720)

4.8 Resolved issues

- CqlBulkOutputFormat is broken by upgrade to Cassandra 2.1. (DSP-4875)
- Spark executors use `cassandra.yaml` and `dse.yaml` to get information about SSL and Kerberos. (DSP-5576)
- Bootstrapping TDE nodes try to read from themselves. (DSP-5701)
- The `dse` startup script hard codes ulimits for the `cassandra` user. (DSP-5713)
- Permissions set by the DSE/Cassandra installation process on the `/var/lib/cassandra` directory are 750 by default. Permissions are now 751. (DSP-5970)
- Remove query docFreq lookup for filters. (DSP-5982)
- `WAIT_FOR_START=10` is too low on an `m1.large` instance type. (DSP-6077)
- Hive connector uses yaml files and property names that are not DSE-compatible. (DSP-6042)
- DSE Spark --jars flag does not work. (DSP-6087)
- The value of `load_max_time_per_core` is too low by default. (DSP-6260)
- Range tombstones might cause unwanted deletes from the index. (DSP-6301)
- `NullPointerException` thrown in Solr when enabling live indexing. (DSP-6336)
- Remove unnecessary contention and improve live indexing reader latency by removing usage of `RAMFile` for deletes. (DSP-6362)
- Failure to close key iterator could lead to reference leaks. (DSP-6377)
- Cannot set `max_solr_concurrency_per_core` at 1. (DSP-6408)
- Single pass queries do not work with `UUID`, `TimeUUID`, `Inet`, `Decimal`, and `Varint` types. (DSP-6443)
- `"_docBoost"` prevents docs from being indexed. (DSP-6486)
- `enable_back_pressure_adaptive_nrt_commit` is not set to true by default. (DSP-6453)
- Changes in `spark-env.sh` do not get picked up properly after restart. (DSP-6502)
- Portfolio manager web interface broken. (DSP-6593)
- Solr re-index progress does not show in OpsCenter Activity. (DSP-6613)
- Allow for configuring the NIO connector via Tomcat `server.xml`. (DSP-6645)

The BIO connector is more performant. However, you can revert back to NIO by setting the protocol to `org.apache.coyote.http11.Http11NioProtocol` in the Tomcat `server.xml` file under `~/dse/resources/tomcat/conf`:

```
<Connector port="${http.port}"
    protocol="org.apache.coyote.http11.Http11NioProtocol"
    connectionTimeout="20000"
    redirectPort="8443" URIEncoding="UTF-8"/>
```

After you start DataStax Enterprise, you can verify the NIO connector with these lines in the Tomcat Catalina log file:

```
Sep 14, 2015 11:22:39 PM org.apache.coyote.http11.Http11NioProtocol init
INFO: Initializing Coyote HTTP/1.1 on http-8983
Sep 14, 2015 11:22:41 PM org.apache.coyote.http11.Http11NioProtocol start
INFO: Starting Coyote HTTP/1.1 on http-8983
```

- Top deletion might cause unwanted deletes from the index. (DSP-6654)
- 2GB SSTable limitation for In-Memory table per node. (DSP-6685)

4.8 Known issue

- DSE Search: Setting Solr concurrency by JMX causes deadlock with index flushing. (DSP-6772)

If you use JMX to set IndexPool Concurrency, a deadlock occurs for index workpool flushing. Indexing stops until the time that is set for `flush_max_time_per_core` is reached.

Workaround: Do not use JMX to set IndexPool Concurrency. If you need to change IndexPool Concurrency, edit the `dse.yaml` file to change `max_solr_concurrency_per_core` and restart the node.

- To improve index performance, set production appropriate [mergeScheduler](#) values for DSE Search with near real time (NRT) indexing. (DSP-9325)

4.8 Cassandra changes

DataStax Enterprise 4.8 certifies Cassandra 2.1.9 with additional production-certified [Cassandra changes](#).

Cassandra changes

Familiarize yourself with the changes and features in the new Cassandra version. A complete list of changes is available in [CHANGES.txt](#). The changes in this document identify only the DataStax Enterprise production-certified changes that are in addition to the specified Cassandra version. Patch releases without Cassandra changes include only the changes for the specified release, and do not include additional production-certified changes.

DataStax Enterprise includes additional production-certified Cassandra changes for:

- Cassandra changes in [DataStax Enterprise 4.8.9](#)
- Cassandra changes in [DataStax Enterprise 4.8.8](#)
- Cassandra changes in [DataStax Enterprise 4.8.7](#)
- Cassandra changes in [DataStax Enterprise 4.8.6](#)
- Cassandra changes in [DataStax Enterprise 4.8.5](#)
- Cassandra changes in [DataStax Enterprise 4.8.4](#)
- Cassandra changes in [DataStax Enterprise 4.8.3](#)
- Cassandra changes in [DataStax Enterprise 4.8.2](#)
- Cassandra changes in [DataStax Enterprise 4.8.1](#)
- Cassandra changes in [DataStax Enterprise 4.8](#)

DataStax Enterprise 4.8.9

DataStax Enterprise 4.8.9 includes these production-certified Cassandra changes that are in addition to the Cassandra 2.1.15 version:

- Account for partition deletions in tombstone histogram. (CASSANDRA-12112)
- Avoid stalling Paxos when the Paxos state expires. (CASSANDRA-12043)
- Remove finished incoming streaming connections from MessagingService. (CASSANDRA-11854)
- Don't try to get SSTables for non-repairing column families. (CASSANDRA-12077)
- Prevent select statements with clustering key greater than 64k. (CASSANDRA-11882)
- Avoid marking too many SSTables as repaired .(CASSANDRA-11696)
- Fix clock skew corrupting other nodes with Paxos. (CASSANDRA-11991)
- Remove distinction between non-existing static columns and existing but null in LWTs. (CASSANDRA-9842)
- Support mlockall on IBM POWER arch. (CASSANDRA-11576)
- Cache local ranges when calculating repair neighbors. (CASSANDRA-11933)
- Allow LWT operation on static column with only partition keys. (CASSANDRA-10532)
- Create interval tree over canonical SSTables to avoid missing SSTables during streaming. (CASSANDRA-11886)
- cqlsh COPY FROM: shutdown parent cluster after forking, to avoid corrupting SSL connections. (CASSANDRA-11749)
- Updated cqlsh Python driver to fix DESCRIBE problem for legacy tables. (CASSANDRA-11055)
- cqlsh: apply current keyspace to source command. (CASSANDRA-11152)

DataStax Enterprise 4.8.8

DataStax Enterprise 4.8.8 includes these production-certified Cassandra changes that are in addition to the Cassandra 2.1.14 version:

- Backport CASSANDRA-11578 (CASSANDRA-11750)
- Clear out parent repair session if repair coordinator dies (CASSANDRA-11824)
- Set default streaming_socket_timeout_in_ms to 24 hours (CASSANDRA-11840)
- Do not consider local node a valid source during replace (CASSANDRA-11848)
- Avoid holding SSTableReaders for duration of incremental repair (CASSANDRA-11739)
- Add message dropped tasks to nodetool netstats (CASSANDRA-11855)
- Don't compute expensive MaxPurgeableTimestamp until we've verified there's an expired tombstone (CASSANDRA-11834)
- Fix paging on DISTINCT queries repeats result when first row in partition change (CASSANDRA-11679)
- Add option to disable use of severity in DynamicEndpointSnitch (CASSANDRA-11737)
- cqlsh COPY FROM fails for null values with non-prepared statements (CASSANDRA-11631)
- Make cython optional in pylib/setup.py (CASSANDRA-11630)
- Change order of directory searching for cassandra.in.sh to favor local one (CASSANDRA-11628)
- cqlsh COPY FROM fails with []{} chars in UDT/tuple fields/values (CASSANDRA-11633)
- cqlsh: COPY FROM throws TypeError with Cython extensions enabled (CASSANDRA-11574)
- cqlsh: COPY FROM ignores NULL values in conversion (CASSANDRA-11549)
- Validate levels when building LeveledScanner to avoid overlaps with orphaned SSTables (CASSANDRA-9935)
- Start L0 STCS-compactions even if there is a L0 -> L1 compaction going (CASSANDRA-10979)

DataStax Enterprise 4.8.7

DataStax Enterprise 4.8.7 includes these production-certified Cassandra changes that are in addition to the Cassandra 2.1.13 version:

- cqlsh: COPY FROM ignores NULL values in conversion (CASSANDRA-11549)
- (cqlsh) Fix potential COPY deadlock when parent process is terminating child processes (CASSANDRA-11505)
- Replace SSTables on DataTracker before marking them as non-compacting during anti-compaction (CASSANDRA-11548)
- Checking if an unlogged batch is local is inefficient (CASSANDRA-11529)
- Fix paging for COMPACT tables without clustering columns (CASSANDRA-11467)
- Fix out-of-space error treatment in memtable flushing (CASSANDRA-11448)
- Backport CASSANDRA-10859 (CASSANDRA-11415)
- COPY FROM fails when importing blob (CASSANDRA-11375)
- Add a -j parameter to scrub/cleanup/upgradesstables to state how many threads to use (CASSANDRA-11179)

DataStax Enterprise 4.8.6

DataStax Enterprise 4.8.6 includes these production-certified Cassandra changes that are in addition to the Cassandra 2.1.13 version:

- Backport CASSANDRA-10679 (CASSANDRA-9598)
- Don't do defragmentation if reading from repaired SSTables (CASSANDRA-10342)
- Fix streaming_socket_timeout_in_ms not enforced (CASSANDRA-11286)
- Avoid dropping message too quickly due to missing unit conversion (CASSANDRA-11302)
- COPY FROM on large datasets: fix progress report and debug performance (CASSANDRA-11053)
- InvalidateKeys should have a weak ref to key cache (CASSANDRA-11176)

- Don't remove FailureDetector history on removeEndpoint (CASSANDRA-10371)
- Only notify if repair status changed (CASSANDRA-11172)
- Add partition key to TombstoneOverwhelmingException error message (CASSANDRA-10888)
- Use logback setting for 'cassandra -v' command (CASSANDRA-10767)
- Fix sstableloader to unthrottle streaming by default (CASSANDRA-9714)
- Fix incorrect warning in 'nodetool status' (CASSANDRA-10176)

DataStax Enterprise 4.8.5

DataStax Enterprise 4.8.5 includes these production-certified Cassandra changes that are in addition to the Cassandra 2.1.13 version:

- Properly release SSTable ref when doing offline scrub (CASSANDRA-10697)
- Improve nodetool status performance for large cluster (CASSANDRA-7238)
- Make it clear what DTCS timestamp_resolution is used for (CASSANDRA-11041)
- Gossiper#isEnabled is not thread safe (CASSANDRA-11116)
- Avoid major compaction mixing repaired and unrepaired SSTables in DTCS (CASSANDRA-11113)
- test_bulk_round_trip_blogposts is failing occasionally (CASSANDRA-10938)
- LCS doesn't do L0 STC on new tables while an L0->L1 compaction is in progress (CASSANDRA-10979)
- Fix isJoined return true only after becoming cluster member (CASSANDRA-11007)
- Fix bad gossip generation seen in long-running clusters (CASSANDRA-10969)
- Avoid NPE when incremental repair fails (CASSANDRA-10909)
- Unmark SSTables compacting once they are done in cleanup/scrub/upgradesstables (CASSANDRA-10829)
- Revert CASSANDRA-10012 and add more logging (CASSANDRA-10961)
- Allow simultaneous bootstrapping with strict consistency when no vnodes are used (CASSANDRA-11005)
- Log a message when major compaction does not result in a single file (CASSANDRA-10847)
- (cqlsh) fix cqlsh_copy_tests when vnodes are disabled (CASSANDRA-10997)
- (cqlsh) fix formatting bytarray values (CASSANDRA-10839)
- (cqlsh) Add request timeout option to cqlsh (CASSANDRA-10686)
- Avoid AssertionError while submitting hint with LWT (CASSANDRA-10477)
- If CompactionMetadata is not in stats file, use index summary instead (CASSANDRA-10676)
- Retry sending gossip syn multiple times during shadow round (CASSANDRA-8072)
- Fix pending range calculation during moves (CASSANDRA-10887)

DataStax Enterprise 4.8.4

DataStax Enterprise 4.8.4 includes these production-certified Cassandra changes that are in addition to the Cassandra 2.1.12 version:

- Sane default (200Mbps) for inter-DC streaming throughput (CASSANDRA-8708)
- Match cassandra-loader options in COPY FROM (CASSANDRA-9303)
- Fix binding to any address in CqlBulkRecordWriter (CASSANDRA-9309)
- Fix the way we replace SSTables after anticompaaction (CASSANDRA-10831)
- cqlsh fails to decode UTF-8 characters for text typed columns (CASSANDRA-10875)
- Log error when stream session fails (CASSANDRA-9294)
- Fix bugs in commit log archiving startup behavior (CASSANDRA-10593)
- (cqlsh) further optimise COPY FROM (CASSANDRA-9302)
- Allow CREATE TABLE WITH ID (CASSANDRA-9179)
- Make Stress compiles within eclipse (CASSANDRA-10807)
- Cassandra Daemon should print JVM arguments (CASSANDRA-10764)

- Allow cancellation of index summary redistribution (CASSANDRA-8805)
- sstableloader will fail if there are collections in the schema tables (CASSANDRA-10700)
- Disable reloading of GossipingPropertyFileSnitch (CASSANDRA-9474)
- Fix Stress profile parsing on Windows (CASSANDRA-10808)

DataStax Enterprise 4.8.3

DataStax Enterprise 4.8.3 includes these production-certified Cassandra changes that are in addition to the Cassandra 2.1.11 version:

- Fix incremental repair hang when replica is down (CASSANDRA-10288)
- Avoid writing range tombstones after END_OF_ROW marker (CASSANDRA-10791)
- Optimize the way we check if a token is repaired in anticompaaction (CASSANDRA-10768)
- Add proper error handling to stream receiver (CASSANDRA-10774)
- Warn or fail when changing cluster topology live (CASSANDRA-10243)
- Status command in debian/ubuntu init script doesn't work (CASSANDRA-10213)
- Some DROP ... IF EXISTS incorrectly result in exceptions on non-existing KS (CASSANDRA-10658)
- DeletionTime.compareTo wrong in rare cases (CASSANDRA-10749)
- Force encoding when computing statement ids (CASSANDRA-10755)
- Properly reject counters as map keys (CASSANDRA-10760)
- Fix the sstable-needs-cleanup check (CASSANDRA-10740)
- (cqlsh) Print column names before COPY operation (CASSANDRA-8935)
- Add Native-Transport-Requests back to tpstats (CASSANDRA-10044)
- Make paging logic consistent between searcher impls (CASSANDRA-10683)
- Fix CompressedInputStream for proper cleanup (CASSANDRA-10012)
- (cqlsh) Support counters in COPY commands (CASSANDRA-9043)
- Try next replica if not possible to connect to primary replica on ColumnFamilyRecordReader (CASSANDRA-2388)
- Limit window size in DTCS (CASSANDRA-10280)
- sstableloader does not use MAX_HEAP_SIZE env parameter (CASSANDRA-10188)
- (cqlsh) Improve COPY TO performance and error handling (CASSANDRA-9304)
- Don't remove level info when running upgradesstables (CASSANDRA-10692)
- Create compression chunk for sending file only (CASSANDRA-10680)
- Make buffered read size configurable (CASSANDRA-10249)
- Forbid compact clustering column type changes in ALTER TABLE (CASSANDRA-8879)
- Reject incremental repair with subrange repair (CASSANDRA-10422)
- Add a nodetool command to refresh size_estimates (CASSANDRA-9579)
- Shutdown compaction in drain to prevent leak (CASSANDRA-10079)
- Invalidate cache after stream receive task is completed (CASSANDRA-10341)
- Reject counter writes in CQLSSTableWriter (CASSANDRA-10258)
- Remove superfluous COUNTER_MUTATION stage mapping (CASSANDRA-10605)
- Improve json2sstable error reporting on nonexistent columns (CASSANDRA-10401)
- (cqlsh) fix COPY using wrong variable name for time_format (CASSANDRA-10633)
- Do not run SizeEstimatesRecorder if a node is not a member of the ring (CASSANDRA-9912)
- CompressionInfo will be fsynced on close (CASSANDRA-10534)
- Reduce contention in CompositeType instance interning (CASSANDRA-10433)

DataStax Enterprise 4.8.2

DataStax Enterprise 4.8.2 includes these production-certified Cassandra changes that are in addition to the Cassandra 2.1.11 version:

- Leak detected, after nodetool drain. (CASSANDRA-10079)

- Improve handling of dead nodes in gossip (CASSANDRA-10298)
- Fix logback-tools.xml incorrectly configured for outputing to System.err (CASSANDRA-9937)
- Fix streaming to catch exception so retry not fail (CASSANDRA-10557)
- Add validation method to PerRowSecondaryIndex (CASSANDRA-10092)
- Support encrypted and plain traffic on the same port (CASSANDRA-10559)
- Do STCS in DTCS windows (CASSANDRA-10276)
- Don't try to get ancestors from half-renamed SSTables (CASSANDRA-10501)
- Avoid repetition of JVM_OPTS in debian package (CASSANDRA-10251)
- Fix potential NPE from handling result of SIM.highestSelectivityIndex (CASSANDRA-10550)
- Fix sorting for queries with an IN condition on partition key columns (CASSANDRA-10363)

DataStax Enterprise 4.8.1

DataStax Enterprise 4.8.1 includes these production-certified Cassandra changes that are in addition to the Cassandra 2.1.11 version:

- Fix paging issues with partitions containing only static columns data (CASSANDRA-10381)
- Fix conditions on static columns (CASSANDRA-10264)
- AssertionError: attempted to delete non-existing file CommitLog (CASSANDRA-10377)
- Merge range tombstones during compaction (CASSANDRA-7953)
- (cqlsh) Distinguish negative and positive infinity in output (CASSANDRA-10523)
- (cqlsh) allow custom time_format for COPY TO (CASSANDRA-8970)
- Don't allow startup if the node's rack has changed (CASSANDRA-10242)
- (cqlsh) show partial trace if incomplete after max_trace_wait (CASSANDRA-7645)
- Fix mmap file segment seeking to EOF (CASSANDRA-10478)
- Allow LOCAL_JMX to be easily overridden (CASSANDRA-10275)
- Mark nodes as dead even if they've already left (CASSANDRA-10205)
- Update internal python driver used by cqlsh (CASSANDRA-10161, CASSANDRA-10507)
- Bulk Loader API could not tolerate even node failure (CASSANDRA-10347)
- Avoid misleading pushed notifications when multiple nodes share an rpc_address (CASSANDRA-10052)
- Fix dropping undroppable when message queue is full (CASSANDRA-10113)
- Fix potential ClassCastException during paging (CASSANDRA-10352)
- Prevent ALTER TYPE from creating circular references (CASSANDRA-10339)
- Fix cache handling of 2i and base tables (CASSANDRA-10155, 10359)
- Fix NPE in nodetool compactionhistory (CASSANDRA-9758)
- Fix rare race where older gossip states can be shadowed (CASSANDRA-10366)
- Fix consolidating racks violating the RF contract (CASSANDRA-10238)

DataStax Enterprise 4.8

DataStax Enterprise 4.8 includes production-certified Cassandra changes that are in addition to the Cassandra 2.1.9 version, including everything listed under 2.1.10 in [CHANGES.txt](#) and:

- (Pig) support BulkOutputFormat as a URL parameter (CASSANDRA-7410)
- BATCH statement is broken in cqlsh (CASSANDRA-10272)
- Added configurable warning threshold for GC duration (CASSANDRA-8907)
- (cqlsh) Make cqlsh PEP8 compliant (CASSANDRA-10066)
- (cqlsh) Fix error when starting cqlsh with --debug (CASSANDRA-10282)
- Scrub, cleanup and upgrade do not unmark compacting until all operations have completed, regardless of the occurrence of exceptions (CASSANDRA-10274)
- Fix handling of streaming EOF (CASSANDRA-10206)
- Only check KeyCache when it is enabled (CASSANDRA-9872)

- Change streaming_socket_timeout_in_ms default to 1 hour (CASSANDRA-8611)
- (cqlsh) update list of CQL keywords (CASSANDRA-9232)
- Disallow decommission when node is in drained state (CASSANDRA-8741)