Exercise 10.2 – Installing Root Certificates on Nodes

This exercise will act as a general guide to create and distribute SSL certificates using OpenSSL and Java keytool. In this exercise you will learn the following:

1. How to create a root CA signing certificate
2. Create a truststore for all nodes
3. Create a Certificate Signing Request (CSR)
4. Sign the CSR or send to an authorized Certificate Authority (CA)
5. Create a local keystore for each node

We typically use SSL certificates for *client-to-node encryption* and *node-to-node encryption*. DataStax supports SSL using well-known Certificate Authority (CA) signed certificates for each node or using a Bring Your Own (BYO) root CA. For production nodes, DataStax recommends using only certificates signed by a CA to reduce SSL certificate management tasks. However, it is possible to use self-signed certificates in DataStax Enterprise (DSE). DSE and DataStax Distributed Apache Cassandra supports Secure Socket Layer (SSL) certificates in local and external keystores.

DataStax recommends using a computer outside the DSE environment to generate and manage SSL certificates. Perform the steps on a dedicated CA server which is fully encrypted and permanently isolated from the network. In development and testing environments you can set up your own root CA to sign DataStax Enterprise node certificates for SSL.

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

Step 1: Creating a Root CA Signing Certificate

Create a directory for the Bring Your Own (BYO) root CA signing certificate/key and then change to that directory:

```
mkdir -p rootca_path &&
cd rootca_path
```

Note: Make certain the root CA files created in these steps are secured on a fully isolated computer dedicated to CA certificate management.

Create a configuration file (*rootca.conf*) with the minimal settings:

```
# rootca.conf
[ req ]
distinguished_name       = CA_DN
prompt                   = no
output_password          = rootca_password
default_bits             = 2048

[ CA_DN ]
C  = CC
O  = org_name
OU = cluster_name
CN = CA_CN
```

Create a root pair, *rootca.key* and *rootca.crt*.

```
openssl req -config rootca.conf \
  -new -x509 -nodes \
  -subj /CN=CA_CN/OU=cluster_name/O=org_name/C=CC/ \
  -keyout rootca.key \
  -out rootca.crt \
  -days 365
```

This method is only for development and test environments. Secure these files in a safe location. Anyone with access to these files can sign certificates.

Tip: BYO CA in a production environment typically uses an intermediary certificate chain.

Verify the root certificate:

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

Confirm the output appears normal and the settings are correct.

Step 2: Creating a Truststore for all Nodes

Create a truststore to be used on all nodes ensuring all nodes recognize the CA. Even when using a well-known certificate authority, DataStax recommends creating a truststore with the signing CA certificate (or certificate chain following the instructions from your CA). Most well-known CA certificates are already available through the DSE Java implementation.

This section uses the following variables:

- Root CA key and signing certificate

- *rootca.key* – Key file for the root certificate
- *rootca.crt* – Certificate used to sign (authorize) DSE node SSL certificates

Important: Anyone with access to the key and signing certificate can authorize hosts as the root certificate authority. Always secure these files.

- Truststore variables:
  - *dse-truststore_name.jks* – Truststore containing the root certificate
  - *key_password* – To be determined by the user
  - *trustedCert* – To be created by the command

Start by creating a single truststore:

```
keytool -keystore dse-truststore.jks \
  -importcert -file 'rootca.crt' \
  -keypass key_password \
  -storepass truststore_password \
  -noprompt
```

The truststore should contain only a single entry.

Verify the truststore using the following command:

```
keytool -list \
  -keystore dseTruststore_name.jks \
  -storepass truststore_password
```

Step 3: Create a Certificate Signing Request

For each node in the cluster, create a *keystore* and *key pair*, and a *Certificate Signing Request (CSR)* using the Fully-Qualified Domain Name (FQDN) of the node.

Note: These steps are required even when using a third-party CA or when adding a node to an existing DSE environment with SSL enabled.

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

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

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* variable is the Fully-Qualified Domain Name (FQDN).

Important: Use a DNS resolvable FQDN for each node. As these nodes are typically internal to the network, use internal DNS to resolve host names. Confirm the information is correct by running 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
```

The Common Name (CN) used to generate the SSL certificate must match the DNS resolvable host name. Mismatches between the CN and node hostname can cause an exception resulting in a refusal of the connection.

Verify each SSL keystore and key pair:

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

Sample results for the keystore with a single entry using the alias *node1*.

```
Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

node1, Jan 23, 2017, PrivateKeyEntry,
Certificate fingerprint (SHA1):
12:B7:45:AA:AD:F0:22:23:3F:13:FC:2C:3D:A4:4F:84:16:96:58:66
```

Generate a signing request from each keystore:

```
keytool -keystore keystore_name.jks \
-alias node_name \
-certreq -file signing_request.csr \
-keypass myKeyPass \
-storepass myKeyPass \
-dname "CN=host_name, OU=cluster_name, O=org_name, C=US"
```

The Certificate Signing Request (CSR) file (*signing_request.csr*) is thus created. Repeat the process for each node in the cluster, confirming the *dname* information matches the node information.

Step 4: Signing the Certificate Signing Request

For each node, sign the certificate signing request. If you created a bring your own (BYO) root CA, follow the instructions below. Alternatively, send the certificate signing request to a well-known CA for signing. For the purpose of this exercise, signing your own certificate should not be a problem.

Sign each node certificate:

```
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 the BYO certificate file was properly signed:

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

Step 5: Create a Local Keystore for each Node

For each node in the cluster, create a keystore and import the signed certificate.

Import the root certificate into each node's keystore:

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

Where following must match the items created in the previous steps:

- *node0.keystore.jks* – keystore created by the certificate signing request
- *node_name* – alias used when creating a certificate signing request
- *rootCa.crt* – root certificate.

Warning: 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:

```
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 when generating the signing request.

Confirmation of the installation appears, repeat both steps for each node's keystore:

```
Certificate reply was installed in keystore
```