## Contents

**Chapter 1. About DSE Advanced Security** ............................................................................... 6

**Chapter 2. Security FAQs** ...................................................................................................... 7

**Chapter 3. Security checklists** ................................................................................................. 10
  - Database security checklist .................................................................................................. 10
  - Search security checklist .................................................................................................... 11
  - Analytics security checklist ............................................................................................... 12
  - Graph security checklist .................................................................................................... 13

**Chapter 4. Securing the environment** .................................................................................. 15
  - Securing ports .................................................................................................................. 15
  - Securing the TMP directory .............................................................................................. 17

**Chapter 5. Authentication and authorization** ...................................................................... 18
  - Configuring authentication and authorization .................................................................. 18
    - About DSE Unified Authentication ............................................................................... 18
    - Configuring security keyspaces .................................................................................... 20
    - Setting up Kerberos ....................................................................................................... 22
    - Enabling authentication and authorization .................................................................. 27
    - Configuring JMX authentication .................................................................................... 37
    - Configuring cache settings ............................................................................................ 40
    - Securing schema information ....................................................................................... 41
  - Managing database access ................................................................................................. 44
    - About RBAC .................................................................................................................. 44
    - Setting up logins and users ......................................................................................... 44
    - Assigning permissions .................................................................................................... 54
  - Providing credentials from DSE tools .............................................................................. 84
    - About clients .................................................................................................................. 84
    - Internal and LDAP authentication ............................................................................... 84
    - Kerberos ........................................................................................................................ 89
    - Nodetool ....................................................................................................................... 96
    - JConsole ........................................................................................................................ 96

**Chapter 6. Auditing database activity** ................................................................................. 97
  - Enabling database auditing ............................................................................................... 97
  - Enable audit logging to a file ......................................................................................... 97
<table>
<thead>
<tr>
<th>Chapter 7. Transparent data encryption</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Transparent Data Encryption</td>
<td>113</td>
</tr>
<tr>
<td>Configuring local encryption</td>
<td>114</td>
</tr>
<tr>
<td>Setting up local encryption keys</td>
<td>114</td>
</tr>
<tr>
<td>Encrypting configuration file properties</td>
<td>116</td>
</tr>
<tr>
<td>Encrypting system resources</td>
<td>117</td>
</tr>
<tr>
<td>Encrypting tables</td>
<td>118</td>
</tr>
<tr>
<td>Rekeying existing data</td>
<td>120</td>
</tr>
<tr>
<td>Troubleshooting encryption key errors</td>
<td>122</td>
</tr>
<tr>
<td>Configuring KMIP encryption</td>
<td>123</td>
</tr>
<tr>
<td>Adding a KMIP host</td>
<td>123</td>
</tr>
<tr>
<td>Encrypting configuration file properties</td>
<td>125</td>
</tr>
<tr>
<td>Encrypting system resources</td>
<td>126</td>
</tr>
<tr>
<td>Encrypting table data</td>
<td>127</td>
</tr>
<tr>
<td>Expiring an encryption key</td>
<td>129</td>
</tr>
<tr>
<td>Rekeying tables using a new key</td>
<td>130</td>
</tr>
<tr>
<td>Troubleshooting KMIP connections</td>
<td>132</td>
</tr>
<tr>
<td>Encrypting Search indexes</td>
<td>132</td>
</tr>
<tr>
<td>Encrypting new Search indexes</td>
<td>132</td>
</tr>
<tr>
<td>Encrypting existing Search indexes</td>
<td>133</td>
</tr>
<tr>
<td>Tuning encrypted Search indexes</td>
<td>135</td>
</tr>
<tr>
<td>Migrating encrypted tables from earlier versions</td>
<td>136</td>
</tr>
<tr>
<td>Bulk loading data between TDE-enabled clusters</td>
<td>137</td>
</tr>
</tbody>
</table>

Chapter 8. Configuring SSL

Steps for configuring SSL
Chapter 9. Securing Spark connections................................................................. 162
Chapter 1. About DSE Advanced Security

What is DataStax Enterprise Advanced Security?

DataStax Enterprise (DSE) Advanced Security provides features that allow you to minimize risk, achieve granular access control, and work toward compliance with regulatory standards.

Key features:

- Unified authentication and role management:
  - Allows you to integrate existing Kerberos, LDAP, and Active Directory users and schemes across DSE resources.

- Data auditing:
  - Provides the ability to understand who looked at what and when; and who changed what and when. Both factors are important to meet security compliance standards.

- Row-level access control and proxy authentication:
  - Restricts which rows a user has access to within a table while preserving client-side identities and privileges in middleware such as web servers.

- Permission management:
  - Uses a GRANT/REVOKE paradigm and Active Directory/LDAP to assign access permissions.

Ready to get started?

Start by reading our DSE Security FAQs and security checklists for DSE components.
Chapter 2. DSE Advanced Security FAQs

Questions and answers about DSE Advanced Security are provided in these categories:

• General
• Authentication and authorization
• Encryption
• Auditing

General

What communication protocols are used?
All communication occurs over TCP sockets and can be secured by using the standard Java Security SSL/TLS implementation in the JVM. Additional application specific protocols like gossip and the CQL Binary Protocol rely on these sockets for transport, for a list of ports used by DSE see Securing DataStax Enterprise ports.

Authentication and authorization

What are the restrictions to the default cassandra user?
The cassandra default account has access to all database resources. When logging in or performing an action, DSE sets the consistency level to QUORUM for this account. In a production environment, using the cassandra account may negatively affect performance and encounter failures. DataStax recommends Adding a superuser login immediately after enabling DSE Unified Authentication with role based access control.

How are user permissions managed?
Superuser permissions allow creation and deletion of other users, and the ability to grant or revoke permissions. Use the default cassandra user only to assist with the initial setup of new users and superusers, and then disable it. The DSE Role Manager determines which roles to assign to authenticated users. See About Role Based Access Control.

How are user groups supported?
DSE supports role management based on LDAP group membership. Configure an LDAP scheme with group lookup, set the Role Management mode option to LDAP, and create roles that match the group names and then assign permissions. See Defining an LDAP scheme.

For efficiency, DataStax recommends using the memberof search method for group lookup; however DataStax also supports directory search.

How are user-action permissions supported?
DSE supports standard object permission management to assign roles specific permissions at the table and row level. Permissions to access all keyspaces, a named keyspace, a table, function, or MBean can be granted to a role. See Setting up logins and users

What authentication mechanisms are supported?
• Internal: Connections provide credentials for a role that has an internally stored password. No additional configuration is required. See Setting up logins and users.

• LDAP: Connections provide LDAP credentials. DSE passes the credentials for verification to LDAP. See Defining an LDAP scheme.

• Kerberos: Connections provide a Kerberos ticket. DSE is configured as a Service Principal (see Setting up Kerberos) and passes the tickets to KDS for verification. See Defining a Kerberos scheme

What LDAP servers are supported?

Can access be restricted using IP whitelisting and blacklisting?
In general, arbitrary client programs do not access the database. Database access by the general user population is controlled at the application layer. Application node to database node access should be controlled by using conventional firewall mechanisms, such as Linux iptables. However, database administrators are an exception to allow connections from DBA hosts.

**What granularity of access to specific elements of data is supported?**
Authorization is granted or revoked at the row level for data.

**What is the difference between RBAC and RLAC?**
Role-based access control (RBAC) refers to authorization to any database resource including row-level access control (RLAC). Row-level access control refers to the feature that allows permissions to be granted/revoked on rows within a table by filtering a text-based partition column.

### Row-level access control (RLAC)
Frequently asked questions about setting row-level permissions.

#### How do I restrict access to a row?
Each table can have a single UTF-8 partition key column on which you build filters to grant access (separate command) to rows within the table. RESTRICT only sets the filtering column name:

```sql
RESTRICT ROWS ON [keyspace_name.]table_name USING partition_key;
```

After setting the column name, use the GRANT command to configure access to rows.

#### Can I unrestrict access to rows in a table with RLAC authorization?
The partition key to filter on using GRANT can be unselected from the table:

```sql
UNRESTRICT ROWS ON [keyspace_name.]table_name USING partition_key;
```

Use the LIST command on the table to display all roles that have been granted permissions.

Unrestricing the column does not grant access to all columns within the table; it invalidates existing filters. Users that are granted access with a filter will be unable to access any rows within the table. To grant permissions to all rows, grant permission on the table to the role.

#### How do I grant permissions for rows in a table?
Configure access to rows within a table by specifying a filtering string that is applied to the partition key column selected in the RESTRICT command. Use case-sensitive literal text in the filter string. Row-level authorization applies only to rows that exactly match the filtering_data. You can create as many RLAC grant variations as required by your security policies. To allow access to rows within a table:

```sql
GRANT permission on 'filtering_data' ROWS IN keyspace_name.table_name to role_name;
```

Use the LIST command to display all permissions a role has on a resource.

#### How do I revoke permissions for rows in a table?
Row permissions are stored based on the filtering string, to remove a permission use the REVOKE command with the exact filtering string you want to remove:

```sql
REVOKE permission on 'filtering_data' ROWS IN keyspace_name.table_name;
```

LIST ALL PERMISSIONS ON TABLE table_name shows all filters granted to roles.

What happens if you run a RESTRICT command on a table that already has a restriction?
Tables have only a single restriction. Running the RESTRICT command replaces the existing restriction. Use DESCRIBE TABLE to view the existing restrictions on the table.

What happens if there a role has access to the keyspace/table level and I grant row access?

Permission is hierarchical, if permission was also granted to the keyspace or table the user has access to all rows in the table. The RLAC permissions have no affect.

Is RLAC supported for use with DSE Graph?

No. Although permissions are shown and errors are not thrown with this statement:

```
GRANT SELECT ON 'custom_key' ROWS IN graph_keyspace.graph_table to 'alice';
```

Permissions are not enforced. RLAC is not supported for use with DSE Graph.

Granting access on rows in a table provides access to data in all graph keyspaces.

Encryption

How are encryption keys secured and managed?

Encryption keys can be managed off-server or locally:

- **KMIP** (Key Management Interoperability Protocol) encryption for encryption keys stored on another server and are cached locally in the memory heap when used by DSE.
- Use local encryption keys and secure using Linux permissions to restrict access.

Can the client-to-node encryption be configured as a two-way SSL?

Yes, although the client certificate DN is not used as a database user principal. Client-to-node encryption protects in-flight data from client machines to a database cluster using SSL (Secure Sockets Layer) and establishes a secure channel between the client and the coordinator node.

How is encryption of at-rest data supported?

Protects sensitive at-rest data using a local encryption key file or remotely stored and managed KMIP encryption key.

Can encryption keys be changed for a particular table?

Yes, by designating transparent data encryption (TDE) on a per table basis. Using encryption, your application can read and write to SSTables that use different encryption algorithms or use no encryption at all. Use a single ALTER TABLE statement to set encryption and compression.

Would encryption of EBS in AWS be a good replacement for using TDE, or is EBS better as a supplement to TDE (or neither)?

EBS encryption is another way to encrypt the data files. EBS encryption ensures encryption of audit logs, system logs, and the SSTable index files, which have partition keys in plain text if using TDE. In general, EBS encryption may be operationally simpler. Primarily, use TDE when full disk encryption is cost prohibitive or not feasible.

Is encryption supported at granular data layers? For example record-level or column- or field-level?

No. Designate transparent data encryption (TDE) only on a per table basis.

Auditing

Which user actions and events are logged?

When you configure audit logging, you can include or exclude categories of database activity such as querying or DML, see Filtering event categories.

Where are audit logs stored and who has access?

Audit logs can be written to either file system log files using logback, or to a database table. Audit events stored in database tables can be secured like any other database table using RBAC. File-based audit logs are stored per-node and can be secured with standard Linux file system permissions. See
Chapter 3. DataStax Enterprise security checklists

DSE Advanced Security is a feature suite that fortifies DataStax Enterprise (DSE) databases against potential harm due to deliberate attack or user error. It includes advanced mechanisms for authentication and authorization, encryption of data in-flight and at-rest, and data auditing. In addition, DataStax Enterprise is compatible with various partner security solutions to meet industry specific requirements and other advanced requirements.

DSE Advanced Security leverages enterprise standards to integrate cohesively with existing technology such as Active Directory (AD), Lightweight Directory Access Protocol (LDAP), Kerberos, Public Key Infrastructure (PKI), and Key Management Interoperability Protocol (KMIP).

DataStax Enterprise includes advanced data protection for enterprise-grade databases:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Database</th>
<th>Search</th>
<th>Analytics</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication (External LDAP or internal)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Partial</td>
</tr>
<tr>
<td>Kerberos authentication</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Authorization (RBAC)</td>
<td>Yes</td>
<td>Partial</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>Row-level permissions (RLAC)</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Client-to-node encryption</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Node-to-node encryption</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Transparent data encryption</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Data auditing</td>
<td>Yes</td>
<td>Yes</td>
<td>Partial</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Some DataStax drivers provide Kerberos support and SSL for client/server communication. Download drivers from DataStax Academy.

DSE database security checklist

Security for DataStax Enterprise database nodes:

- **Authentication:**
  
  Limit connections to the database to only known users. DSE supports user validation with the following authentication methods:
  
  # Internal: Credentials stored in the internal database
  
  # LDAP: External LDAP service, such as Active Directory
  
  # Kerberos: MIT Kerberos tickets checked against an external Key Distribution Server (KDS)

  See Configuring DSE Unified Authentication.

  DSE Unified Authentication is only supported for database connections. To authenticate internode communication, such as gossip, use node-to-node SSL certificates.

- **Authorization:**

  Restrict access to database resources for authenticated users with role-based access control (RBAC). DSE supports role management using the following methods:

  # Internal database: 1-1 mapping of user name or principal name to roles
DataStax Enterprise security checklists

# LDAP: 1-many mapping, where users are assigned all roles that match groups they are members of in LDAP

DataStax only supports RBAC with authentication enabled. See Setting up logins and users and Assigning permissions.

- **Audit activity:**
  Log and monitor activity for database resources, see Setting up database auditing.

- **Transparent data encryption (TDE):**
  Protect data at-rest. DSE provides encryption for sensitive data by encrypting:
  
  # Entire tables (except for partition keys which are always stored in plain text)
  # SSTables containing data, including system tables (such as system.batchlog and system.paxos)
  # Search indexes
  # File-based Hints (in DSE 5.0 and later)
  # Commit logs
  # Sensitive properties in dse.yaml and cassandra.yaml

  Encrypt data using an external KMIP or local service, see About Transparent Data Encryption.

- **Encrypt data in-flight using SSL**
  Secure communication between clients and the database and between nodes in a cluster, see Configuring SSL.

### DSE Search security checklist

DataStax Enterprise supports secure enterprise search. DataStax Enterprise security checklists summarize the security features of DSE Search and other integrated components.

- **Authentication**
  DataStax recommends using Kerberos authentication with the Solr Admin UI and when running commands with cURL using the SolrJ API.
  
  # To authenticate DSE Search clients with Kerberos authentication, use Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO).
  
  # To use the SolrJ API against DSE Search clusters with Kerberos authentication, client applications must use the SolrJ-Auth library and the DataStax Enterprise SolrJ component as described in the solrj-auth-README.md file.
  
  # Define Accessing search indexes from Solr Admin UI (deprecated).
  
  # Perform index management tasks with the CQL shell using Enabling DSE Unified Authentication.

- **Authorization**
  Use role-based access control (RBAC) for authenticated users to provide search index related permissions, see Setting up logins and users, Controlling access to search indexes, and Accessing search indexes from Solr Admin UI (deprecated). Setting row-level permissions with row-level access control (RLAC) is not supported for use with DSE Search or DSE Graph.

- **Audit activity**
  Log and monitor activity for database resources, see Setting up database auditing.

- **Transparent Data Encryption (TDE)**
DataStax Enterprise security checklists

Protect data at-rest. DSE provides KMIP or local encryption for sensitive data in search indexes, see Encrypting Search indexes.

- **Encrypt data in-flight using SSL**
  Encrypt connections using SSL between HTTP clients and CQL shell to with client-to-node encryption on the DSE Search node. See Configuring SSL.

  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.

Security configuration on a DSE Search node is managed by the DataStax Enterprise configuration. Additional configuration is not required for Tomcat and Solr. Change your web.xml or server.xml files only for custom advanced setups.

DSE Search security features have the following limitations:

- **TDE:** Cached data is not encrypted. Encryption occurs only in the DSE database and impacts performance slightly.
- **Authorization:** Permissions apply only to CQL requests, such as search index management and access to data stored in the database. Permissions do not apply to search file resources such as the cache and index configuration.
- **Setting row-level permissions** with row-level access control (RLAC) is not supported for use with DSE Search.
- **HTTP access to the DSE Search data** is protected using SSL (client-to-node encryption). Node-to-node encryption using SSL protects internal Solr communication.

DSE Analytics security checklist

DataStax recommends the following security practices:

- Enable client-to-node encryption using SSL.
- Spark ports for internode communications should run within a secured network without exposure to outside traffic.

Secure DataStax Enterprise Analytics nodes as follows:

- **Authentication:**
  
  1. Distinct secrets for internode and per application, see Configuring Spark nodes.
  2. Native authentication for users of each application executor (run as) and isolation of related data, see Configuring Spark nodes.
  3. Spark UI internal or LDAP authentication, see Monitoring Spark with the web interface.
     1. Internal and LDAP: For DataStax Enterprise Spark applications and tools, use the Spark authentication commands to provide the authentication credentials, see Running spark-submit job with internal authentication.
     2. Kerberos: Defining a Kerberos scheme applies to connecting Spark to DSE database, not authenticating Spark components between each other. The Spark Web UI is not secured, so some parameters passed to the executor in the command line might be visible. However, the DSE
username, password, and delegation token are hidden. By default, when Kerberos is the only authentication scheme, the Spark UI is inaccessible, so UI authorization must be disabled.

- **Authorization:**
  Data pulled from the database for Spark jobs and access control for Spark application submissions is protected by role-based access control (RBAC). The user running the request must have permission to access the data through their role assignment.

  No authorization for the Spark UI master and workers is available.

- **Auditing:**
  
  # Analytic operations performed in Spark are recorded to the Spark Event log, to enable see Configuring Spark logging options.

  # CQL requests are recorded in the database logs, see Setting up database auditing.

- **Transparent Data Encryption (TDE):**
  TDE applies only to data stored in the database. DSE does not support encrypting data that is used by Spark and stored in DSEFS or local temporary directories.

- **Encrypt data in-flight using SSL, TLS, or SASL:**
  
  **SSL/TLS:** Client-to-node encryption protects data in flight for the Spark Executor to DSE database connections by establishing a secure channel between the client and the coordinator node. SSL does not require setting up a shared authentication service. You need to prepare server certificates and enable client-to-node SSL.

  **SASL:** Spark internode and client-to-cluster communication can be encrypted using the SASL Digest-MD5 mechanism for mutual authentication and encryption. SASL encryption is also available for communicating among Spark driver, Spark executors, and the external shuffle service (ExternalShuffleService). See Securing Spark connections for details.

### DSE Graph security checklist

DataStax Enterprise supports secure enterprise graph-database operations. DSE Graph data is completely or partially secured by using DataStax Enterprise security features:

- **Authentication:**
  Allow only authenticated users to access DSE Graph data by enabling DSE Unified Authentication on the transactional database and configure credentials in the DSE Graph remote.yaml, see Using DSE Graph and Gremlin console with Kerberos.

- **Authorization:**
  Limit access to graph data by defining roles for DSE Graph keyspaces and tables, see Controlling access to Graph keyspaces.

  RBAC does not apply to cached data. Setting row-level permissions with row-level access control (RLAC) is not supported for use with DSE Search or DSE Graph.

  Grant execute permissions for the DseGraphRpc object to the defined roles.

- **Audit activity:**
  Log and monitor activity for DSE Graph related database resources, see Setting up database auditing.

- **Transparent Data Encryption:**
  Encrypt data in DSE Graph index tables, see Transparent data encryption

  Cached data is not encrypted. Encryption may slightly impact performance.
DataStax Enterprise security checklists

• **Encrypted database connections using SSL:**
  Encrypt inflight DSE Graph data. Enable SSL client-to-node encryption on the DSE Graph node by setting the `client_encryption_options` in the `cassandra.yaml` file, see Client-to-node encryption.

• **Graph sandbox:**
  Enabled by default, the Graph sandbox can be configured to allow or disallow execution of Java packages, superclasses, and types, see Graph sandbox.

DSE has the following limitations with Graph authorization:

• Limited, as Gremlin queries are not distinguished between query types like CQL.
• Permissions are enforced on a *per vertex label* and registered through CQL at the table level, using individual permissions using CQL.
Securing DataStax Enterprise ports

All network security starts with strict and proper firewall rules on interfaces that are exposed to the internet, allowing only the absolute minimum traffic in or out of the internal network. Firewall security is especially important when running your infrastructure in a public cloud. Wherever you host your clusters, DataStax strongly recommends using a firewall on all nodes in your cluster.

Begin with a restrictive configuration that blocks all traffic except SSH. Then, open up the following ports in compliance with your security requirements to allow communication between the nodes. If these ports are not opened, the node acts as a standalone database server rather than joining the cluster when you start DataStax Enterprise (DSE) on a node.

If the cluster uses SSL only, close any non-SSL ports that have dedicated SSL ports. To ensure communication is not disabled to any non-SSL clients, DataStax recommends testing the configuration in a staging environment before enabling the firewall in production environments.

Do not restrict traffic between DSE Analytics nodes. Traffic between DSE Analytics nodes must be unrestricted to allow communication between DSE Spark Master and Worker nodes.

1. Open the following ports:

<table>
<thead>
<tr>
<th>Port</th>
<th>Service</th>
<th>Configurable in</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Public facing ports</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>SSH (default)</td>
<td>See your OS documentation on sshd.</td>
</tr>
<tr>
<td></td>
<td>DataStax Enterprise public ports</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(random) Spark port for the driver to listen on. Used for communicating with the executors and the standalone Master. To explicitly set the port, set the spark.driver.port property in the Spark driver. If an application is already using the designated port, it will increment the port number up to the setting of the spark.port.maxRetries property. For example, if spark.port.port is set to 11000 and spark.port.maxRetries is set to 10, it will attempt to bind to port 11000. If that fails it will increment the port number and retry, stopping at port 11010.</td>
<td>Ensure traffic is unrestricted between DSE Analytics nodes.</td>
</tr>
<tr>
<td></td>
<td>(random) Spark port for all block managers to listen on. These ports exist on both the driver and the executable. To explicitly set the port, set the spark.blockManager.port property. If an application is already using the designated port, it will increment the port number up to the setting of the spark.port.maxRetries property. For example, if spark.blockManager.port is set to 11000 and spark.port.maxRetries is set to 10, it will attempt to bind to port 11000. If that fails it will increment the port number and retry, stopping at port 11010.</td>
<td>Ensure traffic is unrestricted between DSE Analytics nodes.</td>
</tr>
<tr>
<td></td>
<td>(local hostname) Spark port for communicating with the executors and the standalone Master.</td>
<td>Ensure traffic is unrestricted between DSE Analytics nodes.</td>
</tr>
<tr>
<td>4040</td>
<td>Spark application web site port. If an application is already using the designated port, it will increment the port number up to the setting of the spark.port.maxRetries property. For example, if spark.port.maxRetries is set to 10, it will attempt to bind to port 4041, and repeat until it reaches port 4050.</td>
<td></td>
</tr>
<tr>
<td>5598, 5599</td>
<td>Public/internode ports for DSE File System (DSEFS) clients.</td>
<td>dse.yaml</td>
</tr>
<tr>
<td>7080</td>
<td>Spark Master console port.</td>
<td>spark-env.sh</td>
</tr>
</tbody>
</table>
## Securing the environment

<table>
<thead>
<tr>
<th>Port</th>
<th>Service</th>
<th>Configurable in</th>
</tr>
</thead>
<tbody>
<tr>
<td>7081</td>
<td>Spark Worker web site port.</td>
<td>spark-env.sh</td>
</tr>
<tr>
<td>8182</td>
<td>The gremlin server port for DSE Graph.</td>
<td>See Graph configuration.</td>
</tr>
<tr>
<td>8983</td>
<td>DSE Search (Solr) port and Demo applications web site port (Portfolio, Search, Search log, Weather Sensors)</td>
<td></td>
</tr>
<tr>
<td>8090</td>
<td>Spark Jobserver REST API port.</td>
<td>See Spark Jobserver.</td>
</tr>
<tr>
<td>9042</td>
<td>DSE database native clients port. Enabling native transport encryption in client_encryption_options provides the option to use encryption for the standard port, or to use a dedicated port in addition to the unencrypted native_transport_port. When SSL is enabled, port 9142 is used by native clients instead.</td>
<td>cassandra.yaml</td>
</tr>
<tr>
<td>9091</td>
<td>The DataStax Studio server port.</td>
<td>See DataStax Studio documentation. Configure in dse_studio_install_dir/configuration.yaml.</td>
</tr>
<tr>
<td>9077</td>
<td>AlwaysOn SQL WebUI port.</td>
<td>See Configuring AlwaysOn SQL.</td>
</tr>
<tr>
<td>9142</td>
<td>DSE client port when SSL is enabled. Enabling client encryption and keeping native_transport_port_ssl disabled will use encryption for native_transport_port. Setting native_transport_port_ssl to a different value from native_transport_port will use encryption for native_transport_port_ssl while keeping native_transport_port unencrypted.</td>
<td>See Configuring SSL for client-to-node connections.</td>
</tr>
<tr>
<td>9999</td>
<td>Spark Jobserver JMX port. Required only if Spark Jobserver is running and remote access to JMX is required.</td>
<td></td>
</tr>
<tr>
<td>18080</td>
<td>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.</td>
<td>See Spark history server.</td>
</tr>
</tbody>
</table>

**OpsCenter public ports**

<table>
<thead>
<tr>
<th>Port</th>
<th>Service</th>
<th>Configurable in</th>
</tr>
</thead>
<tbody>
<tr>
<td>8888</td>
<td>OpsCenter web site port. The opscentered daemon listens on this port for HTTP requests coming directly from the browser. See OpsCenter ports reference.</td>
<td>opscentered.conf</td>
</tr>
</tbody>
</table>

**Inter-node ports**

**DSE database inter-node communication ports**

<table>
<thead>
<tr>
<th>Port</th>
<th>Service</th>
<th>Configurable in</th>
</tr>
</thead>
<tbody>
<tr>
<td>5599</td>
<td>Private port for DSEFS inter-node communication port. Must not be visible outside of the cluster.</td>
<td>dse.yaml</td>
</tr>
<tr>
<td>7000</td>
<td>DSE inter-node cluster communication port.</td>
<td>cassandra.yaml</td>
</tr>
<tr>
<td>7001</td>
<td>DSE SSL inter-node cluster communication port.</td>
<td>cassandra.yaml</td>
</tr>
<tr>
<td>7199</td>
<td>DSE JMX metrics monitoring port. DataStax recommends allowing connections only from the local node. Configure SSL and JMX authentication when allowing connections from other nodes.</td>
<td>cassandra-env.sh</td>
</tr>
<tr>
<td>1024 - 65355</td>
<td>JMX reconnection/loopback ports. See the description for port 7199.</td>
<td>See JMX options in Tuning Java Virtual Machine.</td>
</tr>
</tbody>
</table>

**DataStax Enterprise inter-node ports**

<table>
<thead>
<tr>
<th>Port</th>
<th>Service</th>
<th>Configurable in</th>
</tr>
</thead>
<tbody>
<tr>
<td>7077</td>
<td>Spark Master inter-node communication port.</td>
<td>dse.yaml</td>
</tr>
<tr>
<td>8609</td>
<td>Port for inter-node messaging service.</td>
<td>dse.yaml</td>
</tr>
<tr>
<td>10000</td>
<td>Spark SQL Thrift server port. Only required if Spark SQL Thrift server is running.</td>
<td>Set with the -p option with the Spark SQL Thrift server.</td>
</tr>
</tbody>
</table>
Securing the environment

Securing the TMP directory

Enterprise security policies often require mounting the /tmp directory with the no execution (noexec) flag. By default, Cassandra Java Native Access (JNA) is mapped to /tmp; JNA requires an executable directory to start. Remap JNA to a different (executable) directory and change permissions to allow full access to the user that runs the DSE service.

1. Create a directory and grant full access to the Cassandra user.

2. Map the JNA temporary directory; in jvm.options file, add the following flag with the path to the new directory:

   ```
   -Djna.tmpdir=JNA_startup_directory
   ```

3. Restart DataStax Enterprise.

   If DataStax Enterprise fails to start because the JNA startup directory is not available, the following error may appear in the system log:

   ```
   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.
   ```

   See JNA fails to initialize.
Configuring DSE Unified Authentication

Detailed steps to set up authentication and authorization in a DataStax Enterprise environment.

About DSE Unified Authentication

DSE Unified Authentication facilitates connectivity to three primary backend authentication and authorization services. DSE Unified Authentication uses the following services:

- **DSE Authenticator** supports validating user identity against any of the following authentication schemes:
  
  # Internal: Connections provide credentials for a role that has an internally stored password. No additional configuration is required. See [Setting up logins and users](#).
  
  # LDAP: Connections provide LDAP credentials. DSE passes the credentials for verification to LDAP. See [Defining an LDAP scheme](#).
  
  # Kerberos: Connections provide a Kerberos ticket. DSE is configured as a Service Principal (see [Setting up Kerberos](#)) and passes the tickets to KDS for verification. See [Defining a Kerberos scheme](#)

  When a connection request specifies an authentication scheme, DSE Authenticator validates the user against the selected scheme first. If no scheme is specified in the connection request or the validation fails, DSE Authenticator will try the `default_scheme` and then each scheme defined in `other_schemes`.

  It is possible to authenticate users without implementing access control using the DSE Authenticator; however, authentication is required for authorization and role management.

- **DSE Role Manager** assigns roles using one of the following modes:
  
  # Internal: 1-1 mapping using an internally stored password. Requires a role for each account.
  
  # LDAP: 1-many mapping. Assigns DSE roles that match the users' LDAP groups.

    For LDAP role management, DSE disables role nesting; you cannot use `GRANT` to assign a role to another role.

- **DSE Authorizer** analyzes the request against the role permissions on each affected resource before allowing the request to be executed.

  Set and remove permissions on database resources with the CQL commands `GRANT` and `REVOKE`.

  Enable support for row-level access control, which allow permissions to be granted by filtering on a partition column, by setting `authorization_options row_level_access_control` to true, see [Enabling DSE Unified Authentication](#), see [Setting up Row Level Access Control (RLAC)](#).

Steps for new deployment

High-level steps for implementing DSE Unified Authentication in a new deployment.

To implement authentication and authorization in an already established DSE environment, additional precautions and steps are required. See [Steps for production environments](#).

To configure DSE Unified Authentication:
1. Ensure that required data for logins and permission management are accessible and in all datacenters. See Configuring the security keyspaces replication factors.

2. Configure the system settings. See Enabling DSE Unified Authentication.

3. Configuring authentication and authorization methods (schemes):
   - Internally stored passwords. No additional configuration is required create roles with passwords as described in Setting up logins and users.
   - External LDAP. See Defining an LDAP scheme.
   - Kerberos. See Defining a Kerberos scheme.

4. Configuring JMX authentication: Requires changes to the cassandra-env.sh for nodetool and dsetool to run against an authentication enabled cluster.

5. Restart DSE. See Starting and stopping DataStax Enterprise.

   Nodes are vulnerable to malicious activity following the restart. Anybody can access the system using the default cassandra account with password cassandra. DataStax recommends isolating the cluster until after disabling the cassandra account.

6. Set up your own root account and disable or drop the default, cassandra account. See Adding a superuser login.

   Using the default cassandra account may impact performance, because all requests including login execute with consistency level QUORUM. DataStax recommends only using this account to create your root account.

7. Create roles that map to users in the configured schemes and grant permission to allow users access to database resources, such as keyspaces and tables. See Setting up logins and users.

   - Use the latest DataStax certified drivers in all applications connecting to DSE Unified Authentication-enabled transactional nodes. DSE drivers support all the features of the Cassandra drivers and provide additional support for multiple authentication methods as well as externally managed roles assignment. See DataStax drivers.
   - Spark component limitations: DataStax Enterprise provides internal authentication support for connecting Spark to DSE transactional nodes, not for authenticating between Spark components.

What's next:
After enabling authentication and authorization, run tools by supplying credentials:

- Providing credentials for CQLSH
- Using DSE Graph and Gremlin console with Kerberos
- Authenticating using DSE utilities
- Providing credentials with nodetool
- Authenticating using DSE utilities

Steps for production environments
To enable access control without downtime, use transitional modes to temporarily support applications and users that do not have accounts without interrupting services.
Authentication and authorization

1. Update all applications to provide credentials and ensure that they use the latest DSE driver, see DataStax drivers. When using authentication.transitional_mode, you can provide a blank username and password to log in with the anonymous role.

   When authentication and authorization are disabled, the credentials portion of the connection request is ignored. After authentication is enabled all connections must provide credentials.

2. Change the replication factor for the security keyspaces, see Configuring the security keyspaces replication factors.

3. Run a full repair of the system_auth and dse_auth keyspace.

4. On each node, enable authentication and authorization in transitional mode:
   
   - Set authentication_options transitional_mode to permissive.
   - Set authorization_options https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml__auth_transitional_mode to normal.

   See Enabling DSE Unified Authentication.

5. Turn on logging to allow verification that applications can authenticate. See Setting up database auditing.

6. Enable JMX authentication, see Controlling access to JMX MBeans.

7. Perform a rolling restart.

   The transactional nodes are vulnerable to malicious activity following the restart. Anybody can access the system using the default cassandra account with password cassandra. DataStax recommends isolating the cluster until after disabling the cassandra account.

8. After the restarts are complete, use cqlsh to replace the cassandra default account, see Adding a superuser login.

   Using the default cassandra account may impact performance, all requests including login execute with consistency level QUORUM. DataStax recommends only using this account to create your root account.

9. Switch to the new root account and configure roles and assign permissions, see Setting up logins and users.

10. Use the audit logs, or the KDC logs when using Kerberos, to verify that all applications can access the transactional nodes and have the permissions required to execute requests, see Log formats.

11. In the dse.yaml, set authentication_options and authorization_options transitional_mode to disabled.

12. Perform a rolling restart.

Configuring the security keyspaces replication factors

Configure the replication factors appropriately for DSE security in production environments. Change the replication class to NetworkTopologyStrategy and set the replication factor between 3 to 5 for the following security keyspaces:

- system_auth
- dse_security

Default replication factors

The default replication factor for the system_auth and dse_security keyspaces is 1.
To avoid data loss, each of these must be updated in production environments. DataStax recommends changing
the replication factor before enabling authentication. DSE uses a consistency level of \texttt{LOCAL\_ONE} for all security
keyspaces queries, except when using the \texttt{cassandra} role. For the \texttt{cassandra} role, DSE uses the consistency
level \texttt{QUORUM}. Only use the \texttt{cassandra} role to login and create your own full access account; then drop the
\texttt{cassandra} role.

Increase the RF before enabling DSE authentication. The default login account, \texttt{cassandra}, executes all
requests with \texttt{QUORUM} and may fail with an RF of 1.

\section*{Recommended replication factors}

Determine the appropriate RF based on your failure tolerance and the size of your deployment.

- \texttt{system\_auth}: Required for each log in and for every action that affects a database object. Once a user logs
  in, their credentials, roles, and permissions are cached for a period set in the \texttt{cassandra.yaml}. Refer to
  Security properties, which contains LDAP, native authentication, and authorization related data. When the
  keyspace is unavailable, logins and actions may fail. When located on a node in another datacenter, this
  scenario may cause delays that also can lead to failures. The keyspace tables are relatively small.

  DataStax recommends using a replication factor of 3, 4, or 5 per datacenter.

  DSE caches security data. For information about adjusting the cache interval, see Security properties.

- \texttt{dse\_security}: Required for each log in and for related DSE services. Contains DSE Analytic (Spark), DSE
  Client digest tokens, and other Kerberos related data. Less critical for pure database activities.

  DataStax recommends using a replication factor of 3, 4, or 5 per datacenter.

  Never set the replication factor greater than the number of nodes in the datacenter.

To change the replication factors (RF) of the security keyspaces:

\begin{enumerate}
\item Change the \texttt{system\_auth} keyspace RF:

\begin{verbatim}
ALTER KEYSPACE system_auth
    WITH REPLICA\_TION= \{"class" : 'NetworkTopo\_logyStrategy',
    'data\_center\_name' : \texttt{N},
    'data\_center\_name' : \texttt{N};
\end{verbatim}

    Every time you add or remove a datacenter, you must manually reconfigure the \texttt{system\_auth} keyspace.

\item Change the \texttt{dse\_security} keyspace RF:

\begin{verbatim}
ALTER KEYSPACE dse\_security
    WITH REPLICA\_TION= \{"class" : 'NetworkTopo\_logyStrategy',
    'data\_center\_name' : \texttt{N},
    'data\_center\_name' : \texttt{N};
\end{verbatim}

    Every time you add or remove a datacenter, you must manually reconfigure the \texttt{dse\_security}
    keyspace. If DataStax Enterprise or Spark security options are enabled on the cluster, you must also
    increase the replication factor for the \texttt{dse\_leases} keyspace across all logical datacenters.

\item Run the \texttt{nodetool repair} on the security keyspaces.

\begin{verbatim}
nodetool repair --full system_auth
\end{verbatim}
\end{enumerate}
After changing the replication strategy, you must run `nodetool repair` with the `--full` option.

**Setting up Kerberos**

DataStax Enterprise (DSE) authentication with Kerberos protocol uses tickets to prove identity for users and applications without the need to pass credentials, only a Kerberos ticket. This enables a deployment to have fewer attack vectors and can eliminate the need to embed passwords in configuration files.

Also see Use Kerberos authentication for DSE Search in production and DSE Analytics security checklist.

**Kerberos guidelines**

In order to allow DSE clients to authenticate using Kerberos tickets, ensure that your environment meets the DataStax Enterprise requirements and recommendations to establish DSE as a Service Principal. When DSE Authorizer is also enabled, you can map User Principal (including Realm) to roles for database access control.

The [Kerberos Tutorial](#) provides step-by-step instructions on establishing client-DSE Kerberos authentication in DataStax Enterprise and OpsCenter. Kerberos is a network authentication protocol designed to provide strong authentication for client/server applications using secret-key cryptography. For information on installing and setting up Kerberos, see the [MIT Kerberos Consortium](#) documentation.

**Kerberos guidelines**

The following general guidelines apply to setting up Kerberos and configuring DataStax Enterprise as a Kerberos Service Principal:

- Familiarity with Kerberos and understand how to use these commands: `kinit`, `klist`, and `kdestroy`.
- Write permission on the `cassandra.yaml` and `dse.yaml` files and access to all nodes in the cluster.
- Ability to restart the cluster after the configuration is complete.
- Fully operational Kerberos Key Distribution Center (KDC) servers. DataStax recommends using multiple KDC servers, with a single primary or administration KDC server and all others as secondary.
  
  For security purposes, DataStax recommends running the KDC on a secured server outside the DSE cluster. Never install KDC servers on DataStax Enterprise nodes.
- Proper security for the KDC servers, such as firewalls and physically protecting the KDC machines.
- Privilege to manage Kerberos principals and export keytab files (or access to a KDC administrator that does).
- Always secure the DSE keytab files; the DSE service account should own the keytab file and have read/write access (`chmod 600`).
- When using Oracle Java 8, DataStax recommends using the latest version. The minimum version must be at least 1.8.0_151.

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.

**Using Kerberos with DataStax Enterprise**

The following topics provide information on using Kerberos with various DataStax Enterprise features and other software:

- Connecting to SSL-enabled nodes using `cqlsh`
- Using `dsetool` with Kerberos enabled cluster
Authentication and authorization

• 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, see Setting up database auditing.

DataStax recommends using either Kerberos connection encryption (qop=auth-conf) or SSL. Enabling both double encrypts the connection, which waste resources.

Enabling JCE Unlimited

DataStax recommends enabling Java Cryptography Extension (JCE) Unlimited to ensure support for all encryption algorithms, especially AES-256 for Kerberos and SSL when using Oracle Java.

Prior to JDK 1.8.0_151 (8u151), you had to download and install the JCE jurisdiction policy files separately. Those steps are unnecessary in 8u151 and later JDK releases. To enable JCE Unlimited use the crypto.policy Security property introduced in JDK 8u151, as noted in the New Features section of the Oracle JDK 1.8.0_151 Release Notes.

Enabling JCE Unlimited Cryptography

To enable JCE Unlimited Cryptography in environments with JDK 8u151 or later, set the following Security property in the java.security file:

```
crypto.policy-unlimited
```

When set in java.security, or when declared dynamically using the Security.setProperty() call before the JCE framework has been initialized, the unlimited setting is used by the JDK.

Starting in JDK 8u161, JCE Unlimited is enabled by default. Refer to the Release Notes for JDK 8u161.

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.

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 cipher due to export restrictions to certain countries. To use AES-256 with Oracle Java, install the JCE Unlimited Strength Jurisdiction Policy Files.

If your environment uses a JDK version prior to 8u151, which released in October 2017, refer to the download and install steps in the following sections.

Installing JCE Unlimited for pre-8u151 JDK environments on RHEL-based systems

If your JDK on RHEL-based systems must use a pre-8u151 JDK:

1. Install the JCE using the Oracle JAR:
   a. Download the Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files from Oracle Java SE download page under Additional Resources.
   b. Unzip the downloaded file.
   c. Copy local_policy.jar and US_export_policy.jar to the $JAVA_HOME/jre/lib/security directory to overwrite the existing JARS.
   d. Check permissions of installed files so they are readable by all users.

Installing JCE Unlimited for pre-8u151 JDK environments on Debian-based systems

If your JDK on Debian-based systems must use a pre-8u151 JDK:
Authentication and authorization

Install JCE using webupd8 PPA repository:

```
$ sudo apt-get install oracle-java8-unlimited-jce-policy
```

If the repository is not available in your environment, add it and then install. For example:

```
$ sudo add-apt-repository ppa:webupd8team/java
```

Removing AES-256

If you do not use AES-256, you must remove the AES-256 settings as an allowed cipher for each Kerberos principal and then regenerate the keys for the krbtgt principal.

**Prerequisites:** These methods require Kerberos 5-1.2 on the KDC.

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.

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's next: Preparing DSE nodes for Kerberos

**Preparing DSE nodes for Kerberos**

Use these instructions as guidelines for installing the Kerberos client libraries on DSE nodes, verifying DNS entry, and system time settings. 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.

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

Complete the following prerequisites:

- All KDS requirements have been met, see Kerberos guidelines.
- If using Oracle Java 8, DataStax recommends using the latest version, however the minimum version is 1.8.0_151
- Each node has the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files enabled. Refer to Enabling JCE Unlimited. Starting in JDK 8u161, JCE Unlimited is enabled by default. Refer to the Release Notes for JDK 8u161.
  - If you are not using the JCE Unlimited Strength Jurisdiction Policy, make sure that your ticket granting principal does not use AES-256.
Verifying the node hostname and time settings

For Kerberos the hostname is used for the Service Principal name, therefore the hostname must resolve to the correct IP address. (See Principal names and DNS. Kerberos authentication is sensitive to system time, manually set system clocks may cause issues. Ensure that node time is set to a well-known NTP.

1. To verify the hostname:

```bash
nslookup $(hostname --fqdn) && hostname --fqdn && hostname -i
```

Server: 10.200.1.10
Address: 10.200.1.10#53

Name: node.example.com
Address: 10.200.182.183

node.example.com
10.200.182.183

2. On each node, confirm that NTP is configured and running:

```bash
$ ntpq -p
```

<table>
<thead>
<tr>
<th>remote</th>
<th>refid</th>
<th>st</th>
<th>t</th>
<th>when</th>
<th>poll</th>
<th>reach</th>
<th>delay</th>
<th>offset</th>
<th>jitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>*l1506-17.member</td>
<td>209.51.161.238</td>
<td>2</td>
<td>u</td>
<td>331</td>
<td>1024</td>
<td>377</td>
<td>80.289</td>
<td>1.384</td>
<td>1.842</td>
</tr>
<tr>
<td>-tock.eoni.com</td>
<td>216.228.192.69</td>
<td>2</td>
<td>u</td>
<td>410</td>
<td>1024</td>
<td>377</td>
<td>53.812</td>
<td>1.706</td>
<td>34.692</td>
</tr>
<tr>
<td>+time01.muskegon</td>
<td>64.113.32.5</td>
<td>2</td>
<td>u</td>
<td>402</td>
<td>1024</td>
<td>377</td>
<td>59.378</td>
<td>-1.635</td>
<td>1.840</td>
</tr>
<tr>
<td>-time-a.nist.gov</td>
<td>.ACTS.</td>
<td>1</td>
<td>u</td>
<td>746</td>
<td>1024</td>
<td>151</td>
<td>132.832</td>
<td>26.931</td>
<td>55.018</td>
</tr>
<tr>
<td>+golem.canonical</td>
<td>131.188.3.220</td>
<td>2</td>
<td>u</td>
<td>994</td>
<td>1024</td>
<td>377</td>
<td>144.080</td>
<td>-1.732</td>
<td>20.072</td>
</tr>
</tbody>
</table>

Configuring Kerberos connection information for clients

Install Kerberos clients and configure the Kerberos realm and connection details.

**Prerequisites:** From your organizations Kerberos administrator, get the krb5.conf configured for domain that contains the DataStax cluster.

1. Install Kerberos client software on each node using one of the following methods:
   - Tarball-based systems:
     ```bash
     wget http://web.mit.edu/Kerberos/dist/krb5/1.17/krb5-1.17.tar.gz
     ```
   - RHEL-based systems:
     ```bash
     sudo yum install krb5-workstation krb5-libs krb5-pkinit-openssl
     ```
   - Debian-based systems:
     ```bash
     sudo apt-get install krb5-user krb5-config krb5-pkinit
     ```

2. (Optional) In the krb5.conf, verify that the libdefaults settings have DNS and realm lookup disabled.

```bash
[libdefaults]
dns_lookup_kdc = false
```
Authentication and authorization

dns_lookup_realm = false

DataStax recommends not using DNS lookup for KDC and REALM entries. Relying on DNS may negative impact performance and functionality.

3. Distribute the configuration file to each node using one of the following methods:

- **Default location** - Put the krb5.conf file in the /etc directory.
- **Custom location** - When the krb5.conf file is in a location other than default, provide the location using environment variable KRB5_CONFIG.

$ export KRB5_CONFIG="path_to_file"

Refer to MIT Kerberos documentation for full list of default paths.

Creating Kerberos Principals

Add service principals for each node in the DataStax Enterprise cluster.

Use kadmin to perform the following steps

1. Launch Kerberos admin shell with an administrator account that has add privileges.

   $ kadmin -p user_name/admin

2. For each node add two principals:

   - Add a service principal for the database using a random key value:

     addprinc -randkey service_name/FQDN

     where

     # service_name - Name for the DataStax Enterprise database principal, such as dse.
     # FQDN - Fully Qualified Domain Name of the host.

     See Principal names and DNS.

   - Add a service principal for HTTP using a random key value:

     addprinc -randkey HTTP/FQDN

     where FQDN - Fully Qualified Domain Name of the host.

3. Verify that the principals have been added by running the listprincs command within kadmin:

   kadmin: listprincs

   HTTP/node1.example.com@EXAMPLE.COM
   HTTP/node2.example.com@EXAMPLE.COM
   dse/node1.example.com@EXAMPLE.COM
   dse/node2.example.com@EXAMPLE.COM
Authentication and authorization

where node.*.example.com is the FQDN and EXAMPLE.COM is your Kerberos realm, which must be all uppercase.

Creating a Kerberos Keytab file

Save the principal credentials in a keytab file to obtain credentials and authenticate without entering a password each time.

1. Create a keytab file for each node and add the principals keys for each node:

   kadmin: ktadd -k keytabfilename dse/FQDN
   kadmin: ktadd -k keytabfilename HTTP/FQDN

   where ktadd -k creates or appends a key for the DSE service and HTTP principals.

   Example:

   kadmin: ktadd -k /tmp/node1.keytab dse/node1.example.com
   kadmin: ktadd -k /tmp/node1.keytab HTTP/node1.example.com
   kadmin: ktadd -k /tmp/node2.keytab dse/node2.example.com
   kadmin: ktadd -k /tmp/node2.keytab HTTP/node2.example.com

2. Use the klist command to view your principals in each keytab file:

   For example:

   
   sudo klist -e -kt /tmp/node1.keytab

   Keytab name: FILE:/tmp/node1.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 dse/node1FQDN@YOUR_REALM (des3-cbc-sha1)
   2 14/02/16 22:03 dse/node1FQDN@YOUR_REALM (arcfour-hmac)
   2 14/02/16 22:03 dse/node1FQDN@YOUR_REALM (des-hmac-sha1)
   2 14/02/16 22:03 dse/node1FQDN@YOUR_REALM (des-cbc-md5)

   where: -e displays the encryption type and -kt displays the keytab file and its timestamp.

3. Distribute keytab files from the KDC server to the nodes, to ease DSE Kerberos configuration ensure the files have the same name on each node:

   scp /tmp/node1.keytab node_admin@node_hostname:/etc/dse/dse.keytab

4. Change the permissions on dse.keytab so that only the dse_service_account user can read and write to the keytab file:

   sudo chown dse:dse /etc/dse/dse.keytab & & sudo chmod 600 /etc/dse/dse.keytab

Enabling DSE Unified Authentication

DSE Unified Authentication facilitates connectivity to three primary backend authentication and authorization services. DSE Unified Authentication uses the following services:
Authentication and authorization

- **DSE Authenticator**: Provides authentication using internal password authentication, LDAP pass-through authentication, and Kerberos authentication.
- **DSE Role Manager**: Assigns roles by mapping user names to role names or looks up the group membership in LDAP and maps the group names to role names.
- **DSE Authorizer**: Provides access to control for database objects.

By default, DSE Authenticator and DSE Authorizer are disabled. Authenticators other than DseAuthenticator are not supported.

OpsCenter also provides support for LDAP configuration, authenticating users.

**Prerequisites:**

Complete the following before enabling authentication:

- When configuring an external authentication method such as Kerberos or LDAP ensure that the service is active and available. DSE fails to start when an authentication scheme or role management mode is configured but not available.
- Configure the `system_auth` and `dse_security` keyspaces to use a replication factor of 3-5 for each datacenter, see Configuring the security keyspaces replication factors.
- When enabling authentication in an existing environment, upgrade drivers and configure applications to provide credentials. Consider using the transitional mode to allow connections using the anonymous role, see Steps for production environments for more details.

Apply the following updates to each node:

1. In the `cassandra.yaml` file, verify that DSE Unified Authentication and Authorization features are configured:
   
   a. Verify that authenticator is set to DseAuthenticator.
      ```yaml
      authenticator: com.datastax.bdp.cassandra.auth.DseAuthenticator
      ```
   
   b. Verify that authorizer is set to DseAuthorizer.
      ```yaml
      authorizer: com.datastax.bdp.cassandra.auth.DseAuthorizer
      ```
   
   c. Verify that role_manager is set to DseRoleManager.
      ```yaml
      role_manager: com.datastax.bdp.cassandra.auth.DseRoleManager
      ```
   
   d. To secure schema information, enable system_keyspaces_filtering. Users will only be able to see schema information for objects they have access permissions on.
      ```yaml
      system_keyspaces_filtering: true
      ```

2. In the `dse.yaml` file, configure the corresponding options:
   
   a. Configure the DseAuthenticator by uncommenting the `authentication_options` and changing the settings.
      ```yaml
      # authentication_options:
      #   enabled: false
      #   default_scheme: internal
      #   allow_digest_with_kerberos: true
      #   plain_text_without_ssl: warn
      ```
Authentication and authorization

Remove all pound signs (#) at the beginning of the line while preserving the spacing.

- Required settings. Enable DSE Authenticator and select a scheme by uncommenting and setting the values:

```yaml
#     transitional_mode: disabled
#     other_schemes:
#     scheme_permissions: false

#     allow_digest_with_kerberos: true
#     plain_text_without_ssl: warn
#     transitional_mode: disabled
#     other_schemes:
#     scheme_permissions: false
```

If you plan to use only LDAP or Kerberos, include the internal scheme in `other_schemes` to allow access to the default cassandra account and complete the initial set up.

**Table 1: Required authentication_options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Turns on authentication using the default scheme.</td>
</tr>
<tr>
<td>default_scheme</td>
<td>Specifies the authentication scheme when not defined in the connection:</td>
</tr>
<tr>
<td></td>
<td>- <code>internal</code> - Basic authentication using internal login role with password,</td>
</tr>
<tr>
<td></td>
<td>supply the role name and password as credentials. No additional</td>
</tr>
<tr>
<td></td>
<td>configuration required.</td>
</tr>
<tr>
<td></td>
<td>- <code>ldap</code> - Plain text authentication using pass-through LDAP authentication.</td>
</tr>
<tr>
<td></td>
<td>See <code>Defining an LDAP scheme</code>.</td>
</tr>
<tr>
<td></td>
<td>- <code>kerberos</code> - GSSAPI authentication using the Kerberos authenticator.</td>
</tr>
<tr>
<td></td>
<td>See <code>Defining a Kerberos scheme</code>.</td>
</tr>
</tbody>
</table>

- Optional settings:

```yaml
authentication_options:
  enabled: true
  default_scheme: internal
  allow_digest_with_kerberos: true
  plain_text_without_ssl: warn
  transitional_mode: disabled
  other_schemes:
  scheme_permissions: false
```

`scheme_permissions` require EXECUTE permission for the selected scheme. Do not enable this option until after configuring your own root account.

**Table 2: Optional authentication_options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>other_schemes</td>
<td>other_schemes</td>
</tr>
</tbody>
</table>
Authentication and authorization

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheme_permissions</td>
<td>Validate that the role mapped to user matches the authentication scheme. Grant the role permission to the scheme.</td>
</tr>
<tr>
<td>allow_digest_with_kerberos</td>
<td>Allow Kerberos digest-md5 authentication.</td>
</tr>
<tr>
<td>plain_text_without_ssl</td>
<td>Handling of plain text connection requests:</td>
</tr>
<tr>
<td></td>
<td># block - Block the request with an authentication error.</td>
</tr>
<tr>
<td></td>
<td># warn - Log a warning about the request but allow it to continue.</td>
</tr>
<tr>
<td></td>
<td># allow - Allow the request without any warning.</td>
</tr>
<tr>
<td>transitional_mode</td>
<td>Allow access to the database using the anonymous role:</td>
</tr>
<tr>
<td></td>
<td># disabled - Transitional mode is disabled. All connections must provide valid credentials and map to a login-enabled role.</td>
</tr>
<tr>
<td></td>
<td># permissive - Only super users are authenticated and logged in. All other authentication attempts are logged in as the anonymous user.</td>
</tr>
<tr>
<td></td>
<td># normal - Allow all connections that provide credentials. Maps all authenticated users to their role AND maps all other connections to anonymous.</td>
</tr>
<tr>
<td></td>
<td># strict - Allow only authenticated connections that map to a login-enabled role OR connections that provide a blank username and password as anonymous.</td>
</tr>
</tbody>
</table>

b. Configure the DSE Role Manager by uncommenting `role_management_options` and setting the mode:

```yaml
role_management_options:
  mode: internal
```

Remove all pound signs (#) at the beginning of the line while preserving the spacing.

**Table 3: Role Management Modes**

<table>
<thead>
<tr>
<th>scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal</td>
<td>Assign the user name supplied by the authenticator a role that matches the user name, 1 to 1 mapping.</td>
</tr>
<tr>
<td>ldap</td>
<td>Look up the user name in LDAP using the ldap scheme and get the group membership, assign all roles that match a group name, 1 to many mapping. When using Kerberos authentication, identify users by their email address in the LDAP search. The Kerberos Realm must match the domain in the email address.</td>
</tr>
</tbody>
</table>

c. Configure the DSE Authorizer by uncommenting the `authorization_options` and changing the settings.

```yaml
authorization_options:
  enabled: true
  transitional_mode: normal
  allow_row_level_security: true
```

Remove all pound signs (#) at the beginning of the line while preserving the spacing.

- Required. Set enabled to true.

  enabled
  
  Whether to use the DSE Authorizer for role-based access control (RBAC).

  # true - use the DSE Authorizer for role-based access control (RBAC)
Authentication and authorization

# false - do not use the Dse Authorizer

When not set, the default is false.
Default: commented out (false)

- Optional settings:

```
transitional_mode: normal
allow_row_level_security: true
```

**transitional_mode**
Allows the DSE Authorizer to operate in a temporary transitional mode during setup of authorization in a cluster. Set to one of the following values:

- # disabled - Transitional mode is disabled.
- # normal - Permissions can be passed to resources, but are not enforced.
- # strict - Permissions can be passed to resources, and are enforced on authenticated users. Permissions are not enforced against anonymous users.

Default: commented out (disabled)

**allow_row_level_security**
Whether to enable row-level access control (RLAC) permissions; use the same setting on all nodes.

- # true - use row-level security
- # false - do not use row-level

When not set, the default is false.
Default: commented out (false)

3. Configure selected authentication scheme options:

- Defining an LDAP scheme
- Defining a Kerberos scheme

In order for DSE to start up, the external service referenced in the kerberos_options and/or ldap_options must be accessible. If you are not using Kerberos-based authentication, comment out the kerberos_options.

4. Set up JMX authentication to allow `nodetool` and `dsetool` operations, see Configuring JMX authentication.

5. Restart DSE, see Starting DataStax Enterprise as a service or Starting DataStax Enterprise as a standalone process.

What's next: After restarting DSE, log into CQL shell and complete the set up:

- Adding a superuser login
- Create roles and set up permissions, see Setting up logins and users

**Defining a Kerberos scheme**

Prerequisites:
Completely set up Kerberos for DSE nodes before turning on Kerberos authentication. When switching authentication methods, or enabling authentication for the first time in a production environment, DataStax recommends setting up applications to use Kerberos tickets before restricting access to only authenticated
Authentication and authorization

connections. When DSE Authenticator is disabled, the credentials portion of the connection request is ignored. Therefore, you can pass Kerberos tickets to DSE before implementing authentication in the environment.

Change the replication strategy and default replication factor for the system_auth and dse_security keyspaces, see Configuring system_auth keyspace replication

How to add the Kerberos authenticator to cassandra.yaml and Kerberos options to dse.yaml.

1. On each node, edit the cassandra.yaml file to set the authenticator to the DSE Authenticator.

   authenticator: com.datastax.bdp.cassandra.auth.DseAuthenticator

2. Set the native_transport_address and listen_address to the IP address. Do not use localhost or the hostname.

   native_transport_address: 100.200.182.1
   listen_address: 100.200.182.1

3. On each node, edit the dse.yaml file:

   a. Set Kerberos as default or other scheme in the authentication options:

   authentication_options:
      enabled: false
      default_scheme: kerberos
      other_schemes:
        - internal
          scheme_permissions: true
          allow_digest_with_kerberos: true
          plain_text_without_ssl: warn
          transitional_mode: disabled

   When initially enabling authentication, specify the internal scheme as the default or other. After restarting DSE, to establish roles requires using the internal default cassandra account.

Table 4: authentication_options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml_auth_enabled">https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml_auth_enabled</a></td>
<td>Turns on authentication using the default scheme.</td>
</tr>
<tr>
<td>default_scheme</td>
<td>Specifies the authentication scheme when not defined in the connection:</td>
</tr>
<tr>
<td></td>
<td>• internal - Plain text authentication using internal login role with password, supply the role name and password as credentials. No additional configuration required.</td>
</tr>
<tr>
<td></td>
<td>• ldap - Plain text authentication using pass-through LDAP authentication. See Defining an LDAP scheme.</td>
</tr>
<tr>
<td></td>
<td>• kerberos - GSSAPI authentication using the Kerberos authenticator. See Defining a Kerberos scheme.</td>
</tr>
<tr>
<td>other_schemes</td>
<td></td>
</tr>
<tr>
<td>scheme_permissions</td>
<td>Validate that the role mapped to user matches the authentication scheme. Grant the role permission to the scheme.</td>
</tr>
<tr>
<td>allow_digest_with_kerberos</td>
<td>Allow Kerberos digest-md5 authentication.</td>
</tr>
</tbody>
</table>
b. Configure the Kerberos options.

The options are located in the `kerberos_options` section.

Example:

```yaml
kerberos_options:
  keytab: /etc/dse/dse.keytab
  service_principal: dse/_HOST@REALM
  http_principal: HTTP/_HOST@REALM
  qop: auth
```

Table 5: `kerberos_options`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keytab</td>
<td>The keytab file must contain the credentials for both of the fully resolved principal names, which replace <code>_HOST</code> 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.</td>
</tr>
</tbody>
</table>
| service_principal | Sets the principals name for the DSE database and DSE Search (Solr) processes. Use the form `dse/_HOST@REALM`, where dse is the service name. 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:  
  - dse.yaml file  
  - keytab  
  - cqlshrc file (where it is separated into the service/hostname) |
<p>| http_principal | The http_principal is used by the Tomcat application container to run DSE Search. The Tomcat web server uses the GSSAPI 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. |</p>
<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
</table>
| 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:  
  - 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. When adding a Kerberos scheme to an authentication enabled cluster, configure Kerberos roles before restarting DSE, see Setting up logins and users.

What’s next:
When initially configuring authentication complete the set up by:

- Perform a rolling restart.
- Adding a superuser login

Defining an LDAP scheme
DataStax Enterprise supports LDAP for:

- **Authentication**: DSE passes through the credentials to the configured LDAP.
- **Role management**: DSE looks up the authenticated user and retrieves a list of LDAP groups and then matches LDAP group names to DSE role names.

Prerequisites:
Complete Enabling DSE Unified Authentication with the following options:

- For **authentication**, ensure that `authentication_options.scheme:ldap` or `authentication_options.other_scheme:ldap` is set in the `dse.yaml`:

  ```yaml
  authentication_options:
  ...
  scheme: ldap
  ```

- For **role management**, ensure that `role_management_options.mode: ldap` is set in the `dse.yaml`:

  ```yaml
  role_management_options:
  ...
  mode: ldap
  ```

Ensure that a supported LDAP v3 server is available. DataStax Enterprise supports:

- Microsoft Active Directory:
  
  # Windows 2008
  # Windows 2012

- OpenLDAP 2.4.x

- Oracle Directory Server Enterprise Edition 11.1.1.7.0

On every node configure the [https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml__ldap-options](https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml__ldap-options) in the `dse.yaml`. 
Authentication and authorization

For multi-datacenter support, use the nearest available LDAP host.

1. Configure the following options when using an LDAP scheme for authentication or role management:

   Example of AD authentication minimum settings:

   ```
   ldap_options:
   server_host: win2012ad_server.mycompany.lan
   server_port: 389
   search_dn: cn=lookup_user,cn=users,dc=win2012domain,dc=mycompany,dc=lan
   search_password: lookup_user_password
   use_ssl: false
   use_tls: false
   truststore_path: path/to/truststore
   truststore_password: passwordToTruststore
   truststore_type: jks
   user_search_base: cn=users,dc=win2012domain,dc=mycompany,dc=lan
   user_search_filter: (sAMAccountName={0})
   credentials_validity_in_ms: 0
   search_validity_in_seconds: 0
   connection_pool:
   max_active: 8
   max_idle: 8
   ```

   a. Configure all required settings:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_host</td>
<td>Host name of the LDAP server.</td>
</tr>
<tr>
<td>server_port</td>
<td>Port on which the LDAP server listens. The default port for unencrypted</td>
</tr>
<tr>
<td></td>
<td>connections is 389. The default SSL port for LDAP is 636.</td>
</tr>
<tr>
<td>search_dn</td>
<td>Distinguished name (DN) of an account with read access to the</td>
</tr>
<tr>
<td></td>
<td><code>user_search_base</code> and <code>group_search_base</code>. Comment out to use an anonymous</td>
</tr>
<tr>
<td></td>
<td>bind.</td>
</tr>
<tr>
<td>search_password</td>
<td>Password of the <code>search_dn</code> user.</td>
</tr>
<tr>
<td>user_search_base</td>
<td>Distinguished name (DN) of the object to start the recursive search for</td>
</tr>
<tr>
<td></td>
<td>user entries for authentication and role management memberof searches. For</td>
</tr>
<tr>
<td></td>
<td>example to search all users in example.com, <code>ou=users,dc=example,dc=com</code>.</td>
</tr>
<tr>
<td></td>
<td>Active Directory uses a different search base, typically</td>
</tr>
<tr>
<td></td>
<td><code>CN=search,CN=Users,DC=ActDir_domname,DC=internal</code>. For example,</td>
</tr>
<tr>
<td></td>
<td><code>CN=search,CN=Users,DC=example,DC=internal</code>.</td>
</tr>
<tr>
<td>user_search_filter</td>
<td>Attribute that identifies the user. The default setting is <code>(sAMAccountName={0})</code>.</td>
</tr>
<tr>
<td></td>
<td>When using Active Directory set the filter to <code>(sAMAccountName={0})</code>.</td>
</tr>
</tbody>
</table>

   b. Configure optional settings:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_ssl</td>
<td>Set to <code>true</code> to enable SSL connections to the LDAP server. If set to <code>true</code>,</td>
</tr>
<tr>
<td></td>
<td>you might need to change <code>server_port</code> to the SSL port of the LDAP server.</td>
</tr>
<tr>
<td></td>
<td>The default value is <code>false</code>.</td>
</tr>
<tr>
<td>use_tls</td>
<td>Set to <code>true</code> to enable TLS connections to the LDAP server. If set to <code>true</code>,</td>
</tr>
<tr>
<td></td>
<td>change the <code>server_port</code> to the TLS port of the LDAP server. The default</td>
</tr>
<tr>
<td></td>
<td>value is <code>false</code>.</td>
</tr>
<tr>
<td>truststore_path</td>
<td>The path to the truststore for SSL certificates.</td>
</tr>
<tr>
<td>truststore_password</td>
<td>The password to access the trust store.</td>
</tr>
</tbody>
</table>
 Authentication and authorization

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>truststore_type</td>
<td>The type of truststore. The default value is jks.</td>
</tr>
<tr>
<td>credentials_validity_in_ms</td>
<td>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 reflected in DataStax Enterprise during the cache period.</td>
</tr>
<tr>
<td>search_validity_in_seconds</td>
<td>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.</td>
</tr>
<tr>
<td>connection_pool</td>
<td>The configuration settings for the connection pool for making LDAP requests.</td>
</tr>
<tr>
<td>max_active</td>
<td>The maximum number of active connections to the LDAP server. The default value is 8.</td>
</tr>
<tr>
<td>max_idle</td>
<td>The maximum number of idle connections in the pool awaiting requests. The default value is 8.</td>
</tr>
</tbody>
</table>

2. For Role Management mode ldap, choose one of the following:

- Option 1. Configure DSE to get a list of groups from an attribute of the user entry:

  ```
  user_memberof_attribute: memberof
  group_search_type: memberof_search
  ```

  *memberof* is the name of the attribute that contains a list of groups in the default Microsoft Active Directory LDAP scheme. OpenLDAP does not have a member of attribute by default.

Table 6: Options for user attribute

<table>
<thead>
<tr>
<th>Option</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_memberof_attribute</td>
<td>memberof</td>
<td>Attribute that contains a list of group names; role manager assigns DSE roles that exactly match any in the list. Unmatched groups are ignored.</td>
</tr>
<tr>
<td>group_search_type</td>
<td>memberof_search</td>
<td>Recursively search for user entries using the &lt;user_search_base&gt; and &lt;user_search_filter&gt;.</td>
</tr>
</tbody>
</table>

- Option 2. Configure DSE to search all group objects from the search base and return a list of groups that contain the user:

  ```
  group_search_type: directory_search
  group_search_base: DN
  group_search_filter: (uniquemember={0})
  group_name_attribute: CN
  ```

  *uniquemember* is the name of the attribute that contains a list of users in the default Microsoft Active Directory LDAP scheme for group.
Table 7: Options for group objects

<table>
<thead>
<tr>
<th>Option</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_search_type</td>
<td>directory_search</td>
<td>Recursively search for group objects using the group_search_base.</td>
</tr>
<tr>
<td>group_search_base</td>
<td>DN</td>
<td>Identifies the location that role manager starts the recursive check for groups that contain the user. For example to check all internal groups of example.com: cn=internal ou=group,dc=example,dc=com.</td>
</tr>
<tr>
<td>group_search_filter</td>
<td>(uniquemember=(0))</td>
<td>Attribute that matches the user name. In most LDAP services the attribute is uniquemember.</td>
</tr>
<tr>
<td>group_name_attribute</td>
<td>cn</td>
<td>Attribute that contains the group name that role manager matches to a configured DSE role. Group name must match the DSE role name exactly including case. Unmatched groups are ignored.</td>
</tr>
</tbody>
</table>

3. Perform a rolling restart to implement the changes.

When adding LDAP to and authentication enabled DSE environment, DataStax recommends setting up roles for LDAP users and groups before restarting.

Configuring JMX authentication

JMX (Java Management Extensions) technology provides a simple and standard way of managing and monitoring resources related to an instance of a Java Virtual Machine (JVM). This is achieved by instrumenting resources with Java objects known as Managed Beans (MBeans) that are registered with an MBean server. DataStax Enterprise (DSE) supports authentication of JMX users and role-based access control to MBeans, see About DSE Unified Authentication. DSE provides JMX authentication for nodetool and external monitoring tools such as JConsole.

To manage JMX client access, see Controlling access to JMX MBeans.

Java also provides local JMX authentication, which stores credentials and provides access control using a local file. When authenticate and authorization is disabled on the DSE, you can implement file based JMX remote authentication.

Default settings

By default, JMX remote connections are disabled and JMX security authentication is disabled for both local and remote connections in the cassandra-env.sh file:

```
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=false"
```

Enabling DSE Unified Authentication

DSE provides unified authentication from utilities such as dsetool and nodetool as well as external monitoring tools such as JConsole that interface with the database using Java Management Extensions (JMX) MBeans.

To authorize access, see Controlling access to JMX MBeans.

DSE also supports local JMX authentication, which stores credentials and provides access control using a local file. When authenticate and authorization are disabled on DSE, you can implement file based JMX remote authentication.

Prerequisites: To use DSE Unified Authentication for JMX users, complete Enabling DSE Unified Authentication.

Only use Java JMX remote authentication with local files in environments where DSE Unified Authentication and RBAC are disabled.
1. On DSE nodes that you want to allow access, set the JMX remote authenticate to true for remote and/or local:

   ```
   JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=true"
   ```

   Connections are tested to see if they are local, change the first instance to enable authentication on local connections and the second instance (in the else statement) to enable remote.

2. Disable local authentication by commenting out the following lines:

   ```
   #JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.password.file=/etc/cassandra/jmxremote.password"
   #JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.access.file=/etc/cassandra/jmxremote.access"
   ```

3. To enable external authentication using DSE Authenticator, uncomment the following lines:

   ```
   JVM_OPTS="$JVM_OPTS -Dcassandra.jmx.remote.login.config=CassandraLogin"
   JVM_OPTS="$JVM_OPTS -Djava.security.auth.login.config=$CASSANDRA_HOME/conf/cassandra-jaas.config"
   JVM_OPTS="$JVM_OPTS -Dcassandra.jmx.authorizer=org.apache.cassandra.auth.jmx.AuthorizationProxy"
   ```

4. Restart DSE.

**Enabling JMX native authentication**

Use the Java-provided local JMX authentication method, which stores credentials and controls access using a local file. This implementation requires authentication to run utilities such as `nodetool` or `nodesync`. When enabled, ensure that DSE Unified Authentication is disabled.

Generally, JMX settings are inserted into the `cassandra-env.sh` file. However, you can also specify them on the command line:

   ```
   cassandra -Dcom.sun.management.jmxremote.authenticate=true
   -Dcom.sun.management.jmxremote.password.file=/etc/cassandra/jmxremote.password
   ```

1. If it does not already exist, create the `/etc/cassandra` directory from an account with `sudo` privilege.

   ```
   $ sudo mkdir /etc/cassandra
   ```

2. Set the JMX remote authenticate to `true` for remote and/or local:

   ```
   JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=true"
   ```

   Connections are tested to see if they are local; change the first instance to enable authentication on local connections and the second instance (in the else statement) to enable remote.
3. On DSE nodes where you want to disable JMX remote access, ensure `jmxremote.authenticate` is set to `false`:

```
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=false"
```

4. Uncomment the `java.rmi.server.hostname` setting, and change it to the IP address of the node to which you are connected. Example:

```
JVM_OPTS="$JVM_OPTS -Djava.rmi.server.hostname=10.0.100.100"
```

5. On nodes that allow access, set the path to the credentials file:

```
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.password.file=/etc/cassandra/jmxremote.password"
```

   Ensure that the path is accessible to the user who runs as `cassandra`.

6. Create a `jmxremote.password` file that contains a user name and password on each line and save it to the location entered in the previous step. Example:

```
$ touch jmxremote.password
cassandra p4ssw0rd
```

7. Change the ownership and permission of the `jmxremote.password` file. Example:

```
$ chown cassandra:cassandra /etc/cassandra/jmxremote.password
$ chmod 400 /etc/cassandra/jmxremote.password
```

8. (Optional) To limit the types of actions a user can perform, create a `jmxremote.access` file, uncomment the remote access option, and specify the path in the following setting:

```
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.access.file=/etc/cassandra/jmxremote.access"
```

9. If you enabled the remote access in 2, edit the `jmxremote.access` file to include users and their proper permission level. Example:

```
cassandra readwrite
<new_superuser> readwrite
<some_other_user> readonly
```

   The default superuser account is a security hazard! This account is used only for the purposes of illustration.

   The `readonly` permission allows the JMX client to read an MBean's attributes and receive notifications. The `readwrite` permission allows the JMX client to set attributes, invoke operations, and create and remove MBeans, in addition to reading an MBean's attributes and receiving notifications.
Authentication and authorization

The access file must be secured from unauthorized readers. Change the ownership of the `jmxremote.access` file to the user who starts `cassandra`, and change permissions to read only. Example:

```
$ chown cassandra:cassandra /etc/cassandra/jmxremote.access
```

```
$ chmod 400 /etc/cassandra/jmxremote.access
```

This example presumes that `cassandra` is run by the default user `cassandra`.

10. If all nodes on the cluster were updated, perform a rolling restart; otherwise restart only the affected nodes.

11. Verify that authentication is working by running a `nodetool` command with credentials:

```
$ nodetool -u cassandra -pw p4ssw0rd status
```

The results should display.

```
Datacenter: DataStax
---------------
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load  Tokens  Owns  Host ID
Rack
UN 10.200.182.180  316.76 KiB  1 ?  5ca115f6-250a-4964-9a52-c10926031f1b rack1
UN 10.200.182.181  446.76 KiB  1 ?  74a44407-5e26-43d4-83dc-aee9fe35c2f4 rack1
Datacenter: Solr
---------------
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load  Tokens  Owns  Host ID
Rack
UN 10.200.182.183  368.38 KiB  1 ?  d59d912c-dcc9-469f-8ae1-1c14313e16b1 rack1
```

Note: Non-system keyspaces don't have the same replication settings, effective ownership information is meaningless

12. Repeat the configuration on each node in the cluster.

Configuring cache settings

By default, DataStax Enterprise (DSE) caches the login user's roles and corresponding permissions. Caching allows multiple connection requests to occur within the specified period without repeating the entire authorization process (querying the `system_auth` tables) for every action.

Security-related cache settings

Adjust settings in the `cassandra.yaml` file:

```
roles_validity_in_ms
```

Validity period for roles cache in milliseconds. Determines how long to cache the list of roles assigned to the user; users may have several roles, either through direct assignment or inheritance (a role that has been granted to another role). Adjust this setting based on the complexity of your role hierarchy, tolerance for role changes, the number of nodes in your environment, and activity level of the cluster.

Fetching permissions can be an expensive operation, so this setting allows flexibility. Granted roles are cached for authenticated sessions in `AuthenticatedUser`. After the specified time elapses, role validity is rechecked. Disabled automatically when internal authentication is not enabled when using `DseAuthenticator`.
Authentication and authorization

- 0 - disable role caching
- milliseconds - how long to cache the list of roles assigned to the user

Default: 120000 (2 minutes)

**roles_update_interval_in_ms**

Refresh interval for roles cache. After this interval, cache entries become eligible for refresh. On next access, the database schedules an async reload, and returns the old value until the reload completes. If roles_validity_in_ms is non-zero, then this value must also be non-zero. When not set, the default is the same value as roles_validity_in_ms.

Default: commented out (120000)

**permissions_validity_in_ms**

How long permissions in cache remain valid to manage performance impact of permissions queries. Fetching permissions can be resource intensive. Set the cache validity period to your security tolerances. The cache is used for the standard authentication and the row-level access control (RLAC) cache. The cache is quite effective at small durations.

- 0 - disable permissions cache
- milliseconds - time, in milliseconds

REVOKE does not automatically invalidate cached permissions. Permissions are invalidated the next time they are refreshed.

Default: 120000 (2 minutes)

**permissions_update_interval_in_ms**

Sets refresh interval for the standard authentication cache and the row-level access control (RLAC) cache. After this interval, cache entries become eligible for refresh. On next access, the database schedules an async reload and returns the old value until the reload completes. If permissions_validity_in_ms is non-zero, the value for roles_update_interval_in_ms must also be non-zero. When not set, the default is the same value as permissions_validity_in_ms.

Default: commented out (2000)

**permissions_cache_max_entries**

The maximum number of entries that are held by the standard authentication cache and row-level access control (RLAC) cache. With the default value of 1000, the RLAC permissions cache can have up to 1000 entries in it, and the standard authentication cache can have up to 1000 entries. This single option applies to both caches. To size the permissions cache for use with Setting up Row Level Access Control (RLAC), use this formula:

\[ \text{numRlacUsers} \times \text{numRlacTables} + 100 \]

If this option is not present in cassandra.yaml, manually enter it to use a value other than 1000. See Enabling DSE Unified Authentication.

Default: not set (1000)

Disabling Authentication and Authorization caching

To disable configuration of authorization caches (for roles and permissions), uncomment the following line in the jvm.options file:

```
-Dcassandra.disable_auth_caches_remote_configuration=true
```

After setting this option, cache options can be set only in the cassandra.yaml file. To make the new setting take effect, restart Cassandra.

Securing schema information

The schema information and corresponding operation information is stored in system and system_schema keyspaces. By default when DSE Unified Authentication is enabled, roles are granted full access to some tables (listed below). To limit which data in these tables a role can display, enable system_keyspaces_filtering and grant DESCRIBE permission on the keyspace. When a role executes a query against a table in the system or
Authentication and authorization

system_schema keyspace, only rows that match the keyspace that they have DESCRIBE permissions on are returned in the query results.

System and schema keyspace filtering does not apply to superusers. Superusers roles have access to all resources except those to which access is denied by a RESTRICT statement.

When system keyspace filtering is disabled, all users have the following access:

<table>
<thead>
<tr>
<th>Keyspace</th>
<th>Table</th>
<th>Default access</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>size_estimates</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>available_ranges</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>local</td>
<td>FULL*</td>
</tr>
<tr>
<td></td>
<td>peers</td>
<td>FULL*</td>
</tr>
<tr>
<td></td>
<td>sstable_activity</td>
<td>NONE</td>
</tr>
<tr>
<td></td>
<td>IndexInfo</td>
<td>NONE</td>
</tr>
<tr>
<td></td>
<td>built_views</td>
<td>NONE</td>
</tr>
<tr>
<td></td>
<td>views_builds_in_progress</td>
<td>NONE</td>
</tr>
<tr>
<td>system_schema</td>
<td>tables</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>triggers</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>views</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>keyspaces</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>dropped_columns</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>functions</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>aggregates</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>indexes</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>types</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>columns</td>
<td>Full</td>
</tr>
</tbody>
</table>

* All roles have full access to the local and peers tables even if filtering is enabled.

By default, roles do NOT have access to data in the system_auth, system_traces, and system_distributed keyspaces. To allow access grant SELECT permission on the keyspace or table.

Example

The following uses an internal non-superuser account, martin.
1. Create internal login role using cqlsh:

```
CREATE ROLE martin WITH LOGIN = true AND PASSWORD = 'password';
```

2. **Login** as martin:

```
LOGIN martin
```

3. Count the number of tables in `system_schema.tables` that martin can list:

```
SELECT count(*) FROM system_schema.tables;
```

The results is the number of tables that exist cluster-wide.

```
    count
-------
     75
(1 row)
```

4. Set `system_keyspaces_filtering` to `true` and restart the nodes.

5. Log in to cqlsh as martin:

```
$ cqlsh -u martin -p password
```

6. Count the number of tables in `system_schema.tables` that martin can list:

```
SELECT keyspace_name, count(*) AS tables
FROM system_schema.tables
GROUP BY keyspace_name;
```

Assuming martin is not a superuser and does not have describe access on any other tables, the account can only list 18 tables.

```
keyspace_name | tables
--------------+--------
    system_schema |     10
       system |      8
(2 rows)
```

7. To allow martin to see detailed information about cycling tables, grant DESCRIBE access on the cycling keyspace:

```
GRANT DESCRIBE ON KEYSPACE cycling TO martin;
```

8. **Login** as martin:

```
LOGIN martin
```

9. Verify that martin can see the tables in the cycling keyspace:

```
SELECT keyspace_name, count(*) AS tables
FROM system_schema.tables
```
**Authentication and authorization**

```sql
GROUP BY keyspace_name;
```

The results show the number of tables that martin can list in each keyspace.

<table>
<thead>
<tr>
<th>keyspace_name</th>
<th>tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycling</td>
<td>25</td>
</tr>
<tr>
<td>system_schema</td>
<td>10</td>
</tr>
<tr>
<td>system</td>
<td>8</td>
</tr>
</tbody>
</table>

(3 rows)

**Managing database access**

How to set up Role Based Access Control (RBAC).

**About Role Based Access Control**

Role-based access control (RBAC) is available only after completing [Enabling DSE Unified Authentication](#).

A role is a database resource to which privileges to access other database resources are assigned.

**Understanding Role assignment**

The DataStax Enterprise (DSE) Role Manager `mode` controls how a role is assigned to an authenticated user.

- **internal** - Each user has a login role. When the authentication method is external the user name is matched to a role name and the role must have `login set to true`.

  DSE supports nested roles which allows permission to be managed as sets. Use the `GRANT role_name TO role_name` command to assign one role to another as a permission set.

- **ldap** - Looks up the authenticated user's LDAP group membership. Users are assigned all the roles that match an LDAP group name. At least one matching role must have `login set to true`. DSE roles automatically change as LDAP group membership changes.

  DSE does not support nesting roles with the Role Management mode LDAP.

**Setting up logins and users**

The DataStax Enterprise database uses role-based access control (RBAC). Set up roles to manage access control for database objects.

The default `cassandra` role has the same credentials in all environments. DataStax recommends locking down the cluster using `firewall` rules to prevent malicious activity until a new root account has been established.

**Adding a superuser login**

After enabling role-based access control, create your own superuser account and disable or drop the default `cassandra` account. Roles created with the superuser option have full access to the database; users with the role can run any CQL commands on all database resources.

Only provide superuser roles to a limited number of users.

When using LDAP group management (role_management_options. `mode`: ldap), to prevent unintentional role assignment, set `authentication_options. scheme_permissions: true` and bind the superuser role to the internal scheme (see [Binding a role to an authentication scheme](#)).

The DataStax Enterprise database includes the default role `cassandra` with password `cassandra`. The `cassandra` role is a superuser login account that has full access to the database. Requests from the `cassandra` account, including login, use a consistency level of QUORUM. QUORUM may cause significant performance degradation in multiple datacenter environments. For security and performance, DataStax recommends only using the `cassandra` role once during initial RBAC set up to establish your own root account and then disabling or dropping the `cassandra` role.
Requests from all other authenticated accounts have a consistency level of LOCAL_ONE. See How is the consistency level configured? for more details.

Prerequisites: Isolate the DSE cluster and enable RBAC, see Setting up logins and users.

1. Log in to CQL shell (cqlsh) with the Cassandra user:

```
cqlsh -u cassandra -p cassandra
```

2. Create a new superuser account with password stored in the CQL database:

```
CREATE ROLE root_user_name with SUPERUSER = true AND LOGIN = true and PASSWORD = 'password';
```

3. Exit cqlsh:

```
EXIT;
```

4. In order to disable or drop the cassandra role, log in with the new role created in the previous step:

```
cqlsh -u root_user
```

Enter the password at the prompt.

5. Verify that the role was created as a superuser using LIST ROLES:

```
LIST ROLES;
```

```
<table>
<thead>
<tr>
<th>role</th>
<th>super</th>
<th>login</th>
<th>options</th>
</tr>
</thead>
<tbody>
<tr>
<td>root_user</td>
<td>True</td>
<td>True</td>
<td>{}</td>
</tr>
<tr>
<td>cassandra</td>
<td>True</td>
<td>True</td>
<td>{}</td>
</tr>
</tbody>
</table>
```

6. Drop or update the cassandra account:

- Drop the cassandra account:

```
DROP ROLE cassandra;
```

- Update the cassandra role by disabling superuser and changing the password:

```
ALTER ROLE cassandra WITH SUPERUSER = false AND LOGIN = false AND password='new_secret_pw';
```

DataStax recommends dropping the account to secure the database in production environments. When using an external authentication method, such as LDAP, this prevents accidentally assignment with elevated privileges.

7. Verify that the cassandra role was deleted:

```
LIST ROLES;
```

```
<table>
<thead>
<tr>
<th>role</th>
<th>super</th>
<th>login</th>
<th>options</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataStax</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Only the roles created in these steps display:
8. Reopen the firewall to support production CQL traffic.

What's next:
Set up roles that map to the user or group names for the configured authentication schemes:

### Adding database users

A user account stored in the DataStax Enterprise database is a role that has a login and password.

To set up the DSE database user directory feature, set a scheme in the [https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml__authentication](https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml__authentication) to internal.

1. Create a role with login enabled and an internally stored password:

   ```sql
   CREATE ROLE role_name
   WITH LOGIN = true
   AND PASSWORD = 'password_string';
   ```

   where
   - `role_name` - The user name for authentication. Enclose the role names that include uppercase or special characters in double quotes.
   - `LOGIN = true` - Allows the role to access the database.
   - `PASSWORD = 'default_password'` - Stored internally for database managed accounts.
   - (Optional) `superuser = true` - Gives full access to all database objects to the user. See Adding a superuser login.

2. To allow the role to be used for authentication when `scheme_permissions` is true, bind the role to an authentication scheme:

   ```sql
   GRANT EXECUTE
   ON INTERNAL SCHEME
   TO role_name;
   ```

3. To allow another role to manage the new role:

   ```sql
   GRANT AUTHORIZE FOR ALTER, DROP
   ON new_role_name
   TO management_role;
   ```

   All superusers have authorize permissions on all roles. And the role that created the role is granted all permissions on the role.

4. Each user can change their own password with the `ALTER ROLE` command.
Authentication and authorization

a. User logs in with their role name:

```
$ cqlsh -u role_name -p default_password
```

b. Changes the password:

```
ALTER ROLE role_name
WITH password = 'newpassword';
```

What's next: Assign permissions to the role, see Assigning permissions.

Adding roles for LDAP users and groups

DataStax Enterprise supports using LDAP for authentication and/or role management.

Adding roles for LDAP logins

Role manager assigns a DSE role by matching the username provided for authentication with a DSE role name. Create a role for each LDAP user that needs to access the database.

The LDAP **username** attribute is defined in **user_search_filter**, such as the UID (Open LDAP) or SamAccountName (Microsoft Active Directory).

1. Create a login role that matches the **username**:

```
CREATE ROLE ldap_user_role WITH LOGIN = true;
```

   • **ldap_user_role** - The username passed from the authenticator to the role manager; the string is exactly as the user entered it at login, including case.

   • **LOGIN** - Required to allow the user basic access to the database.

   • **SUPERUSER** - Grants full database access, except on objects that a permission has been restricted (see Restricting access to data).

   Do NOT set a password.

2. Bind the assignment to an authentication scheme:

```
GRANT EXECUTE on LDAP SCHEME to ldap_user_role;
```

3. To allow another role to manage the new role:

```
GRANT AUTHORIZE FOR ALTER, DROP
ON new_role_name
TO management_role;
```

All superusers have authorize permissions on all roles. And the role that created the role is granted all permissions on the role.

What's next: Assign permissions to the role, see Assigning permissions.

Adding roles for LDAP groups

When using LDAP authentication with LDAP role management, DSE assigns the user all the roles that match the LDAP groups that they are a member of. At least one DSE role must have login privileges.

DSE does not support nested groups with LDAP role management.
After authentication completes successfully, DSE queries the LDAP again for a list of the groups. DSE either retrieves the list from:

- The user’s member of attribute (user_memberof_attribute) and returns all group names from the group DN (group_name_attribute).
- Searches for groups in the group_search_base that contain the username in the group_search_filter attribute and returns a list of the group names using the group_name_attribute from the group DN.

When role management mode LDAP is enabled with internal authentication, to look up groups the internal role must correspond to the LDAP user id attribute, such as UID or SamAccountname.

1. Create a login role that matches the group_name, where all users that belong to this group can log in to the DSE database:

   ```
   CREATE ROLE group_name WITH LOGIN = true;
   ```

   - group_name - Names are case sensitive, enclose names that contain capital letters in double-quotes. For example, use double quotes to match the cn of the group:
     ```
     cn=DSE_Login_Users,ou=Groups,dc=example,dc=com
     ```
   - LOGIN = true - At least one group the user belongs to must have login privileges for the user to execute requests.
   - SUPERUSER - Grants full database access, except on objects that a permission has been restricted (see Restricting access to data).

2. Bind the assignment to an authentication scheme:

   ```
   GRANT EXECUTE on LDAP SCHEME to group_name;
   ```

What’s next: Assign permissions to the role, see Assigning permissions.

Adding roles for Kerberos principals

When using Role Management mode internal, create a role that matches the user’s principal name to allow them to log in and execute CQL requests.

DSE supports Kerberos authentication with LDAP role management, when the principal name matches the email address attribute of the LDAP user. Configure the group lookup with the user_memberof_attribute option.

1. Create a role that matches the principal name, including the Kerberos REALM, with login enabled:

   ```
   CREATE ROLE "user_name@REALM"
   WITH LOGIN = true;
   ```

   - user_name@REALM - The complete user principal name. Always enclose the role names that include uppercase or special characters in double quotes.
   - LOGIN = true - Allows the role to access the database.
   - (Optional) SUPERUSER = true - Gives full access to all database objects to the user. See Adding a superuser login.
2. Bind the assignment to an authentication scheme:

   ```sql
   GRANT EXECUTE
   ON KERBEROS SCHEME
   TO 'user_name@REALM';
   ```

   Enclose the role name in single quotes, the name is case-sensitive.

3. To allow another role to manage the new role:

   ```sql
   GRANT AUTHORIZE FOR ALTER, DROP
   ON new_role_name
   TO management_role;
   ```

   All superusers have authorize permissions on all roles. And the role that created the role is granted all permissions on the role.

**What's next:** Assign permissions to the role, see Assigning permissions.

**Setting up roles for applications**

DSE proxy management allows roles to log in and execute CQL queries as other roles. This is particularly useful for secure middleware like web servers; the web server can log in once and proxy execute...
queries as its clients, keeping the audit log intact and leveraging DSE role-based access control.

Application authentication and role assignment
The application sends its own credentials, or in the example above a Kerberos ticket, which is validated by DSE. The authenticated user name for the application is passed to DSE Role Manager. The role_management_options mode setting determines how roles are assigned to authenticated users, when using:

- **internal** roles are assigned to authenticated users by matching the user name supplied in the credentials to a role name.
  
  In the example above, the full principle name for the application including REALM must match the role name.

- **ldap** looks up the authenticated user in LDAP and returns a list of groups, then matches the group names to role names.
  
  For LDAP the group names of the application account must match the role that can proxy other roles.
Any members of a group can be assigned the role. When using proxy roles with LDAP group management, create a single group-role pair for the application account. The group in LDAP should only contain one member, the application account.

When creating the primary role for the application account, you must ensure that the correct role is matched by Role Manager to that account.

** Associating proxy roles with the application account

For users that will never directly access DSE, DataStax recommends creating proxy roles that are not bound to an authentication scheme and have no additional options, such as LOGIN, PASSWORD, and SUPERUSER. Therefore, when the authorization_options scheme_permissions are set to true, the role can be used only by the corresponding application account.

To establish an application role using the DSE Role Manager internal mode:

1. Create an application role based on the authentication scheme:
   - Internal scheme:
     ```sql
     CREATE ROLE role_name
     WITH LOGIN = true AND PASSWORD = 'password';
     ```
   - LDAP scheme:
     ```sql
     CREATE ROLE LDAP_service_account
     WITH LOGIN = true;
     ```
   where `LDAP_service_account` matches the attribute specified for the authentication_options user_filter, such as UID or samAccountName or when using role management with LDAP a group name assigned to the service account in LDAP.
   - Kerberos scheme:
     ```sql
     CREATE ROLE 'user_principal_name'
     WITH LOGIN = true;
     ```
   where `user_principal_name` matches the full case-sensitive user principal name, such as jane@EXAMPLE.COM.

   An internally stored password is required when authenticating against the internal scheme. The application role does not require permissions to database objects.

2. Create proxy role without login option:
   ```sql
   CREATE ROLE proxy_role_name;
   ```

   DataStax recommends managing the proxy accounts as permission sets without LOGIN, SUPERUSER, and PASSWORD (internally stored passwords).

3. Assign permissions on database resources to the proxy role:
   ```sql
   GRANT permission_name
   ON resource_name
   ```
Authentication and authorization

Repeat the GRANT statement for each permission you want to assign. For a complete list of permissions and supported database objects, see the CQL Permissions matrix.

4. Associate the application role with the proxy role:
   • Allow the service role to proxy all requests after authenticating:
     ```
     GRANT PROXY.LOGIN ON ROLE proxy_role_name TO service_role_name;
     ```
   • Allow the service role to proxy each request after authenticating:
     ```
     GRANT PROXY.EXECUTE ON ROLE proxy_role_name TO service_role_name;
     ```

5. Test that the application role can run commands with the proxy role:
   a. Start CQL shell with the application role credentials:
      ```
      cqlsh -u application_role_name
      Password:
      ```
      Providing the password from the command line using the -p option provides the password in plain text and may be accessible through the terminal history to any other user with access to the machine.
   b. Start session using the proxy role:
      ```
      EXECUTE AS proxy_role_name;
      ```
   c. Run CQL statements to verify permissions.
   d. Stop executing CQL statements using the proxy role:
      ```
      EXECUTE AS
      ```

Another use case for proxy roles is to track workflows. In this example, create an application account for the cycling web application and corresponding proxy accounts for team management and cyclist expenses workflows.

1. Create a role for the cycling app using internal authentication:
   ```
   CREATE ROLE cycling_app WITH LOGIN=true AND PASSWORD='password';
   GRANT EXECUTE ON INTERNAL SCHEME TO cycling_app;
   ```
   Binding the role to the authentication scheme is required only when the authentication_options scheme_permissions is set to true.

2. Create role with permissions to manage expenses:
a. Create the proxy role:

```
CREATE ROLE cycling_expense_management;
```

b. Assign access permissions to the role:

```
GRANT MODIFY ON cycling.cyclist_expenses to cycling_expense_management;
```

3. Create a role to manage cycling teams:

```
CREATE ROLE update_teams;
GRANT MODIFY ON cycling.cyclist_name TO update_teams;
GRANT MODIFY ON cycling.cyclist_teams TO update_teams;
GRANT MODIFY ON cycling.cyclist_id TO update_teams;
```

4. Allow the cycling application to run CQL commands using the team management and expense management roles:

```
GRANT PROXY.EXECUTE ON ROLE cycling_expense_management to cycling_app;
GRANT PROXY.EXECUTE ON ROLE update_teams to cycling_app;
```

5. Test execution of commands from cycling application role:

```
cqlsh -u cycling_app -p 'password' -e "EXECUTE AS
cycling_expense_management; INSERT INTO cycling.cyclist_expenses
(cyclist_name,expense_id,amount,description) VALUES ('Michael
MATTHEWS',47,25,'lunch');"
```

Since inserts only return a message with errors, the only output should be that the query executed with the correct role:

```
Executing queries as cycling_expense_management.
```

### Binding a role to an authentication scheme

Prevent unintentional role assignment when a group name or user name is found in multiple schemes. When a role has execute permission on a scheme, the role can only be applied to users that authenticated against that scheme.

Enforcing scheme permissions

Unintentional role assignments could occur when managing roles using LDAP (role_management_options.mode: ldap). DSE Role Manager assigns roles by matching the user's groups to a role by name. Users authenticating against the internal scheme automatically get the role associated with their login and password. If the same user exists in LDAP, all matching group-role names are also assigned. Likewise, when an LDAP user authenticates, all group-role matches get assigned. In mixed environments with both internal and LDAP authentication, the potential for overlapping group names and roles used for internal authentication exists. For example, an internal account such as admin might overlap with the LDAP group admin. DataStax recommends enabling scheme_permissions and granting execute on schemes to the corresponding roles.

Scheme permission CQL Syntax

Roles are associated or removed from a scheme using the CQL GRANT and REVOKE commands:

- To associate role with a scheme:

```
GRANT EXECUTE
```
Authentication and authorization

```cql
ON [ALL AUTHENTICATION SCHEMES|INTERNAL SCHEME|LDAP SCHEME|KERBEROS SCHEME]
    TO role_name;
```

- To remove a role from a scheme:

```cql
REVOKE EXECUTE
ON [ALL AUTHENTICATION SCHEMES|INTERNAL SCHEME|LDAP SCHEME|KERBEROS SCHEME]
    FROM role_name;
```

**Prerequisites:** Set `authorization_options.scheme_permissions: true` in `dse.yaml`. Once enabled, roles must be associated with an authentication scheme in order to be assigned.

Roles are resources that can be assigned to another role. Permissions are inherited, that is all the permissions from a resource role are granted to the target role.

- Allow role assignment for users authenticating with any scheme:

```cql
GRANT EXECUTE
ON ALL AUTHENTICATION SCHEMES
    TO role_name;
```

- Allow role assignment only for users authenticating with LDAP:

```cql
GRANT EXECUTE
ON LDAP SCHEME
    TO role_name;
```

- Allow role assignment only for users authenticating with internal:

```cql
GRANT EXECUTE
ON INTERNAL SCHEME
    TO role_name;
```

- Allow role assignment only for users authenticating with Kerberos:

```cql
GRANT EXECUTE
ON KERBEROS SCHEME
    TO role_name;
```

- Allowing role assignment for multiple schemes, such as users authenticating with internal or LDAP, requires executing multiple CQL statements:

```cql
GRANT EXECUTE ON INTERNAL SCHEME TO role_name;
GRANT EXECUTE on LDAP SCHEME to role_name;
```

**Assigning permissions**

Add roles and then grant permission on access database objects.

**About database objects permissions**

DataStax Enterprise supports role-based access control (RBAC) to ensure only authorized users can access database resources.

After creating a role, use the following CQL commands to manage permissions:

- **GRANT** allows access
- **REVOKE** removes access that has been granted
- **RESTRICT** explicitly denies access even if permission is granted directly or inherited
- **UNRESTRICT** removes a restriction
Restrict always take precedence over grant, including access that is inherited or automatically granted to a superuser role. Only superusers can restrict access.

Resource permissions
The following sections shows the relationship between privileges and resources, and describes the resulting permissions. The DataStax Enterprise database role based access control uses modelled hierarchy. Granting a privilege to a top level objects gives the role the same permission to all of the ancestors objects.
Permissions differ between object types.

Data resources
Data resources are keyspaces, types, table, and rows. Access is controlled using modelled hierarchy. Granting and revoking a privilege on a top level object automatically allows the same permission on all ancestors.

Data resources have the following hierarchy:

Synopsis
Use the following syntax for data resource access control:

- **ALL KEYSPACES** syntax:
  ```
  GRANT permission[, permission ...]
  ON ALL KEYSPACES
  TO role_name;
  ```
  Where permissions are **ALL PERMISSIONS, CREATE, DESCRIBE, DROP, MODIFY, and SELECT**.

- **KEYSPACE** syntax:
  ```
  GRANT permission[, permission ...]
  ON KEYSPACE keyspace_name
  TO role_name;
  ```
  Where permissions are **ALL PERMISSIONS, CREATE, DESCRIBE, DROP, MODIFY, and SELECT**.

  User-defined type access control is the same as the privilege the role has on the keyspace.

- **TABLE** syntax:
  ```
  GRANT permission[, permission ...]
  ON [TABLE] keyspace_name.table_name
  TO role_name;
  ```
  Where privileges are **ALL PERMISSIONS, DROP, MODIFY, and SELECT**.
Authentication and authorization

- ROWS syntax:

  ```
  GRANT permission[, permission ...]
  ON 'filter_text' ROWS IN keyspace_name.table_name
  TO role_name;
  ```

  Where privileges are ALL PERMISSIONS, MODIFY and SELECT.

  Row-level access control (RLAC) is disabled by default. To use RLAC, set [https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml__allow_row_level_security](https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml__allow_row_level_security) parameter to true in the dse.yaml.

Permission matrix

The following table describes the CQL statements enabled on the resource when a privilege is granted to a role:

<table>
<thead>
<tr>
<th>Privilege type</th>
<th>Resource names</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL PERMISSIONS</td>
<td>ALL KEYSPACES</td>
<td>CREATE KEYSACE and DROP KEYSACE, as well as all permissions on ancestor objects described in CREATE, ALTER, AUTHORIZE, DESCRIBE, DROP, MODIFY, and SELECT privilege.</td>
</tr>
<tr>
<td>ALL PERMISSIONS</td>
<td>KEYSACE</td>
<td>ALTER, AUTHORIZE, DESCRIBE, and SELECT privileges on the keyspace and CREATE, ALTER, AUTHORIZE, DESCRIBE, DROP, and SELECT privileges on types, tables, and rows.</td>
</tr>
<tr>
<td>ALL PERMISSIONS</td>
<td>TABLE</td>
<td>MODIFY, SELECT, and AUTHORIZE privileges on the table and all privileges on rows.</td>
</tr>
<tr>
<td>ALL PERMISSIONS</td>
<td>ROWS</td>
<td>MODIFY and SELECT privileges on the rows that match the filtering text.</td>
</tr>
<tr>
<td>ALTER</td>
<td>ALL KEYSPACES</td>
<td>ALTER KEYSACE, ALTER TABLE, ALTER TYPE, RESTRICT ROWS, and UNRESTRICT ROWS.</td>
</tr>
<tr>
<td>ALTER</td>
<td>KEYSACE</td>
<td>ALTER TABLE, RESTRICT ROWS, and UNRESTRICT ROWS.</td>
</tr>
<tr>
<td>ALTER</td>
<td>TABLE</td>
<td>ALTER TABLE, RESTRICT ROWS, and UNRESTRICT ROWS.</td>
</tr>
<tr>
<td>CREATE</td>
<td>ALL KEYSPACES</td>
<td>CREATE KEYSACE, CREATE TABLE and CREATE TYPE. Creating a resource automatically grants AUTHORIZE permission to the role that created it.</td>
</tr>
<tr>
<td>CREATE</td>
<td>KEYSACE</td>
<td>CREATE TABLE and CREATE TYPE in specified keyspace.</td>
</tr>
<tr>
<td>CREATE</td>
<td>TABLE</td>
<td>CREATE TABLE in specified keyspace.</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>ALL KEYSPACES</td>
<td>DESCRIBE KEYSACE, DESCRIBE TABLE, and DESCRIBE TYPE in any keyspace.</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>KEYSACE</td>
<td>DESCRIBE KEYSACE, DESCRIBE TABLE, and DESCRIBE TYPE, and DESCRIBE FUNCTION, and DESCRIBE AGGREGATE in specified keyspace.</td>
</tr>
<tr>
<td>DROP</td>
<td>ALL KEYSPACES</td>
<td>DROP KEYSACE, DROP TABLE, and DROP TYPE in any keyspace.</td>
</tr>
<tr>
<td>DROP</td>
<td>KEYSACE</td>
<td>DROP TABLE, and DROP TYPE in specified keyspace.</td>
</tr>
<tr>
<td>DROP</td>
<td>TABLE</td>
<td>DROP TABLE.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>ALL KEYSPACES</td>
<td>INSERT, UPDATE, DELETE and TRUNCATE on all tables.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>KEYSACE</td>
<td>INSERT, UPDATE, DELETE and TRUNCATE on any table in specified keyspace.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>TABLE</td>
<td>INSERT, UPDATE, DELETE and TRUNCATE on specified table. See note for tables with materialized views (MVs).</td>
</tr>
<tr>
<td>MODIFY</td>
<td>ROWS</td>
<td>INSERT, UPDATE, DELETE on the partition that matches the 'filtering_data' for the table.</td>
</tr>
<tr>
<td>Privilege type</td>
<td>Resource names</td>
<td>Permissions</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>SELECT</td>
<td>ALL KEYSPACES</td>
<td>SELECT on any table.</td>
</tr>
<tr>
<td>SELECT</td>
<td>KEYSPACE</td>
<td>SELECT on any table in specified keyspace.</td>
</tr>
<tr>
<td>SELECT</td>
<td>TABLE</td>
<td>SELECT on specified table.</td>
</tr>
<tr>
<td>SELECT</td>
<td>ROWS</td>
<td>SELECT on rows that exactly match the 'filtering_data' in specified table.</td>
</tr>
</tbody>
</table>

To modify a base table that has a materialized view (MV) using an INSERT or UPDATE command if access permissions are enabled, a user must be granted MODIFY or ALL PERMISSIONS on the base table.

**Functions and aggregate resources**

The function syntax applies to user defined functions and aggregates. Although user defined functions and aggregates are located in a keyspace, function permissions are distinct and must be applied separately. Access is controlled using modelled hierarchy. Granting and revoking a privilege on a top level object automatically allows the same permission on all ancestors.

User defined functions are only available in environments that have enable_user_defined_functions set to true in the cassandra.yaml file.

Functions have the following hierarchy.

**Synopsis**

Use the following syntax to control access to functions:

- **ALL FUNCTIONS syntax:**

  ```sql
  GRANT privilege_list
  ON ALL FUNCTIONS
  TO role_name;
  ```

  where the privileges are ALL PERMISSIONS, ALTER, CREATE, DROP, and EXECUTE

- **ALL FUNCTIONS IN KEYSPACE syntax:**

  ```sql
  GRANT privilege_list
  ON ALL FUNCTIONS IN KEYSPACE keyspace_name
  TO role_name;
  ```

  where the privileges are ALL PERMISSIONS, ALTER, CREATE, DROP, and EXECUTE

- **FUNCTION syntax:**

  ```sql
  GRANT privilege_list
  ON FUNCTION function_name (argument_types)
  TO role_name;
  ```

  where the function name is fully qualified and the privileges are ALL PERMISSIONS, ALTER, DROP, and EXECUTE
Authentication and authorization

Revoke permission syntax:

```
REVOKE permission_list
ON resource
FROM role_name;
```

Permission matrix

When a permission is granted to a role, users are able to perform the corresponding operations.

<table>
<thead>
<tr>
<th>privilege_name</th>
<th>resource_name</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL PERMISSIONS</td>
<td>ALL FUNCTIONS</td>
<td>All operations (ALTER, CREATE, DROP, and EXECUTE permissions) on all keyspaces.</td>
</tr>
<tr>
<td>ALL PERMISSIONS</td>
<td>ALL FUNCTIONS IN KEYSPACE</td>
<td>All operations (ALTER, CREATE, DROP, and EXECUTE) on all functions in the selected keyspace.</td>
</tr>
<tr>
<td>ALL PERMISSIONS</td>
<td>FUNCTION</td>
<td>All operations (ALTER, DROP, and EXECUTE) on the selected function.</td>
</tr>
<tr>
<td>ALTER</td>
<td>ALL FUNCTIONS</td>
<td>CREATE OR REPLACE FUNCTION and CREATE OR REPLACE AGGREGATE on existing functions in all keyspaces.</td>
</tr>
<tr>
<td>ALTER</td>
<td>ALL FUNCTIONS IN KEYSPACE</td>
<td>CREATE OR REPLACE FUNCTION and CREATE OR REPLACE AGGREGATE on existing functions in a specific keyspace.</td>
</tr>
<tr>
<td>ALTER</td>
<td>FUNCTION function_name</td>
<td>CREATE OR REPLACE FUNCTION and CREATE OR REPLACE AGGREGATE on an existing function.</td>
</tr>
<tr>
<td>CREATE</td>
<td>ALL FUNCTIONS</td>
<td>CREATE FUNCTION and CREATE AGGREGATE in all keyspaces.</td>
</tr>
<tr>
<td>CREATE</td>
<td>ALL FUNCTIONS IN KEYSPACE</td>
<td>CREATE FUNCTION and CREATE AGGREGATE in specified keyspace.</td>
</tr>
<tr>
<td>DROP</td>
<td>ALL FUNCTIONS</td>
<td>DROP FUNCTION and DROP AGGREGATE in all keyspaces.</td>
</tr>
<tr>
<td>DROP</td>
<td>ALL FUNCTIONS IN KEYSPACE</td>
<td>DROP FUNCTION and DROP AGGREGATE in specified keyspace.</td>
</tr>
<tr>
<td>DROP</td>
<td>FUNCTION function_name</td>
<td>DROP FUNCTION or DROP AGGREGATE specified function.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>ALL FUNCTIONS</td>
<td>Use a function or aggregate in SELECT, INSERT, and UPDATE in all keyspaces. Create an aggregate that contains a function.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>ALL FUNCTIONS IN KEYSPACE</td>
<td>Use a function or aggregate in SELECT, INSERT, and UPDATE in a keyspace. Create an aggregate that contains a function in the keyspace.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>FUNCTION function_name</td>
<td>SELECT, INSERT and UPDATE using specified function and use of the function in CREATE AGGREGATE.</td>
</tr>
</tbody>
</table>

Search indexes

DataStax Enterprise search indexes permissions can only be managed on a Search enabled node. Apply search index permissions in addition to keyspace and table permissions. Access is controlled using modelled hierarchy. Granting and revoking a privilege on a top level object automatically allows the same permission on all ancestors.

Search indexes have the following hierarchy.
Authentication and authorization

Synopsis

Use the following syntax to assign permissions to search resources:

- **ALL SEARCH INDICES**

  \[
  \text{GRANT permission[, permission ...]}
  \text{ON ALL SEARCH INDICES}
  \text{TO role_name;}
  \]

  where permissions are ALL PERMISSIONS, AUTHORIZE [FOR permission_list], SEARCH.ALTER, SEARCH.COMMIT, SEARCH.CREATE, SEARCH.DROP, SEARCH.REBUILD, and SEARCH.RELOAD

- **SEARCH KEYSPACE** *keyspace_name*

  \[
  \text{GRANT permission[, permission ...]}
  \text{ON SEARCH KEYSPACE } \text{keyspace_name}
  \text{TO role_name;}
  \]

  where permissions are ALL PERMISSIONS, AUTHORIZE, SEARCH.ALTER, SEARCH.COMMIT, SEARCH.CREATE, SEARCH.DROP, SEARCH.REBUILD, and SEARCH.RELOAD

- **SEARCH INDEX** [*keyspace_name.*,] *table_name*

  \[
  \text{GRANT permission[, permission ...]}
  \text{ON SEARCH INDEX } \text{[keyspace_name.]table_name}
  \text{TO role_name;}
  \]

  where permissions are ALL PERMISSIONS, AUTHORIZE, SEARCH.ALTER, SEARCH.COMMIT, SEARCH.CREATE, SEARCH.DROP, SEARCH.REBUILD, and SEARCH.RELOAD

### Permission matrix

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Resource</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL PERMISSIONS</td>
<td>ALL SEARCH INDICES</td>
<td>All search index privileges for all search indexes systemwide.</td>
</tr>
<tr>
<td>ALL PERMISSIONS</td>
<td>SEARCH KEYSPACE <em>keyspace_name</em></td>
<td>All search index privileges search indexes on any table in keyspace.</td>
</tr>
<tr>
<td>ALL PERMISSIONS</td>
<td>SEARCH INDEX <em>table_name</em></td>
<td>All search index privileges for on the table.</td>
</tr>
<tr>
<td>SEARCH.ALTER</td>
<td>ALL SEARCH INDICES</td>
<td>ALTER SEARCH INDEX CONFIG and ALTER SEARCH INDEX SCHEMA on all tables in all keyspaces.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Changes to the configuration and schema must be loaded.</td>
</tr>
<tr>
<td>SEARCH.ALTER</td>
<td>SEARCH KEYSPACE <em>keyspace_name</em></td>
<td>ALTER SEARCH INDEX CONFIG and ALTER SEARCH INDEX SCHEMA on all search indexes on any table in the keyspace.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Changes to the configuration and schema must be loaded.</td>
</tr>
</tbody>
</table>
Roles

Role management and role proxy permissions uses the following modelled hierarchy:

ALL ROLES

ROLE

Synopsis

- **ALL ROLES**

  ```sql
  GRANT permission[, permission ...]
  ON ALL ROLES
  TO role_name;
  ```

  where permissions ALL PERMISSIONS, ALTER, AUTHORIZE, CREATE, DESCRIBE, and DROP
• **ROLE**

```sql
GRANT permission[, permission ...]
ON ROLE role_name
TO role_name;
```

where permissions ALL PERMISSIONS, ALTER, AUTHORIZE, CREATE, DESCRIBE, and DROP

• **Role as a permission set (applies if Role Manager mode: internal)**

```sql
GRANT role_name
TO role_name;
```

Nesting roles gives all the permissions of the first role in the statement to the second. With internal role management, use permission set roles to create your own hierarchical permissions structures.

### Permission matrix

**Table 8: Role permission matrix**

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Resource</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL PERMISSIONS</td>
<td>ALL ROLES</td>
<td>All role permissions.</td>
</tr>
<tr>
<td>ALTER</td>
<td>ALL ROLES</td>
<td>ALTER ROLE any role</td>
</tr>
<tr>
<td>ALTER</td>
<td>ROLE role_name</td>
<td>ALTER ROLE specified role</td>
</tr>
<tr>
<td>CREATE</td>
<td>ALL ROLES</td>
<td>CREATE ROLE</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>ALL ROLES</td>
<td>LIST ROLES and LIST PERMISSIONS</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>ROLE role_name</td>
<td></td>
</tr>
<tr>
<td>DROP</td>
<td>ALL ROLES</td>
<td>DROP ROLE on any role</td>
</tr>
<tr>
<td>DROP</td>
<td>ROLE role_name</td>
<td>DROP ROLE specified role</td>
</tr>
<tr>
<td>role_name</td>
<td>role_name</td>
<td>Grant role (as a set of permissions) to another role. Requires AUTHORIZE permission on the permission role and target role.</td>
</tr>
</tbody>
</table>

### Proxy login and execute

Allow a role to execute individual commands or all commands as another role. Typically used when users are interacting with the database through an application that authenticates the users before sending the request to the DataStax Enterprise database.

**Synopsis**

• To allow a role to proxy execute and login with another role

```sql
GRANT PROXY.EXECUTE, PROXY.LOGIN
ON ROLE role_name
TO application_role;
```

Revoke permissions syntax:

```sql
REVOKE permission_name
ON ROLE role_name
```
Proxy permissions

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Resource</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROXY.EXECUTE</td>
<td>ALL ROLES</td>
<td>After authenticating issue individual requests as any specified role.</td>
</tr>
<tr>
<td>PROXY.LOGIN</td>
<td>ROLE role_name</td>
<td>After authenticating issue all requests as any specified role.</td>
</tr>
<tr>
<td>PROXY.EXECUTE</td>
<td>ROLE role_name</td>
<td>After authenticating issue individual requests as a different user.</td>
</tr>
<tr>
<td>PROXY.LOGIN</td>
<td>ROLE role_name</td>
<td>After authenticating issue all requests as a different user.</td>
</tr>
</tbody>
</table>

Authentication scheme resources

When using multiple authentication schemes, there can be overlap between users and groups. For example, an internal login role admin and an LDAP group admin. For each role, you can bind the role to an authentication scheme preventing roles from being assigned to the wrong users.

Only applies when the authentication_options scheme_permissions in the dse.yaml is set to x fv.

Schemes hierarchy:

```
ALL_SCHEMES
  LDAP
  KERBEROS
  INTERNAL
```

Synopsis

**ALL AUTHENTICATION SCHEMES**

Allows role to be assigned to users who authenticated against any scheme.

```
GRANT EXECUTE
ON ALL AUTHENTICATION SCHEMES
TO role_name;
```

**LDAP SCHEME | KERBEROS SCHEME | INTERNAL SCHEME**

Only assigns role to user if they authenticated against the specified scheme:

```
GRANT EXECUTE
ON {LDAP SCHEME | KERBEROS SCHEME | INTERNAL SCHEME}
TO role_name;
```

Permission matrix

<table>
<thead>
<tr>
<th>Permission</th>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECUTE</td>
<td>ALL AUTHENTICATION SCHEMES</td>
<td>Assign role to user from any scheme.</td>
</tr>
<tr>
<td></td>
<td>LDAP SCHEME</td>
<td>Assign role to only LDAP users.</td>
</tr>
<tr>
<td></td>
<td>KERBEROS SCHEME</td>
<td>Assign role to only Kerberos principals.</td>
</tr>
</tbody>
</table>
**Authentication and authorization**

### JMX resources (MBeans) for DSE utilities

After enabling JMX authentication, DataStax Enterprise (DSE) utilities and other third-party tools require MBean access to execute commands. The tools use JMX MBeans to remotely gather information and execute requests. Access is controlled using modelled hierarchy. Granting and revoking a privilege on a top level object automatically allows the same permission on all ancestors.

MBeans have the following modelled hierarchy for access control:

- **ALL MBEANS**
- **MBEANS pattern**
- **MBean name**

MBREAD, MBWRITE, and equivalents are deprecated.

### Synopsis

Use the following syntax to grant access:

- **ALL MBEANS**

  ```
  GRANT permission[, permission ...]
  ON ALL MBEANS
  TO role_name;
  ```

  where permissions are **ALL PERMISSIONS, DESCRIBE, EXECUTE, MODIFY, and SELECT**.

- **MBEANS pattern**

  ```
  GRANT permission[, permission ...]
  ON MBEANS 'class_name:name=value,type=value'
  TO role_name;
  ```

  where DSE supports wildcard characters in the value name to match one or more MBeans and permissions are **ALL PERMISSIONS, DESCRIBE, EXECUTE, MODIFY, and SELECT**.

- **MBean name**

  ```
  GRANT permission[, permission ...]
  ON MBEAN 'class_name:name=value,type=value'
  TO role_name;
  ```

  where permissions are **ALL PERMISSIONS, DESCRIBE, EXECUTE, MODIFY, and SELECT**.

- **Revoke permissions syntax:**

  ```
  REVOKE permission_name
  ON resource
  ```
Authentication and authorization

FROM role_name;

Permission matrix

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Resource</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL PERMISSIONS</td>
<td>ALL MBEANS</td>
<td>All operations that are applicable on all MBEANS.</td>
</tr>
<tr>
<td>ALL PERMISSIONS</td>
<td>MBEAN name</td>
<td>All operations that are applicable on the MBEAN.</td>
</tr>
<tr>
<td>ALL PERMISSIONS</td>
<td>MBEANS pattern</td>
<td>All operations that are applicable on MBEANS that match the wildcard pattern.</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>ALL MBEANS</td>
<td>Use MBQUERYNAMES or MBINSTANCEOF to retrieve information about any mbean.</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>MBEAN name</td>
<td>Use MBQUERYNAMES or MBINSTANCEOF to retrieve information about a named mbean.</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>MBEANS pattern</td>
<td>Use MBQUERYNAMES or MBINSTANCEOF to retrieve information about any mbean matching a wildcard pattern.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>ALL MBEANS</td>
<td>Use MBEXECUTE or MBINVOKE on any mbean.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>MBEAN name</td>
<td>Use MBEXECUTE or MBINVOKE on named mbean.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>MBEANS pattern</td>
<td>Use MBEXECUTE or MBINVOKE on any mbean matching a wildcard pattern.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>ALL MBEANS</td>
<td>Call MBSET on any mbean.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>MBEAN name</td>
<td>Call MBSET on named mbean.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>MBEANS pattern</td>
<td>Call MBSET on any mbean matching a wildcard pattern.</td>
</tr>
<tr>
<td>SELECT</td>
<td>ALL MBEANS</td>
<td>Use MBGET on any mbean.</td>
</tr>
<tr>
<td>SELECT</td>
<td>MBEAN name</td>
<td>Use MBGET on named mbean.</td>
</tr>
<tr>
<td>SELECT</td>
<td>MBEANS pattern</td>
<td>Use MBGET on any mbean matching a wildcard pattern.</td>
</tr>
</tbody>
</table>

Analytic applications

Authorize Spark applications on a DataStax Enterprise Analytics node.

Spark application management permissions use the following modelled hierarchy:

- ANY WORKPOOL
  # WORKPOOL datacenter_name

- ANY SUBMISSION
  # ANY SUBMISSION IN WORKPOOL datacenter_name
  # SUBMISSION id IN WORKPOOL datacenter_name

Synopsis

Use the following syntax to grant access:

- All workpools in the cluster:

  ```
  GRANT permission_list
  ON ANY WORKPOOL
  ```
Authentication and authorization

• Workpool in a datacenter:

  GRANT permission_list
  ON WORKPOOL datacenter_name
  TO role_name;

• All applications in cluster:

  GRANT permission_list
  ON ANY SUBMISSION
  TO role_name;

• All applications in a workpool:

  GRANT permission_list
  ON ANY SUBMISSION
  IN WORKPOOL datacenter_name
  TO role_name;

• Specific application in a workpool:

  GRANT permission_list
  ON SUBMISSION id
  IN WORKPOOL datacenter_name
  TO role_name;

**Permission matrix**

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Resource</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>ANY WORKPOOL</td>
<td>Submit applications to a workpool to any datacenter in cluster.</td>
</tr>
<tr>
<td>CREATE</td>
<td>WORKPOOL datacenter_name</td>
<td>Submit applications to a workpool in the specified datacenter.</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>ANY WORKPOOL</td>
<td>From the Spark UI, able to access all jobs.</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>WORKPOOL datacenter_name</td>
<td>From the Spark UI, able only able to access logs for jobs in the specified datacenter.</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>ANY SUBMISSION</td>
<td>Access all application logs in the Spark UI.</td>
</tr>
</tbody>
</table>
| DESCRIBE    | ANY SUBMISSION IN WORKPOOL
  datacenter_name | From the Spark UI only access application logs in the datacenter specified. |
| DESCRIBE    | SUBMISSION id IN WORKPOOL
  datacenter_name | From the Spark UI only access the logs of an application. |
| MODIFY      | ANY SUBMISSION                    | Manage applications across the entire cluster.             |
| MODIFY      | ANY SUBMISSION IN WORKPOOL
  datacenter_name | Manage applications in the specified datacenter.         |
| MODIFY      | SUBMISSION id IN WORKPOOL
  datacenter_name | Manage a specific application.                           |

**Remote procedure calls**

Syntax for authorizing remote procedure calls use the following modelled hierarchy:

• ALL REMOTE CALLS

  # REMOTE OBJECT object_name

  # REMOTE METHOD object_name.method_name
Authentication and authorization

Synopsis

Use the following syntax when granting permission on remote procedure calls:

- **ALL REMOTE CALLS**

  ```cql
  GRANT permission_list
  ON ALL REMOTE CALLS
  TO role_name;
  ```

  where `permission_list` is a comma separated list that can include ALL PERMISSIONS, AUTHORIZE `[FOR permission_list]`, EXECUTE, MODIFY, and SELECT

- **REMOTE OBJECT** `object_name`

  ```cql
  GRANT permission_list
  ON REMOTE OBJECT object_name
  TO role_name;
  ```

  where `permission_list` is a comma separated list that can include ALL PERMISSIONS, AUTHORIZE `[FOR permission_list]`, EXECUTE, MODIFY, and SELECT

- **REMOTE METHOD** `object_name`.method_name

  ```cql
  GRANT permission_list
  ON REMOTE METHOD object_name method_name
  TO role_name;
  ```

  where `permission_list` is a comma separated list that can include ALL PERMISSIONS, AUTHORIZE `[FOR permission_list]`, EXECUTE, MODIFY, and SELECT

Implementing separation of duties

Use the separation of duties functionality to configure administrator roles for permission management without the ability to execute other CQL commands.

Assigning permission management privileges

In order for administrators and others to use `GRANT` or `REVOKE` to permissions on a resource, their role must have one of the following permissions on the object:

- **AUTHORIZE granted is true** - Manage any permissions that has been granted on the resource; the role also allows the user to execute the CQL commands that correspond to the permission.

  For example, the `admin` role that has both authorize and select on the `all keyspaces` resource.

  ```cql
  GRANT AUTHORIZE, SELECT ON ALL KEYSPACES TO admin;
  ```

  Users with the role can GRANT AND REVOKE both the AUTHORIZATION and SELECT permissions to any other role, including their own:

  ```cql
  LIST ALL PERMISSIONS OF admin;
  ```

<table>
<thead>
<tr>
<th>role</th>
<th>username</th>
<th>resource</th>
<th>permission</th>
<th>granted</th>
<th>restricted</th>
<th>grantable</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>dbadmin</td>
<td>&lt;all keyspaces&gt;</td>
<td>SELECT</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>admin</td>
<td>dbadmin</td>
<td>&lt;all keyspaces&gt;</td>
<td>AUTHORIZE</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

- **grantable is true for a permission** - Manage only the specified permission for other roles, which are not assigned to them. The related commands are executable if granted is also true.
Authentication and authorization

For example, to allow the sec_admin to GRANT and REVOKE permissions for other roles but not access the data in all keyspaces:

```
GRANT AUTHORIZER FOR CREATE, ALTER, DROP, SELECT, MODIFY, DESCRIBE
ON ALL KEYSPACES
TO sec_admin;
```

Verify the permissions:

```
LIST ALL PERMISSIONS OF sec_admin;
```

Granted is false and grantable true.

<table>
<thead>
<tr>
<th>role</th>
<th>username</th>
<th>resource</th>
<th>permission</th>
<th>granted</th>
<th>restricted</th>
</tr>
</thead>
<tbody>
<tr>
<td>sec_admin</td>
<td>sec_admin</td>
<td>&lt;all keyspaces&gt;</td>
<td>CREATE</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
<td>ALTER</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
<td>DROP</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
<td>SELECT</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
<td>MODIFY</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
<td>DESCRIBE</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Roles that are granted ALL PERMISSIONS can delegate resource permission to all roles. When a user creates an object, they are automatically granted ALL PERMISSIONS.

The following rules apply when managing roles and access to database objects:

- Creating a new role requires CREATE granted on ALL ROLES.
- GRANT/REVOKE requires the permission to be grantable (AUTHORIZE FOR permission_name) on the resource.
- Users can not modify their own role properties LOGIN and SUPERUSER. Prevents users with ALTER permissions from making their own account a SUPERUSER or creating a role with a higher level of permission.

Authorize syntax

Use the following to allow the role to GRANT and REVOKE

- AUTHORIZER and any other permission that has been granted to them on the resource.

```
GRANT AUTHORIZER
ON (ALL KEYSPACES | TABLE table_name | 'filter_string' ROWS IN table_name)
TO role_name;
```

- Only the listed permissions.

```
GRANT AUTHORIZER FOR permission_list
ON resource_name
```
### Delegating role management permissions

- When the AUTHORIZE is granted to a role, the target role can delegate any permission it has on the resource to other roles (including itself).

```
GRANT AUTHORIZE ON ALL ROLES
TO role_name;
```

The permission shows as granted when the role's permission are listed.

- When **ALL PERMISsIONS** are granted, the role has the ability to GRANT and REVOKE all permissions to all roles, including itself.

```
ROLE

GRANT permission[, permission ...]
```
ON ROLE role_name
TO role_name;

where permissions ALL PERMISSIONS, ALTER, AUTHORIZE, CREATE, DESCRIBE, and DROP

Table 9: Authorization

<table>
<thead>
<tr>
<th>Permission</th>
<th>Resource</th>
<th>GRANT and REVOKE the permissions the issuing role has been granted on the role.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHORIZE</td>
<td>ALL ROLES</td>
<td></td>
</tr>
<tr>
<td>AUTHORIZE FOR permission_list</td>
<td>ALL ROLES</td>
<td>GRANT and REVOKE the listed permissions.</td>
</tr>
<tr>
<td>AUTHORIZE</td>
<td>ROLE name</td>
<td>GRANT and REVOKE the permissions the issuing role has been granted on the role.</td>
</tr>
<tr>
<td>AUTHORIZE FOR permission_list</td>
<td>ROLE name</td>
<td>GRANT and REVOKE the listed permissions.</td>
</tr>
</tbody>
</table>

Delegating resource management permissions

Authentication schemes

- Delegate privileges to administrators that manage roles.

```sql
GRANT AUTHORIZE [FOR EXECUTE]
ON (ALL AUTHENTICATION SCHEMES | LDAP SCHEME | KERBEROS SCHEME | INTERNAL SCHEME)
TO role_name;
```

# AUTHORIZER - Allows role to delegate the AUTHORIZE permission and if EXECUTE is also granted, the role can delegate execute permissions.

# AUTHORIZER FOR EXECUTE - Allows role to delegate which other roles can assign scheme permissions without changing their own login scheme.

Manage access

Set up roles that can manage permissions on objects without access permission.

1. Create a role. For example `security_admin`.

   ```sql
   CREATE ROLE security_admin;
   ```

2. Allow the role to manage roles:

   ```sql
   GRANT ALTER, CREATE, DROP, DESCRIBE
   ON ALL ROLES
   TO security_admin;
   ```

3. Allow authorization for all permissions with no access privileges.

   - **Data resources**

     ```sql
     GRANT AUTHORIZE FOR ALL PERMISSIONS
     ON ALL KEYSPACES
     TO security_admin;
     ```

     which allows the role to grant AUTHORIZER, CREATE, ALTER, DROP, SELECT, MODIFY, and DESCRIBE permission to other roles.
Authentication and authorization

- **Functions and aggregate resources**
  
  ```
  GRANT AUTHORIZE FOR ALL PERMISSIONS
  ON ALL FUNCTIONS
  TO security_admin;
  ```

  which allows the role to grant AUTHORIZE, CREATE, ALTER, DROP, and EXECUTE permission to other roles.

- **Search indexes**
  
  ```
  GRANT AUTHORIZE FOR ALL PERMISSIONS
  ON ALL SEARCH INDICES
  TO security_admin;
  ```

  which allows the role to grant AUTHORIZE, SEARCH.CREATE, SEARCH.ALTER, SEARCH.DROP, SEARCH.RELOAD, SEARCH.REBUILD, and SEARCH.COMMIT to other roles.

- **Roles**
  
  ```
  GRANT AUTHORIZE FOR ALL PERMISSIONS
  ON ALL ROLES
  TO security_admin;
  ```

  which allows the role to grant AUTHORIZE, CREATE, ALTER, DROP, and DESCRIBE permission to other roles.

- **Authentication scheme resources**
  
  ```
  GRANT AUTHORIZE FOR ALL PERMISSIONS
  ON ALL AUTHENTICATION SCHEMES
  TO security_admin;
  ```

  which allows the role to grant AUTHORIZE and EXECUTE permission to other roles.

- **JMX resources (MBeans) for DSE utilities**
  
  ```
  GRANT AUTHORIZE FOR ALL PERMISSIONS
  ON ALL MBEANS
  TO security_admin;
  ```

  which allows the role to grant SELECT, MODIFY, AUTHORIZE, DESCRIBE, and EXECUTE permission to other roles.

- **Analytic applications**
  
  ```
  # WORKPOOLS
  
  GRANT AUTHORIZE FOR ALL PERMISSIONS
  ON ANY WORKPOOL
  TO security_admin;
  ```

  which allows the role to grant CREATE, DESCRIBE, and AUTHORIZE permission to other roles.
Authentication and authorization

# SUBMISSIONS

```sql
GRANT AUTHORIZE FOR ALL PERMISSIONS
ON ANY SUBMISSION
TO security_admin;
```

which allows the role to grant MODIFY, DESCRIBE, and AUTHORIZE permission to other roles.

- **Remote procedure calls**

```sql
GRANT AUTHORIZE FOR ALL PERMISSIONS
ON ALL REMOTE CALLS
TO security_admin;
```

4. Assign role depending on the Role Management mode:

- **Internal** - Use the GRANT command to assign role to a login or another group role.

```sql
GRANT security_admin TO login_role_name;
```

- **LDAP** - Create a group object with matching CN (security_admin) and assign users as members of the group.

### Controlling access to keyspaces and tables

Authorize roles to access keyspaces and tables using CQL `GRANT` and `REVOKE` commands. Keyspaces, table, and rows are hierarchical. Roles that have a privilege to access a top level object, such as a keyspace automatically have the permission on the child objects (table and rows).

DataStax Enterprise supports this CQL syntax in cqlsh to revoke permissions:

```sql
REVOKE permission_name ON resource_name FROM role_name;
```

- Create a role that has all permissions in all keyspaces:

```sql
CREATE ROLE keyspace_admin;
GRANT ALL PERMISSIONS ON ALL KEYSPACES TO keyspace_admin;
```

- Create an administrator role for a single keyspace:

```sql
CREATE ROLE cycling_admin;
GRANT ALL PERMISSIONS ON KEYSPACE cycling to cycling_admin;
```

- Create a role that can only make data changes, INSERT, UPDATE, DELETE, and TRUNCATE for any table in the keyspace `cycling`:

```sql
GRANT MODIFY ON KEYSPACE cycling TO team_manager;
```

- Create a role that can only select data and use functions in the cycling keyspace:

```sql
CREATE ROLE cyclist_analyst;
GRANT SELECT ON KEYSPACE cycling TO cyclist_analyst;
```
**Setting up Row Level Access Control (RLAC)**

Row-level access control (RLAC) provides authorization to data within a table by matching a filter applied to a text-based partition key. RLAC provides more granular security for tables so that only authorized users are able to view or modify subsets of the data.

If the role has a higher level of permission, such as to the keyspace or the table, the higher level of permission allows the user access to all rows in the table.

Configuring RLAC is a multi-step process:

- **Select partition key:** Only one column can be selected per table; requires MODIFY permission on the table.
- **Grant access using a filter:** Different filters can be granted to any number of roles.

By default row level access control is disabled. To enable RLAC set [https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml__allow_row_level_security](https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml__allow_row_level_security) to true, see Enabling DSE Unified Authentication. Fetching RLAC permissions can be resource intensive and impact performance; therefore RLAC has a separate cache. Tune the RLAC cache settings in cassandra.yaml to manage performance impact.

1. Select a column on the table that you want to configure permissions. Set a UTF-8 partition key column. Only one filtering column per table is allowed:

   ```
   RESTRICT ROWS ON [keyspace_name.]table_name
   USING primary_key_column;
   ```

   Existing filters (if any) now filter on this column. The DESCRIBE TABLE command shows the row restriction.

   ```
   DESCRIBE TABLE table_name;
   ```

2. Assign RLAC to the roles:

   ```
   GRANT permission ON
   'filtering_string' ROWS IN [keyspace_name.]table_name
   TO role_name;
   ```

   Where the `filtering_string` is the case-sensitive text string to exactly match.

   The permission is applied to the role immediately, even for active sessions. Use the LIST command to display the settings:

   ```
   LIST ALL PERMISSIONS OF role_name;
   ```

   The following example uses the `cyclist_expenses` and cycling roles.

   1. Select the `cyclist_name` column as the filtering column:

      ```
      RESTRICT ROWS
      ON cycling.cyclist_expenses
      ```
Authentication and authorization

```
USING cyclist_name;

Show the changes to the table:
DESC cycling.cyclist_expenses;

The restrict statement appears at the end:
```
CREATE TABLE cycling.cyclist_expenses (
  cyclist_name text,
  expense_id int,
  amount float,
  balance float static,
  description text,
  paid boolean,
  PRIMARY KEY (cyclist_name, expense_id)
) WITH CLUSTERING ORDER BY (expense_id ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class':
                         'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
                         'max_threshold': '32',
                         'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64',
                       'class':
                           'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND min_index_interval = 128
    AND speculative_retry = '99PERCENTILE';
```

RESTRICT ROWS ON cycling.cyclist_expenses USING cyclist_name;

2. Get a list of the primary keys to use in the filter:
```
SELECT DISTINCT cyclist_name
FROM cycling.cyclist_expenses;
```

All list of all available values displays:
```
cyclist_name
----------------------
Marianne VOS
Alex FRAME
Steven KRUIKSWIJK
Vera ADRIAN
Michael MATTHEWS
Anna VAN DER BREGGEN
Joe WALLS
Paolo TIRALONGO
```

(8 rows)

3. Assign RLAC:
```
a. Allow role dantest1 to access expenses entered only by Paolo TIRALONGO:

  GRANT SELECT
```
4. Verify permissions:

a. Check dantest1 permissions:

LIST ALL PERMISSIONS OF dantest1;

In this example these are the permissions only for Dan:

<table>
<thead>
<tr>
<th>role</th>
<th>username</th>
<th>resource</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>'Paolo TIRALONGO' rows IN &lt;table cycling.cyclist_expenses&gt;</td>
</tr>
</tbody>
</table>

b. Check Jane's permissions:

LIST ALL PERMISSIONS OF jane;

In this example these are the permissions only for Jane:

<table>
<thead>
<tr>
<th>role</th>
<th>username</th>
<th>resource</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>'Vera ADRIAN' rows IN &lt;table cycling.cyclist_expenses&gt;</td>
</tr>
</tbody>
</table>

When Dan logs in and runs queries, only rows he has permission to access are returned in the results set:

1. Log in as dantest1:

   $ cqlsh -p password -u dantest1

2. Run a query:

   SELECT * FROM cycling.cyclist_expenses;

Only the records that exactly match the filter are displayed:
### Controlling access to search indexes

Assign access privileges to roles for search index management.

#### Permissions and syntax

Setting access to search indexes using the following syntax:

- **Allow access to search index:**

  ```
  GRANT permission_name
  ON (ALL SEARCH INDICES | SEARCH KEYSPACE keyspace_name | SEARCH INDEX [keyspace_name.]table_name)
  TO role_name;
  ```

- **Remove access to search index:**

  ```
  REVOKE permission_name
  ON (ALL SEARCH INDICES | SEARCH KEYSPACE keyspace_name | SEARCH INDEX [keyspace_name.]table_name)
  FROM role_name;
  ```

#### Table 10: Search index permissions

<table>
<thead>
<tr>
<th>permission_name</th>
<th>CQL command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL PERMISSIONS</td>
<td>(all commands below), GRANT, and REVOKE</td>
<td>Perform any search index actions and authorize other roles to use any of the actions below.</td>
</tr>
<tr>
<td>SEARCH.ALTER</td>
<td>ALTER SEARCH INDEX CONFIG and ALTER SEARCH INDEX SCHEMA</td>
<td>Add, drop, and set schema fields and configuration elements in a search index.</td>
</tr>
<tr>
<td>SEARCH.COMMIT</td>
<td>COMMIT SEARCH INDEX</td>
<td>Move the search index from RAM buffer to another on-heap memory location that is suitable for searching.</td>
</tr>
<tr>
<td>SEARCH.CREATE</td>
<td>CREATE SEARCH INDEX</td>
<td>Define a new search index.</td>
</tr>
<tr>
<td>SEARCH.DROP</td>
<td>DROP SEARCH INDEX</td>
<td>Remove a search index.</td>
</tr>
<tr>
<td>SEARCH.REBUILD</td>
<td>REBUILD SEARCH INDEX</td>
<td>Rebuilds the index of the search index.</td>
</tr>
<tr>
<td>SEARCH.RELOAD</td>
<td>RELOAD SEARCH INDEX</td>
<td>Reloads the search index.</td>
</tr>
</tbody>
</table>

#### Requirements to GRANT/REVOKE Search Index permissions

Roles that manage Search Index permissions must have AUTHORIZE on the search index resource:
Authentification and authorization

• Manage permissions for all search indexes:

```sql
GRANT AUTHORIZE FOR permission_name
ON ALL SEARCH INDICES
TO role_name;
```

• Limit permissions to manage permissions to individual tables:

```sql
GRANT AUTHORIZE FOR permission_name
ON SEARCH INDEX [keyspace_name.]table_name
TO role_name;
```

Superuser roles have permission to perform any action; therefore do not require explicitly granting authorize for a permission on search indexes resource.

• Set access to all search indexes:

```sql
GRANT ALL PERMISSIONS ON ALL SEARCH INDICES TO role_name;
```

• Limit access to a specific table:

```sql
GRANT ALL PERMISSIONS ON SEARCH INDEX [keyspace_name.]table_name TO role_name;
```

• Remove previously granted access to all search indexes:

```sql
REVOKE ALL PERMISSIONS ON ALL SEARCH INDICES FROM role_name;
```

• Remove previously granted access to a specific table:

```sql
REVOKE ALL PERMISSIONS ON SEARCH INDEX [keyspace_name.]table_name FROM role_name;
```

• List all roles that have access to search indexes:

```sql
LIST ALL PERMISSIONS ON (ALL SEARCH INDICES|[keyspace_name.]table_name);
```

Controlling access to Graph keyspaces
DSE Graph authentication and authorization is accomplished with DSE Unified Authentication because the underlying storage is the DSE database, see Configuring DSE Unified Authentication. Each graph has the following three keyspaces. Manage permissions to each graph using CQL GRANT/REVOKE commands.

• `graph_name`: Keyspace with the same name as the graph.

• `graph_name_system`: Keyspace with the graph name followed by underscore system that stores shared information.

```sql
GRANT <permission type> ON KEYSPACE <graphname> TO <rolename>;
GRANT <permission type> ON KEYSPACE <graphname_system> TO <rolename>;
```

The following tables show scenarios of how role permissions affect access to graph operations:

**Table 11: Role properties**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Superuser</th>
<th>Non-user</th>
<th>Plain user (no role with graph permissions)</th>
<th>ALL PERMISSIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>Select</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Authentication and authorization

<table>
<thead>
<tr>
<th>Operation</th>
<th>Superuser</th>
<th>Non-user</th>
<th>Plain user (no role with graph permissions)</th>
<th>ALL PERMISSIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema change</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>Add vertex</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>Create/Drop vertex</td>
<td>T,T</td>
<td>F,n/a</td>
<td>F</td>
<td>T,T</td>
</tr>
</tbody>
</table>

Table 12: Permission type on graph keyspaces

<table>
<thead>
<tr>
<th>Operation</th>
<th>CREATE</th>
<th>SELECT</th>
<th>ALTER</th>
<th>MODIFY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Select</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Schema change</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Add vertex</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Create/Drop vertex</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Table 13: Multiple permissions on graph keyspaces

<table>
<thead>
<tr>
<th>Operation</th>
<th>CREATE/SELECT</th>
<th>CREATE/ALTER</th>
<th>CREATE/MODIFY</th>
<th>CREATE/ALTER/DROP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Select</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Schema change</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>Add vertex</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>Create/Drop vertex</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Connection refers to the ability to connect to the DSE Graph Server.

Authorizing remote procedure calls (RPC)

DataStax Enterprise supports authentication and role-based access control for Remote Procedure Calls to Java objects and methods.

The syntax for remote calls for the specified procedure on the remote host is:

```
CALL Object.Method(parameter1, parameter2)
```

RPC permissions

RPC permissions are role-based to provide fine grained control over which roles can execute which commands. The GRANT and REVOKE CQL commands provide and revoke access to objects and methods.

DataStax Enterprise supports this CQL syntax in cqlsh to grant RPC permissions:

```
GRANT permission ON ALL REMOTE CALLS TO role;
GRANT EXECUTE ON REMOTE OBJECT object TO role;
GRANT EXECUTE ON REMOTE METHOD object.method TO role;
```

DataStax Enterprise supports this CQL syntax in cqlsh to revoke RPC permissions:

```
REVOKE EXECUTE ON ALL REMOTE CALLS FROM role;
REVOKE EXECUTE ON REMOTE OBJECT object FROM role;
```
REVOKE EXECUTE ON REMOTE METHOD object.method FROM role;

where:

- EXECUTE is the only permission that applies to RPC.
- role is the role to grant or revoke authorization to.

Managing the permissions of a large number of users can be considerably simplified through the reuse of a small number of high-level roles.

### Authorizing dse client-tool spark and dse spark

To run the `dse client-tool spark` or `dse spark` using a non-superuser role, authorize the role to EXECUTE on the following resources:

- REMOTE OBJECT DseClientTool

When an unauthorized user attempts to run the command, for example launch a spark console session:

```bash
$ dse -u jane -p password spark
```

An unauthorized error displays:

```
The log file is at /home/automaton/.spark-shell.log
ERROR 2018-07-10 19:23:04,382 org.apache.spark.deploy.DseSparkSubmitBootstrapper: Failed to start or submit Spark application because of com.datastax.driver.core.exceptions.UnauthorizedException: User jane has no EXECUTE permission on <rpc method DseClientTool.getSparkMetricsConfig> or any of its parents - see details in the log file(s): /home/automaton/.spark-shell.log
```

1. Log in to `cqlsh` with an account that has authorization permission on remote objects.

   ```bash
   $ cqlsh -u sec_admin
   ```

2. Give EXECUTE permission to the role.

   ```bash
   GRANT EXECUTE ON REMOTE OBJECT DseClientTool TO jane;
   ```

3. (Optional) Test that a user with the role can execute a command.

   ```bash
   $ dse -u jane -p password spark
   ```

   The log file is at /home/automaton/.spark-shell.log

   Creating a new Spark Session
   Welcome to
   
   /\_\_\_/\_/\_/\_/\_/\_\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_/\_
```

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_152)
Type in expressions to have them evaluated.
Type :help for more information.
Controlling access to JMX MBeans

Many monitoring and administrative tools use Management Beans (MBeans) to perform actions on the DataStax Enterprise (DSE) cluster.

When JMX authentication is enabled, non-superuser roles require access to MBeans to use *nodetool* and other DataStax Enterprise (DSE) utilities. Also access to MBeans may be required by third-party monitoring tools or other custom management utilities that interact with the database using JMX.

Many of the DSE utilities use the same or similar MBeans, therefore it can be difficult to create permission sets that limit the commands a user can run. For example, *nodetool status*, *nodetool decommission*, *nodetool removenode*, *nodetool drain*, and others, require EXECUTE permissions on the *StorageService* MBean.

All *nodetool* commands require DESCRIBE on all MBeans.

**Access denied error when permission required**

When the role *martin* tries to execute *nodetool status* without access to MBeans, the request is denied.

```bash
$ nodetool -u martin -pw password status
nodetool: Failed to connect to '127.0.0.1:7199' - SecurityException: 'Access Denied'.
```

- Create an MBEAN administrator role:
  1. Create an MBean administrator role:
     ```sql
     CREATE ROLE mbean_admin;
     ```
  2. Give the role access to all MBeans:
     ```sql
     GRANT ALL PERMISSIONS on ALL MBEANS to mbean_admin;
     ```
  3. Assign the role to a user or group role:
     ```sql
     GRANT mbean_admin TO mike;
     ```

To get a complete list of MBeans, see Using nodetool sjk.

- Minimal permission required for a role to run nodetool commands:
  1. Login to *cqlsh* using a superuser account or a role that has full access to MBeans.
Authentication and authorization

2. Grant the non-superuser role DESCRIBE on all MBEANS:

   GRANT DESCRIBE ON ALL MBEANS TO martin;

3. Grant the role both SELECT and EXECUTE permission on the Storage Service MBean:

   GRANT SELECT, EXECUTE ON MBEAN 'org.apache.cassandra.db:type=StorageService' TO martin;

4. Grant the role EXECUTE on the Endpoint Snitch Information MBean:

   GRANT EXECUTE ON MBEAN 'org.apache.cassandra.db:type=EndpointSnitchInfo' TO martin;

The role martin can execute the nodetool command.

   $ nodetool -u martin -pw password status

Datacenter: Cassandra
--------------
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address  Load  Tokens  Owns  Host ID
   Rack
UN 10.10.100.12  67.45 MiB 1       ?
   8234303e-1f0b-4ced-844b-48e1ccbc3e2 rack1
UN 10.10.100.30  776.03 KiB 1       ?
   ff96a036-a4e3-4257-8787-a66010919c36 rack2

Setting up DSE Spark application permissions

Manage user access to Spark applications. The CQL resources for Spark applications are WORKPOOL and SUBMISSION. Create permissions on the workpool resource controls the ability of a user to submit a Spark application to DSE. Modify permissions on submission resource controls the ability of a user to manage and remove applications.

Use CQL shell (cqlsh) to authorize access to DSE Resource Manager and Spark applications. All commands must be entered on a DSE Analytics node in the cluster.

   • Access required for AlwaysOn SQL roles:

   GRANT ALL PERMISSIONS ON REMOTE OBJECT AlwaysOnSqlRoutingRPC to role_name;
   GRANT ALL PERMISSIONS ON REMOTE OBJECT DseResourceManager TO role_name;
   GRANT ALL PERMISSIONS ON REMOTE OBJECT DseClientTool TO role_name;
   GRANT SELECT, MODIFY ON dse_analytics.alwayson_sql_cache table TO role_name;
   GRANT SELECT, MODIFY ON dse_analytics.alwayson_sql_info TO role_name;

   • Access to only DSE Resource Manager:

   GRANT EXECUTE ON REMOTE OBJECT DseResourceManager TO role_name;

   • Run applications:

   GRANT EXECUTE ON REMOTE OBJECT DseClientTool TO role_name

Each DSE Analytics user must have permission to make remote procedure calls with DSE client tools.
• For roles that are not superusers, access to the following tables is required:

GRANT SELECT ON system.size_estimates TO role_name;
GRANT SELECT, MODIFY ON "HiveMetaStore".sparkmetastore TO role_name;

Additional permissions are required when running AlwaysOn SQL:

GRANT SELECT, MODIFY ON dse_analytics.alwayson_sql_cache_table TO role_name;

• Submit applications:

  # To all datacenters:

  GRANT CREATE ON ANY WORKPOOL TO role_name;

  Use revoke command to remove access:

  REVOKE CREATE ON ANY WORKPOOL FROM role_name;

  # A particular datacenter:

  GRANT CREATE ON WORKPOOL datacenter_name TO role_name;

  Use revoke command to remove access:

  REVOKE CREATE ON WORKPOOL datacenter_name FROM role_name;

  The role used to submit an application is automatically granted permission to MODIFY the application.

• Modify applications:
Authentication and authorization

# All applications:

GRANT MODIFY ON ANY SUBMISSION TO role_name;

Use revoke command to remove access:

REVOKE MODIFY ON ANY SUBMISSION FROM role_name;

# All applications in a particular datacenter:

GRANT MODIFY ON ANY SUBMISSION IN WORKPOOL datacenter_name TO role_name;

Use revoke command to remove access:

REVOKE MODIFY ON ANY SUBMISSION IN WORKPOOL datacenter_name FROM role_name;

# Specific application in a particular datacenter:

GRANT MODIFY ON SUBMISSION id IN WORKPOOL datacenter_name TO role_name;

Use revoke command to remove access:

REVOKE MODIFY ON SUBMISSION id IN WORKPOOL datacenter_name FROM role_name;

• Use DSE GraphFrames:

GRANT EXECUTE ON REMOTE OBJECT DseGraphRpc TO role_name;

Create role for AlwaysOn SQL (alwayson_sql):

CREATE ROLE alwayson_sql WITH LOGIN=true; // role name matches auth_user

// Required if scheme_permissions true
GRANT EXECUTE ON ALL AUTHENTICATION SCHEMES TO alwayson_sql;

// Spark RPC settings
GRANT ALL PERMISSIONS ON REMOTE OBJECT DseResourceManager TO alwayson_sql;
GRANT ALL PERMISSIONS ON REMOTE OBJECT DseClientTool TO alwayson_sql;
GRANT ALL PERMISSIONS ON REMOTE OBJECT AlwaysOnSqlRoutingRPC to alwayson_sql;

// Spark and DSE required table access
GRANT SELECT ON system.size_estimates TO alwayson_sql;
GRANT SELECT, MODIFY ON "HiveMetaStore".sparkmetastore TO alwayson_sql;
GRANT SELECT, MODIFY ON dse_analytics.alwayson_sql_cache_table TO alwayson_sql;
GRANT SELECT, MODIFY ON dse_analytics.alwayson_sql_info TO alwayson_sql;

// Permissions to create and change applications
GRANT CREATE, DESCRIBE ON ANY WORKPOOL TO alwayson_sql;
Denying permission

RESTRICT a privilege on a resource to deny access to user with the role. The user is denied access even if the role is a superuser, privilege has been granted directly to the role or was inherited. Use UNRESTRICT to remove any restrictions the role has on the database resource.

A superuser role has full access to the database. Use restrict to create database administrator accounts that are able to manage database resources and roles but are unable to see or modify data.

GRANT and REVOKE only allow access to database resources that are UNRESTRICT.

1. Log in to CQLSH with a superuser role.

   $ cqlsh -u username

   Logging in with the default role cassandra may impact performance or fail. All requests including login are executed with consistency QUORUM.

2. Create a superuser account with login enabled.

   CREATE ROLE db_admin WITH superuser = true AND login = true
   AND password = 'anypasswordwilldo';

   A password is required for internal accounts but not for LDAP or Kerberos. See CREATE ROLE.

Restricting access to data

The RESTRICT command denies permission on a resource to the role. The user is denied access even if the role is a superuser, the privilege has been granted directly to the role or was inherited. Use UNRESTRICT to remove any restrictions the role has on the database resource.

A superuser role has full access to the database. Use restrict to create database administrator accounts that are able to manage database resources and roles but are unable to see or modify data.

GRANT and REVOKE only allow access to database resources that are UNRESTRICT.

1. Log in to CQLSH with a superuser role.

   $ cqlsh -u username

   Logging in with the default role cassandra may impact performance or fail. All requests including login are executed with consistency QUORUM.

2. Create a superuser account with login enabled.

   CREATE ROLE IF NOT EXISTS db_admin
   WITH superuser = true
   AND login = true
   AND password = 'anypasswordwilldo';

   A password is required for internal accounts but not for LDAP or Kerberos.
3. Restrict the role from accessing the data in the cycling keyspace:

   ```
   RESTRICT MODIFY, SELECT
   ON KEYSSPACE cycling
   TO db_admin;
   ```

4. Verify the restriction:

   ```
   SELECT role, resource, restricted
   FROM system_auth.role_permissions
   WHERE role = 'db_admin';
   ```

   The results show the permissions denied to the role.

<table>
<thead>
<tr>
<th>role</th>
<th>resource</th>
<th>restricted</th>
</tr>
</thead>
<tbody>
<tr>
<td>db_admin</td>
<td>data/cycling</td>
<td>{'MODIFY', 'SELECT'}</td>
</tr>
</tbody>
</table>

Providing credentials from DSE tools

How to provide credentials when connecting to the database from a DataStax Enterprise tool.

About client connections

After enabling DataStax Unified Authentication you must provide credentials if a client request access the database or uses the Java Management Extension.

Upgrading to DSE Unified Authentication usually requires rebuilding applications with the latest driver, which allows authentication scheme selection. DataStax Enterprise drivers support all features of the Cassandra drivers.

Authenticating using DSE utilities

Provide credentials for internal or LDAP pass-through authentication when using DataStax Enterprise (DSE) and native utilities to connect to an authentication enabled cluster.

The following utilities do not require credentials, even when both database and JMX authentication are enabled:

- `dse beeline`
- `dse -v`

When authorization is also enabled, use the `GRANT` command manage access to database objects. See Manage access.

Providing credentials on command line

DataStax Enterprise (DSE) tools interact directly with the database or use the Java Management Extension MBeans to get metrics and perform operations. Use the command line switch that is relative to the type of operation performed by the command.

Each of the tools listed support prompting for the password when none is provided. Using the option to enter the password on the command line prompt is a security risk because the password appears in plain text in the terminal history.

Credentials for database interactions

When DataStax Enterprise database authentication is enabled, you must provide a username and password to run commands that interact with the DSE database.
DSE utilities
DataStax provided utilities typically have the following command line options for credentials:

```
dse -u user_name [ -p password] command
```

Where the command line options for each supported tool are:
Authentication and authorization

- **dse advrep**
  
  ```
  $ dse -u username [-p password] advrep command subcommand
  ```

- **dse fs**
  
  ```
  $ dse -u username [-p password] fs
  ```

- **dse spark**
  
  ```
  $ dse -u username [-p password] spark [options]
  ```

- **dse spark-class**
  
  ```
  $ dse -u username [-p password] spark-class options
  ```

- **dse spark-submit**
  
  ```
  $ dse -u username [-p password] spark-submit options
  ```

- **dse spark-jobserver**
  
  ```
  $ dse -u username [-p password] spark-jobserver options
  ```

- **dse spark-history-server**
  
  ```
  $ dse -u username [-p password] spark-history-server options
  ```

- **dse spark-sql**
  
  ```
  $ dse -u username [-p password] spark-sql-thriftserver options
  ```

- **dse pyspark**
  
  ```
  $ dse -u username [-p password] pyspark options
  ```

- **dse spark-sql**
  
  ```
  $ dse -u username [-p password] spark-sql options
  ```

- **dse SparkR**
  
  ```
  $ dse -u username [-p password] sparkR options
  ```

- **dse client-tool connection options**
  
  ```
  $ dse -u username [-p password] client-tool subcommand options
  ```

- **dse gremlin-console**
  
  ```
  $ dse -u username [-p password] gremlin-console
  ```

Native utilities
The native database utilities typically have the following command line options for credentials:

```bash
$ command -u username [-pw password] subcommand
```

Where the command line options for each supported tool are:

- **nodetool**
  ```bash
  $ nodetool -u username [-pw password] subcommand
  ```

- **SSTable tools**
  ```bash
  $ sstable_command -u username [-pw password] [options]
  ```

- **cassandra-stress tool**
  ```bash
  $ cassandra-stress -mode user=username password=password authprovider=auth-provider=com.datastax.driver.core.AuthProvider options
  ```

**Credentials for Java Management Extensions (JMX) interactions**

When JMX authentication is also enabled, commands that use JMX MBeans to verify status or execute commands require additional credentials. Use the `-a username` and `-b password` before the command.

```bash
dsetool -a jmx_username [ -b jmx_password] command
```

When authentication is enabled, set up permission to access MBeans see Controlling access to JMX MBeans.

- **dse advrep**
- **dsetool status**

The JMX authentication switch for DSE Advanced Replication commands uses:

```bash
dse advrep --jmx-user jmx_username [ --jmx-pwd jmx_password] command
```

Entering the password in plain text from the command line is not secure; the password may be stored in the terminal history. DataStax recommends entering the password at the prompt instead of using the password option from the CLI.

**Providing credentials in a file**

Create a file that contains the credentials for commands that access the database or JMX.

**Path to file**

The credentials file default name is `.dserc` and the default location is the user's home directory. If the file uses the defaults, there is no need to specify it when running commands. If the file is in a different location or has another name, provide the full path with the `-f` option.

**Database authentication parameters**

To provide credentials for commands that require database authentication against an LDAP and/or internal scheme, define the following parameters in the `.dserc` file.

```bash
username=username
```
Authentication and authorization

**password**

The account password in plain text.

**username**

The LDAP user name or internal role name.

**JMX authentication parameters**

To provide credentials for commands that require JMX authentication, define the following parameters in the `.dserc` file.

```plaintext
jmx_username=jmx_username
jmx_password=jmx_password
```

**jmx_username**

Set to name used for JMX login.

**jmx_password**

Set to the password of the JMX user.

DataStax Enterprise supports the built-in JMX authentication against a file or against a DSE authentication scheme. See Configuring JMX authentication.

**Providing credentials using environment variable**

For `dse client-tool`, `dse advrep`, and `dse spark` commands, provide credentials using environment variables. To increase security and prevent authentication credentials from appearing in log files, DataStax recommends using environment variables instead of passing credentials on the command line or in the `.dserc` file.

Add the environment variables to your Bash `.profile` or `.bash_profile` files.

**Database variables**

To specify a user name and password for commands that interact with the database, define variables `DSE_USERNAME` and `DSE_PASSWORD`.

```plaintext
export DSE_USERNAME=username &&
export DSE_PASSWORD=password
```

**Providing credentials for CQLSH**

Provide a username and password to connect to a node on an authentication enabled DataStax Enterprise cluster using credentials:

- Stored in the `cqlshrc` file
- Specified by command line options

**Debugging cqlsh authentication**

If problems authenticating occur, use the `--debug` option to show CQL shell settings and connection details.

- Set credentials in the `cqlshrc` file:

  1. Create or modify the `~/.cassandra/cqlshrc` file by adding an `[authentication]` section with a `username` and `password`.

```plaintext
[authentication]
username = role_name
```
Authentication and authorization

password = password

See cqlshrc.sample for an example.

2. Save the file in 

3. Set permissions on the file to prevent unauthorized access, as the password is stored in plain text.

4. Check the permissions on 

   • On the command line:
     # Prompt for password
     $ cqlsh -u user_name
     # No prompt
     $ cqlsh -u user_name -p password

Configuring DSE Client for Kerberos authentication
Configure Kerberos for the DataStax Enterprise (DSE) clients, including dsetool, DSE FS, DSE Graph, and DSE Spark.

Before you begin
Install MIT Kerberos tools required by these procedures:

   • kinit
   • kadmin

Setting the JAAS configuration file location
Set the location to the configuration file using the .dserc file or environment variables.

Using the DataStax RC file
To get Kerberos credentials without having to enter a password, define the following parameters in the .dserc file:

```plaintext
sasl_protocol=principal_name
login_config=file_name
```

<table>
<thead>
<tr>
<th>sasl_protocol</th>
<th>Set to the user principal_name. For example, <a href="mailto:dse_admin@EXAMPLE.COM">dse_admin@EXAMPLE.COM</a>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>login_config</td>
<td>Set the file_name to the absolute path of the JAAS configuration file. See Providing Kerberos credentials using a keytab file or Providing Kerberos credentials using a ticket cache</td>
</tr>
</tbody>
</table>
Authentication and authorization

Kerberos token variable
Use environment variable DSE_TOKEN to specify a delegation token when Kerberos is used:

```bash
export DSE_TOKEN=delegation_token
```

DSE FS Kerberos variable
When using the `dse fs` command use the DSEFS_SHELL_OPTS environment variable to provide the path to the login file:

```bash
DSEFS_SHELL_OPTS="-Djava.security.auth.login.config=file_name"
```

Set the `file_name` to the absolute path of the JAAS configuration file. See Providing Kerberos credentials using a keytab file or Providing Kerberos credentials using a ticket cache

Spark job variable
If you are using JAAS rather than using a Kerberos ticket, you need to create a JAAS configuration file. The default location for this file is `$USER_HOME/.java.login.config`. If your JAAS configuration file is in a different location, you must specify the location by setting the `java.security.auth.login.config` option to the location of the file.

For example, to set `java.security.auth.login.config` in an environment variable for submitting jobs, set the `SPARK_SUBMIT_OPTS` environment variable to point to your JAAS configuration file:

```bash
export SPARK_SUBMIT_OPTS='"-Djava.security.auth.login.config=/path/jaas.config"'
```

Providing Kerberos credentials using a keytab file
Configure Kerberos for the DataStax Enterprise (DSE) clients, including dsetool, DSE FS, DSE Graph, and DSE Spark.

1. Create the keytab file with `kadmin`:
   Start `kadmin`:
   ```
   $ kadmin
   ```
   Create file:
   ```
   ktadd -k file_name principal_name
   ```

2. Log in using `kinit`:
   ```
   $ kinit -k -t file_name principal_name
   ```

3. Create a JAAS configuration file for the DSE Client:
   a. Create a file named `.java.login.config` and put it in the home directory.
   b. Define the variables required to use a keytab file:
      ```
      DseClient {
          com.sun.security.auth.module.Krb5LoginModule required
          refreshKrb5Config=true
          useKeyTab=true
          keyTab="file_name"
          principal="principal_name";
      }
      ```
where

keyTab

Absolute path to the keytab file. For example, /home/adam/krb5_opsc.keytab.

principal

The fully qualified principal name. For example, dse_admin/dse1.lan@EXAMPLE.COM.

For more information on the available settings see the documentation for

4. (Optional) If the JAAS configuration is not in the default location or have the default name, customize the
   location using on of the following methods:

   • Provide the Kerberos login configuration file location in .dserc file
   • Provide the Kerberos login configuration file location using environment variables

Providing Kerberos credentials using a ticket cache

Configure Kerberos for the DataStax Enterprise (DSE) clients, including dsetool, DSE FS, DSE Graph, and
DSE Spark.

Prerequisites: These steps require MIT Kerberos tools:

• kinit

1. Create a JAAS configuration file for the DSE Client as follows:

   Use the default name .java.login.config and put the file in the user's home directory. Set the DSE
client class to com.sun.security.auth.module.Krb5LoginModule and define the variables as follows:

```java
DseClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useTicketCache=true
    renewTGT=true;
};
```

2. (Optional) If the JAAS configuration is not in the default location or has a different name, customize the
   location using on of the following methods:

   • Setting the JAAS configuration file location
   • Provide the Kerberos login configuration file location using environment variables

3. Use kinit to get a ticket:

   Start kinit:

   ```bash
   $ kinit -f principal_name
   ```

Running Spark jobs with Kerberos

Spark jobs may be run against a Kerberos enabled DataStax Enterprise database. Defining a Kerberos scheme
only connects Spark to DSE database. It does not authenticate Spark components between each other.

Authenticate using the kinit command before starting the Spark job.

With Kerberos authentication, the Spark launcher connects to DSE with Kerberos credentials and requests
DSE to generate a delegation token. The Spark driver and executors use the delegation token to connect to the
cluster.
Authentication and authorization

After the application is finished, the delegation token should be cancelled. It is done automatically when the application is run in client mode. The token is cancelled when the user stops the application. However when applications are deployed in cluster mode, the user needs to cancel the delegation token manually when the application is eventually stopped. DSE will output to the console the exact command to cancel the delegation token when you start the application in cluster mode.

1. If you are using JAAS rather than using a Kerberos ticket, you need to create a JAAS configuration file. The default location for this file is 

   ```
   $USER_HOME/.java.login.config
   ```

   If your JAAS configuration file is in a different location, you must specify the location by setting the `java.security.auth.login.config` option to the location of the file.

   For example, to set `java.security.auth.login.config` in an environment variable for submitting jobs, set the `SPARK_SUBMIT_OPTS` environment variable to point to your JAAS configuration file:

   ```
   export SPARK_SUBMIT_OPTS='-Djava.security.auth.login.config=/path/jaas.config'
   ```

2. To use a JAAS configuration file with Kerberos you must set the `kerberos.use.config.file` option to `true`.

3. You must also set the `kerberos.client.reference.name` option to `DseClient`. For example:

   ```
   $ dse spark -Dkerberos.use.config.file=true
   -Dkerberos.client.reference.name=DseClient
   ```

   Configure the JAAS configuration file to use a keytab file or ticket cache.

Loading data into a remote Kerberos enabled cluster

`sstableloader` tool (also called bulk loader) loads data from an sstable into a target cluster. When loading data into a remote Kerberos enabled cluster, configure the following JVM options:

When running sstableloader with the local DSE cluster, no additional configuration is required.

- 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
  ```

- 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";
  ```
Using DSE Graph and Gremlin console with Kerberos

To run DataStax Enterprise Graph in security enabled environment, configure settings in the Graph remote.yaml:

- **DSE Unified Authentication** - set a user name and password or Kerberos principal name
- **SSL encryption** - enable and set the path to keystore
- **Configure authentication for Gremlin Console connection**:

  # Kerberos configuration

  1. Set the following Kerberos parameters in the remote.yaml:

     ```yaml
     hosts: [KMS_hostname]
     username: null
     password: null
     jaasEntry: DseClient
     # protocol is the the same as the service_principal set in dse.yaml
     protocol: kerberos_principal_name
     ```

     Leave the username and password for Kerberos unset (null). The connector ignores null username and password parameters.

  2. Create a JAAS configuration file for DseClient that defines whether to use a keytab or ticket cache.

     The default JAAS config file and location is `~/.java.login.config`.

     # Ticket cache

     ```java
     DseClient {
     com.sun.security.auth.module.Krb5LoginModule required
     useTicketCache=true
     renewTGT=true;
     }
     ```

     # Keytab file

     ```java
     DseClient {
     com.sun.security.auth.module.Krb5LoginModule required
     refreshKrb5Config=true
     useKeyTab=true
     keyTab="file_path"
     useTicketCache=false;
     }
     ```

  3. (Optional) To use an alternate file name or location for the JAAS configuration file, add it to the system properties using an environment variable before starting the Gremlin console:

     ```bash
     export JAVA_OPTIONS="$JAVA_OPTIONS -Djava.security.auth.login.config=path_to_file"
     ```

     Required if the file is not in the default location, `~/.java.login.config`. 
Authentication and authorization

# Internal or LDAP authentication - set the username and password parameters:

```
username: database_user
password: password
```

Set `search_validity_in_seconds` to a higher value, such as 30 minutes, for better performance.

**Using dsetool with Kerberos enabled cluster**

You must 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=user@realm
-Djava.security.auth.login.config=file_name
```

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;
};
```

**Providing Kerberos credentials when starting CQL shell**

Setup a `cqlshrc` to run `cqlsh` against a Kerberos-enabled cluster.

**Example files**

DataStax Enterprise provides sample files `cqlshrc.sample`:

- Kerberos example
- Kerberos and SSL example

Make changes as appropriate for your environment.

**Kerberos example**

DataStax Enterprise provides a sample `cqlshrc.sample.kerberos` file as a starting point.

Required settings for Kerberos authentication:

```
[connection]
hostname = 192.168.1.2
port = 9042
```
[kerberos]
service = dse ;; If not set, the default is dse
qops = auth ;; Optional, see the paragraph below

The [connection] hostname and [kerberos] service settings must match the values in the dse.yaml configuration file, or be set as 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 environment variables KRB_SERVICE and QOPS override the options in the .cqlshrc file. The loading order for settings is: environment variable, .cqlshrc setting, default.

The default (auth) is used when qops is not specified. On the client side, the qops option is a comma-delimited list of the QOP values allowed by the client for the connection.

- The client (cqlsh) value list must contain at least one of the QOP values 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 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 generating the certificate, be sure to set the CN to the hostname of the node.

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

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 Connecting to SSL-enabled nodes using cqlsh.
The settings for using both Kerberos and SSL are a combination of the Kerberos and SSL sections in these examples.
Authentication and authorization

The supported environmental variables are KRB_SERVICE, SSL_CERTFILE, and SSL_VALIDATE variables.

**Providing credentials with nodetool**

After configuring JMX authentication, using `nodetool` requires the `-u` and `-pw` options to the `nodetool` commands.

1. Run `nodetool` using a username and password:

   ```bash
   $ nodetool -u username -pw password info
   ```

**Providing credentials with JConsole**

After configuring JMX authentication, JConsole requires credentials to connect to DSE transactional nodes.

1. Open JConsole and enter a username and password:

   ![JConsole: New Connection](image)

   2. Click **Connect**.
Chapter 6. Auditing database activity

DataStax Enterprise (DSE) supports capturing database activity to a log file or table.

The audit logger also captures queries and prepared statements submitted by DataStax drivers, which use CQL binary protocol.

Setting up database auditing

Capture DataStax Enterprise (DSE) database activity to a log file or table. Each node only records the events that happen locally. Use the configuration to refine the type of events captured. DSE provides the following customizations:

- **Keyspace filtering** - Capture activity in every keyspace or only targeted keyspaces. Filter keyspace names using regex.

- **Category filtering** - Identify event categories to limit the number of events captured.

- **Role filtering** - Track the activity of particular users or groups by their login role name.

- **Node specific** - Enable auditing on one or more nodes. Allows auditing of only specific nodes, an entire datacenter, or the whole cluster.

Configure logging levels, mask sensitive data, and for the log file set the file name, location, size threshold, and max log files in the `logback.xml` file.

Choosing a File vs Table

Audit logging options are configured on a per node basis and therefore can be different on each node. DataStax Enterprise (DSE) supports the following methods to record database activity:

- **Log file (per node)** - The `SLF4JAuditWriter` [Simple Logging Facade for Java (SLF4J) Audit Writer] logger records all database activity that occurs on the local node to the `audit.log` file. When tracing a multi-node request, collect and parse log files from all the nodes that participated.

- **Unified table** - The `CassandraAuditWriter` logger records all database activity that happens on the local node in the `dse_audit.audit_log` table. Events from all nodes with the logger enabled are captured in the same table. This allows you to easily create reports that include multiple nodes.

Enable audit logging to a file

Use Simple Logging Facade for Java (SLF4J) audit writer (`SLF4JAuditWriter`) logger to record all database activity that occurs on the local node to the `audit.log` file. Secure the log file by controlling access using standard Linux file system permissions.

DSE does not support data encryption for the `audit.log` file. Encryption is only available for the `dse_audit.audit_log` table.

To capture events to the `audit.log` file:

Set the `audit_logging_options` in the `dse.yaml` file:

```yaml
audit_logging_options:
  enabled: true
  logger: SLF4JAuditWriter
```

- **enabled**: `true` - Turns on logging after the next start up.
Auditing database activity

- **logger**: SLF4JAuditWriter - Logger name.

In the Cassandra log directory, DataStax Enterprise creates `audit/audit.log`. 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.

### Enabling audit logging to a table

Use the `CassandraAuditWriter` logger to record all database activity that happens on the local node to the `dse_audit.audit_log` table. Events from all nodes are capture in the same table, allowing you to easily create reports that include multiple nodes.

Using the table option provides a centralized location for all events across the cluster. Secure the table with DataStax role-based access control (RBAC), see Assigning permissions and Transparent Data Encryption (TDE), see Encrypting tables.

To capture events to the `dse_audit.audit_log` table:

1. Set the `audit_logging_options` in the `dse.yaml` file:

   ```yaml
   audit_logging_options:
     enabled: true
     logger: CassandraAuditWriter
     retention_time: 12
   
   cassandra_audit_writer_options:
     mode: sync
     batch_size: 50
     flush_time: 250
     queue_size: 30000
     write_consistency: QUORUM
     dropped_event_log: /var/log/cassandra/dropped_audit_events.log
     day_partition_millis: 3600000
   ```

   - **enabled**: true - Turns on logging after the next start up.
   - **logger**: CassandraAuditWriter - Logger name.
   - **retention_time**: 12 - Number of hours to set the TTL (time-to-live) on the `dse_audit.audit_log` table. Use this setting to automatically expire data. The default is 0 (disabled).

   Customize the `cassandra_audit_writer_options` parameters as required (the default are shown above).

   The `audit_log` table has the following settings:

   ```sql
   DESC TABLE dse_audit.audit_log
   CREATE TABLE dse_audit.audit_log (  
     date timestamp,
     node inet,
     day_partition int,
     event_time timeuuid,
     authenticated text,
     batch_id uuid,
     category text,
     consistency text,
     keyspace_name text,
     operation text,
   ```
source text,
table_name text,
type text,
username text,
PRIMARY KEY ((date, node, day_partition), event_time)
) WITH CLUSTERING ORDER BY (event_time ASC)
AND bloom_filter_fp_chance = 0.01
AND caching = '{keys': 'ALL', 'rows_per_partition': 'NONE'}
AND comment = ''
AND compaction = '{class':
  'org.apache.cassandra.db.compaction.TimeWindowCompactionStrategy',
  'max_threshold': '32', 'min_threshold': '4'}
AND compression = '{chunk_length_in_kb': '64', 'class':
  'org.apache.cassandra.io.compress.LZ4Compressor'}
AND crc_check_chance = 1.0
AND default_time_to_live = 0
AND gc_grace_seconds = 7776000
AND max_index_interval = 2048
AND memtable_flush_period_in_ms = 3600000
AND min_index_interval = 128
AND nodesync = '{enabled': 'true'}
AND speculative_retry = '99PERCENTILE';

2. (Optional) By default, NodeSync is disabled for the audit_log table. To use NodeSync to maintain data consistency, enable it.

```sql
ALTER TABLE dse_audit.audit_log WITH
nodesync = { 'enabled' : true };
```

3. Set the keyspace replication strategy to NetworkTopologyStrategy and set a replication factor for each datacenter in the cluster where auditing is enabled:

```sql
ALTER KEYSPACE dse_audit WITH
replication = {
    'class': 'NetworkTopologyStrategy',
    'datacenter1' : 2,
    'datacenter2' : 1   };
```

4. Force data replication to the newly added datacenters using one of the following methods:

- If NodeSync is enabled:

```bash
$ nodesync -cu user_name -cp password \ validation submit dse_audit.audit_log
```

where `user_name` is an internal login role name or LDAP/Kerberos with permission to access the CQL table.

- If NodeSync is disabled:

```bash
$ nodetool repair dse_audit audit_log
```

### Filtering event categories

Configure which categories to capture in the `audit_logging_options` section of the `dse.yaml`.

```yaml
audit_logging_options:
enabled: true
logger: logger_name
# included_categories:
```
Auditing database activity

```yaml
# excluded_categories:
```

where logger_name is SLF4JAuditWriter or CassandraAuditWriter.

By default, both category parameters are commented out and therefore all events are captured. Use only one of
the following parameters to limit which events are captured:

- included_categories - Includes only listed categories; excludes all others.
- excluded_categories - Excludes listed categories; includes all others.

The category keywords are:

- QUERY
- DML
- DDL
- DCL
- AUTH
- ERROR
- UNKNOWN

Category types

By default, DataStax Enterprise captures all event categories when auditing is enabled. Use either the
included_categories or excluded_categories parameter limit which categories are captured.

All events have both a category and a type. A type usually maps directly to CQL command. The following list
shows all the types in each category.

**QUERY**

Logs the following types of data retrieval events:

- CQL_SELECT - SELECT
- SOLR_QUERY - SELECT statement filtered by the Search index
- GRAPH_TINKERPOP_TRAVERSAL - Calls to a table using the graph traversal instance (g). See
Discovering properties about graphs and traversals
- RPC_CALL_STATEMENT

**DML**

(data manipulation language) Logs the following types of data changes:

- SET_KS - USE
- INSERT - INSERT
- BATCH - BATCH
- TRUNCATE - TRUNCATE
- CQL_UPDATE - UPDATE
- CQL_DELETE - DELETE
- CQL_PREPARE_STATEMENT - DataStax driver prepared statement, such as Java - Prepared Statement
- SOLR_UPDATE
- SOLR_COMMIT_SEARCH_INDEX_STATEMENT - COMMIT SEARCH INDEX
Auditing database activity

**DDL**
(data definition language) Logs the following types of database schema changes:

- **ADD_CF** - CREATE TABLE
- **DROP_CF** - DROP TABLE
- **UPDATE_CF** - ALTER TABLE
- **ADD_KS** - CREATE KEYSPACE
- **DROP_KS** - DROP KEYSPACE
- **UPDATE_KS** - ALTER KEYSPACE
- **CREATE_INDEX** - CREATE INDEX
- **DROP_INDEX** - DROP INDEX
- **CREATE_VIEW** - DROP INDEX
- **DROP_VIEW** - DROP INDEX
- **UPDATE_VIEW** - DROP INDEX
- **CREATE_TYPE** - DROP INDEX
- **DROP_TYPE** - DROP INDEX
- **UPDATE_TYPE** - DROP INDEX
- **CREATE_FUNCTION** - DROP INDEX
- **DROP_FUNCTION** - DROP INDEX
- **CREATE_AGGREGATE** - DROP INDEX
- **DROP_AGGREGATE** - DROP INDEX
- **SOLR_RELOAD_SEARCH_INDEX** - RELOAD SEARCH INDEX
- **SOLR_REBUILD_SEARCH_INDEX** - REBUILD SEARCH INDEX
- **SOLR_GET_RESOURCE** - dsetool create_core
- **SOLR_UPDATE_RESOURCE** - dsetool reload_core
- **SOLR_ALTER_SEARCH_INDEX_STATEMENT** - ALTER SEARCH INDEX CONFIG or ALTER SEARCH INDEX SCHEMA
- **SOLR_CREATE_SEARCH_INDEX_STATEMENT** - CREATE SEARCH INDEX

**DCL**
(data control syntax) - Logs the following types of role and permission management events:

- **CREATE_ROLE** - CREATE ROLE
- **ALTER_ROLE** - ALTER ROLE
- **DROP_ROLE** - DROP ROLE
- **LIST_ROLES** - LIST ROLES
- **GRANT** - GRANT
- **REVOKE** - REVOKE
- **RESTRICT** - RESTRICT
Auditing database activity

- **UNRESTRICT** - UNRESTRICT
- **RESTRICT ROWS** - RESTRICT ROWS
- **UNRESTRICT ROWS** - UNRESTRICT ROWS
- **LIST PERMISSIONS** - LIST PERMISSIONS

**AUTH**

(Authentication) Logs the following types of authentication related events:

- **LOGIN** - Successful logins. **LOGIN** and login requests sent from DataStax drivers.
- **LOGIN_ERROR** - Failed logins.
- **UNAUTHORIZED_ATTEMPT**.

Messages not captured included:

- Successful Kerberos authentication events are not logged by DSE. Audit Kerberos authentication from the KDS (Kerberos Distribution Server). DataStax Enterprise logs failed Kerberos authentication request as type **LOGIN_ERROR**.
- When whitelisting keyspaces, included_keyspaces, AUTH messages are not captured.

**ERROR**

Logs CQL statement failures. The only type is **REQUEST_FAILURE**.

**UNKNOWN**

Logs events where the category and type are both **UNKNOWN**.

Example

Include only data retrieval and manipulation events:

```yaml
audit_logging_options:
  enabled: true
  logger: SLF4JAuditWriter
  included_categories: QUERY, DDL, AUTH
  # excluded_categories:
```

Filtering keyspaces

Configure which keyspaces to capture in the **audit_logging_options** section of the dse.yaml.

```yaml
audit_logging_options:
  enabled: true
  logger: logger_name
  # included_categories:
  # excluded_categories:
  # included_keyspaces:
  # excluded_keyspaces:
```

where **logger_name** is SLF4JAuditWriter or CassandraAuditWriter.

By default, both keyspace parameters are commented out and therefore all events are captured. Use only one of the following parameters to limit which events are captured:

- **included_keyspaces** - Includes only matching keyspaces; excludes all others.
  
  When whitelisting keyspaces, included_keyspaces, AUTH messages are not captured.

- **excluded_keyspaces** - Excludes matching keyspaces; includes all others.

Match keyspaces using a comma separated list of names or a single regular expression.
Example

The system local keyspace is queried on every log in, the following exclusion will show login events without showing additional queries to the system local keyspace.

```
audit_logging_options:
  enabled: true
  logger: logger_name
  excluded_keyspaces: system_local
```

Filtering roles

Set up activity tracking of specific login roles in the `audit_logging_options` section of the `dse.yaml`.

```
audit_logging_options:
  enabled: true
  logger: logger_name
  included_roles:
  excluded_roles: dse_admin, jim
```

where `logger_name` is SLF4JAuditWriter or CassandraAuditWriter.

By default, both roles parameters are commented out and therefore events are captured for all roles. Use only one of the following parameters to limit the roles whose activity is tracked:

- `included_roles` - Includes only matching roles; excludes all others.
- `excluded_roles` - Excludes matching roles; includes all others.

Match roles using a comma separated list of names.

Example

Events of all roles except for dse_admin and jim are captured.

```
audit_logging_options:
  enabled: true
  logger: CassandraAuditWriter
  included_roles:
  excluded_roles: dse_admin, jim
```

Replacing sensitive data in the audit log

When audit logging is enabled and includes DML category events, sensitive data, such as password columns, are exposed in the message portion of the audit log. You can configure the audit log appender (SLF4JAuditWriterAppender) to replace strings in log message to mask the sensitive data. The replace function uses a regular expressions to modify the data. For more information on using the replace filter, see the logback documentation.

Additionally, because the auditing is configured through a text file in the file system, the file is vulnerable to OS-level security breaches. You can secure the file by setting the permissions to 0600 on the audit files. Be aware
Auditing database activity

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.

Before sharing log files, redact sensitive data.

1. Open the logback.xml file in a text editor.

2. Locate the pattern element.

3. Using the replace function on the message text to replace the first term with the second term. For example to find password set to any characters and replace the password characters with xxxxx:

\$replace(\$msg){"password\s*\={\s*'.*', "password='xxxxx'\"}

DataStax automatically masks the password specified in the CREATE ROLE command.

4. Restart the node or wait for the configuration to automatically reload.

Managing the audit logger settings

The logback.xml file sets the SLF4JAuditWriter log file configuration including location, name, roll frequency and retention, and level.

<!--audit log-->
<appender name="SLF4JAuditWriterAppender"
class="ch.qos.logback.core.rolling.RollingFileAppender">
<file>${cassandra.logdir}/audit/audit.log</file> <!-- logfile location -->
<encoder>
<pattern>%-5level [%thread] %date{ISO8601} %F:%L - %msg%n</pattern> <!-- the layout pattern used to format log entries -->
</encoder>
<rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
<fileNamePattern>${cassandra.logdir}/audit/audit.log.%i.zip</fileNamePattern>
<minIndex>1</minIndex>
<maxIndex>20</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>
</appender>

</audit log-->
Auditing database activity

Most database activity level is INFO.

1. Open the logback.xml file in a text editor.

2. To configure data auditing, change the properties.
   - **file** - Path to log file on the location system.
   - **pattern** - Variables to write the log message. To hide sensitive data, see Replacing sensitive data in the audit log.
   - **immediateFlush** - Whether to write event message to log immediately or not.
   - **fileNamePattern** - Name convention for the log file when they roll.
   - **maxFileSize** - Threshold to roll log (maximum log size).

3. Restart the node or wait for the configuration to automatically reload.

Enable authentication debugging

When initially setting up external authentication such as Kerberos or LDAP, use server and authentication loggers with TRACE level to capture internal, LDAP, and Kerberos debugging messages in the debug.log.

1. Locate the logback.xml file and add the following lines before the end of the file:

   ```xml
   <logger name="com.datastax.bdp.transport.server" level="TRACE"/>  
   <logger name="com.datastax.bdp.cassandra.auth" level="TRACE"/>
   ``

   Only activity from the local node is recorded. Enable authentication tracing on the node where you want to troubleshoot authentication.

2. Restart the node or wait for the configuration to automatically reload.

3. Tail the debug.log:

   ```
   $ tail -f /etc/dse/cassandra/debug.log
   ```

4. Log in using an account from the external server.

Output from a successful LDAP login:

```
```
Auditing database activity

Output from a failed LDAP login:

```
username: dantest1 not found in cache
- [ldap-fetch-user] username: dantest1 connection: 
  org.apache.directory.ldap.client.api.LdapNetworkConnection@430680df
bind to connection
user_search_base: ou=users,dc=qaldap,dc=datastax,dc=lan, user_search_filter:
  (uid=dantest1)
found entry for username: dantest1
TRACE [IOThread-0] 2018-04-24 21:18:23,441  LdapUtils.java:433 - [ldap-
bind] userDN: cn=dantest1,ou=users,dc=qaldap,dc=datastax,dc=lan connection: 
  org.apache.directory.ldap.client.api.LdapNetworkConnection@430680df
bind failed for userDN: cn=dantest1,ou=users,dc=qaldap,dc=datastax,dc=lan
...```

Capturing DSE Search HTTP requests

DataStax Enterprise Search nodes support queries through the Apache Solr™ HTTP API. To log HTTP requests, additional configuration is required on each search node. In the web.xml uncomment the filter-mapping element.

1. On all DSE Search nodes, uncomment the filter-mapping element in the 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
```

Log formats

DataStax Enterprise writes events to the log file using pipe-delimited name/value pairs. Each pair is separated from the next by the pipe symbol (|). The name and value portions of each pair are separated by a colon. A
name/value pair, or field, is included in the log line only when a value exists for that particular event. Some fields always have a value, and are always present. Other fields 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.

<table>
<thead>
<tr>
<th>Field Label</th>
<th>Field Value</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>dse node address</td>
<td>no</td>
</tr>
<tr>
<td>source</td>
<td>client address</td>
<td>no</td>
</tr>
<tr>
<td>user</td>
<td>authenticated user</td>
<td>no</td>
</tr>
<tr>
<td>timestamp</td>
<td>system time of log event</td>
<td>no</td>
</tr>
<tr>
<td>category</td>
<td>DML/DDL/QUERY for example</td>
<td>no</td>
</tr>
<tr>
<td>type</td>
<td>API level operation</td>
<td>no</td>
</tr>
<tr>
<td>batch</td>
<td>batch id</td>
<td>yes</td>
</tr>
<tr>
<td>ks</td>
<td>keyspace</td>
<td>yes</td>
</tr>
<tr>
<td>cf</td>
<td>column family</td>
<td>yes</td>
</tr>
<tr>
<td>operation</td>
<td>textual description</td>
<td>yes</td>
</tr>
</tbody>
</table>

The textual description value for the operation field label is currently only present for CQL.

Auditing is completely separate from authorization, although the data points logged include the client address and authenticated user, which may be a generic user if the default authenticator is not overridden. Logging of requests can be activated for any or all of the list of categories described in Setting up database auditing.

**CQL logging examples**

Generally, SELECT queries are placed into the QUERY category. The INSERT, UPDATE, and DELETE statements are categorized as DML. CQL statements that affect schema, such as CREATE KEYSPACE and DROP KEYSPACE, are categorized as DDL.

**CQL USE**

```
USE dsp904;
host:/192.168.56.1|source:/192.168.56.101|user:User allow_all groups=[]|
|timestamp:1351003707937|category:DML|type:SET_KS|ks:dsp904|operation:use dsp904;
```

**CLI USE**

```
USE dsp904;
host:/192.168.56.1|source:/192.168.56.101|user:User allow_all groups=[]|
|timestamp: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
```

---

DataStax 6.7 Security Guide Latest DSE version Latest 6.7 patch: 6.7.7
Auditing database activity

```
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
```

```
DataStax Java Driver queries
```

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

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

```
host:ip-10-85-22-245.ec2.internal/10.85.22.245|source:/127.0.0.1|user:anonymous
```
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.

View events from DSE audit table

The dse_audit.audit_log table stores database activity events for all nodes that have auditing enabled with the CassandraAuditWriter logger. Use CQL to query the table and view results.

Set the replication factor for the dse_audit keyspace to at least one in all datacenters where auditing is enabled.

Audit log columns

The audit log table contains all events from nodes that have auditing is enabled. DataStax Enterprise parses the events into the following columns.

In cqlsh, use DESC TABLE dse_audit.audit_log to view table schema. The table has a compound partition key, date, node, and day_partition and a single clustering column event time.

Table 14: audit_log columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>timestamp</td>
<td>Date of the event. (Partition column 1)</td>
</tr>
<tr>
<td>node</td>
<td>inet</td>
<td>DSE node IP address. (Partition column 2)</td>
</tr>
<tr>
<td>day_partition</td>
<td>int</td>
<td>Current hour of the day * 3600, using GMT. (Partition column 3)</td>
</tr>
<tr>
<td>event_time</td>
<td>timeuuid</td>
<td>A TIMEUUID generated when the event began. (Clustering column)</td>
</tr>
<tr>
<td>authenticated</td>
<td>text</td>
<td>User name or id used to login.</td>
</tr>
<tr>
<td>batch_id</td>
<td>uuid</td>
<td>The UUID of the batch query the event was grouped with when written to Cassandra.</td>
</tr>
<tr>
<td>category</td>
<td>text</td>
<td>The event category name:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• QUERYDML</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DDL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DCL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• AUTH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UNKNOWN</td>
</tr>
<tr>
<td>consistency</td>
<td>text</td>
<td>Request consistency level.</td>
</tr>
<tr>
<td>keyspace_name</td>
<td>text</td>
<td>Keyspace name where the event occurred.</td>
</tr>
<tr>
<td>operation</td>
<td>text</td>
<td>Event description and CQL request if applicable.</td>
</tr>
</tbody>
</table>
## Auditing database activity

### Table of Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>text</td>
<td>Client IP address.</td>
</tr>
<tr>
<td>table_name</td>
<td>text</td>
<td>Table name where the event occurred if applicable.</td>
</tr>
<tr>
<td>type</td>
<td>text</td>
<td>The type of the event. See types in each category.</td>
</tr>
<tr>
<td>username</td>
<td>text</td>
<td>DSE role name of the logged in user.</td>
</tr>
</tbody>
</table>

### Example audit queries

Get all events on a particular node that occurred at:

```
SELECT * FROM dse_audit.audit_log;
```

with sample output from a DSE Graph query:

```
<table>
<thead>
<tr>
<th>Date</th>
<th>Source Address</th>
<th>Port</th>
<th>Event ID</th>
<th>Event Type</th>
<th>Event Args</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>57ff2dc0-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>57ff2f0-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>57ffca00-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>58001820-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>58001821-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>58008d50-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>58012990-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>58019ec0-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>58019ec1-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>5bb86530-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>6dfac5d0-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>6dfbb030-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>70c70530-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>70c70531-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>78fb6480-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>7aadcf70-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>7aadf680-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>7aaee0e0-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>2016-06-22</td>
<td>127.0.0.1</td>
<td>10800</td>
<td>7aaee0e1-3827-11e6-9ea9-2f93eec587f0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null</td>
<td></td>
</tr>
</tbody>
</table>
```
2016-06-22 00:00:00+0000 | 127.0.0.1 | 10800 | 8195b190-3827-11e6-9ea9-2f93ec587f0 | null
| QUERY | ONE | dse Audit |

RequestMessage{, requestID=941d2e1a-7cc9-4c80-8c28-dcc799840b7, op='eval',
processor='session', args={gremlin=g.V().count(),
aliases={g=testQSagain.g}, session=d179c734-813f-4a3e-89d6-bd756b4f65f7, bindings={},
manageTransaction=true, batchSize=64)} | 127.0.0.1:60647 | null |

GRAPH_TINKERPOP_TRAVERSAL | unknown
SELECT "community_id", "member_id" FROM "test"."meal_p" WHERE "~~vertex_exists" = ?
LIMIT ? ALLOW FILTERING;
| /0.0.0.0 | meal_p |

CQL_SELECT | system
SELECT "community_id", "member_id" FROM "test"."ingredient_p" WHERE "~~vertex_exists" = ?
LIMIT ? ALLOW FILTERING;
| /0.0.0.0 | ingredient_p |

CQL_SELECT | system
SELECT "community_id", "member_id" FROM "test"."author_p" WHERE "~~vertex_exists" = ?
LIMIT ? ALLOW FILTERING;
| /0.0.0.0 | author_p |

CQL_SELECT | system
SELECT "community_id", "member_id" FROM "test"."book_p" WHERE "~~vertex_exists" = ?
LIMIT ? ALLOW FILTERING;
| /0.0.0.0 | book_p |

CQL_SELECT | system
SELECT "community_id", "member_id" FROM "test"."reviewer_p" WHERE "~~vertex_exists" = ?
LIMIT ? ALLOW FILTERING;
| /0.0.0.0 | reviewer_p |

CQL_SELECT | system
select * from audit_log;
| /127.0.0.1 | audit_log |

CQL_SELECT | anonymous
SELECT last_updated FROM "test_system".shared_data WHERE dataspace = ?;
| /0.0.0.0 | shared_data |

CQL_SELECT | system
select * from audit_log;
### Auditing database activity

<table>
<thead>
<tr>
<th>Query string not found in prepared statement cache</th>
<th>bind variable values unavailable</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQL_SELECT</td>
<td>system</td>
</tr>
<tr>
<td>select * from audit_log ;</td>
<td>/0.0.0.0</td>
</tr>
<tr>
<td>CQL_SELECT</td>
<td>anonymous</td>
</tr>
<tr>
<td>SELECT last_updated FROM &quot;test_system&quot;.shared_data WHERE dataspace = ?;</td>
<td>/0.0.0.0</td>
</tr>
<tr>
<td>CQL_PREPARE_STATEMENT</td>
<td>system</td>
</tr>
<tr>
<td>Query string not found in prepared statement cache</td>
<td>bind variable values unavailable</td>
</tr>
<tr>
<td>CQL_SELECT</td>
<td>system</td>
</tr>
<tr>
<td>Query string not found in prepared statement cache</td>
<td>bind variable values unavailable</td>
</tr>
<tr>
<td>CQL_SELECT</td>
<td>system</td>
</tr>
<tr>
<td>select * from audit_log ;</td>
<td>/127.0.0.1</td>
</tr>
<tr>
<td>CQL_SELECT</td>
<td>anonymous</td>
</tr>
<tr>
<td></td>
<td>/127.0.0.1</td>
</tr>
</tbody>
</table>
Chapter 7. Transparent data encryption

DataStax recommends encrypting sensitive configuration properties in the dse.yaml and cassandra.yaml files.

About Transparent Data Encryption

Protects sensitive at-rest data using a local encryption key file or remotely stored and managed KMIP encryption key.

- **Configuration file properties**: Protects LDAP search password, LDAP truststore password, SSL truststore passwords.
- **System resources**: Protected properties using the same key for the system.batchlog and system.paxos tables, hint files and commit logs.
- **Database tables**: Protects all data in the table except for the primary key columns. Different tables can use different keys.

SSTable data files are immutable once they have been flushed to disk and are only encrypted during the write to disk. To encrypt existing data, use the `nodetool upgradestables` with the `-a` option to rewrite the tables to disk with encryption.

Do NOT store sensitive data in Primary Key columns. Transparent data encryption does not encrypt partition keys and clustering columns.

Data that is not encrypted

DSE does not encrypt the following:

- Table partition key and clustering columns
- Database files other than the commit log and SSTable data files
- DSEFS data files

Requirements

To use the DataStax Enterprise (DSE) Transparent Data Encryption (TDE) feature, enable the Java Cryptography Extension (JCE).

Starting in JDK 8u161, JCE Unlimited is enabled by default. Refer to the Release Notes for JDK 8u161.

When using TDE secure local file system. Encryption keys are stored remotely with KMIP encryption or locally with on-server encryption.

TDE limitations and recommendations

The following utilities cannot access encrypted data, but will operate on all unencrypted data.

Compression and encryption introduce performance overhead.

Data is not directly protected by TDE when you access the data using the following utilities.

<table>
<thead>
<tr>
<th>Utility</th>
<th>Reason utility is not encrypted</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodetool</td>
<td>Uses only JMX, so data is not accessed.</td>
</tr>
<tr>
<td>sstableloader</td>
<td>Operates directly on the SSTables.</td>
</tr>
<tr>
<td>sstablescrub</td>
<td>Operates directly on the SSTables.</td>
</tr>
</tbody>
</table>

DataStax 6.7 Security Guide Latest DSE version Latest 6.7 patch: 6.7.7
Transparent data encryption

<table>
<thead>
<tr>
<th>Utility</th>
<th>Reason utility is not encrypted</th>
</tr>
</thead>
<tbody>
<tr>
<td>sstableutil</td>
<td>Operates directly on the SSTables.</td>
</tr>
<tr>
<td>sstableverify</td>
<td>Operates directly on the SSTables.</td>
</tr>
</tbody>
</table>

Lifecycle Manager (LCM) is not compatible when `config_encryption_active` is `true` in DSE and OpsCenter. For LCM limitations, see Encrypted DSE configuration values.

**TDE options**

To get the full capabilities of TDE and to ensure full algorithm support, enable JCE.
Starting in JDK 8u161, JCE Unlimited is enabled by default. Refer to the Release Notes for JDK 8u161.

**Configuring local encryption**

Use locally-stored symmetric encryption keys to protect the following assets:

- **Configuration file property values**: LDAP search, LDAP truststore, and SSL truststore passwords.
- **Sensitive system resources**: System batchlog and paxos tables, hint files, and commit logs.
- **Table data**: Any table.
- **Search indexes**: All search indexes.

**Local encryption guidelines**

When you encrypt tables, hint files, commit logs, and configuration properties using a local key:

- Create any number of local encryption keys using the `dsetool createsystemkey` command.

  ```
  # Tables can use different encryption keys.
  DataStax Enterprise creates a unique key for each combination of cipher algorithm, key strength, and external local encryption key used in a table definition and stores it in the `dse_system.encrypted_keys` table. The local encryption key file is used to encrypt/decrypt the table key.
  # Configuration properties use the same key file that is defined by the `config_encryption_key_name` property.
  # All system resources use the same key file. (The file is not selectable.)
  # Distribute all local encryption key files cluster-wide. Put keys on all nodes in the same folder and define the location in the `system_key_directory` property in `dse.yaml`.
  # Ensure that the DataStax Enterprise account owns the `system_key_directory` and has read/write permission.
  ```

  To change an encryption key, see Rekeying existing data.

**Setting up local encryption keys**

Use `dsetool createsystemkey` to generate local encryption/decryption key files.

  To change an encryption key, see Rekeying existing data.

**Setting up local encryption keys for production environments**

After installing DSE, create a local encryption key file, distribute it to the same location on all nodes in the cluster, and update the `dse.yaml system_key_directory` and `config_encryption_key_name` properties.

  To change an encryption key, see Rekeying existing data.

**Prerequisites**: To ensure support for all encryption algorithms, enable JCE.
Starting in JDK 8u161, JCE Unlimited is enabled by default. Refer to the Release Notes for JDK 8u161.

1. If the directory does not exist, create the /conf directory based on your DataStax Enterprise (DSE) installation type:
   - Package installation
     ```
     $ /etc/dse/conf
     ```
   - Tarball installation
     ```
     $ installation_location/resources/dse/conf
     ```

2. Configure the file name and the location of the encryption key in the dse.yaml file:
   a. Set `system_key_directory` property to the path where you want to store the encryption keys.
      ```yaml
      system_key_directory: /etc/dse/conf
      ```
   b. Change the directory owner to the DSE account and ensure that the DSE account has read/write permissions.
   c. Set the `config_encryption_key_name` to the `key_name`. The default name is `system_key`.
      ```yaml
      config_encryption_key_name: system_key
      ```

      Encryption key files can have any valid Unix name.

3. Go to the `system_key_directory` and then create an encryption key using the dsetool createsystemkey command:
   For example:
   ```
   $ cd /etc/dse/conf
   $ dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 key_name
   ```

   Where `key_name` is the name of the key file to create. If no file name is specified, the key file is named `system_key`.

   Encryption key files can have any valid Unix name.

   DSE supports the following JCE cipher algorithms and corresponding length:

   **cipher_algorithm[/mode/padding]**
   
   DSE supports the following JCE cipher algorithms:
   
   - AES/CBC/PKCS5Padding (valid with length 128, 192, or 256).
   - AES/ECB/PKCS5Padding (valid with length 128, 192, or 256)
   - DES/CBC/PKCS5Padding (valid with length 56)
   - DESede/CBC/PKCS5Padding (valid with length 112 or 168)
   - Blowfish/CBC/PKCS5Padding (valid with length 32-448)
   - RC2/CBC/PKCS5Padding (valid with length 40-128)
Transparent data encryption

Default value: AES/CBC/PKCS5Padding (with length 128).

If `config_encryption_active` is set to `true` in dse.yaml, a warning is generated, but the system key is still successfully generated.

4. Copy the key file to all other nodes in the cluster. Put keys on all nodes in the same directory.

5. Update the `system_key_directory` and `config_encryption_key_name` in dse.yaml.

   dsetool reads current values from the dse.yaml. A restart is not required to continue setting up encryption.

6. Ensure that the DSE account owns the key files and has read/write access on them. If necessary, change the ownership of the file to the DSE user.

   $ chown cassandra /etc/dse/conf/system_key

Setting up local encryption keys to embed in installation package for development environments

You can create a local encryption/decryption key file that can be embedded in a distribution (tarball). In development environments, this distribution package can then be used by other users. This strategy is especially helpful when using scripts with IT automation tools such as Ansible.

The current user must have write permission to the directory where you want to generate the key files.

1. Specify the key file output directory when you create the encryption key with the dsetool createsystemkey command:

   For example:

   $ dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 -d /home/jane/keys

   Result: A key file /home/jane/keys/system_key is created.

2. In the distribution tarball, create a directory for the system key file. Use the default location (/etc/dse/conf) or add a new location.

3. If you used a new location, update the `system_key_directory` property in dse.yaml as appropriate.

Encrypting configuration file properties

Configure DSE to use a local encryption key to decrypt properties in the configuration file. Use passwords encrypted with the local key for the following properties:

- **dse.yaml LDAP values:**
  
  ldap_options.search_password
  ldap_options.truststore_password

  Use plain text for the KMIP keystore or truststore passwords.

- **cassandra.yaml SSL values:**
  
  server_encryption_options.keystore_password
  server_encryption_options.truststore_password
  client_encryption_options.keystore_password
**Transparent data encryption**

**Prerequisites:** Complete the key setup described in Setting up local encryption keys.

When using a local encryption key file, set the location `system_key_directory` and ensure that the key file is owned by the account running DSE.

1. For each property, replace plain text passwords with encrypted passwords returned by running the `dsetool encryptconfigvalue` command:
   - a. Encrypt the password:
     
     ```
     $ dsetool encryptconfigvalue
     ```

     ```
     Using system key system_key
     Enter value to encrypt:
     Enter again to confirm:
     Your encrypted value is:
     +Vj5oHCR/jqfA+OJE2m8zA==
     ```
   - b. Replace the old value with the new value in the configuration file, for example the SSL truststore password in the `cassandra.yaml`:
     
     ```
     $ truststore_password: +Vj5oHCR/jqfA+OJE2m8zA==
     ```

     After the configuration file property encryption is enabled, DSE startup fails if any of the protected properties are not encrypted.

2. In `dse.yaml`, enable configuration file property encryption:
   - a. Set `config_encryption_active` to true.
     
     ```
     config_encryption_active: true
     ```

     When set to true, the configuration values must be encrypted or commented out.

     Lifecycle Manager (LCM) is not compatible when `config_encryption_active` is true in DSE and OpsCenter. For LCM limitations, see Encrypted DSE configuration values.

   - b. Set the local key encryption filename:
     
     ```
     config_encryption_key_name: key_filename
     ```

3. Update the `dse.yaml` and `cassandra.yaml` on all nodes in the cluster.

4. Set up encryption for system resources, see Encrypting system resources.

5. Perform a rolling restart.

**Encrypting system resources**

Encrypt data in the `system.batches` and `system.paxos` tables, hint files, and commit logs using a local encryption key.
Transparent data encryption

If tracing is enabled, the system_traces keyspace also contains sensitive data; encrypt tables in the system_traces keyspace following the instructions in Encrypting tables.

Prerequisites: Complete the key setup described in Setting up local encryption keys.

When using a local encryption key file, set the location system_key_directory and ensure that the key file is owned by the account running DSE.

1. In the dse.yaml file, configure encryption settings for system tables, the commit log, and the hints files.

   ```yaml
   system_info_encryption:
     enabled: true
     cipher_algorithm: cipher_name
     secret_key_strength: key_length
     chunk_length_kb: default_table_chunk_size
   ```

   • Required. Set enabled to true.

   • Optional. Configure the type of encryption key to use:

     # cipher_algorithm: Set the name of a supported JCE cipher algorithm to use. For a list of support
     algorithms, see cipher_algorithm

     # secret_key_strength: Specify the key length.

     # chunk_length_kb: Size of SSTables. The default (64) is used if the option is excluded.

     When these properties are set, DSE only uses a key that matches; if no matching key exists, start up fails.

2. Perform a rolling restart of DSE.

3. To encrypt existing data, run `nodetool upgradesstables -a system batchlog paxos` on all nodes in the cluster.

Encrypting tables

Configure Transparent Data Encryption (TDE) to protect all data in a table, except for the primary key columns. Different tables can use different keys.

Primary keys are stored in plain text. Do not put sensitive information in partition key or clustering columns.

Two keys are used for table encryption:

• **Local encryption key**: Encrypts/decrypts internal table encryption key values.

• **Table encryption key**: DSE creates a single key entry in the dse_system.encrypted_keys table for each cipher algorithm, key strength, and local encryption key combination that is defined for table encryption.

   Tables with the same encryption settings use the same encryption key.

Data is encrypted when written to SSTables on disk. Applications can read and write to SSTables that use different encryption algorithms or no encryption at all.

Creating a table with encryption and compression

DataStax recommends creating tables with both encryption and compression enabled, using EncryptingLZ4Compressor as the encryption class.

Prerequisites: Complete the key setup described in Setting up local encryption keys.

When using a local encryption key file, set the location system_key_directory and ensure that the key file is owned by the account running DSE.
1. Change to the keyspace where you want to create the table. The following examples use test as the keyspace name:

   $ cqlsh

   USE test;

2. Create the table with encryption and compression.

   The following example encrypts a table named encryption_test using the DESede algorithm, with a key length of 112. Data is compressed using the EncryptingLZ4Compressor compressor.

   A local encryption key called system_key must exist in the directory specified by system_key_directory. This file was created when Setting up local encryption keys.

   If the DSE account does not have read/write permission or the file is missing, an error message Failed to initialize Encryptor displays.

   ```
   CREATE TABLE test.encryption_test (d int PRIMARY KEY) WITH COMPRESSION = {
     'class': 'EncryptingLZ4Compressor',
     'cipher_algorithm' : 'DESede/CBC/PKCS5Padding',
     'secret_key_strength' : 112,
     'system_key_file' : 'system_key'
   };
   ```

   See Table encryption options and syntax for more information.

3. To change the encryption settings, use the ALTER TABLE command and specify the settings to modify.

   The following command changes the encryption key used to encrypt the table data, and modifies the key strength.

   ```
   ALTER TABLE test.encryption_test WITH COMPRESSION = {
     'class': 'EncryptingLZ4Compressor',
     'cipher_algorithm' : 'AES/ECB/PKCS5Padding',
     'secret_key_strength' : 128,
     'system_key_file' : 'system_key'
   };
   ```

4. If you changed encryption settings, run the following command on all nodes in the cluster to rewrite the SSTables using the new encryption key:

   ```
   $ nodetool upgradestables -a test encryption_test
   ```

**Table encryption options and syntax**

When a table definition uses an encryption class, all table data except for primary keys is encrypted with a key entry from the dse_system.encrypted_keys table. If no keys match the cipher_algorithm, secret_key_strength, and system_key_file settings, a new key is created and added to the table.

The following syntax only shows encryption options. All other compression options, such as chunk_length_in_kb, are also available.

**Syntax**

```
COMPRESSION = {
  'class' : 'encryption_class',
  'cipher_algorithm' : 'cipher_algorithm_type',
  'secret_key_strength' : length
};
```
Transparent data encryption

'system_key_file': 'key_filename']

Options

encryption_class
Specifies the encryption type. Use one of the class names from the following table. (Required)

<table>
<thead>
<tr>
<th>Name</th>
<th>Encrypts</th>
<th>Compresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryptor [1]</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>EncryptingLZ4Compressor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>EncryptingDeflateCompressor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>EncryptingSnappyCompressor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

[1] When using the Encryptor class, specify a larger young generation heap (the -Xmn parameter) to improve garbage collection (GC). For example, set the size to: -Xmn1600M when running cassandra-stress.

cipher_algorithm_type
Sets the type of encryption key. DSE supports the following JCE algorithms and corresponding length.

cipher_algorithm[/mode/padding]
DSE supports the following JCE cipher algorithms:

- AES/CBC/PKCS5Padding (valid with length 128, 192, or 256).
- AES/ECB/PKCS5Padding (valid with length 128, 192, or 256)
- DES/CBC/PKCS5Padding (valid with length 56)
- DESede/CBC/PKCS5Padding (valid with length 112 or 168)
- Blowfish/CBC/PKCS5Padding (valid with length 32-448)
- RC2/CBC/PKCS5Padding (valid with length 40-128)

Default value: AES/CBC/PKCS5Padding (with length 128).

length
Specifies the length of the encryption key.
Default: 128. (Optional)

data_key_filename
Specifies the file name of the local encryption key used to encrypt the table key. Local keys are specified in system_key_directory.
Default: system_key. (Optional)

Rekeying existing data
Create a new local encryption key, change the table key filename, and re-encrypt the SSTables using the new key. When changing the system key, all existing data must be re-encrypted before removing the old key.

Prerequisites: These steps require the following privileges:

- DataStax Enterprise node administrator or superuser account with read/write/modify permission on DSE resources and configuration directories.
- If DSE database authentication and authorization is enabled, a database account with ALTER TABLE permission on the encrypted tables.

1. Back up SSTables.
2. Create a new local encryption key and distribute to nodes in the cluster:
a. Go to the key file directory, which is defined in the system_key_directory setting of the dse.yaml.

b. Use dsetool createsystemkey to create a new system key in the key file directory:

```
dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 new_system_key
```

Both the new and old key are required for re-encryption. Do NOT remove the old key until after changing the table schema and rekeying the existing SSTables.

c. Verify that the database account has read and write access to the files:

```
la -l
```

In this example DSE is the account that runs the database.

```
-rw------- 1 dse dse 50 May 19 10:54 system_key
-rw------- 1 dse dse 50 May 19 11:20 new_system_key
```

d. Copy the new key to the system key directory on all nodes in the cluster.

Ensure that the new key has the correct permissions.

3. Change the key filename in the table schemas:

a. Get a list of all the encrypted tables that you want to change.

Use the DESC KEYSPACE keyspace_name cqlsh command to show all table properties in a keyspace.

b. For each table where you want use the new encryption key, change the key filename in the table schema:

```
ALTER TABLE keyspace_name.table_name
    WITH compression = { 'sstable_compression' : 'EncryptingSnappyCompressor',
    'cipher_algorithm' : 'AES/ECB/PKCS5Padding',
    'secret_key_strength' : 128,
    'chunk_length_kb' : 128,
    'system_key_file': 'new_system_key'
}
```

c. Ensure that the schema change has replicated to all nodes in the cluster:

```
nodetool describecluster
```

The following example shows a small three node cluster where all three nodes have the same schema. If any of the nodes have a different schema, wait until the schema changes are propagated before going onto the next step.

```
Name: Cluster1
Snitch: org.apache.cassandra.locator.DynamicEndpointSnitch
Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
Schema versions:
  25743512-6b6f-3f76-96bc-1122d441f539: [node1_IP, node2_IP, node3_IP]
```

4. Use nodetool upgradesstables to rewrite the encrypted SSTables using the new key. Run the following command on every node in the cluster:
Transparent data encryption

- Target only specific tables:

```bash
$ nodetool upgradesstables --include-all-sstables keyspace_name table_name
```

- Target specific keyspace:

```bash
$ nodetool upgradesstables --include-all-sstables keyspace_name
```

- All keyspaces and tables:

```bash
$ nodetool upgradesstables --include-all-sstables
```

5. After completing the above steps, remove the old key and ensure that the old key is not used for any tables or configuration file property encryption.

The old key is required to access the backed up SSTables created in the first step.

Troubleshooting encryption key errors

Failed to initialize Encryptor

When creating or altering a table to use a local key, the commands fails. For example, creating a test table with LZ compression and encryption:

```cql
CREATE TABLE test.encryption_test (a int primary key) WITH COMPRESSION = {
    'class': 'EncryptingLZ4Compressor',
    'cipher_algorithm': 'AES/ECB/PKCS5Padding',
    'secret_key_strength': 256,
    'system_key_file': 'AES-256'
};
```

The following error occurs:

```
ConfigurationException: EncryptingLZ4Compressor.create() threw an error:
java.lang.RuntimeException Failed to initialize Encryptor
```

Solution

1. Verify that the account running the DSE database is the owner of the encryption key file on the local system:

```bash
$ ls -l /etc/dse/conf
```

In this case, the account dse only has read/write access to the `system_key` file.

```
total 8
-rwx----- 1 joe joe 70 Aug  8 15:48 AES-256
-rwx----- 1 joe joe 50 Aug  2 15:06 system_key
```

2. Change the ownership of the file to the DSE user, and ensure that the file has read/write permissions.

```bash
$ chown cassandra /etc/dse/conf/system_key
```

3. Rerun the CQL command.
Configuring KMIP encryption

Set up encryption using keys from a KMIP (Key Management Interoperability Protocol) host to protect sensitive configuration file properties, system resources, and tables.

DataStax recommends using KMIP key server security policies to limit the number of nodes in the cluster that can remotely manage keys, due to the risks associated with expiring, revoking, and destroying keys.

Use OpsCenter to monitor KMIP server status. See Configuring an alert for KMIP errors.

Adding a KMIP host

DataStax Enterprise supports using encryption keys from one or more remote KMIP hosts to encrypt/decrypt table data and/or sensitive properties in the dse.yaml and cassandra.yaml configuration files. Follow these steps to add a KMIP server information to the list of available hosts.

DataStax recommends limiting the number of nodes that can remotely manage KMIP keys using a security policy on the KMIP host.

Perform all steps on every node in the cluster.

1. Set up KMIP agents and registered DSE with the KMIP service:

   Refer to the KMIP key provider documentation for detailed steps.

   a. Download and install the KMIP agent.

   b. Connect to the KMIP host.

   c. Register the DSE node.

   d. Locate the SSL key pair generated by the KMIP agent.

2. Convert the key pair from PEM to a DSE compatible JKS format:

   a. Secure the KMIP agent private key files by removing read access for all users, for example the Vormetric DSM agents creates two files named kmip-key.pem and kmip-host_name.pem.

   b. Copy both keys to another directory, such as your home directory.

   c. Generate a PKCS12 format file from the PEM files:

      ```
      $ openssl pkcs12 -export -out kmip_keystore.p12 -inkey kmip-key.pem -in kmip-host_name.pem
      ```

      Where kmip_keystore.p12 is the output file name and kmip-host_name.pem is part of the key pair created by the KMIP agent.

   d. Create a JKS keystore:

      ```
      $ keytool -importkeystore -destkeystore kmip_keystore.jks -srcstoretype PKCS12 -srcstore kmip_keystore.p12
      ```

      Where

      - kmip_keystore.jks is the keystore file name that is created
      - kmip_keystore.p12 is the PKC12 file generated in the previous step

      Enter a password for the keystore at the prompt and fill out the host information.
e. Install the KMIP root certificate into the JKS truststore:

\[
\text{
\$ keytool -import -alias kmipCA -file kmip-host_CA.pem -keystore kmip_truststore.jks
}
\]

Enter a password for the truststore at the prompt and fill out the host information.

f. Move the keystore and truststore to a directory accessible by DSE and change the file to allow the DSE account read/write access.

g. Delete or secure the files used to create the keystore and truststore.

3. Add the host details to the `kmip_hosts` section of the `dse.yaml`:

```
kmp_hosts:
  kmip_group_name:
    hosts: FQDN[, FQDN, ...]
    keystore_path: /etc/dse/conf/kmip_keystore.jks
    keystore_type: jks
    keystore_password: password
    truststore_path: /etc/dse/conf/kmip_truststore.jks
    truststore_type: jks
    truststore_password: password
    key_cache_millis: N
    timeout: N
    protocol: protocol
    cipher_suites: supported_cipher
```

- Required settings:
  
  # kmip_group_name: User-defined group name that identifies the KMIP host in DSE related commands.
  
  # hosts: Comma separated list of fully qualified domain names (FQDN) of KMIP hosts. DSE tries the hosts in the order listed.
  
  # keystore_path: Location of the keystore created in 2.

  # keystore_type: jks Keystore format. Must be set to jks.

  # keystore_password: Password of the keystore file created in 2.

  # truststore_path: Location of the truststore file created in 2.

  # truststore_type: jks Truststore format. Must be set to jks.

  # truststore_password: Password of the truststore file created in 2.

- Optional settings:

  # key_cache_millis: N where N is the interval at which DSE refreshes the key cache on the node in milliseconds. The default is 300000 (five minutes).

  # timeout: N where N is the socket timeout in milliseconds. The default is 1000.

  # protocol: protocol for communicating between the node and KMIP key server. When not specified, JVM default is used.

  # cipher_suites: supported_cipher for communicating between the node and KMIP key server. When not specified, JVM default is used.
4. Verify that the node can connect to the KMIP host by listing encryption keys on the remote KMIP server:

```
$ dsetool managekmip list kmip_group_name
```

dsetool picks up dse.yaml changes without requiring a restart. If problems connecting to the KMIP server occur, see Troubleshooting KMIP connections.

5. Repeat steps on all nodes in the cluster.

### Encrypting configuration file properties

Configure DSE to use a KMIP encryption key to decrypt sensitive configuration properties. Use passwords encrypted with the KMIP key for the following properties:

- **dse.yaml LDAP values:**
  ```
  ldap_options.search_password
  ldap_options.truststore_password
  ```

  Use plain text for the KMIP keystore or truststore passwords.

- **cassandra.yaml SSL values:**
  ```
  server_encryption_options.keystore_password
  server_encryption_options.truststore_password
  client_encryption_options.keystore_password
  client_encryption_options.truststore_password
  ```

DataStax Enterprise caches encryption keys from the KMIP host and refreshes the cache at the interval set by the `key_cache_millis` property; the default setting is five minutes.

### Prerequisites:

Complete the steps in Adding a KMIP host.

If any of the defined KMIP groups are not available, DSE startup fails.

1. Back up the configuration files.

2. Get the URL of the KMIP encryption key:

   - To create a new key and get the URL:
     ```
     $ dsetool createsystemkey 'AES/ECB/PKCS5' 128 -k kmip_group_name
     ```

     The example output shows the URL for the `host_name` in the dse.yaml with ID: 02-1655.

     kmip://host_name/02-1655

   - To use an existing KMIP key, the URL syntax is `kmip://kmip_group_name/ID`. To look up the key ID:
     ```
     $ dsetool managekmip list kmip_group_name
     ```

     For example, the `host_name` has the following keys:

     Keys on `host_name`:
## Transparent data encryption

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>02-449</td>
<td>82413ef3-4fa6-4d4d-9dc8-71370d731fe4_0</td>
<td>AES/CBC/PKCS5</td>
</tr>
<tr>
<td></td>
<td>Deactivated</td>
<td>Mon Apr 25 20:25:47 UTC 2016</td>
</tr>
<tr>
<td>02-540</td>
<td>0eb2277e-0acc-4adb-9241-1dd84ddee691c_0</td>
<td>AES</td>
</tr>
<tr>
<td></td>
<td>Active</td>
<td>Tue May 31 12:57:59 UTC 2016</td>
</tr>
</tbody>
</table>

The URL of the second key in the list is `kmip://host_name/02-540`.

3. **Configure** system property encryption **settings** in the `dse.yaml`.
   a. **Enable** system property encryption:

   ```
   config_encryption_active: true
   ```

   b. **Set the URL of the KMIP key used to decrypt properties**:

   ```
   config_encryption_key_name: KMIP_key_URL
   ```

   Where `KMIP_key_URL` format is `kmip://kmip_group_name/key-id`, for example `kmip://host_name/02-1655`.

4. **For each property**, replace plain text passwords with encrypted passwords returned by running the `dsetool encryptconfigvalue` command:
   a. **Encrypt the password**:

   ```
   $ dsetool encryptconfigvalue
   ```

   Using system key `system_key`

   Enter value to encrypt:
Enter again to confirm:

   Your encrypted value is:

   ```+Vj5oHCR/jqfA+OJE2m8zA==```

   b. **Replace the old value with the new value in the configuration file**, for example the SSL truststore password in the `cassandra.yaml`:

   ```
   $ truststore_password: +Vj5oHCR/jqfA+OJE2m8zA==
   ```

   After the configuration file property encryption is enabled, DSE startup fails if any of the protected properties are not encrypted.

5. **Optional. Set up system resource encryption**.

6. **Perform a rolling restart**.

### Encrypting system resources

Use a KMIP key to encrypt the `system.batchlog` and `system.paxos` tables, hint files and commit logs.

**Prerequisites:**
Complete the steps in Adding a KMIP host.

If any of the defined KMIP groups are not available, DSE startup fails.

1. In the `dse.yaml` file, configure encryption settings for system tables, the commit log, and the hints files.

   ```yaml
   system_info_encryption:
     enabled: (true | false)
     cipher_algorithm: cipher_name
     secret_key_strength: length
     key_provider: KmipKeyProviderFactory
     kmip_host: kmip_group_name
     chunk_length_kb: 64
   ```

   • Required properties:
     - `enabled`: Set to `true`. On the next startup, system resources are encrypted. If the system tables have existing data, use `nodetool upgradesstables` to apply encryption.
     - `key_provider`: Set to `KmipKeyProviderFactory`.
     - `kmip_host`: Use the group name from the `kmip_hosts` section.

   • Optional. To ensure that KMIP generates a compatible key, configure the type of encryption key to use:
     - `cipher_algorithm`: Set the name of a supported JCE cipher algorithm to use. DSE supports the following algorithms:
       ```
       | cipher_algorithm | secret_key_strength |
       |------------------|--------------------|
       | AES              | 128, 192, or 256   |
       | DES              | 56                 |
       | DESede           | 112 or 168         |
       | Blowfish         | 32-448             |
       | RC2              | 40-128             |
       ```
     - `secret_key_strength`: Specify the key length.
     - `chunk_length_kb`: Configures chuck size for SSTables. The default (64) is used if the option is excluded.

     When these properties are set, DSE only uses a key that matches; if no matching key exists, start up fails.

2. Perform a rolling restart.

3. To encrypt existing data, run `nodetool upgradesstables -a system batchlog paxos` on all nodes in the cluster.

**Encrypting table data**

Encrypt data stored in a table data using a KMIP key.

Primary keys are stored in plain text. Do not put sensitive information in partition key or clustering columns.

**Prerequisites:**

Complete the steps in Adding a KMIP host.
Transparent data encryption

If any of the defined KMIP groups are not available, DSE startup fails.

1. To create a new encrypted table using a key from a KMIP server:

   • Encryption without compression:

   ```
   CREATE TABLE customers
   ...
   WITH COMPRESSION =
   { 'class': 'Encryptor',
     'key_provider': 'KmipKeyProviderFactory',
     'kmip_host': 'kmip_group_name'
     ['key_namespace' = 'kmip_namespace'],
     '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_group_name' specifies the user-defined KMIP key server group name defined in the kmip_hosts section of the dse.yaml file.

   # 'kmip_host': 'kmip_group_name' ['key_namespace' = 'kmip_namespace'] specify an optional KMIP namespace. Using namespaces allows you to granularly manage keys on a per table or keyspace basis.

   • Compression and encryption:

   ```
   CREATE TABLE customers
   ...
   WITH COMPRESSION =
   { 'class': 'EncryptingDeflateCompressor',
     'key_provider': 'KmipKeyProviderFactory',
     'kmip_host': 'kmip_group_name',
     'cipher_algorithm': 'AES/ECB/PKCS5Padding',
     'secret_key_strength': 128 };
   ```

2. To encrypt an already existing table:

   a. Change the table compression settings:

   • Encryption without compression:

   ```
   ALTER TABLE customers
   ...
   WITH COMPRESSION =
   { 'class': 'Encryptor',
     'key_provider': 'KmipKeyProviderFactory',
     'kmip_host': 'kmip_group_name'
     ['key_namespace' = 'kmip_namespace'],
     '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_group_name' specifies the user-defined KMIP key server group name defined in the kmip_hosts section of the dse.yaml file.
Transparent data encryption

# ['key_namespace' = 'kmip_namespace'] specify an optional KMIP namespace. Using namespaces allows you to granularly manage keys on a per table or keyspace basis.

- Compression and encryption:

```sql
ALTER TABLE customers
...
WITH COMPRESSION =
{ 'class': 'EncryptingDeflateCompressor',
'key_provider': 'KmipKeyProviderFactory',
'kmip_host': 'kmip_group_name',
'cipher_algorithm': 'AES/ECB/PKCS5Padding',
'secret_key_strength': 128 };
```

b. Encrypt existing data on all nodes in the cluster:

```
$ nodetool upgradesstables -a [keyspace_name [table_name[ tablename]...]
```

Expanding an encryption key

Security policies generally limit the amount of time an encryption key is in use; this section describes how to expire a key without re-encrypting the exiting data. After a key expires, it is no longer used to encrypt new data, but is still used to decrypt existing data.

To change the key used for both encryption and decryption, see Rekeying tables using a new key.

1. Get a list of the available keys and states from the KMIP server:

```
$ dsetool managekmip list kmip_group_name
```

For example, a host named vormetricgroup that has two keys:

<table>
<thead>
<tr>
<th>Keys on vormetricgroup:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>Activation Date</td>
</tr>
<tr>
<td>Namespace</td>
</tr>
<tr>
<td>02-449</td>
</tr>
<tr>
<td>Mon Apr 25 20:25:47 UTC 2016</td>
</tr>
<tr>
<td>02-540</td>
</tr>
<tr>
<td>Tue May 31 12:57:59 UTC 2016</td>
</tr>
</tbody>
</table>

DSE supports one or more KMIP hosts. Each KMIP host is defined under a user-defined `kmip_group_name` in the `kmip_hosts` section of the `dse.yaml`.

2. Expire the key:
Transparent data encryption

- Immediately expire the key:

```
$ dsetool managekmip expirekey kmip_group_name key_id
```

- Schedule an expiration date:

```
$ dsetool managekmip expirekey kmip_group_name key_id datetime
```

After the key expires, the database gets a new key for encryption the next time it refreshes the key cache (`key_cache_millis`); the default setting is five minutes. Expired keys are still available to decrypt data.

3. Optionally, force a refresh of the DSE key cache by performing a rolling restart.

Rekeying tables using a new key

Change the encryption key that is used for both encrypting new data and decrypting the existing data. Use these steps to secure the data after an event that potentially compromised an encryption key, such as a change in security administration staff. Before destroying the old key, revoke the compromised KMIP key, wait for the database key cache refresh, and then re-encrypt existing SSTables with the new key.

The database caches the encryption keys and refreshes the cache at an interval set by the `key_cache_millis` (default setting is 5 minutes). To get a new key, either wait for the key cache refresh interval or perform a rolling restart.

The dsetool managekmip commands proxies KMIP commands to the corresponding host; the commands affect encryption keys as follows:

- **expirekey**: Database stops using the key for encryption at the specified time and continues to use the expired key to decrypt existing data. Data re-keying is not required.

  Use this command to satisfy security policies that require periodically switching the encryption key.

- **revoke**: Permanently disables the key on the KMIP server. Database can no longer use the key for encryption, but continues to use the key for decryption of existing data. Re-encrypt existing data before completely removing the key from the KMIP server.

  Use this command as the first step when replacing a compromised key.

- **destroy**: Completely removes the key from the KMIP server. Database can no longer use the key for encryption or decryption. Existing data that has not been re-encrypted becomes inaccessible.

  Use this command only after revoking a key and re-encrypting existing data.

1. Back up SSTables.

2. Revoke the compromised key using the `dsetool managekmip revoke`:

   a. Get the ID of the KMIP encryption key you want to revoke from the KMIP server:

```
$ dsetool managekmip list kmip_groupname
```

The following is an example of a KMIP server that has two keys, active and deactivated.

<table>
<thead>
<tr>
<th>ID</th>
<th>Namespace</th>
<th>Activation Date</th>
<th>Name</th>
<th>Cipher</th>
<th>Creation Date</th>
<th>Protect Stop Date</th>
</tr>
</thead>
</table>
DSE supports one or more KMIP hosts. Each KMIP host is defined under a user-defined `kmip_group_name` in the `kmip_hosts` section of the `dse.yaml`.

b. Revoke the key you want to replace using the ID:

```
$ dsetool managekmip expirekey kmip_groupname key_id
```

Revoking permanently deactivates the key on the KMIP server. When the key cache refreshes a new key for encryption/decryption is automatically created. Revoked keys are used to decrypt existing data. DO NOT destroy the revoked key until after re-encrypting the existing data.

c. Verify that the key State is Deactivated.

```
$ dsetool managekmip list kmip_groupname
```

The following is an example of a KMIP server that has two keys:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Cipher</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>02-449</td>
<td>82413ef3-4fa6-4d4d-9dc8-71370d731fe4_0</td>
<td>AES/CBC/PKCS5</td>
<td>Deactivated</td>
</tr>
<tr>
<td></td>
<td>02-449</td>
<td>Mon Apr 25 20:25:47 UTC 2016</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>02-540</td>
<td>0eb2277e-0acc-4adb-9241-1dd84de691c_0</td>
<td>AES</td>
</tr>
<tr>
<td></td>
<td>02-540</td>
<td>Tue May 31 12:57:59 UTC 2016</td>
<td>n/a</td>
</tr>
</tbody>
</table>

3. Refresh the database key cache using one of the following methods:

- Wait the amount of time specified in the `key_cache_millis` setting before continuing to the next step.
- Perform a rolling restart, see Starting and stopping DataStax Enterprise.

KMIP keys are cached on the DSE node. DSE refreshes the cache and a new key is automatically generated by the KMIP server after `key_cache_millis` lapses; the default setting is 5 minutes.

4. (Optional) Get a list of the affected tables to re-encrypt using the new key using the `DESC` keyspace command:

For example to find all tables in the `cycling` keyspace that use the KMIP group:

```
DESC KEYSPACE cycling
```

5. Use `nodetool upgradesstables` to rewrite the encrypted SSTables using the new key. Run the following command on every node in the cluster:
Transparent data encryption

• Target only specific tables:

```bash
$ nodetool upgradesstables --include-all-sstables keyspace_name table_name
[|table_name ...]
```

• Target specific keyspace:

```bash
$ nodetool upgradesstables --include-all-sstables keyspace_name
```

• All keyspaces and tables:

```bash
$ nodetool upgradesstables --include-all-sstables
```

6. (Optional) Remove the encryption key so that it is no longer available for decryption:

```bash
$ dsetool managekmip destroy key_id
```

The backed up SSTables are only accessible using the old key. Ensure that the data is accessible before removing the key.

Troubleshooting KMIP connections

The following error message may occur if the DataStax Enterprise SSL certificate is self-signed or from a CA that has not been added to the certificate chain on the KMIP server.

```
WARN 14:46:42,928 Unable to connect to KMIP host: 10.120.15.100:9005
com.cryptsoft.kmip.TTLVReadException: Error reading TTLV ResponseMessage. Got end-of-stream after reading 0 byte(s). Read bytes: (0) []
```

Ensure that trust between DSE and KMIP host has been properly configured, see Creating local SSL certificate and keystore files.

Encrypting Search indexes

DSE Search uses transparent data encryption (TDE) to encrypt data, including DSE Search index files and the DSE Search commit log. Cached data is not encrypted. DSE Search index encryption shares the setup with SSTable encryption, including secret key management and cipher creation.

DSE Search encryption is on when:

• The backing database table is also encrypted. The backing CQL table for a search core contains the system key (secret key). This backing CQL table must be encrypted to enable encryption of search indexes. Every new index file is created with the latest encryption setup of the backing database table.

• The Search index config class for directoryFactory is `solr.EncryptedFSDirectoryFactory`.

Table encryption can be dynamically enabled, changed, and disabled without restarting a DataStax Enterprise node. The index encryption setup changes with the table.

All encrypted files have a header that contains the required information to reconstruct cipher transformation that is used for the file.

Encryption with DSE Search introduces a slight performance overhead.

Encrypting new Search indexes

You can enable encryption for new search cores when you create them.
Using SolrJ Auth to implement encryption

To use the SolrJ-Auth libraries to implement encryption, follow instructions in the solrj-auth-README.md file.

These SolrJ-Auth libraries are included in the clients directory in DataStax Enterprise distribution. The SolrJ-Auth code is public.

Prerequisites:

When using TDE secure local file system. Encryption keys are stored remotely with KMIP encryption or locally with on-server encryption.

Encryption is enabled per core.

1. To enable encryption for a new core, edit the search index config file to change the class for directoryFactory to `solr.EncryptedFSDirectoryFactory`.
   - Recommended: Use the dsetool `dsetool create_core` command with automatic resource generation. Specify the class for directoryFactory to `solr.EncryptedFSDirectoryFactory` with the handy `coreOptionsInline` argument:
     ```bash
     $ dsetool create_core keyspace_name.table_name generateResources=true coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
     ```
   - You can also use the dsetool `dsetool create_core` command this way:
     ```bash
     $ dsetool create_core keyspace_name.table_name schema=schema.xml solrconfig=solrconfig.xml
     ```
     where the `solrconfig.xml` file specifies the required directoryFactory:
     ```xml
     <directoryFactory name="DirectoryFactory" class="solr.EncryptedFSDirectoryFactory"/>
     ```
     The `generateResources=true` option generates resources only if resources do not exist in the solr_resources table.
     After you create an encrypted search core, a node restart is not required.

What’s next:

To disable encryption, disable encryption for the backing CQL table. No node restart is required.

Encrypting existing Search indexes

Encrypting DSE Search indexes turns on encryption only for new files. Additional steps are required to encrypt existing data. When you encrypt existing DSE Search indexes, a node restart is required.

There are several ways to encrypt existing DSE Search indexes. Choose the method that is appropriate for your environment.

Prerequisites:

When using TDE secure local file system. Encryption keys are stored remotely with KMIP encryption or locally with on-server encryption.

Encryption is enabled per core.

1. To enable encryption for an existing core, change the search index config:
   ```xml
   <directoryFactory name="DirectoryFactory" class="solr.EncryptedFSDirectoryFactory"/>
   ```

2. Upload the changed `solrconfig.xml` file.
Transparent data encryption

For example, you can use dsetool to upload the changed resource file:

```
$ dsetool reload_core keyspace_name.table_name
```

3. Restart the DataStax Enterprise node.

   Enabling encryption does not require a node restart. However, the directoryFactory changes require a node restart.

   Encryption is on only for new files. Additional steps are required to encrypt existing data.

4. To encrypt existing files, use one of these methods:

   • Slowest option: Use dsetool to reload the core and reindex while the node is running

     ``
     $ dsetool reload_core keyspace_name.table_name deleteAll=true reindex=true
     ```

   • Fastest option: Requires that the local node is offline

     When your DataStax Enterprise cluster has a second remote node that is running, use dsetool to encrypt index files on the local offline node. The encryption configuration is read from the remote node.

     Requirements:

     # The remote node that contains the encryption configuration must be running.

     # The local node is offline.

     # The user that runs this command must have read and write permissions to the directory that contains the index files.

     a. Enable encryption for all nodes that have a search index.

     b. Run the dsetool command:

     ``
     $ dsetool upgrade_index_files keyspace_name.table_name -h IP_address [options]
     ```

     You can specify the following options for offline index encryption:

     **keyspace_name.table_name**
     Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

     **-h IP_address**
     Required. Node hostname or IP address of the remote node that contains the encryption configuration that is used for index encryption. The remote node must be running.

     **-c port**
     The DSE port on the remote node that contains the encryption configuration.

     **--backup**
     Preserves the index files from the current index as a backup after successful upgrade. The preserved index file backup is moved to the `--workspace` directory. When not specified, index files from the current index are deleted.

     **--workspace directory**
     The workspace directory for the upgrade process. The upgraded index is created in this directory. When not specified, the default directory is the same directory that contains the search index files.

     **--index directory**
     The data directory that contains the search index files. When not specified, the default directory is inferred from the search index name.

   • Simple option: But eventual and uncertain
Just wait. As new data is added to your index, encryption occurs when indexes are compacted or merged. Since new files are encrypted, your files will eventually be encrypted.

5. To verify which files are encrypted, use this command to list all DSE Search index files for the specified search core on the local node:

```
$ dsetool list_index_files keyspace_name.table_name [--index directory]
```

where `--index directory` specifies the data directory that contains the search index files. When not specified, the default directory is inferred from the search core name.

6. After you verify which files are encrypted, you can encrypt existing files using a different option.

What’s next:

To disable encryption, disable encryption for the backing CQL table. No node restart is required.

Tuning encrypted Search indexes

The primary configuration settings for tuning search index encryption are in the `dse.yaml` configuration file.

1. In the `dse.yaml` file on each node, adjust the index encryption options:

```
# solr_encryption_options:
#    decryption_cache_offheap_allocation: true
#    decryption_cache_size_in_mb: 256
```

   a. `decryption_cache_offheap_allocation` - Specify whether to allocate search decryption cache off JVM heap.

   b. `decryption_cache_size_in_mb` - Sets the maximum size of shared search decryption cache, in megabytes (MB).

2. Additional properties in the Search index config are available to tune encryption, although DataStax recommends using the default settings:

```
<directoryFactory name="DirectoryFactory" class="solr.EncryptedFSDirectoryFactory">
  <bool name="solr.crypto.cache.enabled">true</bool>
  <int name="solr.crypto.encryption.workers">2</int>
  <int name="solr.crypto.encryption.bufferpool.sizeinbytes">10240</int>
</directoryFactory>
```

3. To monitor the decryption cache, use JMX and the Solr Admin UI to view the Apache Solr metrics that are provided for Block Cache.

   For tuning purposes, review the hitratio metric. If hitratio is below 1.00, the cache is too small to fit entire decrypted index into memory.

4. To verify which files are encrypted, use this command to list all DSE Search index files for the specified search core on the local node:

```
$ dsetool list_index_files keyspace_name.table_name [--index directory]
```

where `--index directory` specifies the data directory that contains the search index files. When not specified, the default directory is inferred from the search core name.
Migrating encrypted tables from earlier versions

Steps to migrate encrypted tables from earlier versions to the latest version of DataStax Enterprise.

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 6.7, following the DataStax Upgrade Guide instructions.

4. Restart the cluster.

5. Check that the dse_system.encrypted_keys table was created using the cqlsh DESCRIBE KEYSPACE 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)
   );
   
   EverywhereStrategy is the default replication strategy for the dse_system and solr_admin keyspaces. Do not use or alter any other instances of EverywhereStrategy.
   ```

7. Rewrite all SSTables.

   ```
   $ nodetool upgradesstables --include-all-sstables
   ```

8. Verify if the search index files are encrypted:

   ```
   $ dsetool list_index_files keyspace_name table_name
   ```

9. Perform offline index encryption.

   ```
   $ dsetool upgrade_index_files keyspace_name.table_name -h IP_address [options]
   ```

   You can specify the following options:

   - **keyspace_name.table_name**
     Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

   - **-h IP_address**
     Required. Node hostname or IP address of the remote node that contains the encryption configuration that is used for index encryption. The remote node must be running.

   - **-c port**
     The DSE port on the remote node that contains the encryption configuration.
Transparent data encryption

--backup
Preserves the index files from the current index as a backup after successful upgrade. The preserved index file backup is moved to the --workspace directory. When not specified, index files from the current index are deleted.

--workspace directory
The workspace directory for the upgrade process. The upgraded index is created in this directory. When not specified, the default directory is the same directory that contains the search index files.

--index directory
The data directory that contains the search index files. When not specified, the default directory is inferred from the search index name.

Bulk loading data between TDE-enabled clusters

A common operation in database environments is to bulk load data between clusters. For example, to facilitate testing of new functionality, you may need to load large amounts of data from a production environment to your development environment. When Transparent Data Encryption (TDE) is enabled, these secure environments require additional steps to ensure that the valid encryption keys are in place.

There are two types of keys used while streaming encrypted data:

1. Decryptor
   Used to decrypt the SSTable during streaming. The decryptor must be the same key used to encrypt the data on the source cluster.

2. Encryptor
   Used to encrypt the SSTable on the target cluster. The key is the one configured in the encryption option for the CQL table schema on the target cluster.

The decryptor and encryptor could be the same key, or different keys. If you encounter errors during bulk data loading between clusters, the cause may be that your environment uses different keys, and the wrong key was used during decryption.

To bulk load data between two TDE-enabled clusters, follow these steps:

1. Copy the encryption key file used on the source cluster to the target cluster. The key resides in the directory identified by the system_key_directory option in dse.yaml. The default directory for the encryption key file is /etc/dse/conf. Do not change the name of encryption key when you copy the key from the source to the target cluster. For example, if the key file is named our_system_key on the source cluster, the same file name must be used on the target cluster, and placed in the target cluster's designated system_key_directory.

   The default key file name, system_key, is often used on different clusters. If that's true for your environment, a problem would occur if you were to copy the key file from the source cluster to the target cluster. Two different keys with the same name cannot exist in the same directory. To avoid this scenario, rekey the target cluster to use a different key name. You can rename the existing key or generate a new key. Refer to Rekeying existing data.

2. On the source cluster, get the key's entries from the dse_system.encrypted_keys table. Example:

   ```sql
   SELECT * from dse_system.encrypted_keys;
   key_file                | cipher | strength | key_id                                                                 | key
   ------------------------+--------+----------+------------------------------------------------------------------------|------
   ```
Transparent data encryption

<table>
<thead>
<tr>
<th>our_system_key</th>
<th>AES</th>
<th>128</th>
<th>d9b3dd70-c764-11e7-abc4-793ec23f8a8c</th>
</tr>
</thead>
<tbody>
<tr>
<td>kmbYEiKLkmW3Hzg7d1Pt1rWk3j6hR+gM7bxd/pRd7gU=</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. On the target cluster, insert the same key entry. Example:

```
INSERT INTO dse_system.encrypted_keys (key_file, cipher, strength, key_id, key)
VALUES
('our_system_key', 'AES', 128, 'd9b3dd70-c764-11e7-abc4-793ec23f8a8c',
 'kmbYEiKLkmW3Hzg7d1Pt1rWk3j6hR+gM7bxd/pRd7gU=');
```

4. On the target cluster, verify that your added entry is in the `dse_system.encrypted_keys` table. Example:

```
SELECT * from dse_system.encrypted_keys;
```

<table>
<thead>
<tr>
<th>key_file</th>
<th>cipher</th>
<th>strength</th>
<th>key_id</th>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>our_system_key</td>
<td>AES</td>
<td>128</td>
<td>d9b3dd70-c764-11e7-abc4-793ec23f8a8c</td>
<td>kmbYEiKLkmW3Hzg7d1Pt1rWk3j6hR+gM7bxd/pRd7gU=</td>
</tr>
<tr>
<td>system_key_dev</td>
<td>AES</td>
<td>256</td>
<td>81847700-c99d-11e7-b9d9-23f36e5077c2</td>
<td>6YXE87AcEv61jvY6x7rdj6Ahde0N60HxzAlNRnW1s7nVDFPQBArh64LousF8bXmy</td>
</tr>
</tbody>
</table>

If you use the same key as decryptor and encryptor, the `SELECT` output will show only one key.

5. If you change the encryption setting on the target cluster, run the following command on all nodes in the target cluster to rewrite the SSTables using the new encryption key:

```
$ nodetool upgradesstables --include-all-sstables
```

After performing the steps listed above, `sstableloader` should be able to run successfully during bulk data loading operations between two TDE-enabled clusters.
Chapter 8. Configuring SSL

How to configure SSL for DataStax Enterprise

Configure SSL for DataStax Enterprise (DSE) by implementing Client Certificate Authentication. Following this approach, each node verifies the service or client making a request against a local truststore to validate that the certificate was issued by a known Certificate Authority (CA).

Creating SSL certificates, keystores, and truststores

You can implement SSL using CA signed certificates signed by well-known CAs, or by creating your own root CA. DataStax recommends using certificates signed by a CA to reduce SSL certificate management tasks. However, you can use self-signed certificates with DSE, which supports SSL certificates in local and external keystores.

Creating your own CA in a production environments typically involves using an intermediary certificate chain, where the root CA signs one or more intermediate certificates with its private key. These intermediary certificates chain together to link back to the root CA, which owns one or more trusted roots.

Where to configure SSL

DSE supports SSL encryption between nodes (node-to-node communication) and between clients and nodes (client-to-node communication). You can use SSL to encrypt in-flight data for the following DSE services and clients.

DSE services

Use SSL to encrypt data in the following node-to-node connections:

- DSE Core
- DSE Search with Apache Solr™
- DSE Analytics with Apache Spark™
- DSE Graph

DSE clients

Use SSL to secure connections from a client to the coordinator node to establish client-to-node connections:

- DSE drivers
- CQL shell (cqlsh)
- DataStax Studio
- DataStax Bulk Loader
- DataStax Apache Kafka Connector
- DSE tools

Configuring SSL for DSE

Complete the following procedures to configure SSL for DSE:

1. Create SSL certificates, keystores, and truststores.
2. Configure SSL for DSE services (node-to-node communication).

3. Configure SSL for DSE clients (client-to-node communication).

After creating the necessary SSL certificates and configuring SSL for DSE services, use `cqlsh` to connect to your SSL-enabled cluster.

Creating SSL certificates, keystores, and truststores

Before configuring SSL for DataStax Enterprise (DSE) services, you must create SSL certificates, keystores, and truststores. DSE supports both remote keystore SSL providers and local keystore files.

Complete the procedure depending on whether you want to use a remote PKCS11 keystore provider or create your own local keystore files:

- Using a remote PKCS11 keystore provider
- Creating local SSL certificate and keystore files

After creating and configuring SSL dependencies, configure SSL for node-to-node connections and client-to-node connections.

Using a remote PKCS11 keystore provider

DataStax Enterprise (DSE) database uses the Java Cryptography API (JCA) to implement SSL providers. The JCA is a pluggable architecture that abstracts the actual cryptography implementation from the algorithm requested. To support swapping out different implementations, DSE database uses `Cipher.getInstance("AES")`.

The JCA architecture `Provider` class allows multiple implementations to register using a different service provider interface (SPI). Java comes with multiple providers and supports installation of additional providers, such as PKCS11.

The DSE database `keystore type` parameter in `cassandra.yaml` determines which SPI to use.
Configuring SSL

Differences between PKCS11 and PKCS12

PKCS11 and PKCS12 are part of the RSA Public Key Cryptography Standards for storing private key and certificate information.

**PKCS12** is typically used to store private key and certificate information on files. The default keystore type in Java is JKS, though you can specify PKCS12 with the `-storetype` option when creating a keystore with `keytool`.

**PKCS11** provides an interface to connect with hardware keystore devices. This type of keystore can store private keys, secret keys, and certificates like PKCS12, but is designed for Hardware Storage Modules (HSM).

Installing additional providers

Install providers using the `java.security` configuration that comes with the JRE.

For more detailed instructions, see [How to implement a Provider (Oracle)](https://docs.oracle.com/javase/8/docs/technotes/guides/security/ProviderIndex.html).

Add the location to the `java.security` configuration file, which is located in `$JAVA_HOME/lib/security/`:

```
security.provider.10=SunPKCS11
```

Creating local SSL certificate and keystore files

Use these steps as a general guide to create and distribute SSL certificates using OpenSSL and Java `keytool`.

Use SSL certificates for client-to-node encryption and node-to-node encryption. DataStax supports SSL using well-known CA signed certificates for each node or you can create your own root Certificate Authority (CA). DataStax recommends using certificates signed by a well-known CA to reduce SSL certificate management tasks. However, you can use self-signed certificates with DataStax Enterprise (DSE), which supports SSL certificates in local and external keystores.

Use this method of generating a root key/certificate pair only for development and test environments. Secure these files, as anyone with access to the root CA files can sign certificates and authorize hosts as the root certificate authority.

Creating your own CA in a production environments typically involves using an intermediary certificate chain, where the root CA signs one or more intermediate certificates with its private key. These intermediary certificates chain together to link back to the root CA, which owns one or more trusted roots.

OpsCenter Lifecycle Manager can configure DSE clusters to use client-to-node and node-to-node encryption, and automates the process of preparing server certificates. See [Configuring SSL using LCM](https://docs.datastax.com/en/opscenter/c65/opscenter-6.0/opscenter-configuration-guide/ssl/configuration.html).

DataStax recommends using a computer outside of the DSE environment to generate and manage SSL certificates. Complete the steps on a dedicated CA server that is fully secured and permanently isolated from the network.

Using a well-known CA

If using a third-party signed certificate, or when adding a node using an existing root CA, skip to the steps for Creating a key and certificate for each node.

Creating a root CA certificate

In development and testing environments, you can set up your own root Certificate Authority (CA) to sign DataStax Enterprise (DSE) node certificates for SSL. In this model, generate your own root certificate that will be used to sign the certificate on every node, generate certificates for individual nodes, sign them, and generate corresponding keystores for every node.

If you want to use a remote keystore provider instead, see [Using a remote PKCS11 keystore provider](#).

---

DataStax 6.7 Security Guide Latest DSE version Latest 6.7 patch: 6.7.7
1. Create a directory for the root CA signing certificate/key, and then change to that directory:

```bash
$ mkdir -p rootca_path

$ cd rootca_path
```

`rootca_path`
Directory where root certificate is created and stored. DataStax recommends securing this directory, ideally on a computer isolated from the network.

2. Create a `rootca.conf` configuration file:

```bash
$ touch rootca.conf
```

`rootca.conf`
Root CA configuration file.

3. Edit the `rootca.conf` file and add the following minimal settings:

```ini
# rootca.conf
[ req ]
distinguished_name       = CA_DN
prompt                    = no
output_password           = rootca_password
default_bits             = 2048

[ CA_DN ]
C = CC
O = org_name
OU = cluster_name
CN = CA_CN
```

`CA_DN`
Title for the section containing the Distinguished Name (DN) properties for the CA.

`rootca_password`
Password for the generated file used to sign certificates.

`CC`
Two letter country code, such as `US` for United States or `JP` for Japan. See Nations Online for a complete list of country codes.

`org_name`
Name of your organization.

`cluster_name`
Name of your DataStax Enterprise (DSE) cluster.

`CA_CN`
Common Name (CN) for the root CA.

4. Use `openssl` to create a root key/certificate pair:

```bash
$ openssl req -config rootca.conf \ -new -x509 -nodes \ -keyout rootca.key \ -out rootca.crt \ -days 3650
```

The `-x509` option outputs a signed certificate used as the root CA. The number of days specified by the `-days` option affects the duration that all signed certificates are valid for. Indicating a higher number of days means that the certificates are valid for a longer period. In the previous example, the root CA is valid for 3650 days, which is approximately 10 years.
Configuring SSL

Two files are created: rootca.key and rootca.crt.

5. Verify the root certificate:

```bash
$ openssl x509 -in rootca.crt -text -noout
```

```
Certificate:
  Data:
    Version: 1 (0x0)
    Serial Number: serial_number (0xcd4bc943beeb35ce)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=US, O=datastax, OU=pw-j-dse, CN=rootCa
    Validity
      Not Before: Jul 23 20:15:06 2019 GMT
      Not After : Jul 23 20:15:06 2020 GMT
    Subject: C=US, O=datastax, OU=pw-j-dse, CN=rootCa
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
        Modulus:
          00:d8:71:e0:51:07:ad:f1:f7:0b:4d:2c:10:4c:24:
          ...
        Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
```

Creating a truststore for all nodes

Create a truststore that is used to ensure that all nodes recognize the certificate authority (CA). Even when using a well-known certificate authority, DataStax recommends creating a truststore with the signing CA certificate or certificate chain (following the instructions from your CA). Most well-known CA certificates are already available through the DataStax Enterprise (DSE) Java implementation.

1. Create a single truststore, and add the root certificate to the truststore. For the `keystore_type`, enter JKS:

   ```bash
   $ keytool -keystore dse-truststore.jks -storetype keystore_type -importcert -file 'rootca.crt' -keypass keystore_password -storepass truststore_password -alias rootca_name -noprompt
   
   dse-truststore.jks
   Truststore that contains the root certificate.
   Use the same truststore that contains the root certificate on all nodes.

   keystore_type
   Valid types are JKS, JCEKS, PKCS11, or PKCS12. For file-based keystores, use PKCS12.
   Default: commented out (JKS)

   rootca.crt
   Certificate used to sign (authorize) DSE node SSL certificates.

   keystore_password
   Password used to protect the private key of the key pair.

   truststore_password
   Password required to access the keystore.

   rootca_name
Configuring SSL

Name (alias) used to identify the root certificate when importing into the node's keystore. A message displays indicating that the certificate was added. The truststore now contains a single entry.

Certificate was added to keystore

2. Verify the truststore to ensure that it contains the root certificate:

$ keytool -list \ -keystore dse-truststore.jks \ -storepass truststore_password

The command output indicates the keystore type, provider, number of entries, creation date, and certification details.

Keystore type: jks
Keystore provider: SUN
Your keystore contains 1 entry
rootca_name, Aug 8, 2019, trustedCertEntry,
Certificate fingerprint (SHA1): SHA1-hash

Creating a key and certificate for each node

For each node in the cluster, create a keystore, key pair, and certificate signing request using the Fully Qualified Domain Name (FQDN) of the node.

These steps are required even when using a third-party CA, or when adding a node to an existing DSE environment with SSL enabled.

Prerequisites: On each node, run the following command to obtain the FQDN for each node:

$ nslookup $(hostname --fqdn) && hostname --fqdn && hostname -i

Server: 10.200.1.10
Address: 10.200.1.10#53
Name: ip-10-200-182-183.example.com
Address: 10.200.182.183
ip-10-200-182-183.example.com
10.200.182.183

In this example, ip-10-200-182-183.example.com is the Common Name (CN), which is used to generate the SSL certificate. The CN must match the DNS resolvable host name. Mismatches between the CN and node hostname cause an exception and the connection is refused.

1. Create a directory to store the keystores and change to the directory:

$ mkdir -p dse/keystores

$ cd dse/keystores

2. For each node, generate a keystore with key pair. Each node will have its own keystore, such as node1-keystore.jks:
Ensure the passwords entered for `truststore_password` and `keystore_password` are the same. If the passwords are different, DSE fails to start and returns an error message: "Cannot recover key".

```bash
$ keytool -genkeypair -keyalg RSA -alias node_name -keystore node-keystore.jks
-storepass truststore_password -keypass keystore_password -validity 730
-keysize 2048 -dname "CN=node_name, OU=cluster_name, O=org_name, C=CC"
-ext "san-ip:node_ip_address"
```

The `-validity` option specifies how long the generated key pair for the node is valid for. In the previous example the key pair is valid for 730 days, which is approximately 2 years.

- **node_name**: Fully Qualified Domain Name (FQDN) of the node, such as `ip-10-200-182-183.example.com`. If using the FQDN as the `node_name`, you can add the IP address as a subject alternative name (SAN) so that the certificate protects the IP address in addition to the domain name.

- **node-keystore.jks**: Keystore for the individual node.

- **truststore_password**: Password required to access the keystore.

- **keystore_password**: Password used to protect the private key of the key pair.

- **cluster_name**: Name of your DataStax Enterprise (DSE) cluster.

- **org_name**: Name of your organization.

- **CC**: Two letter country code, such as `US` for United States or `JP` for Japan. See [Nations Online](http://www.nationsonline.com) for a complete list of country codes.

- **node_ip_address**: If using the domain name as the `node_name` for the CA, add `san-ip:ip_address` to the `-ext` option. Using the IP address as a subject alternative name (SAN) ensures that the certificate protects the IP address in addition to the domain name. For example:

```bash
-ext "san-ip:10.200.100.52"
```

3. Verify each SSL keystore and key pair:

```bash
$ keytool -list -keystore node-keystore.jks -storepass truststore_password
```

The command output indicates the keystore type, provider, and number of entries. The alias used the example is `dcl_node1`.

- **Keystore type**: JKS
- **Keystore provider**: SUN

Your keystore contains 1 entry

- `dcl_node1, Jul 23, 2019, PrivateKeyEntry`,

---

Page 146
Configuring SSL

Certificate fingerprint (SHA1): SHA1_hash

4. Generate a signing request from each keystore:

```bash
$ keytool -keystore node-keystore.jks -alias node_name -certreq -file
signing_request.csr -keypass node_key_password -storepass keystore_password 
```

**node_key_password**
Password used to protect the individual private key.

**keystore_password**
Password used to protect the private key of the key pair.

The certificate signing request file (`signing_request.csr`) is created.

5. Repeat the previous steps on each node to generate a signing request, ensuring that the `dname` information matches the node information (such as `node_name` and `cluster_name`).

**Signing the certificate signing request**

For each node, sign the certificate signing request. If you created your own root CA, follow the instructions below. Alternatively, send the certificate signing request to a well-known CA for signing.

1. Sign each node certificate:

```bash
$ openssl x509 -req -CA 'path/to/rootca.crt' -CAkey 'path/to/rootca.key' -in
signing_request.csr -out signing_request.crt_signed -days 3650 -CAcreateserial 
-passin pass:rootca_password -extfile san_config_file.conf
```

**signing_request.csr**
Certificate signing request (CSR) that is passed to the Certificate Authority (CA) to sign the certificate. The CSR typically includes the public key, plus associated metadata such as the Common Name (CN), Organization (O), Organization Unit (OU), and Country (C).

**signing_request.crt_signed**
The signed certificate file to create, using the certificate signing request (CSR) (`signing_request.csr`) as the input file.

**san_config_file.conf**
If using the domain name as the `node_name` and the node IP address as a subject alternative name (SAN), create a temporary configuration file and pass it in using the `-extfile` option. In the configuration file, use the `subjectAltName` parameter to specify the DNS and IP. For example:

```
subjectAltName=DNS:node_name,IP:node_ip_address
```

You can specify multiple SANs in the same configuration file:

```
subjectAltName=DNS:domain1,IP:10.200.100.52
subjectAltName=DNS:domain1,IP:10.200.101.63
subjectAltName=DNS:domain1,IP:10.200.111.74
subjectAltName=DNS:domain1,IP:10.200.121.85
```

Use the `rootca_password` entered when Creating the root CA signing certificate.

A signed certificate file `signing_request.crt_signed` is created.
2. Verify the root certificate file was properly signed:

```
$ openssl verify -CAfile 'path/to/rootca.crt' signing_request.crt_signed
signing_request.crt_signed: OK
```

3. Delete the temporary configuration file `san_config_file.conf` to protect the SAN for a node.

```
$ rm -f san_config_file.conf
```

**Importing the signed certificate into the node keystore**

For each node in the cluster, create a keystore and import the signed certificate. The variables you enter in the following commands must match the information that you entered in Creating a key and certificate for each node.

1. Import the root certificate (`rootca.crt`) into each node's keystore:

   You created the root certificate when Creating a truststore for all nodes.

   ```
   $ keytool -keystore node-keystore.jks -alias rootca_name -importcert -file 'path/to/rootca.crt' -keystorepass keystore_password -storepass truststore_password -noprompt
   ```

   **rootca_name**
   
   Name (alias) used to identify the root certificate when importing into the node's keystore.

   If the signed certificate for the node is imported before the root certificate, an error occurs:

   ```
   keytool error: java.lang.Exception: Failed to establish chain from reply
   ```

2. Import the node's signed certificate (`signing_request.crt_signed`) into the corresponding keystore on the node:

   ```
   $ keytool -keystore node-keystore.jks -alias node_name -importcert -file signing_request.crt_signed -keystorepass keystore_password -storepass truststore_password -noprompt
   ```

   The alias **node_name** must match the alias name used to generate the signing request. See Creating a key and certificate for each node.

   Confirmation of the installation appears:

   ```
   Certificate was added to keystore
   ```

3. Verify your keystore again, which should now contain two entries. One entry is for the node keystore, and the other for the imported root certificate:

   ```
   $ keytool -list -keystore node-keystore.jks -storepass truststore_password
   ```

   Each keystore entry is identified by the name you entered for the -alias:

   ```
   Keystore type: jks
   Keystore provider: SUN
   Your keystore contains 2 entries
   ```
4. Repeat the previous steps on each node to import the root certificate, and then import the signed certificate into the keystore.

Configuring SSL for node-to-node connections

Node-to-node (internode) encryption protects data in-flight between nodes in a cluster using SSL.

**Prerequisites:** Create SSL certificates, keystores, and truststores. You can either create local keystore files or use a remote keystore provider.

OpsCenter Lifecycle Manager can configure DataStax Enterprise clusters to use node-to-node encryption and automates the process of preparing server certificates using an internal certificate authority and deploys the resulting keystore and truststore to each node automatically.

1. Edit cassandra.yaml and make the following changes to the server_encryption_options section to enable SSL:

   a. Set internode_encryption to one of the following options to limit which traffic between nodes is encrypted:

      - all - Encrypt all inter-node communications
      - none - No encryption
      - dc - Encrypt the traffic between the datacenters
      - rack - Encrypt the traffic between the racks

   b. Set require_client_auth to true to require two-way host certificate validation.

   c. Set require_endpoint_verification to true to verify that the connected hostname matches the certificate.

2. Configure the keystore and truststore, depending on whether you are using local keystore files or a remote keystore provider. All settings are configured in the server_encryption_options section of cassandra.yaml:

   - **Local files:** use the following settings.

```
server_encryption_options:
  internode_encryption: all
  keystore_type: JKS
  keystore: path_to_keystore.jks
  keystore_password: keystore_password
  require_client_auth: true
  require_endpoint_verification: true
  truststore_type: JKS
  truststore: path_to_truststore.jks
  truststore_password: truststore_password
```

To encrypt the truststore and keystore passwords for local encryption, see Encrypting configuration file properties or for KMIP see Encrypting configuration file properties.
Configuring SSL

- **Remote keystore provider**: use the following settings. Unused options can be blank or commented out.

  Requires installation of a provider. See Using a remote PKCS11 keystore provider.

```yaml
server_encryption_options:
  internode_encryption: all
  keystore_type: PKCS11
  require_client_auth: true
  require_endpoint_verification: true
  truststore_type: PKCS11
```

**internode_encryption**

Encryption options for inter-node communication using the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite for authentication, key exchange, and encryption of data transfers. Use the DHE/ECDHE ciphers, such as TLS_DHE_RSA_WITH_AES_128_CBC_SHA if running in (Federal Information Processing Standard) FIPS 140 compliant mode.

**keystore_type**

Valid types are JKS, JCEKS, PKCS11, or PKCS12. For file-based keystores, use PKCS12.

Default: commented out (JKS)

**keystore**

Relative path from DSE installation directory or absolute path to the Java keystore (JKS) suitable for use with Java Secure Socket Extension (JSSE), which is the Java version of the Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols. The keystore contains the private key used to encrypt outgoing messages.

Default: resources/dse/conf/.keystore

**keystore_password**

Password for the keystore. This must match the password used when generating the keystore and truststore.

Default: cassandra

**require_client_auth**

Whether to enable certificate authentication for node-to-node (internode) encryption. When not set, the default is false.

Default: commented out (false)

**require_endpoint_verification**

Whether to verify the connected host and the host name in the certificate match. When not set, the default is false.

Default: commented out (false)

**truststore_type**

Valid types are JKS, JCEKS, or PKCS12. For file-based truststores, use PKCS12.

Default: commented out (JKS)

**truststore**

Relative path from DSE installation directory or absolute path to truststore containing the trusted certificate for authenticating remote servers.

Default: resources/dse/conf/.truststore

**truststore_password**

Password for the truststore.

Default: cassandra

3. Save and close the `cassandra.yaml` file.

4. Restart DSE.

### Configuring SSL for client-to-node connections

Client-to-node encryption protects in-flight data from client machines to a database cluster using SSL (Secure Sockets Layer) and establishes a secure channel between the client and the coordinator node.
Configuring SSL

Complete the following procedure on every node in the cluster to configure SSL for client-to-node connections.

On a DSE Search node, enabling SSL for the database automatically enables SSL in the DSE Search web.xml file and configures an SSL connector in Tomcat using the authentication/authorization filters. No changes are required for the web.xml or server.xml files.

If the TomcatSolrRunner doesn’t find a connector in server.xml, it creates a default connector. The default connector binds to the native_transport_address.

If you are not using the JCE Unlimited Strength Jurisdiction Policy, make sure that your ticket granting principal does not use AES-256. Starting in JDK 8u161, JCE Unlimited is enabled by default. Refer to the Release Notes for JDK 8u161. If your ticket granting principal 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_256_CBC_SHA as it isnt supported by the socket
```

Prerequisites: Create SSL certificates, keystores, and truststores. You can either create local keystore files or use a remote keystore provider.

1. Edit cassandra.yaml and make the following changes in the client_encryption_options to enable SSL:
   a. Set enabled to true to enable SSL.
   b. Set optional to false (default) to only allow SSL connections.
   c. Set require_client_auth to true to require two-way host certificate validation.

2. Configure the keystore and truststore, depending on whether you are using local keystore files or a remote keystore provider. All settings are configured in the client_encryption_options section of cassandra.yaml:
   - **Local files**: use the following settings.
     ```yaml
     client_encryption_options:
     enabled: true
     optional: false
     keystore_type: JKS
     keystore: path_to_keystore
     keystore_password: keystore_password
     require_client_auth: true
     truststore_type: JKS
     truststore: path_to_truststore
     truststore_password: truststore_password
     protocol: ssl
     algorithm: SunX509
     store_type: JKS
     cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA]
     ```
   - **Remote keystore provider**: use the following settings. Unused options can be blank or commented out.
     Requires installation of a provider. See Using a remote PKCS11 keystore provider.
     ```yaml
     client_encryption_options:
     enabled: true
     optional: false
     keystore_type: PKCS11
     require_client_auth: true
     truststore_type: PKCS11
     ```
Configuring SSL

```yaml
protocol: ssl  
algorithm: SunX509  
store_type: JKS  
cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA]
```

**enabled**
- Whether to enable client-to-node encryption.
  Default: false

**optional**
- Whether to allow unsecured connections when client encryption is enabled.
  Default: false

**keystore_type**
- Valid types are JKS, JCEKS, PKCS12, and PKCS11. For file-based keystores, use PKCS12.
  DSE 6.0 uses `store_type` for this option. In DSE 6.7, you may use `store_type` or `keystore_type`. If you specify both, `keystore_type` takes precedence. By default, `store_type` is not listed in `cassandra.yaml`.
  Default: commented out (JKS)

**keystore**
- Relative path from DSE installation directory or absolute path to the Java keystore (JKS) suitable for use with Java Secure Socket Extension (JSSE), which is the Java version of the Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols. The keystore contains the private key used to encrypt outgoing messages.
  Default: `resources/dse/conf/.keystore`

**keystore_password**
- Password for the keystore.
  Default: cassandra

**require_client_auth**
- Whether to enable certificate authentication for client-to-node encryption. When not set, the default is false.
  When set to true, client certificates must be present on all nodes in the cluster.
  Default: commented out (false)

**truststore_type**
- Valid types are JKS, JCEKS, and PKCS12.
  Default: commented out (JKS)

**truststore**
- Relative path from DSE installation directory or absolute path to truststore containing the trusted certificate for authenticating remote servers.
  Truststore password and path is only required when `require_client_auth` is set to true.
  Default: `resources/dse/conf/.truststore`

**truststore_password**
- Password for the truststore. This must match the password used when generating the keystore and truststore.
  Truststore password and path is only required when `require_client_auth` is set to true.
  Default: cassandra

3. Save and close the `cassandra.yaml` file.

4. Complete a rolling DSE restart.

**Configuring JMX on the server side**

Complete the following procedure to configure JMX on the server side when enabling secure client-to-node connections using SSL.
1. If the $LOCAL_JMX setting is present, change it to no.

   "$LOCAL_JMX" = "no"

2. Enable JMX authentication by setting -Dcom.sun.management.jmxremote.authenticate to true:

   JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=true"

3. Uncomment the following settings in the cassandra-env.sh file. You must specify the path to appropriate keystore and truststore, including passwords for each.

   You can also use the jvm.options file as described in Setting system properties during startup.

   JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl=true"
   JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl.need.client.auth=true"
   JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.registry.ssl=true"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStore=path_to_keystore.jks"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStorePassword=keystore-password"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStore=path_to_truststore.jks"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStorePassword=truststore-password"

   com.sun.management.jmxremote.ssl
   Set to true to enable SSL for JMX.

   com.sun.management.jmxremote.ssl.need.client.auth
   Set to true to enable two-way certificate authentication.

   com.sun.management.jmxremote.registry.ssl
   Set to true to create an RMI registry protected by SSL, and configure a management agent when the JVM starts.

Configuring SSL for nodetool, nodesync, dsetool, and Advanced Replication

Complete the following procedure to configure JMX for using nodetool, nodesync, dsetool, and DataStax Enterprise (DSE) Advanced Replication with SSL.

Make these changes in the cassandra-env.sh file on each node in the cluster.

Prerequisites:

1. Create SSL certificates with a self-signed CA.
2. Configure client-to-node encryption.
3. Configure JMX on the server side.

For production environments, secure an entire cluster using JKS files. For a single-node development environment, you can use a simpler single-node, local keystore file and truststore file.

1. Open the cassandra-env.sh file.
2. Restart DSE.
3. **nodetool**: To configure the client settings for nodetool, create a `.cassandra/nodetool-ssl.properties` file in your home or client program directory on the node where you will run the command. Add the following settings, depending on whether you are running the command in a production or development environment.

```bash
$ touch ~/.cassandra/nodetool-ssl.properties
```

**Production environment:**

<table>
<thead>
<tr>
<th>Settings</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dcom.sun.management.jmxremote.ssl=true</code></td>
<td></td>
</tr>
<tr>
<td><code>-Dcom.sun.management.jmxremote.ssl.need.client.auth=false</code></td>
<td></td>
</tr>
<tr>
<td><code>-Dcom.sun.management.jmxremote.registry.ssl=true</code></td>
<td></td>
</tr>
<tr>
<td><code>-Djavax.net.ssl.keyStore=path_to_keystore</code></td>
<td></td>
</tr>
<tr>
<td><code>-Djavax.net.ssl.keyStorePassword=keystore-password</code></td>
<td></td>
</tr>
<tr>
<td><code>-Djavax.net.ssl.trustStore=path_to_truststore</code></td>
<td></td>
</tr>
<tr>
<td><code>-Djavax.net.ssl.trustStorePassword=truststore-password</code></td>
<td></td>
</tr>
</tbody>
</table>

**Development environment:**

<table>
<thead>
<tr>
<th>Settings</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dcom.sun.management.jmxremote.ssl.need.client.auth=true</code></td>
<td></td>
</tr>
<tr>
<td><code>-Djavax.net.ssl.keyStore=path_to_keystore</code></td>
<td></td>
</tr>
<tr>
<td><code>-Djavax.net.ssl.keyStorePassword=keystore-password</code></td>
<td></td>
</tr>
<tr>
<td><code>-Djavax.net.ssl.trustStore=path_to_truststore</code></td>
<td></td>
</tr>
<tr>
<td><code>-Djavax.net.ssl.trustStorePassword=truststore-password</code></td>
<td></td>
</tr>
</tbody>
</table>

4. **nodesync**: To configure the client settings for nodesync, create a `.cassandra/nodesync-ssl.properties` file in your home or client program directory on the node where you will run the command. Add the following settings to the file.

```bash
$ touch ~/.cassandra/nodesync-ssl.properties
```

The file for nodesync is equivalent to the `.cassandra/nodetool-ssl.properties` file used by nodetool, except that it defines properties shared by JMX and CQL.

```bash
$ touch ~/.cassandra/nodesync-ssl.properties
```

The JVM properties for nodesync should be the same as those set for nodetool, but defined in a separate file, such as `nodesync-jvm.options`. DataStax recommends maintaining separate option files for nodetool and nodesync. For example, you might need SSL only in the CQL connection, but not in JMX. In this case, nodetool would not require the JVM properties, while nodesync would need them defined.

5. Start the appropriate tool using the following options to establish an encrypted connection with username and password credentials, or an auth provider class (for CQL). If you provide a username option but not a password, you are prompted to enter one.
**Configuring SSL**

### nodetool

$ nodetool --ssl -u jmx_username -pw jmx_password command

### nodesync (JMX, CQL, or both)

$ nodesync --jmx-ssl --jmx-username jmx_username --jmx-password jmx_password command

$ nodesync --cql-ssl --cql-username cql_username --cql-password cql_password command

$ nodesync --cql-ssl --cql-auth-provider cql-auth-provider-ClassName command

$ nodesync --jmx-ssl --jmx-username jmx_username --jmx-password jmx_password --cql-ssl --cql-username cql_username --cql-password cql_password command

$ nodesync --jmx-ssl --jmx-username jmx_username --jmx-password jmx_password --cql-ssl --cql-auth-provider cql-auth-provider-ClassName command

### dsetool

$ dsetool --ssl -a jmx_username -b jmx_password command

### dse advrep

$ dse advrep --ssl -u jmx_username command

---

**Setting up SSL for JConsole (JMX)**

Using JConsole with SSL requires the same JMX changes to cassandra-env.sh as nodetool. See Configuring SSL for nodetool, nodesync, dsetool, and Advanced Replication. You do not need to create a nodetool-ssl.properties file, but the same JVM keystore and truststore options must be specified with jconsole on the command line.

**Prerequisites:**

1. Create SSL certificates with a self-signed CA for production environments, or create SSL certificates for development environments.

2. Configure client-to-node encryption

1. Copy the keystore and truststore files to the node where JConsole will be run. In this example, the files are server-keystore.jks and server-truststore.jks.

2. Run jconsole using the JVM options:

   ```
   $ jconsole -Djavax.net.ssl.keyStore=server-keystore.jks -Djavax.net.ssl.keyStorePassword=keystore-password -Djavax.net.ssl.trustStore=server-truststore.jks -Djavax.net.ssl.trustStorePassword=truststore-password
   ```

   If no errors occur, JConsole starts. If connecting to a remote node, select Remote Process and enter the hostname and JMX port. If using authentication, enter the username and password. See Using JConsole for more information.
Configuring SSL

Connecting SSTableloader to a secured cluster

The sstableloader tool is also called bulk loader. If you run sstableloader from a DataStax Enterprise (DSE) node that has been configured for Kerberos or client-to-node/node-to-node encryption using SSL, no additional configuration is required for securing sstableloader operations. sstableloader automatically detects the configuration. On a development machine without SSL, configure Kerberos or SSL as follows:

To use SSL to connect to an unsecured DSE node from a development system, use the sstableloader script 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_node_truststore.jks -tspw truststore-password
```

If you want to configure require_client_auth=true on the target, add the path to the keystore and keystore password, as shown in the following example:

```
$ resources/cassandra/bin/sstableloader -d 192.168.56.102 /var/lib/cassandra/data/Keyspace1/Standard1 -tf org.apache.cassandra.thrift.SSLTransportFactory -ts path_to_node_truststore.jks -tspw truststore-password -ks path_to_node_keystore.jks -kspw keystore-password
```

Connecting to SSL-enabled nodes using cqlsh

Use the CQL shell (cqlsh) to connect to nodes where SSL is configured for client-to-node connections. cqlsh can use its own key and certificate that can be signed by the same root Certificate Authority (CA) used for nodes in the cluster, or signed by a different CA.

Use the DataStax Enterprise sample cqlshrc.sample.ssl file as a starting point. See the SSL example for reference.

The environment variables (SSL_CERTFILE and SSL_VALIDATE) override any options set in the cqlshrc file.

If problems authenticating occur, use the --debug option to show CQL shell settings and connection details.

1. Create a client.conf configuration file:

```
$ touch client.conf
```

2. Edit the client.conf file and add the following settings:

```
# client.conf
[ req ]
distinguished_name = CA_DN
prompt = no
output_password = rootca-cqlsh_password
default_bits = 2048

[ CA_DN ]
C = CC
O = org_name
OU = cluster_name
```

DataStax 6.7 Security Guide Latest DSE version Latest 6.7 patch: 6.7.7
3. Generate a separate key and certificate for cqlsh, using the client.conf file you created in the previous step.

```
$ openssl req -newkey rsa:2048 -nodes \ -keyout client_key.key \ -out
    signing_request.csr \ -config
    client.conf
```

4. Sign the certificate using the same root CA certificate used on the node where you are running cqlsh. You created the root CA to sign DSE node certificates for SSL.

```
$ openssl x509 -req -CA 'path/to/rootca.crt' \ -CAkey 'path/to/rootca.key' \ -in
    signing_request.csr \ -out client_cert.crt_signed \ -days 3650 \ -CAcreateserial \ 
    -passin pass:rootca_password
```

- **rootca.crt**: Certificate used to sign (authorize) DSE node SSL certificates.

5. Copy the cqlshrc.sample.ssl file to the ~/.cassandra directory. The following example uses the default location for a package installation:

```
$ cp /etc/dse/cassandra/cqlshrc.sample.ssl ~/.cassandra
```

6. Rename the file to cqlshrc. The file is typically located in ~/.cassandra/cqlshrc.

   If cqlsh finds the cqlshrc file located in the home directory, cqlsh moves the file to ~/.cassandra/cqlshrc upon its next invocation and shows a message that the file moved.

7. Specify the location of the SSL certificate file, either using the SSL_CERTFILE environment variable or the [ssl] cqlshrc parameters.

   If you created your own root CA, use the root certificate rootca.crt. If using an external certificate from a well-known root CA, extract the certificate from the dse-truststore.jks truststore.

   - **Environment variable**:
     
     Use the SSL_CERTFILE variable to specify the path to the certificate file:

     ```
     $ EXPORT SSL_CERTFILE='path/to/rootca.crt'
     ```

   - **cqlshrc parameter**:
     
     In the [ssl] section of the cqlshrc file, use the certfile parameter to specify the path to the root certificate:

     ```
     [ssl]
     certfile = path/to/rootca.crt
     validate = true
     userkey = client_key.key
     usercert = client_cert.crt_signed
     ```

   - **rootca.crt**: Certificate used to sign (authorize) DSE node SSL certificates.
   - **client_key.key**: Key certificate used for cqlsh.
   - **client_cert.crt_signed**: Signed security certificate to use when connecting to a node using cqlsh.

8. Restart cqlsh.
Configuring SSL

Example: SSL example

DataStax Enterprise provides a sample `cqlshrc.sample.ssl` file that you can use as a starting point.

```plaintext
[authentication]
username = fred
password = !!bang!!$

[connection]
hostname = 127.0.0.1
port = 9042
factory = cqlshlib.ssl.ssl_transport_factory

[ssl]
certfile = path/to/rootca.crt
; Optional, true by default.
validate = true
userkey = client_key.key
usercert = client_cert.crt_signed
[certfiles]
; Optional section, overrides the default certfile in the [ssl] section.
10.209.182.160 = ~/keys/cassandra01.cert
10.68.65.199 = ~/keys/cassandra02.cert
```

When validate is enabled, you must create a PEM key which is used in the `cqlshrc` file.

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.

Enabling SSL encryption for DSEFS

There are two parts to enabling SSL encryption for the DataStax Enterprise File System (DSEFS):

- Node-to-node encryption
- Client-to-node encryption

Enabling node-to-node encryption in DSE automatically enables encrypted communication between DSEFS nodes. DSE nodes with client-to-node encryption enabled allow SSL connections from the DSEFS shell.

Configuring the DSEFS shell to use SSL encryption

In most cases, you don't need to add any DSEFS shell settings to connect using SSL. If a `~/.dse/dsefs-shell.yaml` configuration file cannot be found, DSEFS shell attempts to load server-side configuration and SSL settings from DSE configuration files.

To manually configure SSL, create and edit the DSEFS shell configuration file. The DSEFS shell is configured in the `~/.dse/dsefs-shell.yaml` configuration file. Add the following settings to enable SSL encryption:

```yaml
encryption_options:
  enabled: true
  optional: true
  truststore:
    truststore_type:
    truststore_password:
  keystore:
    keystore_type:
    keystore_password:
```
Configuring SSL

```yaml
protocol:
algorithm:
cipher_suites:
require_endpoint_verification: false
```

The same settings can be given as `dse fs` command-line options, except `keystore_password`, `truststore_password`, and `cipher_suites`. If passwords are not given in the configuration file, they will be prompted for at the DSEFS shell startup. The command line options override settings read from the configuration file.

If a non-optional secure connection is established, a `[secure]` flag will appear in the prompt of the DSEFS shell.

### enabled
Whether to enable client-to-node encryption.
Default: false

### optional
Whether to allow unsecured connections when client encryption is enabled.
Default: false

### truststore
Relative path from DSE installation directory or absolute path to truststore containing the trusted certificate for authenticating remote servers.

Truststore password and path is only required when `require_client_auth` is set to `true`.
Default: `resources/dse/conf/.truststore`

#### truststore_type
Valid types are JKS, JCEKS, and PKCS12.
Default: commented out (JKS)

#### truststore_password
Password for the truststore. This must match the password used when generating the keystore and truststore.

Truststore password and path is only required when `require_client_auth` is set to `true`.
Default: `cassandra`

### keystore_type
Valid types are JKS, JCEKS, PKCS12, and PKCS11. For file-based keystores, use PKCS12.

DSE 6.0 uses `store_type` for this option. In DSE 6.7, you may use `store_type` or `keystore_type`. If you specify both, `keystore_type` takes precedence. By default, `store_type` is not listed in `cassandra.yaml`.
Default: commented out (JKS)

### keystore
Relative path from DSE installation directory or absolute path to the Java keystore (JKS) suitable for use with Java Secure Socket Extension (JSSE), which is the Java version of the Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols. The keystore contains the private key used to encrypt outgoing messages.
Default: `resources/dse/conf/.keystore`

#### keystore_password
Password for the keystore.
Default: `cassandra`

### protocol
Default: commented out (TLS)

### algorithm
Default: commented out (SunX509)

### cipher_suites
Supported ciphers:

- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
Configuring SSL

- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

Default: commented out

require_client_auth
Whether to enable certificate authentication for client-to-node encryption. When not set, the default is false.
When set to true, client certificates must be present on all nodes in the cluster.
Default: commented out (false)

Reference: SSL instruction variables

The following variables are used throughout this section to demonstrate how to create local keystore files when configuring SSL on a two node, single datacenter cluster.

Anyone with access to the CA key and signing certificate can authorize hosts as the root certificate authority. Always secure these files.

Root certificate authority (CA) variables

rootca_path
Directory where root certificate is created and stored. DataStax recommends securing this directory, ideally on a computer isolated from the network.

rootca.conf
Root CA configuration file.

Distinguished Name (DN) properties

CA_DN
Title for the section containing the Distinguished Name (DN) properties for the CA.

rootca_password
Password for the generated file used to sign certificates.

CC
Two letter country code, such as US for United States or JP for Japan. See Nations Online for a complete list of country codes.

org_name
Name of your organization.

cluster_name
Name of your DataStax Enterprise (DSE) cluster.

CA_CN
Common Name (CN) for the root CA.

Key and signing certificate

rootca.key
Key file for the root CA certificate.

rootca.crt
Certificate used to sign (authorize) DSE node SSL certificates.

Truststore and keystore variables

dse-truststore.jks
Truststore that contains the root certificate.
   Use the same truststore that contains the root certificate on all nodes.

node-keystore.jks
Keystore for the individual node.
node-key_password
  Password used to protect the individual private key.

keystore_password
  Password used to protect the private key of the key pair.

truststore_password
  Password required to access the keystore.

node_certificate_path
  Location where the certificate file for each DSE node is created. Typically, SSL certificates and keys are
generated on a secure system that is isolated from the network.

node_name
  Fully Qualified Domain Name (FQDN) of the node, such as `ip-10-200-182-183.example.com`. If using
  the FQDN as the `node_name`, you can add the IP address as a subject alternative name (SAN) so that
  the certificate protects the IP address in addition to the domain name.

node_ip_address
  If using the domain name as the `node_name` for the CA, add `san-ip:ip_address` to the `-ext` option.
  Using the IP address as a subject alternative name (SAN) ensures that the certificate protects the IP
  address in addition to the domain name. For example:

  
  -ext "san-ip:10.200.100.52"

signing_request.csr
  Certificate signing request (CSR) that is passed to the Certificate Authority (CA) to sign the certificate.
The CSR typically includes the public key, plus associated metadata such as the Common Name (CN),
Organization (O), Organization Unit (OU), and Country (C).

signing_request.crt_signed
  The signed certificate file to create, using the certificate signing request (CSR) (`signing_request.csr`) as the input file.

san_config_file.conf
  If using the domain name as the `node_name` and the node IP address as a subject alternative name
  (SAN), create a temporary configuration file and pass it in using the `-extfile` option. In the configuration
  file, use the `subjectAltName` parameter to specify the DNS and IP. For example:

  
  subjectAltName=DNS:node_name,IP:node_ip_address

You can specify multiple SANs in the same configuration file:

  subjectAltName=DNS:domain1,IP:10.200.100.52
  subjectAltName=DNS:domain1,IP:10.200.101.63
  subjectAltName=DNS:domain1,IP:10.200.111.74
  subjectAltName=DNS:domain1,IP:10.200.121.85
Chapter 9. Securing Spark connections

Communication between Spark applications and transactional nodes, masters and workers, and intercommunication between Spark drivers and executors can be encrypted. Encryption must be configured individually on each node in the cluster.

Encryption between the Spark processes and DSE is configured by enabling client-to-node encryption in cassandra.yaml.

Encryption between Spark applications, including between the Spark master and worker, is configured by enabling Spark security in dse.yaml.

Encryption between the Spark driver and executors in client applications is configured by enabling Spark security in the application configuration properties, or by default in spark-defaults.conf and spark-daemon- defaults.conf in the Spark configuration directory.

The default location of the Spark configuration files depends on the type of installation:

- **Package installations:** /etc/dse/spark/
- **Tarball installations:** installation_location/resources/spark/conf

1. Enable mutual authentication and encryption between Spark master and worker nodes in dse.yaml.

   In DSE 6.7.4 and later, when DSE authentication is enabled with authentication_options in dse.yaml, Spark security is enabled regardless of these settings.

   a. Enable mutual authentication by setting spark_security_enabled to true.

   b. Enable encryption by setting spark_security_encryption_enabled to true.

   ```
   spark_security_enabled: true
   spark_security_encryption_enabled: true
   ```

   To enable encryption, you must also enable mutual authentication.

2. To encrypt communication between the Spark application and master, DSE inherits the client-to-cluster connection encryption options.

3. The Spark web UI by default uses client-to-cluster encryption settings to enable SSL security in the web interface. To enable SSL security separately from DSE client-to-cluster encryption, change the settings in dse.yaml under spark_ui_options. Set encryption to custom, then set the keystore settings in encryption_options.

   The Spark web UI server automatically discards all cipher algorithms that end with a suffix SHA, SHA1 or MD5. If you manually specify cipher suites for encryption, make sure the specified cipher suites are not ignored by the server and are supported by the web browsers used to access the Spark web UI. For example, two 256 bit cipher suites that are supported by both the server and current web browsers are TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 and TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384.

4. The AlwaysOn SQL server can optionally use SSL to encrypt client connections.

5. Restart the node to recognize the changes.

6. For each Spark application, set the following client encryption options to true in the spark-defaults.conf file to enable encryption between the Spark driver and executors.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spark.authenticate</code></td>
<td>Enables or disables mutual authentication using a secret key. The default is false.</td>
</tr>
<tr>
<td><code>spark.authenticate.enableSaslEncryption</code></td>
<td>Enables or disables SASL encryption between a Spark driver and its executors. The default is false.</td>
</tr>
<tr>
<td><code>spark.io.encryption.enabled</code></td>
<td>Enables or disables encryption for data stored by drivers directly on disk.</td>
</tr>
</tbody>
</table>

To enable encryption by default for all Spark applications, modify the options in the `spark-defaults.conf` file in the Spark configuration directory. To encrypt data stored on the server by default, modify the options in the `spark-daemon-defaults.conf` file.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spark.io.encryption.enabled</code></td>
<td>Enables or disables encryption for data stored by executors and shuffle services directly on disk.</td>
</tr>
</tbody>
</table>