# DSE Security HowTo - Enable SSL/TLS for DSE

## System Info

- 1 DSE cluster: 2-node, version 5.0.7, graph/spark/solr enabled
  - node1 (privarte) IP: 10.240.0.6
  - node2 (privarte) IP: 10.240.0.7
- 1 OpsCenter instance, version 6.0.7
  - (private) IP: 10.240.0.8
- OS: Ubuntu 16.04.2 LTS

## Procedures

### 1. Install Java Cryptography Extension (JCE)

For Debian based system, JCE can be installed via APT. Run the following commands on every DSE and OpsCenter node (for Java8 JCE):

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt update
sudo apt install oracle-java8-unlimited-jce-policy
```

### 2. Prepare Keystore and Truststore

Please **note** that the methods described in this section is the most basic approach of how to prepare the keystores and truststores used for DSE. It should only be used for small DSE cluster for DEV/TEST purpose and is listed here for concept clarification purpose.

For larger or production DSE deployment, it is recommended to use the approach of validation through a Certificate Authority (CA). For DSE version 5.0+ managed by OpsCenter 6.0+, the Life Cycle Manager (LCM) can do the SSL configuration automatically for a DSE cluster. I will describe these 2 approaches in a separate blog post.

**== Keystore ==**

On any node, use keytool to generate the keystore for each node, as per the following command:

```
keytool -genkey -keyalg RSA -alias <alias_name> -keystore <keystore_name> -storepass <keystore_password> -keypass <keystore_password> -dname "CN=<node_ip>, OU=None, O=None, L=None, C=None"
```

For example, for the 3 nodes in this scenario, the commands are:

```
## For DSE nodes

keytool -genkey -keyalg RSA -alias dsenode0 -keystore dsenode0.keystore -storepass dsestore123 -keypass dsestore123 -dname "CN=10.240.0.6, OU=None, O=None, L=None, C=None"

keytool -genkey -keyalg RSA -alias dsenode1 -keystore dsenode1.keystore -storepass dsestore123 -keypass dsestore123 -dname "CN=10.240.0.7, OU=None, O=None, L=None, C=None"

## For OpsCenter node

keytool -genkey -keyalg RSA -alias opscnode -keystore opscnode.keystore -storepass dsestore123 -keypass dsestore123 -dname "CN=10.240.0.8, OU=None, O=None, L=None, C=None"
```

**== Truststore ==**

For the keystores created in the previous step, export the public certificate part out of the keystore and import it into a common truststore, as per the following commands:

```
keytool -export -alias <alias_name> -file <certificate_name> -keystore <keystore_name> -storepass <kestore_password>

keytool -import -v -trustcacerts -alias <alias_name> -file <certificate_name> -keystore <common_truststore_name> -storepass <truststore_password>
```

- note: when prompted for "Trust this certificate?", enter "yes"

For example, for the 3 nodes in our scenario, the commands are as below. Please note that commands below choose to use the same truststore password as the keystore password. You're free to choose a different password.

```
## DSE node 0

keytool -export -alias dsenode0 -file dsenode0.cer -keystore dsenode0.keystore -storepass dsestore123

keytool -import -v -trustcacerts -alias dsenode0 -file dsenode0.cer -keystore dse.truststore -storepass dsestore123

## DSE node 1

keytool -export -alias dsenode1 -file dsenode1.cer -keystore dsenode1.keystore -storepass dsestore123

keytool -import -v -trustcacerts -alias dsenode1 -file dsenode1.cer -keystore dse.truststore -storepass dsestore123

## OpsCenter node

keytool -export -alias opscnode -file opscnode.cer -keystore opscnode.keystore -storepass dsestore123

keytool -import -v -trustcacerts -alias opscnode -file opscnode.cer -keystore dse.truststore -storepass dsestore123
```

## 3. Place the Keystore and Truststore to DSE Cluster

Copy each generated keystore file to its corresponding DSE node or OpsCenter node. Copy the common truststore to every node. The target location can be any directory, but practically I prefer to put it under the main DSE/OpsCenter configuration folder, such as /etc/dse/security/, or /etc/opscenter/security.

For example, for the 3 nodes in this scenario, the commands are as below. Please note that for configuration convenience purpose, when the keystore and truststore files are copied over to the target node, they are renamed as ".keystore" and ".truststore" files respectively.

```
## DSE node0

scp dsenode0.keystore automaton@10.240.0.6:/usr/local/lib/dse/security/.keystore

scp dse.truststore automaton@10.240.0.6:/usr/local/lib/dse/security/.truststore

## DSE node1

scp dsenode1.keystore automaton@10.240.0.7:/usr/local/lib/dse/security/.keystore

scp dse.truststore automaton@10.240.0.7:/usr/local/lib/dse/security/.truststore

## OpsCenter node

scp opscnode.keystore automaton@10.240.0.8:/etc/opscenter/security/.keystore

scp dse.truststore automaton@10.240.0.8:/etc/opscenter/security/.truststore
```

## 4. Configure DSE Node for Client-to-Node and Node-to-Node SSL Encryption

In "cassandra.yaml" file, make the following changes.

Most of these configuration items are quite straightforward and don't need much explanation. The only configuration item that needs a bit more explanation is "require_client_auth" which means 2-way certificate authentication. When SSL encryption is enabled, it is recommended to enable this for node-to-node encryption, but keep it as disabled for client-to-node encryption.

```
#authenticator: AllowAllAuthenticator
authenticator: com.datastax.bdp.cassandra.auth.DseAuthenticator

server_encryption_options:

    internode_encryption: all
    keystore: /etc/dse/security/.keystore
    keystore_password: dsestore123
    truststore: /etc/dse/security/.truststore
    truststore_password: dsestore123
    # More advanced defaults below:
    # protocol: TLS
```

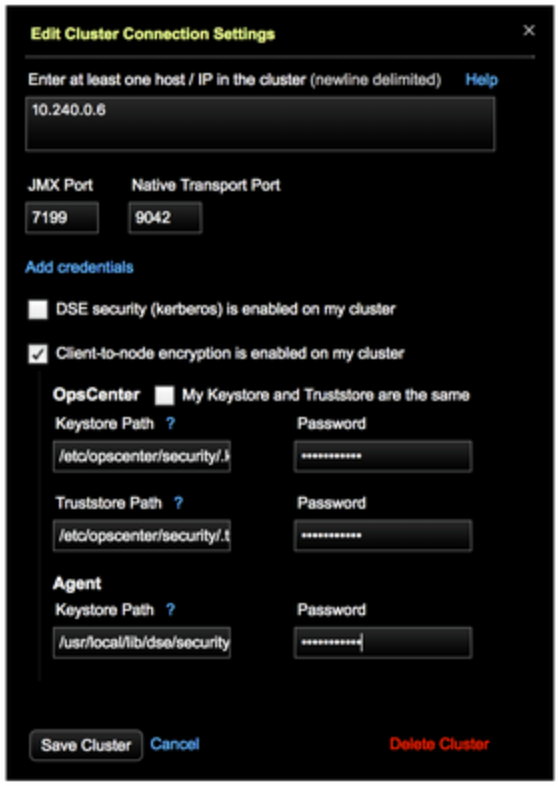
```
        # algorithm: SunX509
        # store_type: JKS
        # cipher_suites:
        [TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_SH
        A,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_
        AES_256_CBC_SHA]
        require_client_auth: true


    client_encryption_options:

        enabled: true
        # If enabled and optional is set to true encrypted and unencrypted connections are handled.
        optional: false
        keystore: /etc/dse/security/.keystore
        keystore_password: dsestore123
        # require_client_auth: false
        # Set trustore and truststore_password if require_client_auth is true
        # truststore: /etc/dse/security/.truststore
        # truststore_password: dsestore123
        # More advanced defaults below:
        # protocol: TLS
        # algorithm: SunX509
        # store_type: JKS
        # cipher_suites:
        [TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_SH
        A,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_
        AES_256_CBC_SHA]
```

Restart DSE cluster after making the above changes and verify Cassandra system log file for the following messages:

"Starting Encrypted Messaging Service on SSL port 7001"

## 5. Configure OpsCenter and datastax-agent when DSE client-to-node SSL encryption is enabled

Once DSE client-to-node SSL encryption is enabled, OpsCenter and datastax-agent needs to be configured for SSL encryption as well since they're considered as clients to a DSE cluster. The easiest way to configure OpsCenter and datastax-agent SSL communication with DSE cluster is through OpsCenter WebUI:




The key configuration items in the UI (as per the screenshot above) are:

- [OpsCenter] Keystore Path:  the absolute file path for the keystore for OpsCenter. For our example, this would be "/etc/opscenter/security/.keystore"
- [OpsCenter] Keystore Password: the password for OpsCenter keystore. For our example, this would be "dsestore123"
- [OpsCenter] Truststore Path:  the absolute file path for the keystore for OpsCenter. For example, for our example, this would be "/etc/opscenter/security/.truststore"
- [OpsCenter] Truststore Password: the password for OpsCenter keystore. For our example, this would be "dsestore123"
- [Agent] Keystore Path: the absolute file path for the keystore for datastax-agent. Since datastax-agent resides on DSE node, this would be the same as the keystore generated for each DSE node. For our example, this would be "/usr/local/lib/dse/security/.keystore"
- [Agent] Keystore Password: the password for OpsCenter datastax-agent. For our example, this would be "dsestore123"

The SSL configuration items from UI, once saved, are written in OpsCenter's cluster specific config file (e.g. /etc/opscenter/clusters/<cluster_name>.conf). See the ***bold*** text below:

[agents]
***ssl_keystore_password = dsestore123***
***ssl_keystore = /usr/local/lib/dse/security/.keystore***

[cassandra]
***ssl_truststore_password = dsestore123***
cql_port = 9042
seed_hosts = 10.240.0.6
password =
***ssl_keystore_password = dsestore123***
***ssl_keystore = /etc/opscenter/security/.keystore***
***ssl_truststore = /etc/opscenter/security/.truststore***
username =

## 6. Configure CQLSH when DSE client-to-node SSL encryption is enabled

Similarly, when DSE client-to-node SSL encryption is enabled, CQLSH also needs to be configured accordingly to enable SSL communication between CQLSH client and DSE node. The keystore format generated above is of JKS store type (Java Key Store format). CQLSH tool, however, requires PEM type (OpenSSL format) for setting up SSL connection. In order to convert a JKS format (.jks) to PEM format (.pem), an intermediate PKCS12 format (.pk12) is needed. The commands to do so are as below:

keytool -importkeystore -srckeystore <JKS_keystore_name> -destkeystore <PKCS12_keystore_name> -deststoretype PKCS12

```
-srcstorepass <keystore_password> -deststorepass <keystore_password>
openssl pkcs12 -in <PKCS12_keystore_name> -out <PEM_keystore_name> -passin pass:<keystore_password>
```

In our example, if we run CQLSH from OpsCenter node, the following commands can generate the PEM keystore used for CQLSH:

```
keytool -importkeystore -srckeystore opscnode.keystore -destkeystore opscnode.p12 -deststoretype PKCS12 -srcstorepass
dsestore123 -deststorepass dsestore123
openssl pkcs12 -in opscnode.p12 -out opscnode.pem -passin pass:dsestore123
```

Once the required PEM file is generated, add the following sections in .cassandra/cqlshrc file:

```
[connection]
hostname = <default_cqlsh_connection_IP>
port = 9042
factory = cqlshlib.ssl.ssl_transport_factory

[ssl]
certfile = <CQLSH SSL PEM file for OpsCenter node>    ## (e.g. ~/opscnode.pem)
validate = false

[certfiles]
<dsenode0_IP> = <CQLSH SSL PEM file for DSE node 0>    ## (e.g. ~/dsenode0.pem)
<dsenode1_IP> = <CQLSH SSL PEM file for DSE node 1>    ## (e.g. ~/dsenode1.pem)
... ...
```

Then run "**cqlsh --ssl**" command to initiate CQLSH connection to DSE node with SSL encryption

## 7. Configure SSL communication between OpsCenter daemon and datastax-agent

The communication between OpsCenter server and datastax-agent by default is not SSL-enabled. OpsCenter installation has a folder that contains the keystore files needed for SSL communication between OpsCenter server and datastax-agents. For packaged installation, the folder is ***/var/lib/opscenter/ssl*** and the keystore file needed is ***agentKeyStore***. The procedure of setting OpsCenter/datastax-agent SSL communication is as below (package installation):

- Copy keystore file ***/var/lib/opscenter/ssl/agentKeyStore*** from OpsCenter node to every datastax-agent node. The target location folder is: **/var/lib/datastax-agent/ssl/.** Since datastax-agent installation doesn't have "ssl" sub-folder under /var/lib/datastax-agent/ folder, please create it manually and make sure its user/group ownership is cassandra.
- On OpsCenter node, edit the main OpsCenter configuration file (opscenterd.conf) and add the following section. Restart OpsCenter daemon after.

  ```
  [agents]

  use_ssl = true
  ```

- On each datastax-agent node, edit the main datastax-agent configuration file (address.yaml) and add/update the following item. Restart datastax-agent after.

  ```
  use_ssl: 1
  ```

## 8. Cassandra Client Driver SSL

The following web page gives a detailed description on SSL configuration for Cassandra java client driver: http://docs.datastax.com/en/developer/java-driver/3.1/manual/ssl/.

For other programming language driver SSL configuration, please check corresponding Cassandra client driver document for that language.