Exercise 10.03 – Configuring SSL Connections

This exercise will instruct users on how to configure SSL connections between nodes. As we have only built a single node at present, these instructions will serve as a jumping-off point for later configurations. Node-to-node or *internode* encryption protects data in-flight between nodes in a cluster using SSL (Secure Sockets Layer).

Optional: Students can clone an existing VM instance and set up an SSL connection between the two VMs.

When done with this exercise you should be able to perform the following actions:

1. Enable SSL server encryption for node-to-node encryption
2. Be able to configure SSL connections for client-to-node connections

Step 1: Enable SSL Server Encryption for node-to-node Connections

To enable node-to-node SSL encryption, set the *server_encryption_options* in the *cassandra.yaml* file on every node requiring secure communications. Each step below explains the available options and which settings should be used:

To encrypt traffic between nodes, set the option *internode_encryption* to one of the following options:

- **all** - Encrypt all inter-node communications
- **none** - No encryption
- **dc** - Encrypt the traffic between the datacenters (server only)
- **rack** - Encrypt the traffic between the racks (server only)

Enforce client-to-node encryption by setting the option *require_client_auth* to **true**.

Tip: After enabling you must configure clients, such as *nodetool* and *cqlsh,* to use SSL.

- Verify that the connected hostname matches the certificate.
- Set require_endpoint_verification to **true**.

Configure the keystore and truststore:

- Local keystore and truststore files:
    - keystore_type: *PKCS12*
    - keystore: *Relative path from DSE installation directory or absolute path to the keystore file*
    - keystore_password: *Password to access the keystore*
    - truststore_type: *PKCS12*
    - truststore: *Relative path from DSE installation directory or absolute path to truststore file*
    - truststore_password: *Password to access truststore*

The following example shows possible options using local files:

```
server_encryption_options:
    internode_encryption: all
    keystore_type: PKCS12
    keystore: resources/dse/conf/keystore.jks
    keystore_password: myPassKey
    truststore_type: PKCS12
    truststore: resources/dse/conf/truststore.jks
```

```
    truststore_password: truststorePass
    require_client_auth: true
    require_endpoint_verification: true
```

- Remote keystore and truststore:
    - keystore_type: *PKCS11*
    - keystore: *blank or commented out*
    - keystore_password: *blank or commented out*
    - truststore_type: *PKCS11*
    - truststore: *blank or commented out*
    - truststore_password: *blank or commented out*

Note: A provider is required for a remote keystore and truststore installation.

Example code for a remote keystore and truststore:

```
server_encryption_options:
    internode_encryption: all
    keystore_type: PKCS11
    keystore:
    keystore_password:
    truststore_type: PKCS11
    truststore:
    truststore_password:
   require_client_auth:
    require_endpoint_verification:
```

Restart DSE to activate the changes.

Step 2: Enable SSL for client-to-node Connections

Client-to-node encryption protects in-flight data from client machines to a database cluster using SSL (Secure Sockets Layer) and establishes a secure channel between the client and the coordinator node.

Note: On a DSE Search node, enabling SSL for the database automatically enables SSL in the DSE Search *web.xml* file and configures an SSL connector in Tomcat using the authentication/authorization filters. No changes are required for the *web.xml* or *server.xml* files.

If the *TomcatSolrRunner* does not find a connector in the *server.xml* file, it creates a default connector. The default connector binds to the option set in *native_transport_address*.

CAUTION: If you are not using the JCE Unlimited Strength Jurisdiction Policy, make certain the ticket granting principal does *not* use AES-256. If the ticket granting principal is using AES-256, a warning similar to the following may appear in the logs:

```
WARN [StreamConnectionEstablisher:18] 2015-06-22 14:12:18,589
SSLFactory.java (line 162) Filtering out
TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE
_RSA_WITH_AES_256_CBC_SHA as it isnt supported by the socket
```

Configure SSL for client-to-node in Production Environments

Set up SSL for client-to-node connections for a cluster in a production environment. Perform the following steps on each node:

Note: DSE Search and Spark nodes require the truststore entries in the *cassandra.yaml* file.

1. Enable SSL. In the section labeled *client_encryption_options*, set the *enabled* option to **true**.
2. Permit only SSL connections. Set the *optional* setting to **false** (default is **false**).
3. Require two-way host certificate validation. Set the *require_client_auth* option to **true**.
4. Configure the keystore and truststore:
    - Local files – use the following settings:
        - keystore_type: *PKCS12*
        - keystore: *Relative path to the keystore file*
        - keystore_password: *Password to access the keystore*
        - truststore_type: *PKCS12*
        - truststore: *Relative path from DSE installation directory or absolute path to truststore file*
        - truststore_password: *Password to access truststore*

Note: The *truststore* password and path is only required when the option *require_client_auth* is set to **true**.

Consult the following example configuration setting using local files:

```
client_encryption_options:
    enabled: true
    keystore_type: PKCS12
    keystore: resources/dse/conf/.keystore
    keystore_password: cassandra
    require_client_auth: true
    truststore_type: PKCS12
    truststore: resources/dse/conf/.truststore
    truststore_password: cassandra
    protocol: ssl
    algorithm: SunX509
    store_type: JKS
    cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA]
```

Remote device using custom provider:

- keystore_type: *PKCS11*
- keystore: *blank or commented out.*
- keystore_password: *blank or commented out.*
- truststore_type: *PKCS11*
- truststore: *blank or commented out.*
- truststore_password: *blank or commented out.*

Note: A provider is required for a remote keystore and truststore installation.

Consult the following example configuration setting with remote client-to-node options:

```
client_encryption_options:
    enabled: true
    keystore_type: PKCS12
    keystore: resources/dse/conf/.keystore
    keystore_password: cassandra
    require_client_auth: false
    protocol: ssl
    algorithm: SunX509
    store_type: JKS
    cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA]
```

Restart DataStax Enterprise to make the changes effective.

Configure SSL for client-to-node connections in a development environment

Set up SSL for client-to-node connections for a cluster in a development, test, or demonstration environment. In the *cassandra.yaml* file, change the following settings in the *client_encryption_options* section:

- Set *enabled* to **true**.
- Provide the passwords used when generating the *keystore* and *truststore*.
- Set the paths to the following files: *.keystore* and *.truststore*.
- If two-way certificate authentication is required, set the variable *require_client_auth* to **true**.
- Enabling two-way certificate authentication allows the SSL tools to connect to a remote node.

Make certain all the previously required steps for using SSL certificate and keystore files have been completed.

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

```
client_encryption_options:
  enabled: true
  optional: false
  keystore_type: PKCS12
  keystore: resources/dse/conf/.keystore
  keystore_password: cassandra
  require_client_auth: false
  truststore_type: PKCS12
  truststore: resources/dse/conf/.truststore
  truststore_password: cassandra
  # protocol: TLS
  # algorithm: SunX509
  cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA]
```

Restart DataStax Enterprise to make the changes effective.