Chapter 40.

April 2020

Welcome to the April 2020 edition of DataStax Developer's Notebook (DDN). This month we answer the following question(s);

> My company has started using more cloud instances for tasks like proof of concepts, and related. We used to just leave these boxes wide open, since they generally contain zero sensitive data. But, things being what they are, we feel like we should start securing these boxes. Can you help ?
>
> *Excellent question ! In the March/2019 edition of this document, we detailed how to implement native authentication using DataStax Enterprise (DSE). In this edition, we detail how to implement SSL between DSE server nodes (in the event you go multi-cloud), and then also SSL from client (node) to DSE cluster.*

Software versions

The primary DataStax software component used in this edition of DDN is DataStax Enterprise (DSE), currently release 6.8 EAP (Early Access Program). All of the steps outlined below can be run on one laptop with 16 GB of RAM, or if you prefer, run these steps on Amazon Web Services (AWS), Microsoft Azure, or similar, to allow yourself a bit more resource.

For isolation and (simplicity), we develop and test all systems inside virtual machines using a hypervisor (Oracle Virtual Box, VMWare Fusion version 8.5, or similar). The guest operating system we use is Ubuntu Desktop version 18.04, 64 bit.

40.1 Terms and core concepts

As stated above, ultimately the end goal is to secure a development server that is operating in the public cloud. A few comments:

- In the March/2019 edition of this document, we detailed how to implement passwords, authentication and authorization, using DataStax Enterprise (DSE) native subsystems.
- In this document, we detail how to enable DSE node to DSE node SSL using a self-signed certificate, and how to enable SSL from DSE cluster to a client (node).
- Effectively we center this detail on the DSE Core functional component, and do not discuss additional server side security, like encryption additional (data) files used by DSE Search, DSE Analytics, or DSE Graph.

  Basically, we are treating this as a development server.

Prerequisites/assumptions to the material that follows

- You have a 2 node version 6.8 EAP DSE cluster operating, and can find and modify the cassandra.yaml file. Also, that you can successfully restart each of the participating DSE nodes.
- The steps outlined in this document are for pre-production systems only. To save steps, we concurrently reboot all DSE server nodes to enact the necessary configuration file change that support SSL. This document does not discuss minimizing any outages that might be preferred when performing these steps on production type systems.
- Your DSE server hostname resolves via DNS.
- All work is done at the command line.

Setting up SSL, Just Operating System Steps

Example 40-1lists the 20 or so steps needed to prepare the operating system to support SSL. A code review follows.

Example 40-1 Operating systems steps

```
cd /opt
mkdir ca
cd ca
```



```
vi rootCa_cert.conf
   # rootCa_cert.conf
   [ req ]
   distinguished_name = TestSSL
   prompt = no
   output_password = pswd2020
   default_bits = 2048
   [ TestSSL ]
   C = US
   O = TestOrg
   OU = TestCluster
   CN = rootCa

which openssl

openssl req -config rootCa_cert.conf \
   -new -x509 -nodes \
   -subj /CN=rootCa/OU=TestCluster/O=TestOrg/C=US/ \
   -keyout rootCa.key \
   -out rootCa.crt \
   -days 3650

>> Generating a 2048 bit RSA private key
>> ..........................................+++++
>>
.............................................................
.....................+++++
>> writing new private key to 'rootCa.key'
>> -----


ls -l
>> total 12
>> -rw-r--r-- 1 root root  175 Jan 27 11:48 rootCa_cert.conf
```

```
>> -rw-r--r-- 1 root root 1111 Jan 27 11:49 rootCa.crt
>> -rw------- 1 root root 1700 Jan 27 11:49 rootCa.key

# validate
openssl x509 -in rootCa.crt -text -noout

>> Certificate:
>>     Data:
>>         Version: 1 (0x0)
>>         Serial Number:
>>             f1:8d:0a:fe:bb:eb:09:ff
>>         Signature Algorithm: sha256WithRSAEncryption
>>         Issuer: CN = rootCa, OU = TestCluster, O = TestOrg, C
= US
>>         Validity
>>             Not Before: Jan 27 19:49:44 2020 GMT
>>             Not After : Jan 24 19:49:44 2030 GMT
>>         Subject: CN = rootCa, OU = TestCluster, O = TestOrg, C
= US
>>         Subject Public Key Info:
>>             Public Key Algorithm: rsaEncryption
>>                 RSA Public-Key: (2048 bit)
>>                 Modulus:
>>
00:bc:d8:02:ef:50:5d:7b:fe:6a:ad:fb:9a:3e:32:
>>
c0:42:51:04:a4:ef:25:f7:29:45:af:ce:8f:48:c1:
>>
75:c5:1a:91:48:ee:d2:ed:74:bd:1c:2d:76:2f:e3:
>>
ea:b8:cc:96:69:2a:71:2a:cb:51:ca:28:16:72:16:
>>
89:7e:2e:ee:17:1d:83:53:b9:2e:e0:bb:57:5d:4c:
>>
```



```
06:3b:bc:7c:24:de:43:8b:c1:c9:1a:cc:2b:92:63:
>>
eb:18:ab:0b:11:37:d8:b0:b1:7b:92:4f:5d:85:c7:
>>
26:8e:98:4b:98:2d:54:e6:47:6c:cc:35:7d:57:49:
>>
aa:cd:e7:28:16:f7:5d:3c:37:23:b2:cd:2e:61:19:
>>
af:6d:66:f9:4b:6b:24:ee:b7:41:b8:42:33:bc:3d:
>>
6b:86:80:a9:59:01:f0:4b:f6:4d:b4:e5:aa:03:50:
>>
6b:d4:9f:74:af:67:e7:62:f6:d3:0c:02:81:68:6a:
>>
b5:a5:78:79:14:65:65:e1:9c:e9:56:1a:b5:c5:73:
>>
7b:bb:ae:ca:f5:30:16:47:b2:81:83:74:1f:d1:e3:
>>
ac:85:a8:41:f7:55:2d:87:fa:e1:4a:a7:dd:d1:62:
>>
b6:85:52:67:7d:51:51:19:82:43:fd:3d:83:65:b8:
>>
d3:f1:c1:88:62:9b:2b:94:9c:b2:fc:2c:8b:b8:38:
>>                     01:19
>>                 Exponent: 65537 (0x10001)
>>     Signature Algorithm: sha256WithRSAEncryption
>>
01:72:de:af:9d:ba:67:b9:6a:57:11:95:ec:8e:01:12:67:32:
>>
1a:32:57:2c:6a:c6:ca:1c:ac:39:3f:8a:6e:1d:92:7f:a6:3a:
>>
19:61:25:61:2a:18:15:49:b7:b8:05:dc:c8:97:b1:53:c3:d9:
>>
56:9a:74:0b:58:31:dc:3d:81:b7:85:7d:a2:ab:6b:f0:70:eb:
```

```
>>
aa:92:a2:d8:9d:d2:80:ef:05:89:80:66:e2:09:ae:11:e1:61:
>>
68:ec:f0:e3:f2:a2:41:7e:a8:0c:36:df:c9:85:a9:a7:e5:dd:
>>
67:b9:8b:ae:cd:1e:84:b9:14:4c:d0:e0:cd:b1:e0:f1:b3:2f:
>>
d6:37:af:51:de:fc:a9:e8:1e:87:ee:a2:8b:8a:6b:7a:94:bb:
>>
69:6a:2d:e6:5a:1b:cd:d9:48:88:63:e8:47:9c:94:19:4c:0a:
>>
27:1d:4a:7c:ac:9f:9e:23:3c:52:db:8f:4e:a8:41:ff:8b:ad:
>>
e4:5e:40:31:19:94:7a:c7:1c:62:4c:2c:21:7b:b3:6c:74:15:
>>
33:57:bb:69:10:6e:37:9a:29:be:a2:c7:ba:32:fc:7e:a4:38:
>>
8f:da:34:31:fa:f3:23:b2:77:99:bd:a3:00:d2:81:9c:03:41:
>>
b0:bf:ae:fa:71:72:a1:f6:47:c0:08:82:c0:32:91:b6:9e:6f:
>>          15:ee:c1:39


keytool -keystore dse-truststore.jks \
   -alias rootCa -importcert -file './rootCa.crt' \
   -keypass pswd2020 \
   -storepass pswd2020 \
   -noprompt

>> Certificate was added to keystore


      -----------------
```



```
cd /opt
mkdir keystores
cd keystores


keytool -genkeypair -keyalg RSA \
   -alias node \
   -keystore keystore.jks \
   -keypass pswd2020 \
   -storepass pswd2020 \
   -validity 3650 \
   -keysize 2048 \
   -dname "CN=test, OU=TestCluster, O=org_name, C=US"

>> # ls -l
>> total 4
>> -rw-r--r-- 1 root root 2162 Jan 27 14:20 keystore.jks


keytool -list -keystore keystore.jks -storepass pswd2020

>> Keystore type: jks
>> Keystore provider: SUN
>>
>> Your keystore contains 1 entry
>>
>> node, Jan 27, 2020, PrivateKeyEntry,
>> Certificate fingerprint (SHA1):
29:38:B8:C9:87:D7:B9:7D:75:23:6D:1E:BB:E9:3C:B5:8F:AE:41:D4
>>
>> Warning:
>> The JKS keystore uses a proprietary format. It is recommended
to migrate to \
```

```
>>    PKCS12 which is an industry standard format using "keytool
-importkeystore \
>>    -srckeystore keystore.jks -destkeystore keystore.jks
-deststoretype pkcs12".


keytool -keystore keystore.jks \
   -alias node \
   -certreq -file signing_request.csr \
   -keypass pswd2020 \
   -storepass pswd2020 \
   -dname "CN=test, OU=cluster_name, O=org_name, C=US"

>> Warning:
>> The JKS keystore uses a proprietary format. It is recommended
to migrate \
>>    to PKCS12 which is an industry standard format using
"keytool -importkeystore \
>>    -srckeystore keystore.jks -destkeystore keystore.jks
-deststoretype pkcs12".


openssl x509 -req -CA '/opt/ca/rootCa.crt' \
   -CAkey '/opt/ca/rootCa.key' \
   -in signing_request.csr \
   -out node0.crt_signed \
   -days 3650 \
   -CAcreateserial \
   -passin pass:pswd2020

>> Signature ok
>> subject=C = US, O = org_name, OU = cluster_name, CN = test
>> Getting CA Private Key
```

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

>> node0.crt_signed: OK


keytool -keystore keystore.jks \
   -alias rootCa \
   -importcert -file '/opt/ca/rootCa.crt' \
   -noprompt  -keypass pswd2020 \
   -storepass pswd2020

>> Certificate was added to keystore
>>
>> Warning:
>> The JKS keystore uses a proprietary format. It is recommended
to \
>>   migrate to PKCS12 which is an industry standard format using
\
>>    "keytool -importkeystore -srckeystore keystore.jks
-destkeystore \
>>    keystore.jks -deststoretype pkcs12".


keytool -keystore keystore.jks \
   -alias node \
   -importcert -noprompt \
   -file node0.crt_signed \
   -keypass pswd2020 \
   -storepass pswd2020

>> Certificate reply was installed in keystore
>>
>> Warning:
```



```
>> The JKS keystore uses a proprietary format. It is recommended
to \
>>   migrate to PKCS12 which is an industry standard format using
\
>>    "keytool -importkeystore -srckeystore keystore.jks
-destkeystore \
>>     keystore.jks -deststoretype pkcs12".


>> # ls -l
>> total 12
>> -rw-r--r-- 1 root root 3731 Jan 27 14:28 keystore.jks
>> -rw-r--r-- 1 root root 1111 Jan 27 14:25 node0.crt_signed
>> -rw-r--r-- 1 root root 1047 Jan 27 14:23 signing_request.csr


cd /opt
mkdir ssl
cd ssl

cp /opt/keystores/keystore.jks .
cp /opt/ca/dse-truststore.jks .

cd /opt

chmod -R 400 ssl


>> # ls -l
>> total 24
>> drwxr-xr-x 2 root root 4096 Jan 27 14:37 ca
>> drwxr-xr-x 8 root root 4096 Jan 27 13:45 dse
>> drwxr-xr-x 3 root root 4096 Dec 25 14:43 google
>> drwxr-xr-x 2 root root 4096 Jan 27 14:37 keystores
```

```
>> dr-------- 2 root root 4096 Jan 27 14:37 ssl
>> drwxrwxrwx 3 root root 4096 Dec 25 14:36 vmware_tools
```

Relative to Example 40-1, the following is offered:

- Lines starting with pound symbols ("#"), are comments. Lines starting with two right angle brackets (">>") represent output. Neither of these types of lines need to be executed.

  All work is performed as the root user.

- We make 3 directories total, "ca", "keystores", and "ssl", all under /opt.

  Each of these contain different types of content.

- First we create a 'cert' file using vi(C), and entering the contents as shown.
- Using openssl, and keytool, we first generate the 'cert' file, then the key file. Using keytool, we import to the Linux keystore.
- All files are set to Linux permission, 400.
- At least 4 validation steps are listed above. There really isn't much that can go wrong here.
- Repeat the above steps on all DSE server node boxes. This work can be performed while DSE is operating.

Setting up SSL, Just DSE Steps

Example 40-2 displays additional steps related to just DataStax Enterprise (DSE). A code review follows.

Example 40-2 DSE system steps

```
server_encryption_options:
    internode_encryption: all
    keystore_type: JKS
    keystore: /opt/ssl/keystore.jks
    keystore_password: pswd2020
    require_client_auth: false
    require_endpoint_verification: false
    truststore_type: JKS
    truststore: /opt/ssl/dse-truststore.jks
    truststore_password: pswd2020
    protocol: TLS

client_encryption_options:
```



```
enabled: true
algorithm: SunX509
optional: false
keystore_type: JKS
keystore: /opt/ssl/keystore.jks
keystore_password: pswd2020
truststore_type: JKS
truststore: /opt/ssl/dse-truststore.jks
truststore_password: pswd2020
protocol: TLS
```

Relative to Example 40-2, the following is offered:

- All of these settings are made by editing the cassandra.yaml file. All of these changes can be made while DataStax Enterprise (DSE) is operating. Some of these changes involve un-commenting a setting, some involve changing the value to a setting.
- Edit the cassandra.yaml on each DSE server node, then reboot all nodes concurrently.

Connecting using DSE CQLSH

While there are other means to get CQLSH operating on a stand alone (client) node, we often just copy the entire DataStax Enterprise (DSE) server distribution to wherever we need it, and access its program binaries.

First we need to copy one of the rootCa.crt files from an earlier step to this client (node). We copied ours to, /opt/ssl/rootCa.crt

We need to edit (or create) the cqlshrc file in the user's HOME directory, and under a relative subdirectory titled, .cassandra, a hidden directory. Contents as shown in Example 40-3.

Example 40-3 Contents of $HOME/.cassandra/cqlshrc

```
[authentication]
username = root
password = password

[connection]
hostname = 172.16.2.183
port = 9042

[ssl]
```



```
certfile = /opt/ssl/rootCa.crt
validate = false
```

Above, we programmed to operate as the root user, with a password equal to, password.

To test all of the above, attempt to run DSE CQLSH via a,

```
cqlsh -ssl
```

40.2 Complete the following

Implement the changes above using the steps provided. Test for encryption on the line using a packet analyzer. (Instructions for this work not provided here.)

Also, implement the authentication/authorization steps, and test same, from the March/2019 edition of this document.

40.3 In this document, we reviewed or created:

This month and in this document we detailed the following:

- How to enable DataStax Enterprise (DSE) node to node SSL.
- How to enable DSE server node to client (code) SSL.

Persons who help this month.

Kiyu Gabriel, Dave Bechberger, and Jim Hatcher.

Additional resources:

Free DataStax Enterprise training courses,

```
https://academy.datastax.com/courses/
```

Take any class, any time, for free. If you complete every class on DataStax Academy, you will actually have achieved a pretty good mastery of DataStax Enterprise, Apache Spark, Apache Solr, Apache TinkerPop, and even some programming.

This document is located here,

```
https://github.com/farrell0/DataStax-Developers-Notebook
```

```
https://tinyurl.com/ddn3000
```

