DataStax 5.1 Security Guide Earlier DSE version Latest 5.1 patch: 5.1.17
# Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>About DSE Advanced Security</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Security FAQs</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Security checklists</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>- Database security checklist</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>- Search security checklist</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>- Analytics security checklist</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>- Graph security checklist</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>Securing the environment</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>- Securing ports</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>- Securing the TMP directory</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>Configuring DSE Unified Authentication</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>- About DSE Unified Authentication</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>- Implementing DSE Unified Authentication</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>- Implementing without downtime in production</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>- Setting security keyspace replication factors</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>- Setting up Kerberos</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>- Kerberos guidelines</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>- Enabling JCE Unlimited</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>- Preparing DSE nodes for Kerberos</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>- Enabling DSE Unified Authentication</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>- Defining a Kerberos scheme</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>- Defining an LDAP scheme</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>- Configuring JMX authentication</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>- DSE Unified Authentication</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>- JMX native authentication</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>- Managing credentials, role, and permissions cache settings</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>Connecting to authentication enabled clusters</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>- About client connections</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>- Providing credentials with DSE tools</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>- Providing credentials with nodetool</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>- Providing credentials with JConsole</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>- Providing credentials with cqlsh</td>
<td>45</td>
</tr>
</tbody>
</table>
Chapter 7. Managing roles

About roles
Creating superuser accounts
Creating roles for internal mode
Creating roles for LDAP mode
Creating roles for Kerberos principals
Binding a role to an authentication scheme
Configuring proxy roles for applications

Chapter 8. Authorizing access to database resources

About permissions
Managing keyspace and table permissions
Setting row-level permissions
Managing access to DSE Graph keyspaces
Authorizing remote procedure calls for CQL execution
JMX MBean permissions
Search index permissions
Managing Spark application permissions

Chapter 9. Auditing activity

Enabling data auditing
Configuring audit logging
Log formats
Configuring audit logging to a database table
CassandraAuditWriter table columns
Configuring auditing for DSE Search

Chapter 10. Transparent data encryption

About Transparent Data Encryption
Configuring local encryption
Setting up local encryption keys
Encrypting configuration file properties
Encrypting system resources
Encrypting tables........................................................................................................................................... 91
Rekeying existing data..................................................................................................................................... 94
Troubleshooting encryption key errors........................................................................................................... 95
Configuring KMIP encryption......................................................................................................................... 96
Adding a KMIP host......................................................................................................................................... 96
Encrypting configuration file properties.......................................................................................................... 98
Encrypting system resources............................................................................................................................ 100
Encrypting table data....................................................................................................................................... 101
Expanding an encryption key............................................................................................................................ 102
Rekeying tables using a new key...................................................................................................................... 103
Troubleshooting KMIP connections................................................................................................................ 105
Encrypting Search indexes............................................................................................................................... 105
Encrypting new Search indexes......................................................................................................................... 106
Encrypting existing Search indexes................................................................................................................ 106
Tuning encrypted Search indexes...................................................................................................................... 108
Migrating encrypted tables from earlier versions............................................................................................. 108
Bulk loading data between TDE-enabled clusters.......................................................................................... 110

Chapter 11. Configuring SSL......................................................................................................................... 112

About SSL........................................................................................................................................................ 112
Setting up SSL certificates.............................................................................................................................. 112
Securing internal transactional node connections............................................................................................ 117
Securing client to cluster connections............................................................................................................ 118
Securing Spark connections............................................................................................................................. 119
Using CQL shell (cqlsh) with SSL.................................................................................................................. 121
Setting up SSL for nodetool, dsetool, and dse advrep...................................................................................... 122
Setting up SSL for jconsole (JMX).................................................................................................................. 124
Connecting sstableloader to a secured cluster.................................................................................................. 124
Enabling SSL encryption for DSEFS................................................................................................................ 125
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 Advanced 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

New security features in DSE 5.1

New security features include granular access control that allows permissions on table rows and search indexes; role assignment managed by LDAP groups; DSE proxy authentication for database clients; and JMX authentication integrated with DSE Unified Authentication (LDAP or internal).

- Row-level access control (RLAC) for row-level permissions
- Role-based access control (RBAC) for search indexes
- Configuring proxy roles for applications
- Integrated Creating roles for LDAP mode role management
- Enabling DSE Unified Authentication with internal or LDAP

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 Creating superuser accounts 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 roles.

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 Managing roles

What authentication mechanisms are supported?

DataStax 5.1 Security Guide Earlier DSE version Latest 5.1 patch: 5.1.17
DSE Advanced Security FAQs

- Internal: Connections provide credentials for a role that has an internally stored password, no additional configuration is required, see Managing roles.

- LDAP: Connections provide LDAP credentials and 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:

```
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:

```
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.

Unrestricting 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:

```
GRANT permission
on 'filtering_data' ROWS IN keyspace_name.table_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:

```
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 specify which categories of audit events (administration, authentication, DML, DDL, DCL, and query operations) to log, and whether to omit operations against specific keyspaces from audit logging.

**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. 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).

New features include:

- Row level access control (RLAC)
- Role-based access control for search indexes
- DSE proxy authentication for database clients
- Integrated LDAP role management
- JMX authentication with internal or LDAP

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 store in the internal database
    - LDAP: External LDAP service, such as Active Directory
    - Kerberos: MIT Kerberos tickets checked against an external Key Distribution Server (KDS)
Security checklists

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
  - 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 Managing roles and Authorizing access to database resources.

- **Audit activity:**
  Log and monitor activity for database resources, see Enabling data auditing in DataStax Enterprise.

- **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. 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 cores 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 Managing roles, Controlling access to search indexes, and Accessing cores 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 Enabling data auditing in DataStax Enterprise.

• **Transparent Data Encryption (TDE)**
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:**
  # Distinct secrets for internode and per application, see Configuring Spark nodes.
  # Native authentication for users of each application executor (run as) and isolation of related data, see Configuring Spark nodes.
  # Spark UI internal or LDAP authentication, see Monitoring Spark with the web interface.
Security checklists


# 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.

# 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 Enabling data auditing in DataStax Enterprise.

• 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 Providing credentials for DSE Graph.

• Authorization:

Limit access to graph data by defining roles for DSE Graph keyspaces and tables, see Managing access to DSE 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 Enabling data auditing in DataStax Enterprise.

- **Transparent Data Encryption:**
  Encrypt data in DSE Graph index tables, see Transparent data encryption
  
  Cached data is not encrypted. Encryption may slightly impact performance.

- **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.
Chapter 4. Securing the environment

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 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 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>Public facing ports</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>SSH (default)</td>
<td>See your OS documentation on sshd.</td>
</tr>
<tr>
<td>DataStax Enterprise public ports</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(random)</td>
<td>Spark port for all block managers to listen on. These ports exist on both the driver and the executors.</td>
<td>Ensure traffic is unrestricted between DSE Analytics nodes.</td>
</tr>
<tr>
<td>(random)</td>
<td>Spark port for the driver to listen on. Used for communicating with the executors and the standalone Master.</td>
<td>Ensure traffic is unrestricted between DSE Analytics nodes.</td>
</tr>
<tr>
<td>(local hostname)</td>
<td>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, DSE then tries 4041 and continues incrementing until it finds an open port.</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>
<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>Port</td>
<td>Service</td>
<td>Configurable in</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------</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></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 Securing client to cluster connections.</td>
</tr>
<tr>
<td>9160</td>
<td>DSE client port (Thrift port). OpsCenter agents make Thrift requests to their local node on this port.</td>
<td>cassandra.yaml</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>
<tr>
<td></td>
<td><strong>OpsCenter public ports</strong></td>
<td></td>
</tr>
<tr>
<td>8888</td>
<td>OpsCenter web site port. The opscenterd daemon listens on this port for HTTP requests coming directly from the browser. See OpsCenter ports reference.</td>
<td>opscenterd.conf</td>
</tr>
<tr>
<td></td>
<td><strong>Inter-node ports</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>DSE database inter-node communication ports</strong></td>
<td></td>
</tr>
<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></td>
<td>See JMX options in Tuning Java resources.</td>
<td></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 resources.</td>
</tr>
<tr>
<td></td>
<td><strong>DataStax Enterprise inter-node ports</strong></td>
<td></td>
</tr>
<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>8984</td>
<td>DSE Search inter-node communication port used for releases earlier than 5.0 and during upgrades to 5.0. Deprecated for DataStax Enterprise 5.0 and later.</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.
Chapter 5. Configuring DSE Unified Authentication

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 Managing roles.
  # LDAP: Connections provide LDAP credentials and 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 first tries the `default_scheme` and then each scheme defined in `other_schemes` in order.

To provide credentials from clients and applications, see Connecting to authentication enabled clusters.

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** process used to assign roles to a user:
  # Internal: One to one mapping. Matches the user name to a DSE role. Requires a role for each user. See Creating roles for internal mode.
  # LDAP: One to many mapping. Matches the user LDAP group names to DSE roles. Users can have more than one role. See Creating roles for LDAP mode.
    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 row-level permissions.

Implementing DSE Unified Authentication

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 Implementing without downtime in production.

To configure DSE Unified Authentication:
Configuring DSE Unified Authentication

1. **Setting security keyspaces replication factors:** Ensure that required data for logins and permission management are accessible and in all datacenters.

2. **Enabling DSE Unified Authentication:** By default DSE Unified Authentication is disabled.

3. Configuring authentication and authorization methods (schemes):
   - Internally stored passwords, no additional configuration is required create roles with passwords as described in Managing roles
   - External LDAP, see Defining an LDAP scheme
   - Kerberos, see Defining a Kerberos scheme

4. **Enabling DSE Unified 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.
   
   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 Creating superuser accounts.
   
   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.

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 Managing roles
   - Use the latest DataStax certified drivers in all applications connecting to transactional nodes that have DSE Unified Authentication enabled. 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 Spark components between each other.

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

- Providing credentials with `cqlsh`
- Providing credentials for DSE Graph
- Providing credentials with DSE tools
- Providing credentials with `nodetool`
- Providing credentials with DSE tools

**Implementing without downtime in production**
To enable access control without downtime, use transitional modes to temporarily support applications and users that do not have accounts without interrupting services.
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 Setting 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 transitional_mode to normal.

   See Enabling DSE Unified Authentication.

5. Turn on logging to allow you to verify that applications are able to authenticate, see Enabling data auditing in DataStax Enterprise.

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 Creating superuser accounts.

   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 Managing roles.

10. Use the audit logs or when using Kerberos the KDC logs to verify that all applications are able to access the transactional nodes and have the permissions required to execute requests, see Formats of DataStax Enterprise logs.

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

12. Perform a rolling restart.

Setting security keyspaces replication factors

You must configure the replication factors appropriate for using DSE Security in production environments. The keyspaces that require an increased replication factor are:

   - system_auth
   - dse_security

DSE Unified Authentication stores authentication and authorization in the security keyspaces. Read access to security_auth and dse_security keyspaces is implicitly given to every authenticated user.
In order to log in for the first time using default user cassandra, you must increase the `system_auth` keyspace RF. If the keyspace is not available, the cassandra account log in fails. The cassandra account uses a consistency level of QUORUM for all requests.

**Security keyspaces and tables**

Read access to these system tables is implicitly given to every authenticated user because the tables are used by most DSE tools:

- **system_auth keyspace**
  Contains authorization and internal authentication data.

**Table 1: system_auth tables**

<table>
<thead>
<tr>
<th>Table</th>
<th>Columns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource_role_permissions_index</td>
<td>resource (PK), role</td>
<td>Stores the role and a resource that the role has a set permission.</td>
</tr>
<tr>
<td>role_permissions</td>
<td>role (PK), resource, permissions</td>
<td>Stores the role, resource (for example keyspace_name/table_name), and the permission that the role has to access the resource.</td>
</tr>
<tr>
<td>role_members</td>
<td>role (PK), member</td>
<td>Stores the roles and role members.</td>
</tr>
<tr>
<td>roles</td>
<td>role (PK), can_login, is_superuser, member_of, salted_hash</td>
<td>Stores the role name, whether the role can be used for login, whether the role is a superuser, what other roles the role may be a member of, and a bcrypt salted hash password for the role.</td>
</tr>
</tbody>
</table>

- **dse_security keyspace**
  Contains DSE Spark, Kerberos digest data, and role options.

**Table 2: dse_security tables**

<table>
<thead>
<tr>
<th>Table</th>
<th>Columns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role_options</td>
<td>role, options</td>
<td>Role options.</td>
</tr>
<tr>
<td>digest_tokens</td>
<td>id, password</td>
<td>Kerberos digest tokens when enabled.</td>
</tr>
<tr>
<td>spark_security</td>
<td>dc, shared_secret</td>
<td>Share secret for Spark.</td>
</tr>
</tbody>
</table>

**Default replication factors**

The default replication factor for the `system_auth` and `dse_security` keyspaces is 1.

All analytics keyspaces are initially created with the `SimpleStrategy` replication strategy and a replication factor (RF) of 1. Each of these must be updated in production environments to avoid data loss. DataStax recommends changing the replication factor before enabling authentication. DSE uses a consistency level of `LOCAL_ONE` for all security keyspaces queries, except when using the cassandra role. For the cassandra role, DSE uses the consistency level `QUORUM`. Only use the cassandra role to login and create your own full access account and then drop the cassandra role.

**Recommended replication factors**

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

- **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 cassandra.yaml, see Security properties. 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, 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, to adjust cache interval see Security properties.
Configuring DSE Unified Authentication

- **dse_security**: Contains DSE Analytic (Spark), DSE Client digest tokens, and other Kerberos related data. Required for related DSE services, less critical for pure database activities. DataStax recommends using a replication factor of 1 per datacenter if none of these services are present. See the corresponding set up instructions for recommended dse_security if these services are present.

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

1. **Change the system_auth keyspace RF:**
   ```sql
   ALTER KEYSPACE system_auth
   WITH REPLICATION= {'class' : 'NetworkTopologyStrategy',
                     'data_center_name' : N,
                     'data_center_name' : N};
   ```

   Every time you add or remove a datacenter, you **must** manually reconfigure the system_auth keyspace.

2. **Change the dse_security keyspace RF:**
   ```sql
   ALTER KEYSPACE dse_security
   WITH REPLICATION= {'class' : 'NetworkTopologyStrategy',
                     'data_center_name' : N,
                     'data_center_name' : N};
   ```

   Every time you add or remove a datacenter, you **must** manually reconfigure the dse_security keyspace. If DataStax Enterprise or Spark security options are enabled on the cluster, you must also increase the replication factor for the dse_leases keyspace across all logical datacenters.

3. **Run the nodetool repair on the security keyspaces.**
   ```
   nodetool repair --full system_auth
   nodetool repair --full dse_security
   ```

   After changing the replication strategy, you **must** run nodetool repair with the --full option.

### Setting up Kerberos

DataStax Enterprise 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`).
- Latest build of a Technology Compatibility Kit (TCK) Certified OpenJDK version 8 or Oracle Java SE Runtime Environment 8 (JRE or JDK). Earlier or later versions are not supported.
  Although Oracle JRE/JDK 8 is supported, DataStax does more extensive testing on OpenJDK 8. This change is due to the end of public updates for Oracle JRE/JDK 8. Java 9 is not supported.

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:

- Using CQL shell (`cqlsh`) with SSL
- Using `dsetool` with Kerberos enabled cluster
- 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 Enabling data auditing in DataStax Enterprise.

DataStax recommends choosing either Kerberos or SSL because of overlap in features, and using it for both encryption and authentication.

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:

```java
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:

Install JCE using webupd8 PPA repository:

```bash
$ 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:

```bash
$ 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"`. 
Configuring DSE Unified Authentication

- 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 Upgrade restrictions and limitations.

Prerequisites:

Complete the following prerequisites:

- Each node has a DNS entry that resolves the hostname to the correct IP address
- Each node uses NTP for the system time
- All KDS requirements have been met, see Kerberos guidelines.
- Latest build of a Technology Compatibility Kit (TCK) Certified OpenJDK version 8 or Oracle Java SE Runtime Environment 8 (JRE or JDK). Earlier or later versions are not supported.
  
  Although Oracle JRE/JDK 8 is supported, DataStax does more extensive testing on OpenJDK 8. This change is due to the end of public updates for Oracle JRE/JDK 8. Java 9 is not supported.
- Each node has the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files enabled, see Enabling JCE Unlimited.
  
  If you are not using the JCE Unlimited Strength Jurisdiction Policy, make sure that your ticket granting principal does not use AES-256.

1. Verify the DNS resolves hostname and NTP settings on each node:

   a. For Kerberos the hostname is used for the Service Principal name, therefore the hostname must resolve to the correct IP address. To verify:

      `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
b. Kerberos authentication is sensitive to system time, manually set system clocks may cause issues. On each node, confirm that NTP is configured and running:

```
$ ntpq -p
```

2. On each node, install Kerberos:

- RHEL-based systems:
  ```
sudo yum install krb5-workstation krb5-libs krb5-pkinit-openssl
  ```

- Debian-based systems:
  ```
sudo apt-get install krb5-user krb5-config krb5-pkinit
  ```

3. Your the krb5.conf for your REALM in the /etc directory on each DataStax Enterprise node.

   The krb5.conf file contains Realm configuration required by Kerberos, see MIT Kerberos krb5.conf documentation. DataStax recommends not using DNS lookup for KDC, and REALM entries. Relying on DNS may negative impact performance and functionality. Verify that the libdefaults section contains the following entries.

   ```
   [libdefaults]
   dns_lookup_kdc = false
   dns_lookup_realm = false
   ```

4. On the Key Distribution Center (KDC) server, create a Service Principal and keytab for each node:

   a. Add a DSE Service Principal and HTTP Principal for each node using the kadmin command:

   ```
kadmin -p user_name/admin
addprinc -randkey service_name/FQDN
addprinc -randkey HTTP/FQDN
quit
```
### Configuring DSE Unified Authentication

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FQDN</td>
<td>The fully qualified domain name of the DSE node.</td>
</tr>
<tr>
<td>-randkey</td>
<td>Sets the key of the principal to a random value.</td>
</tr>
</tbody>
</table>

For more details on using `kadmin`, see the `kadmin help`.

**Example:**

```
kadmin -p kadmin/admin
kadmin: addprinc -randkey dse/node1.example.com
kadmin: addprinc -randkey HTTP/node1.example.com
kadmin: addprinc -randkey dse/node2.example.com
kadmin: addprinc -randkey HTTP/node2.example.com
```

**b.** 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
kadmin/admin@EXAMPLE.COM
```

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

**c.** 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
```

**d.** 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)
```
where: -e displays the encryption type and -kt displays the keytab file and its timestamp.

e. 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
```

f. 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:

- **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 DSE Authenticator are not supported.

OpsCenter also provides support for LDAP configuration, authenticating users.

**Prerequisites:**

Complete the following procedures 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 Setting 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 Implementing without downtime in production 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:
Configuring DSE Unified Authentication

a. Verify that **authenticator** is set to `DseAuthenticator`.

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

b. Verify that **authorizer** is set to `DseAuthorizer`.

   authorizer: com.datastax.bdp.cassandra.auth.DseAuthorizer

c. Verify that **role_manager** is set to `DseRoleManager`.

   role_manager: com.datastax.bdp.cassandra.auth.DseRoleManager

d. If you are **Setting row-level permissions** with RLAC, tune these **Security-related cache settings**:

   permissions_validity_in_ms: 2000
   permissions_update_interval_in_ms: 2000
   permissions_cache_max_entries: 1000

2. In the `dse.yaml` file, configure the corresponding options:

   a. Configure the DSE Authenticator 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
      #   transitional_mode: disabled
      #   other_schemes:
      #   scheme_permissions: false
      ```

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

      • Required settings. Enable DSE Authenticator and select a scheme:

      ```yaml
      enabled: true
      default_scheme: internal
      ```

      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>
<thead>
<tr>
<th>Table 3: Required authentication_options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
</tr>
<tr>
<td>------------------------------------------</td>
</tr>
<tr>
<td><a href="https://docs.datastax.com/en/dse/5.1/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml_auth_enabled">https://docs.datastax.com/en/dse/5.1/dse-admin/datastax_enterprise/config/configDseYaml.html#configDseYaml_auth_enabled</a></td>
</tr>
</tbody>
</table>
Configuring DSE Unified Authentication

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_scheme</td>
<td>Specifies the authentication scheme when not defined in the connection:</td>
</tr>
<tr>
<td></td>
<td># internal - Basic 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>
</tbody>
</table>

**Table 4: Optional authentication_options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>other_schemes</td>
<td>You cannot use other_schemes with DSE components that use Thrift, such as CFS and the CassandraThriftstore in Analytics datacenters. Only the default_scheme is used when using components that use Thrift drivers.</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>
<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. Default.</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># permissive - Allow all connections that provide credentials. Maps authenticated superusers to their role AND maps all other users to anonymous.</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 5: 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>

3. Configure selected authentication scheme options:

- Defining an LDAP scheme
- Defining a Kerberos scheme
For DSE to start, 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 `dseutil` operations, see Enabling DSE Unified Authentication.

5. Restart DSE, see Starting DataStax Enterprise as a service or Starting DataStax Enterprise as a stand-alone process.

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

- Creating superuser accounts
- Create roles and set up permissions, see Managing roles

**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 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.

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

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

   ```
   rpc_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 cassandara account.
### Configuring DSE Unified Authentication

Table 7: **authentication_options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://docs.datastax.com/en/dse/5.1/dse-admin/datastaxenterprise/config/configDseYaml.html#configDseYaml__auth_enabled">https://docs.datastax.com/en/dse/5.1/dse-admin/datastaxenterprise/config/configDseYaml.html#configDseYaml__auth_enabled</a></td>
<td>Turns on authentication using the default scheme.</td>
</tr>
<tr>
<td><strong>default_scheme</strong></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><strong>other_schemes</strong></td>
<td>You cannot use <code>other_schemes</code> with DSE components that use Thrift, such as CFS and the CassandraHiveMetastore in Analytics datacenters. Only the default_scheme is used when using components that use Thrift drivers.</td>
</tr>
<tr>
<td><strong>scheme_permissions</strong></td>
<td>Validate that the role mapped to user matches the authentication scheme. Grant the role permission to the scheme.</td>
</tr>
<tr>
<td><strong>allow_digest_with_kerberos</strong></td>
<td>Allow Kerberos digest-md5 authentication.</td>
</tr>
<tr>
<td><strong>plain_text_without_ssl</strong></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. Default.</td>
</tr>
<tr>
<td></td>
<td>• allow - Allow the request without any warning.</td>
</tr>
<tr>
<td><strong>transitional_mode</strong></td>
<td>Sets the behavior when authentication fails and credentials are missing:</td>
</tr>
<tr>
<td></td>
<td>• permissive - Allow all connections that provide credentials. Maps authenticated superusers to their role AND maps all other users to anonymous.</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 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 8: **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 _HOST with the FQDN of the host in the service_principal and http_principal settings. The UNIX user running DataStax Enterprise must also have read permissions on the keytab.</td>
</tr>
</tbody>
</table>
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</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) |
| http_principal | The `http_principal` is used by the Tomcat application container to run DSE Search. The Tomcat web server uses GSS-API mechanism (SPNEGO) to negotiate the GSSAPI security mechanism (Kerberos). Set `REALM` to the name of your Kerberos realm. In the Kerberos principal, `REALM` must be uppercase. |
| qop | 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 Managing roles.

What’s next:

When initially configuring authentication complete the set up by:

- Perform a rolling restart.
- Creating superuser accounts
- Creating roles for internal mode
- Creating roles for LDAP mode

**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.

DataStax Enterprise allows you to adjust the LDAP connection using the LDAP tuning switches. To debug LDAP connections enable logging, see Enabling authentication debug.

**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:
```
Configuring DSE Unified Authentication

... scheme: ldap

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

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 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 <code>user_search_base</code> and <code>group_search_base</code>. Comment out to use an anonymous bind.</td>
</tr>
<tr>
<td>search_password</td>
<td>Password of the <code>search_dn</code> user.</td>
</tr>
</tbody>
</table>
### Configuring DSE Unified Authentication

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_search_base</td>
<td>Distinguished name (DN) of the object to start the recursive search for user entries for authentication and role management memberof searches. For example to search all users in example.com, <code>ou=users,dc=example,dc=com</code>. Active Directory uses a different search base, typically <code>CN=search,CN=Users,DC=ActDir_domname,DC=internal</code>. For example, <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>{uid={0}}</code>. 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>, you might need to change <code>server_port</code> to the SSL port of the LDAP server. 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>, change <code>server_port</code> to the TLS port of the LDAP server. The default 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>
<tr>
<td>truststore_type</td>
<td>The type of truststore. The default value is <code>jks</code>.</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:

  ```
  usermemberof_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.
Configuring DSE Unified Authentication

Table 9: 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 user_search_base and user_search_filter.</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 10: 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.

What’s next:

- Creating superuser accounts
- Creating roles for internal mode
- Creating roles for LDAP mode

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. Only use this method to require authentication to run utilities such as nodetool when DSE Unified Authentication is disabled on the DSE.

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. 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. On DSE nodes where you want to disable access, set the JMX remote authenticate to false for remote and/or local:

      ```
      JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=false"
      
      Disabling local may cause unexpected behavior, such as preventing dsetool from running on the node.
      
   3. On nodes that allow access, set the path to the credentials file:

      ```
      JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.password.file=/etc/dse/cassandra/jmxremote.password"
      
      Ensure that the path is accessible to the user who runs cassandra.
      
   4. Create a file that contains a user name and password on each line and save it to the location entered in the previous step.

      ```
      sallyjones 123DSEadmin
      
   5. 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/dse/cassandra/jmxremote.access"
      
   6. If all nodes on the cluster were updated, perform a rolling restart; otherwise restart only the affected nodes.

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

```
nodetool -u sallyjones -pw 123DSEadmin status
```

The results should display.

```
Datacenter: DataStax
```
### Configuring DSE Unified Authentication

<table>
<thead>
<tr>
<th>Status=Up/Down</th>
<th>State=Normal/Leaving/Joining/Moving</th>
<th>Address</th>
<th>Load</th>
<th>Tokens</th>
<th>Owns</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rack</td>
<td></td>
<td>UN 10.200.182.180</td>
<td>316.76 KiB</td>
<td>1</td>
<td>?</td>
<td>5ca115f6-250a-4964-9a52-c10926031f1b rack1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UN 10.200.182.181</td>
<td>446.76 KiB</td>
<td>1</td>
<td>?</td>
<td>74a44407-5e26-43d4-83dc-aae9fe35c2f4 rack1</td>
</tr>
</tbody>
</table>

Datacenter: Solr

---

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

## Managing credentials, role, and permissions cache settings

By default, DataStax Enterprise (DSE) caches user credentials, role, and permissions. Caching allows multiple connection requests to occur within the specified period without repeating the entire authenticating process and issue multiple requests without querying the `system_auth` table for every action.

### Security-related cache settings

Adjust settings in the `cassandra.yaml` file:

**roles_validity_in_ms**

Default: 2000. Validity period for roles cache; set to 0 to disable. 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.

**roles_update_interval_in_ms**

Default: 2000. Enable to refresh interval for roles cache. Defaults to the same value as `roles_validity_in_ms`. 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 must be also.

**credentials_validity_in_ms**

Default: 2000. How many milliseconds credentials in the cache remain valid. This cache is tightly coupled to the provided PasswordAuthenticator implementation of `IAuthenticator`. If another `IAuthenticator` implementation is configured, the database does not use this cache, and these settings have no effect.

Credentials are cached in encrypted form. This may cause a performance penalty that offsets the reduction in latency gained by caching.

Cache credentials and permissions are not automatically invalidated after issuing a `REVOKE` statement.

This setting is disabled when set to 0.

**credentials_update_interval_in_ms**
Configuring DSE Unified Authentication

Default: same value as credentials_validity_in_ms. After this interval, cache entries become eligible for refresh. The next time the cache is accessed, the system schedules an asynchronous reload of the cache. Until this cache reload is complete, the cache returns the old values.

If credentials_validity_in_ms is nonzero, this property must also be nonzero.

permissions_validity_in_ms
Default: 2000. Fetching permissions can be resource intensive. Define how many milliseconds permissions in cache remain valid to manage performance impact of permissions queries. 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.

Cache credentials and permissions are not automatically invalidated after issuing a REVOKE statement.

This setting is disabled when set to 0.

permissions_update_interval_in_ms
Default: same value as permissions_validity_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 nonzero, roles_update_interval_in_ms must also be non-zero.

permissions_cache_max_entries
Default: 1000. 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 row-level permissions, use this formula:

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

If this option is not present in cassandra.yaml, manually enter it. See Enabling DSE Unified Authentication.

Row-level permissions cache
Configure the RLAC cache to manage performance impact of permissions queries when using row-level permissions. A properly configured RLAC cache can reduce performance impact on a node with only a minor latency penalty, but at the cost of more CPU consumption.

Disabling Authentication and Authorization caching
To disable configuration of authentication and authorization caches (credentials, roles, and permissions) via JMX, 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.
Chapter 6. Connecting to authentication enabled clusters

About client connections

After enabling DataStax Unified Authentication you must provide credentials in connection requests.

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

DataStax supports authentication on connections from the following clients:

- Astyanax
- cqlsh
- Drivers
- Hector
- pycassa
- JMX, nodetool and JConsole

Providing credentials with DSE tools

Through authentication, the database establishes the identity of the person or system that is attempting an operation. Authentication works with any combination of DSE authentication, LDAP pass-through authentication, and Kerberos authentication.

Authentication is supported for use with dse commands, dse client-tool, and the dsetool utility.

- To use dsetool with SSL encryption, see Setting up SSL for nodetool, dsetool, and dse advrep.
- To configure external client access to DataStax Enterprise commands, see Authorizing remote procedure calls for CQL execution.

Providing credentials

You can provide authentication credentials in several ways:

Command line

Provide login credentials on the command line:

```
$ dse [connection_options] subcommand
```

```
$ dse client-tool [connection_options] subcommand
```

```
$ dsetool [connection_options] subcommand
```

where the connection_options include:
Connecting to authentication enabled clusters

- `-f config_file` - the path to a configuration file that stores credentials. If not specified, then use `~/.dserc` if it exists.

  The configuration file can contain DataStax Enterprise and JMX login credentials. For example:

  ```
  username=username
  password=password
  jmx_username=jmx_username
  jmx_password=jmx_password
  ```

  The credentials in the configuration file are stored in clear text. DataStax recommends restricting access to this file only to the specific user.

- `dse -u username` is the user name to authenticate for database access.

- `dsetool -l username` is the user name to authenticate for database access.

- `-p password` is the password to authenticate for database access. If you do not provide a password on the command line, you are prompted to enter one.

- `-a jmx_username` is the user name for authenticating when local JMX authentication is enabled.

- `-b jmx_password` is the password for authenticating when local JMX authentication is enabled. If you do not provide a password on the command line, you are prompted to enter one.

- `subcommand` is any dse, dse client-tool, or dsetool subcommand.

`~/.dserc` file

Create a file named `.dserc` in your home directory. The `~/.dserc` file contains the cassandra user name and password:

```
username=username
password=password
```

When you launch a password-protected tool and authentication is not provided on the command line, the credentials in the `~/.dserc` file are used. The `~/.dserc` is ignored when a configuration file is specified with `-f`.

Environment variables for dse client-tool, dse gremlin-console, and Spark commands

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.

Use environment variables `DSE_USERNAME` and `DSE_PASSWORD` to specify a user name and password:

```bash
export DSE_USERNAME=username
export DSE_PASSWORD=password
```

Use environment variable `DSE_TOKEN` to specify a delegation token when Kerberos is used:

```bash
export DSE_TOKEN=delegation_token
```

Providing credentials with nodetool

After configuring JMX authentication, using `nodetool` requires the `-u` and `-pw` options to the `nodetool` commands.
Connecting to authentication enabled clusters

1. Run `nodetool` using a **username** and **password**:

   ```
   $ 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**.

Providing credentials with `cqlsh`

Typically, after configuring authentication, logging into `cqlsh` requires the `-u` and `-p` options to the `cqlsh` command. To set credentials for use when launching `cqlsh`, create or modify the `.cassandra/cqlshrc` file. When present, this file passes default login information to `cqlsh`. See the `cqlshrc.sample`.

1. Create or modify the `cqlshrc` file that specifies a role name and password.

   ```
   [authentication]
   username = fred
   password = !!bang!!$
   ```

2. Save the file in **home/.cassandra** directory and name it `cqlshrc`.
Connecting to authentication enabled clusters

3. Set permissions on the file to prevent unauthorized access, as the password is stored in plain text. The file must be readable by the user that starts cassandra.

   ```
   $ chmod 440 home/.cassandra/cqlshrc
   ```

4. Check the permissions on `home/.cassandra/cqlshrc_history` to ensure that plain text passwords are not compromised.

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=service_name
-Djava.security.auth.login.config=path_to_login_config
```

where:

- `service_name` is the service name component of the `service_principal` that is defined in the `dse.yaml` file
- `path_to_login_config` is the JAAS configuration file with the following options declared in it:

```
DseClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useTicketCache=true
    renewTGT=true;
};
```

Using cqlsh with Kerberos or user authentication

To use cqlsh with DataStax Unified Authentication, you can configure credentials in a file to avoid having to pass them for every login. You can use the sample files as a starting point, and use the --debug option to identify and resolve authentication problems.

**Example files**

DataStax Enterprise provides sample files and examples to help configure authentication:

- Kerberos example
- SSL 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.
Connecting to authentication enabled clusters

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 Using CQL shell (cqlsh) with SSL.

The settings for using both Kerberos and SSL are a combination of the Kerberos and SSL sections in these examples.

The supported environmental variables are KRB_SERVICE, SSL_CERTFILE, and SSL_VALIDATE variables.

**Debugging cqlsh authentication**

Use the `--debug` option to troubleshoot authentication problems with cqlsh. Pass the `--debug` option to cqlsh to populate the debug log message with the type of authentication that cqlsh is attempting.

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=dse@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";
  }
  ```

Providing credentials for DSE Graph

To run DataStax Enterprise Graph with DSE Unified Authentication, configure a user name and password or Kerberos in the Graph remote.yaml.
Connecting to authentication enabled clusters

• Use Kerberos guidelines for DSE Graph in production.

Authenticate DSE Graph users with Kerberos authentication using Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO). To use Kerberos authentication and SSL encryption with the Gremlin console, modify the remote.yaml to include the following:

```
hosts: [hostname-because-it's-kerberos]
username: null
password: null
jaasEntry: DseClient
# protocol is the same as the service_principal set in dse.yaml
protocol: your_dse_principal
connectionPool: {enableSsl: true}
```

A password set to null is ignored in the remote.yaml.

• To configure Graph for internal or LDAP authentication with DSE Unified Authentication:

Enter credentials in the remote.yaml file to use Gremlin console. For instance, the following lines are added:

```
username: realuser
password: password
connectionPool: {enableSsl: false}
```

A password is required for internal or LDAP authorization.

DSE has a credential cache, and the setting for this cache can greatly affect the performance of graph queries. The credentials_validity_in_ms is the critical setting in the dse.yaml file. Setting to 0 will cause DSE to re-authenticate the user on all operations requiring authentication. An additional setting search_validity_in_seconds will perform better if set to a higher value, such as 30 minutes.

• Set an environment variable to pass the Jaas configuration file location to the gremlin console:

```
$ export JAVA_OPTIONS=-Djava.security.auth.login.config=$HOME/jaas.config; dse gremlin-console
```

Required when the jaas.config file is not in the default location.

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.

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.
Connecting to authentication enabled clusters

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
```

Here is an example JAAS configuration file:

```
DseClient
{
   com.sun.security.auth.module.Krb5LoginModule required
       ...;
};
```
Chapter 7. Managing roles

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

About roles

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

A role is a database resource to which privileges to access other database resources are assigned. A role is also an individual user for authentication purposes when using the internal authentication scheme. To execute CQL commands in DSE authorization enabled environments, a user must have at least one role with login privileges.

Understanding role assignment

The DSE Role Manager mode effects how a role is assigned to an authenticated user:

- **internal**: Uses a primary role for each individual user and is a 1-to-1 mapping. A role is assigned by matching the authenticated user name to a role name.

  Manage permissions by nesting roles. Use GRANT role_name TO role_name to give all the permissions of the first role to the second role.

- **ldap**: Uses LDAP group membership and is a 1-to-many mapping. The DSE Role Manager looks up the user in LDAP and returns a list of groups. Users are assigned all the roles that match their group name.

  Roles for each individual user are not required and the users DSE role automatically changes as LDAP group membership changes.

  Manage permissions for each group. DSE disables nested roles when LDAP mode is selected; using GRANT role_name TO role_name results in an error.

Configure the assignment method by setting role_management_options mode in the dse.yaml.

Use the CQL CREATE ROLE, ALTER ROLE, and DROP ROLE commands to manage roles.

Managing roles

Creating a role to manage access to database objects is a two step process, first you CREATE ROLE and then you GRANT permissions on a resource.

Use the CQL CREATE ROLE, ALTER ROLE, and DROP ROLE commands to manage roles using the following syntax:

```
(CREATE | ALTER | DROP ) role_name
[WITH (LOGIN = true | SUPERUSER = true | password = 'password')];
```

Role properties:

- **SUPERUSER**: (Default: false) Execute ALL CQL commands, equivalent to a root account.

  DataStax recommends giving superuser status only to a limited number of users and managing superuser authentication with the internal scheme. 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).

- **LOGIN**: (Default: false) Required to execute CQL requests.
Managing roles

When managing permissions using roles as an assignable permission set, the permission set role does not require login.

- **PASSWORD**: (Default: null) Stores a bcrypt-salted hash password for the role in the `system_auth` keyspace; a role that has a password represents an individual user that can authenticate against the internal scheme (also requires LOGIN = true).

Use the **LIST ROLES** command to show all roles that you have describe permission on or all roles assigned to a role.

```
LIST ROLES [of role_name];
```

Only SUPERUSER roles or roles with CREATE, ALTER, and DROP permission can manage roles.

**Permissions to create, modify, and drop roles**

To manage roles the following permissions are required:

### Table 11: Role management permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Database resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>ALL ROLES</td>
<td>ROLE role_name</td>
</tr>
<tr>
<td>CREATE</td>
<td>ALL ROLES</td>
<td>ROLE role_name</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>ALL ROLES</td>
<td>ROLE role_name</td>
</tr>
<tr>
<td>DROP</td>
<td>ALL ROLES</td>
<td>ROLE role_name</td>
</tr>
</tbody>
</table>

**Permissions to manage access control**

In order to manage permissions, the role with the **GRANT** or **REVOKE** command must have **AUTHORIZE** permission on both the target role and the resource.

To allow a role to manage permissions of a database object:

```
GRANT AUTHORIZE on resource_name to management_role;
```

To allow a role to manage permissions of a role:

```
GRANT AUTHORIZE on target_role to management_role;
```

To allow a role to create proxy roles:

```
GRANT AUTHORIZE on proxy_role to management_role;
GRANT AUTHORIZE on application_role to management_role
```

where AUTHORIZE gives GRANT and REVOKE permission to the role for managing permissions of other roles, `resource_name` is any database resources, see database resources.

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

- Creating a new role requires CREATE permission on all roles. The role used to create a role automatically gets permissions to AUTHORIZE the role.
- **GRANT/REVOKE** requires AUTHORIZE permission on the target role and database resource.
• A role can only modify roles other than itself. Prevents users with ALTER permissions from making their own account a SUPERUSER or creating a role with a higher level of permission.

Creating superuser accounts

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. DataStax recommends giving superuser status only to a limited number of users and managing superuser authentication with the internal scheme. 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).

DSE 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 multi-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 Managing roles.

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:
   
   ```
   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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6. Drop or update the cassandra account:
Managing roles

- 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;
```

Only the roles created in these steps display:

<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>
</tbody>
</table>

(1 rows)

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:

- Creating roles for internal mode
- Creating roles for LDAP mode

Creating roles for internal mode

DataStax Enterprise Role Manager internal mode matches the name of the authenticated user directly to a role. DSE supports internal role management for users that were authenticated from the following schemes:

- **Internal Scheme:**

  # Role name: User name for authentication.

  # Role options: LOGIN = true and PASSWORD = 'some_password'

  # Scheme permissions: GRANT EXECUTE on INTERNAL SCHEME to role_name;

  When both mode and authentication scheme is internal, the role that matches the user name supplied in the connection request is assigned.

- **Kerberos Scheme:**

  # Role name: Exactly matches the Kerberos User Principal Name including REALM. Matching is case sensitive, enclose the role name in quotes. For example, 'jdoe@EXAMPLE.COM'.

  # Role options: LOGIN = true

  # Scheme permissions: GRANT EXECUTE on KERBEROS SCHEME to role_name;
Managing roles

- LDAP Scheme:
  
  # Role name: Exactly match the user name provided in the connection string. The matching is case sensitive.
  
  # Role options: LOGIN = true
  
  # Scheme permissions: GRANT EXECUTE on LDAP SCHEME to role_name;

Prerequisites:
These instructions apply to environments that have the following settings:

- Authenticator, authorizer, and role manager in the cassandra.yaml file:

  authenticator: com.datastax.bdp.cassandra.auth.DseAuthenticator
daformer: com.datastax.bdp.cassandra.auth.DseAuthorizer
dRoleManager: com.datastax.bdp.cassandra.auth.DseRoleManager

- Authentication, role management, and authorization options in the dse.yaml file:

  authentication_options:
   default_scheme: ldap
   other_schemes:
     - kerberos
     - internal
   ...

  authorization_options:
   enabled: true
   ...

  role_management_options:
   mode: internal

- For internal users, create a role with login enabled and an internally stored password:

  CREATE ROLE jane WITH LOGIN = true AND PASSWORD = 'Abc123Jane';

  Use the ALTER ROLE command to change the password.
  Bind the assignment to an authentication scheme:

  GRANT EXECUTE on INTERNAL SCHEME to jane;

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

- For Kerberos User Principal create a role that includes the REALM, user_name@REALM, with login enabled:

  CREATE ROLE 'jane@EXAMPLE.COM' WITH LOGIN = true;

  Wrap the role name in single quotes to preserve the case. You must set login to true for all Kerberos roles, do NOT set a password.
  Bind the assignment to an authentication scheme:

  GRANT EXECUTE on KERBEROS SCHEME to 'jane@EXAMPLE.COM';

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

- For LDAP users create a role that matches their UID or samAccountName (user filter attribute):

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

  You must set login to true for all LDAP roles, do NOT set a password.

  Bind the assignment to an authentication scheme:

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

Creating roles for LDAP mode

When Role Management mode is set to LDAP, the DataStax Enterprise Role Manager looks up the user in LDAP and returns a list of groups. The group lookup is configured in the `ldap_options`.

The following rules apply:

- LDAP group name to role name matching is case sensitive and must exactly match the value in the group name attribute.
- One or more of the user's LDAP group names must match a role to allow CQL requests.
- At least one role matching the user's group must have LOGIN enabled.
- Any user that is a member of an LDAP group that matches a role name is assigned a role.
- Create a role for each LDAP group that has a distinct permission set. DSE disables nested roles when LDAP mode is selected; using `GRANT role_name TO role_name` results in an error.

DSE supports looking up users whose name matches the user name filter for LDAP but authenticated with a different method such as internal or Kerberos. Typically, a Kerberos full principal name matches the user's email address. To use Kerberos authentication with LDAP group lookup specify the user attribute that matches the Kerberos principal name in the LDAP user filter, for example `mail: {0}`.

Prerequisites:

These instructions apply to environments that have the following settings:

- Authenticator, authorizer, and role manager in the `cassandra.yaml` file:

  ```
  authenticator: com.datastax.bdp.cassandra.auth.DseAuthenticator
  authorizer: com.datastax.bdp.cassandra.auth.DseAuthorizer
  role_manager: com.datastax.bdp.cassandra.auth.DseRoleManager
  ```

- Authentication, role management, and authorization options in the `dse.yaml` file:

  ```
  authentication_options:
  default_scheme: ldap
  other_schemes:
  - kerberos
  - internal
  ...
  
  authorization_options:
  enabled: true
  ...
  
  role_management_options:
  mode: ldap
  ```

1. Create a role.
Managing roles

This examples shows a role for users in the analyst LDAP group:

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

Superusers can grant and revoke permissions to any role but their own.

2. Bind the assignment to the LDAP authentication scheme:

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

Enclose the role name in single quotes to use uppercase, the name is case sensitive.

Role manager only assigns the role to users that authenticated with the LDAP scheme. Only required when scheme_permissions is set to true.

3. Allow SELECT in cycling keyspace:

```
GRANT SELECT on KEYSPACE cycling to analyst;
```

Any user in the analyst group can query all tables in the cycling keyspace.

Creating 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 Creating superuser accounts.

2. Bind the assignment to an authentication scheme:

```
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:

```
GRANT AUTHORIZE FOR ALTER, DROP
ON new_role_name
```
Managing roles

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 Authorizing access to database resources.

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:

  ```cql
  GRANT EXECUTE 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 authentication_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. ALL PERMISSIONS cannot be used with ALL AUTHENTICATION SCHEMES.
• Allow role assignment for users authenticating with any scheme:
  
  ```sql
  GRANT EXECUTE ON ALL AUTHENTICATION SCHEMES to role_name;
  ```

• Allow role assignment only for users authenticating with LDAP:
  
  ```sql
  GRANT EXECUTE ON LDAP SCHEME TO role_name;
  ```

• Allow role assignment only for users authenticating with internal:
  
  ```sql
  GRANT EXECUTE ON INTERNAL SCHEME TO role_name;
  ```

• Allow role assignment only for users authenticating with Kerberos:
  
  ```sql
  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:
  
  ```sql
  GRANT EXECUTE ON INTERNAL SCHEME TO role_name;
  GRANT EXECUTE ON LDAP SCHEME TO role_name;
  ```

### Configuring proxy 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.
Managing roles

Figure 2: 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.

Application authentication and role assignment
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:
     ```
     CREATE ROLE role_name WITH LOGIN = true AND PASSWORD = 'password';
     ```
   - LDAP scheme:
     ```
     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:
     ```
     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:
   ```
   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:
   ```
   GRANT permission_name ON resource_name TO proxy_role_name;
   ```

   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 proxy_role_name TO service_role_name;
     ```
   - Allow the service role to proxy each request after authenticating:
     ```
     GRANT PROXY.EXECUTE ON 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:
Managing roles

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.
```
Chapter 8. Authorizing access to database resources

About permissions
After creating a role, use GRANT and REVOKE to control access to CQL database resources.

Available permissions
Permissions are granted on a resource to a role; grant a privilege at any level of the resource hierarchy. The full set of available privileges is:

- ALL PERMISSIONS
- ALTER
- AUTHORIZE
- CREATE
- DESCRIBE
- DROP
- EXECUTE
- MODIFY
- PROXY.EXECUTE
- PROXY.LOGIN
- SEARCH.ALTER
- SEARCH.COMMIT
- SEARCH.CREATE
- SEARCH.DROP
- SEARCH.REBUILD
- SEARCH.RELOAD
- SELECT
- role_name

Database resource structure
DataStax Enterprise database resources are objects to which permissions are applied, and hierarchy applies. Grant permissions on a resource higher in the chain will automatically grant that same permission on all resources lower down.

Not all privileges apply to every type of resource. For instance, EXECUTE is only relevant in the context of functions, MBeans, RPC, and authentication schemes. Attempting to grant privileges on a resource that the permission is not applicable results in an error.

Available objects:
Authorizing access to database resources

Functions

- ALL FUNCTIONS
  # ALL FUNCTIONS IN KEYS加倍keName
  # FUNCTION keyspace_name.function_name

Data

- ALL KEYSPACES
  # KEYS加倍ke keyspace_name
  # TABLE table_name
    • Rows ('filtering_data' ROWS IN table_name)
    • Indexes (other than search indexes) belong to a table but permission cannot be
      directly assigned. ALTER permission on a table allows users to CREATE and DROP
      indexes.

Search index

- ALL SEARCH INDICES
  # SEARCH INDICES [keyspace_name.]table_name

JMX

- ALL MBEAMS
  # MBEAN mbean_name
  # MBEAMS pattern

Roles

- ALL ROLES
  # ROLE role_name

Roles are both an assignable permission set and a database resource. GRANT role_name TO
role_name gives all the permissions of the first role in the statement to the second role. Use roles
to create your own hierarchical permissions structures.

Remote procedure calls (RPC)

- ALL OBJECTS
  # OBJECT object_name
  # METHOD

Authentication schemes

- ALL SCHEMES
  # (LDAP | KERBEROS | INTERNAL) SCHEME
Managing keyspace and table permissions

DataStax Enterprise supports role-based access control to data on transactional nodes. The **grant** and **revoke** CQL commands provide and revoke access to objects and methods. Permission is hierarchical, granting permission to a parent object automatically allows full access to all ancestors; data objects have the following structure:

**ALL KEYSPACES**

- **KEYSPACE** `keyspace_name`
  
  - **TABLE** `table_name`
    
    - `'filtering_string' ROWS`

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

```cql
GRANT permission_name ON resource_name TO role_name;
```

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

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

Where permissions that apply to each data resources type is described below:

<table>
<thead>
<tr>
<th>permission_name</th>
<th>resource_name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>ALL KEYSPACES</td>
<td>ALTER KEYSPACE, ALTER TABLE, ALTER TYPE, RESTRICT ROW in any keyspace.</td>
</tr>
<tr>
<td>ALTER</td>
<td>KEYSpace <code>keyspace_name</code></td>
<td>ALTER KEYSPACE, ALTER TABLE, ALTER TYPE, and RESTRICT ROW in specified keyspace.</td>
</tr>
<tr>
<td>ALTER</td>
<td>TABLE <code>table_name</code></td>
<td>ALTER TABLE and RESTRICT ROW of specified table.</td>
</tr>
<tr>
<td>ALTER</td>
<td><code>'filtering_data' ROWS in </code>table_name</td>
<td>ALTER</td>
</tr>
<tr>
<td>CREATE</td>
<td>ALL KEYSPACES</td>
<td>CREATE KEYSPACE, CREATE TABLE, CREATE FUNCTIONS, and CREATE TYPE in any keyspace.</td>
</tr>
<tr>
<td>CREATE</td>
<td>KEYSpace <code>keyspace_name</code></td>
<td>CREATE TABLE and CREATE TYPE in specified keyspace.</td>
</tr>
<tr>
<td>DROP</td>
<td>ALL KEYSPACES</td>
<td>DROP KEYSPACE, DROP TABLE, and DROP TYPE in any keyspace</td>
</tr>
<tr>
<td>DROP</td>
<td>KEYSpace <code>keyspace_name</code></td>
<td>DROP TABLE and DROP TYPE in specified keyspace</td>
</tr>
<tr>
<td>MODIFY</td>
<td>ALL KEYSPACES</td>
<td>INSERT, UPDATE, DELETE and TRUNCATE rows in any table.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>KEYSpace <code>keyspace_name</code></td>
<td>INSERT, UPDATE, DELETE and TRUNCATE rows in any table in the specified keyspace.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>TABLE <code>table_name</code></td>
<td>INSERT, UPDATE, DELETE and TRUNCATE any rows in the specified table. See note for tables with materialized views (MV).</td>
</tr>
<tr>
<td>MODIFY</td>
<td><code>'filtering_data' ROWS in </code>table_name</td>
<td>INSERT, UPDATE, DELETE and TRUNCATE rows that match the `'filtering_data' on rows in a table that match the filtering criteria.</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.
Authorize access to database resources

For more details, see Access control matrix.

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

  ```
  CREATE ROLE keyspace_admin;
  GRANT ALL PERMISSIONS ON ALL KEYSPACES TO keyspace_admin;
  GRANT keyspace_admin to martin;
  ```

- Create an administrator role for a single keyspace:

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

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

  ```
  GRANT MODIFY ON KEYSPACE cycling TO team_manager;
  GRANT team_manager to sandy;
  ```

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

  ```
  CREATE ROLE cyclist_analyst;
  GRANT SELECT ON KEYSPACE cycling TO cyclist_analyst;
  GRANT EXECUTE ON ALL FUNCTIONS IN KEYSPACE cycling TO cyclist_analyst;
  GRANT cyclist_analyst TO wilson;
  ```

### Setting row-level permissions

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 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
  ```
Authorizing access to database resources

```
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;
```

1. Select the cyclist_name column as the filtering column:

```
RESTRICT ROWS ON cycling.cyclist_expenses 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 dclocal_read_repair_chance = 0.1  
AND default_time_to_live = 0  
AND gc_grace_seconds = 864000  
AND max_index_interval = 2048  
AND memtable_flush_period_in_ms = 0  
AND min_index_interval = 128  
AND read_repair_chance = 0.0  
AND speculative_retry = '99PERCENTILE';
```
RESTRICT ROWS ON cycling.cyclist_expenses USING cyclist_name;

2. Get a list of the primary keys to use in the filter:

```sql
SELECT DISTINCT cyclist_name FROM cycling.cyclist_expenses ;
```

<table>
<thead>
<tr>
<th>cyclist_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marianne VOS</td>
</tr>
<tr>
<td>Alex FRAME</td>
</tr>
<tr>
<td>Steven KRUIKSWIJK</td>
</tr>
<tr>
<td>Vera ADRIAN</td>
</tr>
<tr>
<td>Michael MATTHEWS</td>
</tr>
<tr>
<td>Anna VAN DER BREGGEN</td>
</tr>
<tr>
<td>Paolo TIRALONGO</td>
</tr>
</tbody>
</table>

(7 rows)

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

```
GRANT SELECT ON 'Paolo TIRALONGO' ROWS IN cycling.cyclist_expenses TO dantest1;
```

   b. Allow role jane to access only Vera ADRIAN:

```
GRANT SELECT ON 'Vera ADRIAN' ROWS IN cycling.cyclist_expenses TO jane;
```

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>dantest1</td>
<td>dantest1</td>
<td>'Paolo TIRALONGO' rows IN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cycling.cyclist_expenses&gt;</td>
</tr>
</tbody>
</table>
|          |          | | SELECT

   b. Check jane permissions:

```
LIST ALL PERMISSIONS OF jane;
```

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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cycling.cyclist_expenses&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Authorizing access to database resources

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

1. Log in as dantest1:
   ```
cqlsh -p password -u dantest1
   ```

2. Run a query:
   ```
dantest1@cqlsh> SELECT * FROM cycling.cyclist_expenses;
   ```

Only the records that exactly match the filter are displayed:

<table>
<thead>
<tr>
<th>cyclist_name</th>
<th>expense_id</th>
<th>balance</th>
<th>amount</th>
<th>description</th>
<th>paid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paolo TIRALONGO</td>
<td>11</td>
<td>null</td>
<td>10</td>
<td>dinner</td>
<td>null</td>
</tr>
<tr>
<td>Paolo TIRALONGO</td>
<td>12</td>
<td>null</td>
<td>10</td>
<td>dinner</td>
<td>null</td>
</tr>
<tr>
<td>Paolo TIRALONGO</td>
<td>24</td>
<td>null</td>
<td>10</td>
<td>lunch</td>
<td>null</td>
</tr>
<tr>
<td>Paolo TIRALONGO</td>
<td>25</td>
<td>null</td>
<td>11</td>
<td>dinner</td>
<td>null</td>
</tr>
<tr>
<td>Paolo TIRALONGO</td>
<td>26</td>
<td>null</td>
<td>12</td>
<td>lunch</td>
<td>null</td>
</tr>
<tr>
<td>Paolo TIRALONGO</td>
<td>27</td>
<td>null</td>
<td>13</td>
<td>lunch</td>
<td>null</td>
</tr>
<tr>
<td>Paolo TIRALONGO</td>
<td>28</td>
<td>null</td>
<td>14</td>
<td>lunch</td>
<td>null</td>
</tr>
<tr>
<td>Paolo TIRALONGO</td>
<td>29</td>
<td>null</td>
<td>15</td>
<td>dinner</td>
<td>null</td>
</tr>
<tr>
<td>Paolo TIRALONGO</td>
<td>30</td>
<td>null</td>
<td>16</td>
<td>lunch</td>
<td>null</td>
</tr>
<tr>
<td>Paolo TIRALONGO</td>
<td>31</td>
<td>null</td>
<td>17</td>
<td>dinner</td>
<td>null</td>
</tr>
<tr>
<td>Paolo TIRALONGO</td>
<td>32</td>
<td>null</td>
<td>18</td>
<td>breakfast</td>
<td>null</td>
</tr>
</tbody>
</table>

(11 rows)

Managing access to DSE 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_pvt**: Keyspace with the same name as the graph followed by underscore pvt. Contains the partitioned vertex tables.
- **graph_name_system**: Keyspace with the graph name followed by underscore system that stores shared information.

```
GRANT <permission type> ON KEYSPACE <graphname> TO <rolename>;
GRANT <permission type> ON KEYSPACE <graphname_pvt> 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:
### Authorizing access to database resources

Table 12: 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>
<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 13: 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 14: 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 for CQL execution

DataStax Enterprise supports authentication and role-based access control for Remote Procedure Calls to the DSE database.

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;
```
**Authorizing access to database resources**

```sql
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:

```sql
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, see the Access Control Matrix for more details
- role is the role to grant or revoke authorization to.

**Granting RPC permissions to DseClientTool**

You must configure RPC permissions for external clients to run the dse client-tool command and to launch Spark:

```sql
GRANT EXECUTE ON REMOTE OBJECT DseClientTool TO USER;
```

RPC permission for the DseClientTool object is required to run Spark because the DseClientTool object is called implicitly by the Spark launcher.

If access is attempted without permissions, you will see an error message similar to:

```
com.datastax.driver.core.exceptions.UnauthorizedException: User X has no EXECUTE permission on <rpc method DseClientTool.getSparkMasterAddress> or any of its parents
```

To resolve the problem, set RPC permissions for the DseClientTool object. For example:

```sql
GRANT EXECUTE ON REMOTE OBJECT DseClientTool TO ADMINROLE;
```

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

**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,

```
$ nodetool -u martin -pw password status
```

the request is denied.

```
nodetool: Failed to connect to '127.0.0.1:7199' - SecurityException: 'Access Denied'.
```

- Create an MBEAN administrator role:
  1. Create an MBean administrator role:
     ```
     CREATE ROLE mbean_admin;
     ```
  2. Give the role access to all MBeans:
     ```
     GRANT ALL PERMISSIONS on ALL MBEANS to mbean_admin;
     ```
  3. Assign the role to a user or group role:
     ```
     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.
  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-48e1ccbcce3e2 rack1
```
Authorizing access to database resources

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 KEYSNAME keyspace_name | SEARCH INDEX
      [keyspace_name.]table_name)
  TO role_name;
  ```

- Remove access to search index:

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

Table 15: 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:

- Manage permissions for all search indexes:

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

- Limit permissions to manage permissions to individual tables:

  ```
  GRANT AUTHORIZE FOR permission_name
  ON SEARCH INDEX [keyspace_name.]table_name
  ```
Authorizing access to database resources

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:

  GRANT ALL PERMISSIONS ON ALL SEARCH INDICES TO role_name;

- Limit access to a specific table:

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

- Remove previously granted access to all search indexes:

  REVOKE ALL PERMISSIONS ON ALL SEARCH INDICES FROM role_name;

- Remove previously granted access to a specific table:

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

- List all roles that have access to search indexes:

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

Managing 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 to 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 ON "HiveMetaStore".sparkmetastore TO role_name;
  GRANT MODIFY ON "HiveMetaStore".sparkmetastore 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:
# 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;
Enabling data auditing in DataStax Enterprise

The audit logger logs information only on nodes set up for logging. For example, node 0 has audit turned on, node 1 does not. This configuration means that issuing updates and other commands on node 1 does not affect the node 0 audit log. For maximum information from data auditing, turn on data auditing on every node.

Audit logs can be written to either file system log files or to a database table:

<table>
<thead>
<tr>
<th>System log files</th>
<th>Database tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>When you turn on audit logging, the default is to write to logback file system log files. Logging functionality uses Simple Logging Facade for Java (SLF4J) with a logback backend.</td>
<td>As audit logs increase in size, logging audit data to a table is more useful.</td>
</tr>
<tr>
<td>File-based logs are stored per node and are secured with standard Linux file system permissions.</td>
<td>Audit events stored in database tables can be secured like any other table using RBAC, see Managing roles. For example, store database table-based logs in encrypted SSTables. Control access to the tables with object permissions.</td>
</tr>
<tr>
<td>The log files can be read from a terminal for troubleshooting queries or managing security.</td>
<td>Larger clusters use tables because logback audit logs become cumbersome. The data can be queried like any other table, making analysis easier and custom audit reports possible.</td>
</tr>
</tbody>
</table>

Audit logging of queries and prepared statements submitted to the DataStax drivers, which use the CQL binary protocol, is supported.

When using audit logging with Kerberos authentication, the login events take place on Kerberos and are not logged in DataStax Enterprise. Authentication history is available only on Kerberos. When DataStax Enterprise is unable to authenticate a client with Kerberos, a LOGIN_ERROR event is logged.

1. Open the dse.yaml file in a text editor.
2. In the audit_logging_options section, set enabled to true.

```yaml
# Audit logging options
audit_logging_options:
  enabled: true
```

3. Set the logger option to either:
   - CassandraAuditWriter
     Logs to a table.
   - SLF4JAuditWriter
     Logs to the SLF4J logger.

4. To include or exclude event categories from being logged, add the event types included_categories or excluded_categories and specify the categories in a comma separated list. You can set either event type, but not both.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH</td>
<td>Logs login events.</td>
</tr>
<tr>
<td>DML</td>
<td>Logs insert, update, delete and other data manipulation language (DML) events.</td>
</tr>
<tr>
<td>DDL</td>
<td>Logs object and role create, alter, drop, and other data definition language (DDL) events.</td>
</tr>
<tr>
<td>DCL</td>
<td>Logs grant, revoke, create role, drop role, and list roles events.</td>
</tr>
</tbody>
</table>
Auditing activity

<table>
<thead>
<tr>
<th>Setting</th>
<th>Logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY</td>
<td>Logs all queries.</td>
</tr>
</tbody>
</table>

The authenticator determines whether DataStax Enterprise audit logs include login events:

- When using audit logging with DataStax Unified Authentication, the login events with the native protocol authentication are logged in DataStax Enterprise.
- When using audit logging with internal authentication, the login events take place as part of connection negotiation and are not logged in DataStax Enterprise.

5. To include or exclude database keyspaces from being logged, add a comma separated list of keyspaces to the included_keyspaces or excluded_keyspaces options. You can set either one, but not both.

   When whitelisting keyspaces, included_keyspaces, AUTH messages are not captured.

6. If you are logging to a database table, set the retention time for logged events by setting the retention_time option to the number of hours the events should be retained. The default value is 0, which retains all event data indefinitely.

7. Configure the audit logging writer.
   - SLF4JAuditWriter
   - Database table

   The following example sets the audit logger to log to a database table.

```
# Audit logging options
audit_logging_options:
  enabled: true
  logger: CassandraAuditWriter
```

Configuring audit logging

If you've enabled audit logging and set the logger to output to the SLF4JAuditWriter as described in Enabling data auditing in DataStax Enterprise, you can configure the logger by setting options in logback.xml.

DataStax Enterprise places the audit log in the directory defined in the logback.xml configuration file. After the log file reaches the configured size threshold, it rolls over, and the log file name is changed. The file names include a numerical suffix that is determined by the maxBackupIndex property.

Sensitive data in log files

Because auditing is configured through a text file in the file system, the file is vulnerable to OS-level security breaches. You can address this issue by changing DataStax Enterprise’s unmask setting, which is 0700, by setting the permissions to 0600 on the audit files. Be aware that if other tools look at the data, changing this setting can cause read problems. Alternately, you can store the audit file on an OS-level encrypted file system such as Vormetric.

Redact sensitive data before you share log files for troubleshooting purposes.

For example, when:

- A password is inserted in a table in a column named password.
- And the audit logging options in dse.yaml are set to included_categories: DML, ... to include DML (insert, update, delete and other data manipulation language events) in the audit log.
Auditing activity

- You can redact the values for that column so that passwords do not show in the log. Use the following to replace that string in the logback.xml file:

```xml
%-5level [%thread] %date{ISO8601} %X{service} %F:%L - %replace(%msg){"password='.*'", "password='xxxxx'"}%n
```

The replace layout option uses regular expressions to modify the data before it ends up in the log file. For more information on using the replace filter, see the logback documentation.

Configuring data auditing

You can configure which categories of audit events (administration, authentication, DML, DDL, DCL, and query operations) to log, and whether to omit operations against specific keyspaces from audit logging.

The audit logger logs at INFO level, so the DataAudit logger must be configured at INFO (or lower) level in logback.xml. Setting the logger to a higher level, such as WARN, prevents any log events from being recorded, but it does not completely disable the data auditing. Some overhead occurs that is caused by regular processing.

1. Open the logback.xml file in a text editor.
2. To configure data auditing, accept the default settings or change the properties:

```xml
<appender name="SLF4JAuditWriterAppender" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${cassandra.logdir}/audit/audit.log</file> <!-- logfile location -->
  <encoder>
    <pattern>%-5level [%thread] %date{ISO8601} %F:%L - %msg%n</pattern> <!-- the layout pattern used to format log entries -->
    <immediateFlush>true</immediateFlush>
  </encoder>
  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
    <fileNamePattern>${cassandra.logdir}/audit/audit.log.%i.zip</fileNamePattern>
    <minIndex>1</minIndex>
    <maxIndex>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 name="SLF4JAuditWriter" level="INFO" additivity="false">
  <appender-ref ref="SLF4JAuditWriterAppender"/>
</logger>
```

3. Generate Kerberos debug output:

```xml
...<logger name="com.datastax.bdp.transport.server" level="TRACE"/>
...<logger name="com.datastax.bdp.cassandra.auth" level="TRACE"/>
...```

4. Generate LDAP debug output:

```xml
...<logger name="com.datastax.bdp.cassandra.auth" level="TRACE"/>
...```

5. Restart the node to see changes in the log.
Formats of DataStax Enterprise logs

The log format is a simple set of pipe-delimited name/value pairs. The pairs themselves are separated by the pipe symbol (|), and the name and value portions of each pair are separated by a colon. A name/value pair, or field, is 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 Enabling data auditing in DataStax Enterprise.

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 KEYSAPCE, 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:1351003707937|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=[]
```
Auditing activity

CQL BATCH

BEGIN BATCH
  INSERT INTO t0(id, field0) VALUES (0, 'foo')
  INSERT INTO t0(id, field0) VALUES (1, 'bar')
DELETE FROM t1 WHERE id = 2
APPLY BATCH;

CQL DROP KEYSSPACE

DROP KEYSSPACE dsp904;

CQL prepared statement

INSERT INTO cf (id, name) VALUES (?, ?) [id=1,name=vic]

Thrift batch_mutate

INSERT INTO t0 VALUES (0, 'foo')
DELETE FROM t1 WHERE id = 2

DataStax Java Driver queries

ADD KEYSPACE Keyspace1
Batch updates

Batch updates, whether received via a Thrift batch_mutate call, or in CQL BEGIN BATCH....APPLY BATCH block, are logged in the following way: A UUID is generated for the batch, then each individual operation is reported separately, with an extra field containing the batch id.

Configuring audit logging to a database table

If you’ve enabled audit logging and set the logger to output to a database table as described in Enabling data auditing in DataStax Enterprise, you can configure the logger by setting options in dse.yaml. Audit events are written to the dse_audit.audit_log table. The default compaction strategy for the dse_audit.audit_log table is TimeWindowCompactionStrategy (TWCS). DataStax recommends changing the compaction strategy for tables that were created before DataStax Enterprise 4.8.0:

```
ALTER TABLE dse_audit.audit_log WITH COMPACTION={'class': 'TimeWindowCompactionStrategy'};
```

The logger can be run synchronously or asynchronously. By default, the logger runs synchronously. Use the GRANT or REVOKE CQL commands to manage the permissions for accessing the dse_audit.audit_log table.

When run synchronously, an event does not complete until the event has been written to the table. If there is a failure after the event has been written to the table but before the event completed, the log may contain events that were never completed. For example, a query may be logged in the table but it did not successfully complete.

When run asynchronously, audit events are queued for writing to the table, but may not be logged before the event is completed. For example, when logging a query, the query may execute before the audit event is written to the table. A pool of writer threads handles logging audit events from the queue, writing to the table in batch queries. The advantage of writing audit events asynchronously is better performance under load, however if there is a failure before an audit event is written to the table, the audit event may not be logged even though the event has completed.

1. Open dse.yaml in a text editor.

2. Set these audit_logging_options:

   **cassandra_audit_writer_options**

   Configuration options for CassandraAuditWriter.

   ```yaml
   cassandra_audit_writer_options:
   mode: sync
   batch_size: 50
   flush_time: 500
   ```
3. Save the file and restart DataStax Enterprise.

**CassandraAuditWriter table columns**

When logging audit data to a database table using the `CassandraAuditWriter` logger, the audit data is stored in the `dse_audit.audit_log` table. This table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date of the event.</td>
</tr>
<tr>
<td>node</td>
<td>DSE node address.</td>
</tr>
<tr>
<td>day_partition</td>
<td></td>
</tr>
<tr>
<td>event_time</td>
<td>The system timestamp of the event.</td>
</tr>
<tr>
<td>batch_id</td>
<td>The UUID of the batch query the event was grouped with when written to Cassandra.</td>
</tr>
<tr>
<td>category</td>
<td>The event category.</td>
</tr>
<tr>
<td>keyspace_name</td>
<td>The keyspace of the event.</td>
</tr>
<tr>
<td>operation</td>
<td>The query or event description.</td>
</tr>
<tr>
<td>source</td>
<td>The IP address of the client.</td>
</tr>
<tr>
<td>table_name</td>
<td>The table affected by the event.</td>
</tr>
<tr>
<td>type</td>
<td>The type of the event.</td>
</tr>
<tr>
<td>username</td>
<td>The authenticated user triggering the event. If authentication isn't enabled, the user is anonymous.</td>
</tr>
</tbody>
</table>

Using cqlsh, a SELECT statement can access the data if a user/role has permission:

```
SELECT * FROM dse_audit.audit_log;
```

with sample output from a DSE Graph query:

```
2016-06-22 00:00:00+0000 | 127.0.0.1 | 10800 | 57ff2dc0-3827-11e6-9ea9-2f93ec587f0 |
null | QUERY | null | null |
2016-06-22 00:00:00+0000 | 127.0.0.1 | 10800 | 57ffa2f0-3827-11e6-9ea9-2f93ec587f0 |
null | DML | null | test |
2016-06-22 00:00:00+0000 | 127.0.0.1 | 10800 | 57ffca00-3827-11e6-9ea9-2f93ec587f0 |
null | QUERY | ONE | test |
2016-06-22 00:00:00+0000 | 127.0.0.1 | 10800 | 58001820-3827-11e6-9ea9-2f93ec587f0 |
null | DML | null | test |
2016-06-22 00:00:00+0000 | 127.0.0.1 | 10800 | 58001821-3827-11e6-9ea9-2f93ec587f0 |
null | QUERY | ONE | test |
2016-06-22 00:00:00+0000 | 127.0.0.1 | 10800 | 58008d50-3827-11e6-9ea9-2f93ec587f0 |
null | DML | null | test |
2016-06-22 00:00:00+0000 | 127.0.0.1 | 10800 | 5800db70-3827-11e6-9ea9-2f93ec587f0 |
null | DML | null | test |
2016-06-22 00:00:00+0000 | 127.0.0.1 | 10800 | 5800db71-3827-11e6-9ea9-2f93ec587f0 |
null | QUERY | ONE | test |
2016-06-22 00:00:00+0000 | 127.0.0.1 | 10800 | 58012990-3827-11e6-9ea9-2f93ec587f0 |
null | DML | null | test |
```
<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Remote Address</th>
<th>Request Code</th>
<th>Query Code</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-06-22 00:00:00+0000</td>
<td>127.0.0.1</td>
<td>DML</td>
<td>test</td>
<td>58019ec0-3827-11e6-9ea9-2f93eeec587f0</td>
</tr>
<tr>
<td>2016-06-22 00:00:00+0000</td>
<td>127.0.0.1</td>
<td>QUERY</td>
<td>ONE</td>
<td>58019ec1-3827-11e6-9ea9-2f93eeec587f0</td>
</tr>
<tr>
<td>2016-06-22 00:00:00+0000</td>
<td>127.0.0.1</td>
<td>QUERY</td>
<td>ONE</td>
<td>5bb86530-3827-11e6-9ea9-2f93eeec587f0</td>
</tr>
<tr>
<td>2016-06-22 00:00:00+0000</td>
<td>127.0.0.1</td>
<td>QUERY</td>
<td>QUORUM</td>
<td>test_system</td>
</tr>
<tr>
<td>2016-06-22 00:00:00+0000</td>
<td>127.0.0.1</td>
<td>QUERY</td>
<td>ONE</td>
<td>dse_audit</td>
</tr>
<tr>
<td>2016-06-22 00:00:00+0000</td>
<td>127.0.0.1</td>
<td>QUERY</td>
<td>QUORUM</td>
<td>test_system</td>
</tr>
<tr>
<td>2016-06-22 00:00:00+0000</td>
<td>127.0.0.1</td>
<td>QUERY</td>
<td>QUORUM</td>
<td>dse_system</td>
</tr>
<tr>
<td>2016-06-22 00:00:00+0000</td>
<td>127.0.0.1</td>
<td>QUERY</td>
<td>ONE</td>
<td>dse_system</td>
</tr>
<tr>
<td>2016-06-22 00:00:00+0000</td>
<td>127.0.0.1</td>
<td>QUERY</td>
<td>QUORUM</td>
<td>test_system</td>
</tr>
</tbody>
</table>

RequestMessage{, requestId=941d2e1a-7cc9-4c80-8c28-dccb799840b7, op='eval', processor='session', args={gremlin=g.V().count(), aliases={g=testQSagain.g}, session=d179c734-813f-4a3e-89d6-bd756b4fcf57, bindings={}, manageTransaction=true, batchSize=64}) | /127.0.0.1:60647 | null | unknown |

SELECT "community_id", "member_id" FROM "test"."meal_p" WHERE "~~vertex_exists" = ? LIMIT ? ALLOW FILTERING; | /0.0.0.0 | meal_p |

SELECT "community_id", "member_id" FROM "test"."ingredient_p" WHERE "~~vertex_exists" = ? LIMIT ? ALLOW FILTERING; | /0.0.0.0 | ingredient_p |

SELECT "community_id", "member_id" FROM "test"."author_p" WHERE "~~vertex_exists" = ? LIMIT ? ALLOW FILTERING; | /0.0.0.0 | author_p |

SELECT "community_id", "member_id" FROM "test"."book_p" WHERE "~~vertex_exists" = ? LIMIT ? ALLOW FILTERING; | /0.0.0.0 | book_p |

SELECT "community_id", "member_id" FROM "test"."recipe_p" WHERE "~~vertex_exists" = ? LIMIT ? ALLOW FILTERING; | /0.0.0.0 | recipe_p |
Configuring auditing for a DSE Search cluster

If auditing is enabled, DSE Search nodes do not require additional configuration. If the filter-mapping element in the Apache Solr™ web.xml file is commented out, the auditor cannot log anything from Solr.

1. If necessary, uncomment the filter-mapping element in the Solr web.xml file.

```xml
<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
```
Chapter 10. 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 upgradesstables` with the `-a` option to rewrite the tables to disk with encryption.
  Primary keys are stored in plain text. Do NOT put sensitive information in partition key or clustering columns.

**Data that is not encrypted**

DSE does not encrypt the following:

- Table partition key columns
- Database files other than the commit log and SSTable data files
- DSEFS data files
- Spark spill files

**Requirements**

To use the DataStax Enterprise (DSE) Transparent Data Encryption (TDE) feature, enable the Java Cryptography Extension (JCE).

When using TDE on a 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.

<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>
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 Configuration encryption.

**TDE options**

To get the full capabilities of TDE and to ensure full algorithm support, enable JCE Unlimited.

**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 (DSE) 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 of the `dse.yaml`.
  
  - Ensure that the DSE account owns the `system_key_directory` and has read/write permission.
  
**Setting up local encryption keys**

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 Unlimited.

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.

```
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`.

```
config_encryption_key_name: system_key
```

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`. DSE supports the following JCE cipher algorithms and corresponding length:

`cipher_algorithm[/mode/padding]`

DSE supports the following JCE cipher algorithms and corresponding length:

- 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: `AES/CBC/PKCS5Padding` (with length 128).

Encryption key files can have any valid Unix name.

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 and update the `system_key_directory` and `config_encryption_key_name` in dse.yaml.
Transparent data encryption

`dsetool` reads current values from `dse.yaml`. A restart is not required to continue configuring encryption.

5. 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.

```bash
$ chown cassandra /etc/dse/conf/system_key
```

Encrypting configuration file properties

Configure DSE to use a local encryption key to encrypt 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`
  - `client_encryption_options.truststore_password`

**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:

   ```bash
   $ 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`:

   ```yaml
   truststore_password: +Vj5oHCR/jqfA+OJE2m8zA==
   ```

Once 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.

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.batchlog and system.paxos tables, hint files, and commit logs using a local encryption key.

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.

```
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.
Transparent data encryption

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.

  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.

  Tables with the same encryption settings use the same encryption key.

**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,
   ```
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 upgradesstables -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,
    '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 and corresponding length:

- 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: AES/CBC/PKCS5Padding (with length 128).

**length**

Specifies the length of the encryption key.

Default: 128. (Optional)
Transparent data encryption

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 the 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.
      
      ```
      ALTER TABLE keyspace_name.table_name
      WITH compression = ( 'sstable_compression' : 'EncryptingSnappyCompressor',
                      'cipher_algorithm' : 'AES/ECB/PKCS5Padding',
                      'secret_key_strength' : 128,
                      'chunk_length_kb' : 128,
      ```

DataStax 5.1 Security Guide Earlier DSE version Latest 5.1 patch: 5.1.17
Transparent data encryption

```java
'system_key_file': 'new_system_key'
```

c. Ensure that the schema change has replicated to all nodes in the cluster:

```bash
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:

- **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
  ```

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 fail. For example, creating a test table with LZ compression and encryption:

```sql
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:

```java
ConfigurationException: EncryptingLZ4Compressor.create() threw an error:
java.lang.RuntimeException Failed to initialize Encryptor
```

Solution
Transparent data encryption

1. Verify that the account running the DSE database is the owner of the encryption key file on the local system:

   ```
   $ ls -l /etc/dse/conf
   ``

   In this case, the account dse only has read/write access to the system_key file.

   ```
   total 8
   -rw------- 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.

   ```
   $ 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 -srckeystore 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:

```
$ 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`:

```
k mip_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.
Transparent data encryption

```yaml
# 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:

  ```yaml
  # 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:

   ```sh
   $ 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:
     
     | ID    | Name                  | Cipher       |
     |-------|-----------------------|--------------|
     | 02-449| 82413ef3-4fa6-4d4d-9dc8-71370d731fe4_0 | AES/CBC/PKCS5 |
     |       | deactivated           |              |
     |       | Mon Apr 25 20:25:47 UTC 2016 | n/a          |
     | 02-540| 0eb2277e-0acc-4adb-9241-1dd84dde691c_0 | AES          |
     |       | active                |              |
     |       | Tue May 31 12:57:59 UTC 2016 | n/a          |
     |       | n/a                   |              |
     
     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:
Transparent data encryption

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==
```

Once 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.batches` 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.

```
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:

**Table 17: Supported cipher algorithms names**

<table>
<thead>
<tr>
<th>cipher_algorithm</th>
<th>secret_key_strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>128, 192, or 256</td>
</tr>
<tr>
<td>DES</td>
<td>56</td>
</tr>
<tr>
<td>DESede</td>
<td>112 or 168</td>
</tr>
<tr>
<td>Blowfish</td>
<td>32-448</td>
</tr>
<tr>
<td>RC2</td>
<td>40-128</td>
</tr>
</tbody>
</table>

# secret_key_strength: Specify the key length.
Transparent data encryption

# chunk_length_kb: Configures chunk 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.

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:

```sql
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:

```sql
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.

# ['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:

```
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]...]
```

**Expiring 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, the `host_name` has two keys:

<table>
<thead>
<tr>
<th>Keys on host_name:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>02-449</td>
</tr>
</tbody>
</table>

DataStax 5.1 Security Guide Earlier DSE version Latest 5.1 patch: 5.1.17
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:**
   - 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.

   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 proxy KMIP commands to the corresponding host; the commands effect encryption keys as follows:

- **dsetool managekmip 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.

- **dsetool managekmip 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.

- **dsetool managekmip 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.**

DataStax 5.1 Security Guide Earlier DSE version Latest 5.1 patch: 5.1.17
2. Revoke the compromised key using the `dsetool`:
   a. Get the ID of the KMIP encryption key you want to revoke from the KMIP server:

   ```bash
   $ 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>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>Mon Apr 25 20:25:47 UTC 2016</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>02-540</td>
<td>0eb2277e-0acc-4adb-9241-1dd84de691c_0</td>
<td>AES</td>
<td>Active</td>
</tr>
<tr>
<td></td>
<td>Tue May 31 12:57:59 UTC 2016</td>
<td>n/a</td>
<td>Thu Jul 27</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`.

   b. Revoke the key you want to replace using the ID:

   ```bash
   $ 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.

   ```bash
   $ 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>Mon Apr 25 20:25:47 UTC 2016</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>02-540</td>
<td>0eb2277e-0acc-4adb-9241-1dd84de691c_0</td>
<td>AES</td>
<td>Active</td>
</tr>
<tr>
<td></td>
<td>Tue May 31 12:57:59 UTC 2016</td>
<td>n/a</td>
<td>Thu Jul 27</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.

   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:

- Target only specific tables:

```
$ nodetool upgradesstables --include-all-sstables keyspace_name table_name
    [table_name ...]
```

- Target specific keyspace:

```
$ nodetool upgradesstables --include-all-sstables keyspace_name
```

- All keyspaces and tables:

```
$ nodetool upgradesstables --include-all-sstables
```

6. (Optional) Remove the encryption key so that it is no longer available for decryption:

```
$ 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 Setting up SSL certificates.

### 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.
Transparent data encryption

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 on a 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` to change the class for directoryFactory to `solr.EncryptedFSDirectoryFactory`.
   
   For example, you can use the `dsetool create_core` command with automatic resource generation. Specify the class for directoryFactory to `solr.EncryptedFSDirectoryFactory` with the handy `coreOptionsInline` argument:
   
   ```
   $ dsetool create_core keyspace_name.table_name generateResources=true coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
   ```

   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 on a 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, edit the `solrconfig.xml` file to change the class for directoryFactory to `solr.EncryptedFSDirectoryFactory`:
   
   ```
   <directoryFactory name="DirectoryFactory" class="solr.EncryptedFSDirectoryFactory"/>
   ```

2. Upload the changed `solrconfig.xml` file.
   
   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]
```
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:

   ```yaml
   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:

   ```xml
   <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.

It is important to note that if any old key is lost in the process of migration, the data will be lost forever. Exercise caution when using this procedure!

1. Back up the entire keyspace that has a `dse_system.encrypted_keys` table.

2. Back up all system keys.

3. Upgrade the cluster to DataStax Enterprise 5.1, 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 KEYSPACES` command.
   
   If you need to restore the dse_system.encrypted_keys table, load the table. Do not truncate or delete anything.

6. If the dse_system.encrypted_keys table was created, go to the next step; otherwise, create the table manually:

   ```
   CREATE KEYSPACE dse_system WITH replication = {'class': 'EverywhereStrategy'};
   USE dse_system;
   CREATE TABLE encrypted_keys (
       key_file text,
       cipher text,
       strength int,
       key_id timeuuid,
       key text,
       PRIMARY KEY (key_file, cipher, strength, key_id)
   );
   ```

   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.

   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.

   ```
   $ 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.

   **--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.
Transparent data encryption

--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:

   ```
   SELECT * from dse_system.encrypted_keys;
   
   key_file | cipher | strength | key_id                               | key
   ├─────+--------+----------+--------------------------------------+---------
   our_system_key | AES    | 128      | d9b3dd70-c764-11e7-abc4-793ec23f8a8c | kmbYE1KLkmW3Hzg7dIPt1rw3j6hR+gM7bxj/pRd7gU=
   ```

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
   ```
4. On the target cluster, verify that your added entry is in the `dse_system.encrypted_keys` table. Example:

```sql
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>kmbYE1LKmW3Hzg7dIPt1rWk3j6hR+gM7bxd/pRd7gU=</td>
</tr>
<tr>
<td>system_key_dev</td>
<td>AES</td>
<td>256</td>
<td>81847700-c99d-11e7-b9d9-23f36e5077c2</td>
<td>6YXE07AcEv61jyT6x7zdj6Ade0N6OHzxALNnh1s7nVFPPQArh64LousF#bXmy</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 11. Configuring SSL

About SSL
DataStax Enterprise supports SSL encryption for data in-flight for the following components:

• DSE transactional nodes
• DSE Search with Apache Solr™
• DSE Analytics with Apache Spark™
• DSE Graph
• DSE tools
• DSE drivers
• DSE OpsCenter

Setting up SSL certificates
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 with bring your own (BYO) root Certificate Authority.

DataStax recommends using certificates signed by a CA to reduce SSL certificate management tasks. However, it is possible to use self-signed certificates in DSE.

These steps walk you through the general process to generate and distribute SSL certificates using OpenSSL and Java keytool. Use an intermediary certificate chain to securely implement SSL in a production environment, see the OpenSSL root CA instructions.

OpsCenter Lifecycle Manager can configure DataStax Enterprise clusters to use client-to-node and node-to-node encryption and automates the process of preparing server certificates. See Configuring SSL/TLS for DSE using LCM.

Prerequisites:
Perform the following steps on a dedicated CA server which is fully encrypted and permanently isolated from the network. When using BYO CA, always create the root pair for certificate signing in a secure environment. Anybody with access to the root CA files can use it to sign certificates.

DataStax recommends using a computer outside the DSE environment to generate and manage SSL certificates.

The Common Name (CN) that is used to generate the SSL certificate must match the DNS resolvable host name. Mismatches between the CN and node hostname cause an exception and the connection is refused.

Skip to step 3 when using a third-party signed certificate or when adding a node using an existing rootCA.

1. **BYO root CA only:** create your own root CA for signing node certificates:
   a. Create a directory for the CA and then change to that directory:

   ```bash
   mkdir -p dse/root/ca
   ```
cd dse/root/ca

Ensure that the root CA files created in these steps are secured on a fully isolated computer dedicated to CA certificate management.

b. Create a configuration file in the ca directory:

```
# gen_rootCa_cert.conf
[ req ]
distinguished_name = req_distinguished_name
prompt    = no
output_password  = myPass
default_bits  = 2048

[ req_distinguished_name ]
C     = US
O     = org_name
OU    = cluster_name
CN    = rootCa
```

Where you define the variables for your environment as follows:

- `gen_rootCa_cert.conf` is the configuration file name
- `myPass` is the CA password
- `US` is the two character country code
- `org_name` is the name of your organization
- `cluster_name` is the name of your DataStax Enterprise cluster

c. Create a root pair.

```
openssl req -config gen_rootCa_cert.conf \
-new -x509 -nodes \
-subj /CN=rootCa/O=cluster_name/O=DataStax/C=US/ \
-keyout rootCa.key \
-out rootCa.crt \
-days 365
```

The root pair, `rootCa.key` key file and `rootCa.crt` are created. These instructions are for development and test environments, for a production environment these files would be carefully secured as described in the OpenSSL documentation.

d. Verify the root certificate:

```
openssl x509 -in rootCa.crt -text -noout
```

Certificate:

```
Data:
Version: 1 (0x0)
Serial Number: 14793138693831603662 (0xcd4bc943bee35ce)
Signature Algorithm: sha256WithRSAEncryption
Issuer: C=US, O=datastax, OU=pw-j-dse, CN=rootCa
Validity
    Not Before: Jan 23 20:15:06 2017 GMT
    Not After : Jan 23 20:15:06 2018 GMT
Subject: C=US, O=datastax, OU=pw-j-dse, CN=rootCa
```

DataStax 5.1 Security Guide Earlier DSE version Latest 5.1 patch: 5.1.17
Configuring SSL

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

Public-Key: (2048 bit)

Modulus:

Exponent: 65537 (0x10001)

Signature Algorithm: sha256WithRSAEncryption

2. Create a single truststore:

```
keytool -keystore dse-truststore.jks \
-importcert -file '../ca/rootCa.crt' \
-keypass myPass \ 
-storepass truststorePass \ 
-noprompt
```

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 DSE Java implementation.

The truststore contains a single entry. Verify the truststore using the following command:

```
keytool -list \
-keystore dse-truststore.jks \
-storepass truststorePass
```

Start here when using a third-party CA or when adding a node to an existing DSE environment with SSL enabled.
3. For each node in the cluster, create a keystore and key pair, and certificate signing request using FQDN of the node.

   a. Create a directory to store the keystores and change to the directory:

   ```
   mkdir -p dse/keystores
   cd dse/keystores
   ```

   b. For each node, generate a keystore with key pair:

   ```
   keytool -genkeypair -keyalg RSA
   -alias node_name
   -keystore keystore_name.jks
   -storepass myKeyPass
   -keypass myKeyPass
   -validity 365
   -keysize 2048
   -dname "CN=host_name, OU=cluster_name, O=org_name, C=US"
   ```

   where the `host_name` is the FQDN (Fully Qualified Domain Name).

   Use a DNS resolvable FQDN (Full Qualified Domain Name) for each node, to ensure the information you are using is correct run the following commands on each node:

   ```
   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

   c. Verify each SSL keystore and key pair:

   ```
   keytool -list -keystore keystore_name.jks -storepass myKeyPass
   ```

   Results for keystore with single entry with alias node1.

   Keystore type: JKS
   Keystore provider: SUN

   Your keystore contains 1 entry

   node1, Jan 23, 2017, PrivateKeyEntry,
   Certificate fingerprint (SHA1):

   d. Generate a signing request from each keystore:

   ```
   keytool -keystore keystore_name.jks
   -alias node_name
   -certreq -file signing_request.csr
   -keypass myKeyPass
   -storepass myKeyPass
   ```
Configuring SSL

-dname "CN=host_name, OU=cluster_name, O=org_name, C=US"

The certificate signing request file (signing_request.csr) is created. Repeat for each node, ensuring that the dname information matches the node information.

4. Sign the certificate signing request of each node:
   - BYO root CA: Using the root CA created in step 1, sign each nodes certificate:

     ```bash
     openssl x509 -req -CA '../ca/rootCa.crt' \
     -CAkey '../ca/rootCa.key' \
     -in node0.csr \
     -out node0.crt_signed \
     -days 365 \
     -CAcreateserial \
     -passin pass:myPass
     ```

     A signed certificate file is created, verify that was properly signed:

     ```bash
     openssl verify -CAfile '../ca/rootCa.crt' node0.crt_signed
     ```

     node0.crt_signed: OK

   - Send the certificate signing request to a well-known CA for signing.

5. For each node in the cluster, import the signed certificates into the keystores:
   - Import the root certificate into each node's keystore:

     ```bash
     keytool -keystore node0.keystore.jks \
     -alias node_name \
     -importcert -file '../ca/rootCa.crt' \
     -noprompt -keypass myKeyPass \
     -storepass myKeyPass
     ```

     where the following must match the items created in the previous steps:
     - `node0.keystore.jks` - keystore created in 3.b
     - `node_name` is the alias used in 3.b, `rootCa`
     - `rootCa.crt` - root certificate

     An error occurs, keytool error: java.lang.Exception: Failed to establish chain from reply, if the signed certificate for the node is imported before the root certificate.

   - Import the node's signed certificate into corresponding keystore:

     ```bash
     keytool -keystore node0.keystore.jks \
     -alias node_name \
     -importcert -noprompt \
     -file node0.crt_signed \
     -keypass myKeyPass \
     -storepass myKeyPass
     ```

     where the alias name must match the alias name used to generate the signing request in step 3.d.
6. Move the truststores and keystores to a computer with access to the DSE nodes and distribute to each node:
   
   a. Create a directory for the certificates in the DSE configuration directory on each node:

   ```
   mkdir -p dse/conf/certs
   ```

   b. Copy the truststore to each node using a generic name:

   ```
   scp dse-truststore.jks \
   node0:dse/conf/certs/truststore.jks
   ```
   
   Using the same name and location on all nodes allows the same configuration for SSL in the `cassandra.yaml`.

   c. Copy the corresponding keystore to each node using a generic name:

   ```
   scp node0.keystore.jks \
   node0:dse/conf/certs/keystore.jks
   ```
   
   Be sure to copy the correct keystore to the correct nodes.

### Securing internal transactional node connections

Node-to-node (internode) encryption protects data transferred between nodes in a cluster using SSL (Secure Sockets Layer). For information about generating SSL certificates, see Setting up SSL certificates.

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.

To enable node-to-node SSL encryption:

1. Set the `server_encryption_options` in the `cassandra.yaml` file on each node:

   - `internode_encryption`: Encrypts traffic between nodes, options: `none`, `all`, `dc`, or `rack`.
   - `keystore`: Relative path from DSE installation directory or absolute path to the keystore file.
   - `keystore_password`: Password to access the keystore.
   - `truststore`: Relative path from DSE installation directory or absolute path to truststore file.
   - `truststore_password`: Password to access truststore.
   - `require_client_auth`: Enable two way encryption. After enabling you must configure clients, such as nodetool and cqlsh to use SSL.
   - `require_endpoint_verification`: Optional, verify the connected host and the host name in the certificate match.
Configuring SSL

To encrypt the truststore and keystore passwords with KMIP, see Encrypting table data.

```
server_encryption_options:
    internode_encryption: all
    keystore: resources/dse/conf/keystore.jks
    keystore_password: myPassKey
    truststore: resources/dse/conf/truststore.jks
    truststore_password: truststorePass
    require_client_auth: true
    require_endpoint_verification: true
```

2. Restart DSE.

Securing client to cluster 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. Unlike Kerberos, does not require setting up a shared authentication service.

If you are using DSE Search, enabling SSL automatically enables it in the DSE Search `web.xml` file and configures an SSL connector in Tomcat using the authentication/authorization filters. You do not have to change your `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 `rpc_address` in `cassandra.yaml`.

Prerequisites:

- DataStax recommends Enabling JCE Unlimited.
- Prepare an SSL certificate, Setting up SSL certificates.

To enable encryption, perform these steps on each node:

DSE Search and Spark nodes require the truststore entries in `cassandra.yaml`.

1. Production clusters: In the `cassandra.yaml` file, in the `client_encryption_options` section:
   - Set `enabled` to true.
   - Set the paths to your `.keystore` and `.truststore` files.
   - Provide the passwords that were used when generating the keystore and truststore.
   - To enable client certificate authentication, set `require_client_auth` to true.
   - Use an SSL production certificate.

```
client_encryption_options:
    enabled: true
    keyStore: resources/dse/conf/.keystore    ## Path to your .keystore file
    keyStore_password: keystore_password    ## Password that you used to generate the keystore
    require_client_auth: true
    trustStore: resources/dse/conf/.truststore  ## Path to your .truststore
    trustStore_password: truststore_password  ## Password that you used to generate the truststore
    protocol: ssl
    algorithm: SunX509
```
Configuring SSL

store_type: JKS
cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA]

For information about using Kerberos with SSL, see Using CQL shell (cqlsh) with SSL. To encrypt the truststore and keystore passwords with KMIP, see Encrypting table data.

2. Development clusters: In the cassandra.yaml file, in the client_encryption_options section:

   • Set enabled to true.
   • Provide the passwords that were used when generating the keystore and truststore.
   • Set the paths to your .keystore and .truststore files.
   • If two-way certificate authentication is desired, set require_client_auth to true. Enabling two-way certificate authentication allows tools to connect to a remote node.
   • Complete Setting up SSL certificates.

   For local access to run cqlsh on a local node with SSL encryption, require_client_auth can be set to false.

   ```yaml
   client_encryption_options:
     enabled: true
     # If enabled and optional is set to true encrypted and unencrypted connections are handled.
     optional: false
     keystore: conf/keystore.node0
     keystore_password: cassandra
     require_client_auth: true
     # Set trustore and truststore_password if require_client_auth is true
     truststore: conf/truststore.node0
     truststore_password: cassandra
     protocol: TLS
     algorithm: SunX509
     store_type: JKS
     cipher_suites: [TLS_RSA_WITH_AES_256_CBC_SHA]
   ```

3. If the client_encryption_options are set in dse.yaml file, remove them.

4. If you are not using the JCE Unlimited Strength Jurisdiction Policy, make sure that your ticket granting principal does not use AES-256.

   If your ticket granting principal uses AES-256, you might see a warning like this in the logs:

   ```java
   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
   ```

5. Restart DataStax Enterprise.

Securing Spark connections

Communication between Spark applications and transactional nodes, masters and workers, and intercommunication between Spark drivers and executors can be encrypted. You must configure encryption on each node in your cluster.

Encryption between the Spark driver and DSE is configured by enabling client encryption in cassandra.yaml.
Encryption between Spark nodes, 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 in the Spark configuration directory.

The default location of the Spark configuration files depends on the type of installation:

- **Package installations and Installer-Services**: /etc/dse/spark/
- **Tarball installations and Installer-No Services**: installation_location/resources/spark/conf

1. Enable mutual authentication and encryption between Spark master and worker nodes in dse.yaml.

   In DSE 5.1.15 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 driver 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 Spark SQL Thrift 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 to enable encryption between the Spark driver and executors. To enable encryption by default for all Spark applications modify the options in the spark-defaults.conf file in the Spark configuration directory.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spark.authenticate</td>
<td>Enables or disables mutual authentication using a secret key. The default is false.</td>
</tr>
<tr>
<td>spark.authenticate.enableSaslEncryption</td>
<td>Enables or disables SASL encryption between a Spark driver and its executors. The default is false.</td>
</tr>
</tbody>
</table>
Using CQL shell (cqlsh) with SSL

To use cqlsh with Kerberos and SSL, use the sample files as a starting point and make changes as appropriate for your environment.

Example files
DataStax Enterprise provides sample files and examples to help configure authentication for Kerberos, SSL, and Kerberos and SSL:

- SSL example
- Kerberos and SSL

Make changes as appropriate for your environment.

See the cqlshrc.sample.

SSL example
DataStax Enterprise provides a sample cqlshrc.sample.ssl file that you can use as a starting point.

```
[authentication]
username = fred
password = !!bang!!$

[connection]
hostname = 127.0.0.1
port = 9042

[ssl]
certfile = ~/keys/cassandra.cert
validate = false ;; Optional, true by default. See the paragraph below.

[certfiles] ;; Optional section, overrides the default certfile in the [ssl] section.
10.209.182.160 = /etc/dse/cassandra/conf/dsenode0.cer
10.68.65.199 = /etc/dse/cassandra/conf/dsenode1.cer
```

When validate 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
DataStax Enterprise provides a sample cqlshrc.sample.kerberos_ssl file that you can use as a starting point.

For information about using Kerberos with SSL, see Using CQL shell (cqlsh) with SSL.

The settings for using both Kerberos and SSL are a combination of the Kerberos and SSL sections in these examples.

The supported environmental variables are KRB_SERVICE, SSL_CERTFILE, and SSL_VALIDATE variables.

Debugging cqlsh authentication
Use the --debug option to troubleshoot authentication problems with cqlsh. Pass the --debug option to cqlsh to populate the debug log message with the type of authentication that cqlsh is attempting.
Configuring SSL

Setting up SSL for nodetool, dsetool, and dse advrep

Using nodetool, dsetool, and dse advrep with SSL requires some JMX setup.

**Prerequisites:** Complete Setting up SSL certificates. Additionally, configure client-to-node encryption.

A high-level overview of the required configuration to set up nodetool, dsetool, and dse advrep for use with SSL:

1. Configure JMX SSL on the server side with changes on each node in the cluster.
2. Restart DSE.
3. Configure the client settings in your home or client program directory on the node on which the command will run.

Configure JMX SSL on the server side:

Make these changes in the cassandra-env.sh file on each node in the cluster.

1. If the $LOCAL_JMX setting is present, change it to no:

   ```
   "$LOCAL_JMX" = "no"
   ```

2. Add the following settings:

   You can also use the jvm.options file as described in start-up parameters.

   **For production:**

   ```
   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 -Dcom.sun.management.jmxremote.ssl.enabled.protocols=<enabled-protocols>"
   #JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl.enabled.cipher.suites=<enabled-cipher-suites>"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStore=/usr/local/lib/cassandra/conf/server-keystore.jks"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStorePassword=myKeyPass"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStore=/usr/local/lib/cassandra/conf/server-truststore.jks"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStorePassword=truststorePass"
   ```

   **For development:**

   ```
   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 -Dcom.sun.management.jmxremote.ssl.enabled.protocols=<enabled-protocols>"
   #JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl.enabled.cipher.suites=<enabled-cipher-suites>"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStore=keystore.node0"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStorePassword=cassandra"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStore=truststore.node0"
   JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStorePassword=truststorePass"
   ```

   where:

   - com.sun.management.jmxremote.ssl=true enables SSL for JMX.
• `com.sun.management.jmxremote.ssl.need.client.auth=true` enables two-way certificate authentication.
• `com.sun.management.jmxremote.registry.ssl=true` creates an RMI registry protected by SSL and configures an out-of-the-box management agent when the Java VM is started.
• `com.sun.management.jmxremote.registry.ssl=true` requires that `com.sun.management.jmxremote.ssl.need.client.auth=true` is also enabled.

You must:
• Set appropriate paths to the keystore and truststore files.
• Set the passwords to the passwords set during keystore and truststore generation.

3. Restart DSE.

4. To configure the client settings, create a `.cassandra/nodetool-ssl.properties` file in your home or client program directory with the following settings on the node on which the command will run.

   For production:
   ```
   -Dcom.sun.management.jmxremote.ssl=true
   -Dcom.sun.management.jmxremote.ssl.need.client.auth=false
   -Dcom.sun.management.jmxremote.registry.ssl=true
   -Djavax.net.ssl.keyStore=/usr/local/lib/dse/resources/dse/conf/.keystore
   -Djavax.net.ssl.keyStorePassword=cassandra
   -Djavax.net.ssl.trustStore=/usr/local/lib/cassandra/conf/.truststore
   -Djavax.net.ssl.trustStorePassword=cassandra
   ```

   For development:
   ```
   -Djavax.net.ssl.keyStore=keystore.node0
   -Djavax.net.ssl.keyStorePassword=cassandra
   -Djavax.net.ssl.trustStore=truststore.node0
   -Djavax.net.ssl.trustStorePassword=cassandra
   -Dcom.sun.management.jmxremote.ssl.need.client.auth=true
   -Dcom.sun.management.jmxremote.registry.ssl=true
   ```

   To use nodetool, dsetool, and dse advrep with SSL for an encrypted connection for any operation:

5. Start the command with the `--ssl` option.
   nodetool example:
   ```
   $ nodetool --ssl command
   ```

   dsetool example:
   ```
   $ dsetool --ssl command
   ```

   dse advrep example:
   ```
   $ dse advrep --ssl command
   ```

6. Start the command with the `--ssl` option for an encrypted connection and specify the username and password for authentication and authorization for any operation. If you do not enter a password, you are prompted to enter one.
Configuring SSL

nodetool example:

```
$ nodetool --ssl -u username -pw password command
```

dsetool example:

```
$ dsetool --ssl -a jmx_username -b jmxpassword command
```

dse advrep example:

```
$ dse advrep --ssl -u username command
```

Setting up SSL for jconsole (JMX)

Using jconsole with SSL requires the same JMX changes to cassandra-env.sh as described in using nodetool (JMX) with SSL encryption. There is no need to create nodetool-ssl.properties, but the same JVM keystore and truststore options must be specified with jconsole on the command line.

Prerequisites:
Prepare SSL certificates with a self-signed CA for production, or prepare SSL certificates for development. Additionally, configure client-to-node encryption.

1. Copy the keystore and truststore files created in the prerequisite 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=myKeyPass
-Djavax.net.ssl.trustStore=server-truststore.jks
-Djavax.net.ssl.trustStorePassword=truststorePass
```

If no errors occur, jconsole will start. If connecting to a remote node, enter the hostname and JMX port, in Remote Process. If using authentication, enter the username and password.

Connecting sstableloader to a secured cluster

The sstableloader tool is also called bulk loader. If you run sstableloader from a DataStax Enterprise node that has been configured for Kerberos or client-to-node/node-to-node encryption using SSL, no additional configuration is needed for securing sstableloader operations. sstableloader automatically detects the configuration. On an unconfigured development machine, however, configure Kerberos or SSL as follows:

To use SSL to connect to an unsecured DataStax Enterprise 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/truststore \
```
If you want to configure require_client_auth=true on the target, set these additional options:

```
resources/cassandra/bin/sstableloader -d 192.168.56.102 /var/lib/cassandra/data/Keyspace1/
  Standard1 
  -tf org.apache.cassandra.thrift.SSLTransportFactory 
  -ts /path/to/truststore 
  -tspw truststore_password 
  -ks /path/to/keystore 
  -kspw keystore_password
```

## 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:

```
encryption_options:
  enabled: true
  optional: true
  truststore:
    truststore_password:
    store_type:
    keystore:
      keystore_password:
      protocol:
      algorithm:
      cipher_suites:
      require_client_auth: 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
Configuring SSL

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_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

store_type
Valid types are JKS, JCEKS, and 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.
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
- 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)