

# mod17\_ex03: Creating Local Keystore

The purpose of this exercise is to create a host keystore.

## Reference Information

The following documents provide information related to this exercise.

- [TBD](#)

## 1. Introducing Java Keystore

### 1.1 Defining Java Keystore

When a large enterprise embraces TLS one immediate result will be the requirement to manage an increasing number of private keys and public certificates. Such a volume of files will become a management challenge. The Java community create a solution named Java Keystores. As the name implies the solution is intended to store many keys in a single binary file, called a keystore. This reduces the administration of keys and certificates.

The keystore is a binary file that holds keys. It is commonly used to hold both private keys and public keys. It will hold keys from a variety of formats, the two most common being pem and pkcs. The keys held within the keystore are used to validate other hosts and services for the node. They are used during the handshake establishing an encrypted channel for communications.

### 1.2 Keystore Alias and Password

## Alias

The keystores will hold several keys and certificates. It identifies each element with an alias. It is important to manage these aliases as these are used to lookup the required key or certificate.

## Passwords

A keystore requires a password to gain access. Additionally, each key must have a password. Requesting a key requires passing in two passwords.

### 1.3 The difference between keystores and truststores.

Java uses two methods to distinguish between the types of keystores. These are:

- `javax.net.ssl.keyStore`
- `javax.net.ssl.trustStore`

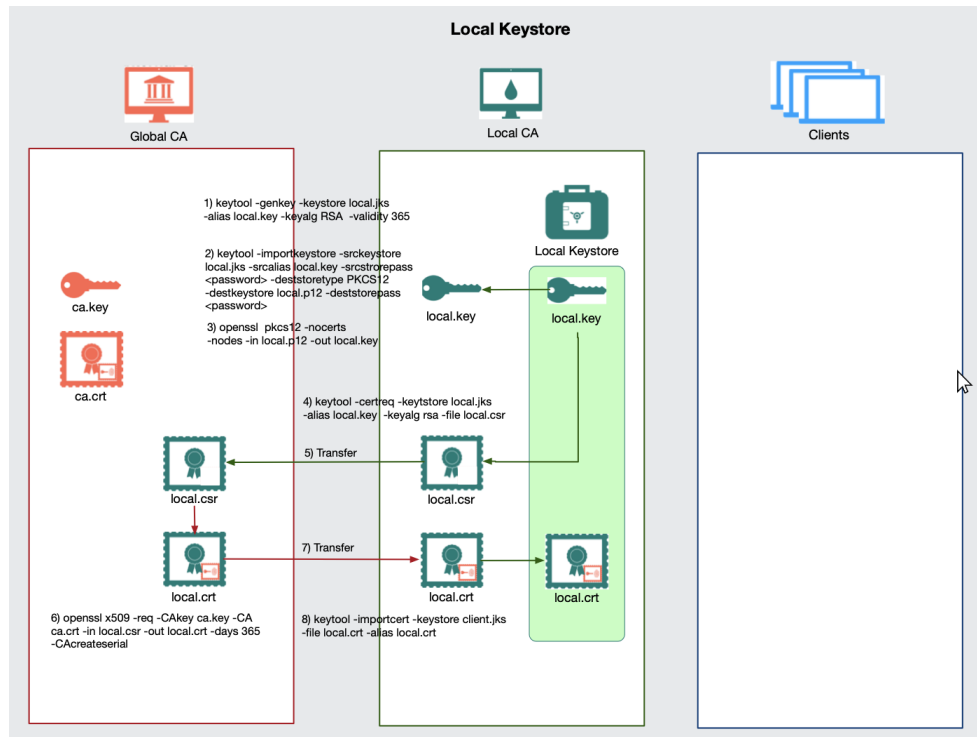
They are used to specify which keystores to use, for two different purposes.

- **KeyManager:** Determines which authentication credentials to send to the remote host.
- **TrustManager:** Determines whether the remote authentication credentials (and thus the connection) should be trusted.

### 1.4 Java keytool

The Java keytool is the command to create and manage Java keystores.

## 1.5 Overview creating Java keystores



1. The keytool command will create the local keystore with a private key.
  2. The keytool command does not have a function for exporting the private key. The practice is to change the keystore format to PKCS12.
  3. The openssl command will convert the PKCS12 private key into the PEM format.
  4. The keytool command will create a certificate signing request (.csr).
  5. Transfer the .csr file to the CA.
  6. The CA uses the openssl command to sign the csr creating the certificate file.
  7. Transfer the .crt file to local.
  8. The keytool command imports the signed certificate file into the local keystore.
2. Open a Mate terminal as the user training.

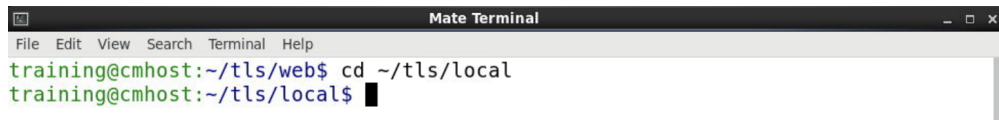
```

Mate Terminal
File Edit View Search Terminal Help
training@cmhost:~$

```

3. Create the local keystore.

### 3.1 Change directory to local.



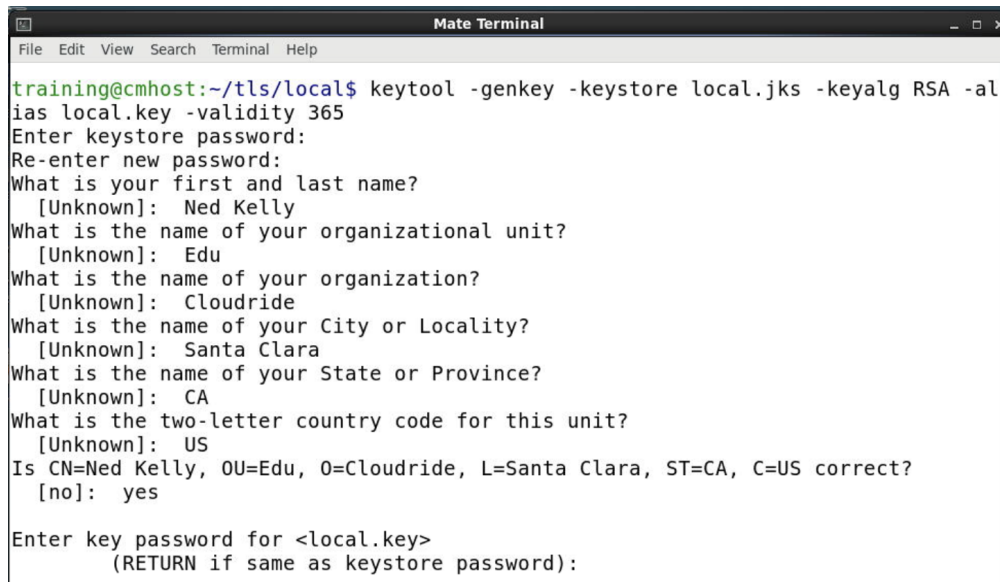
```

training@cmhost:~/tls/web$ cd ~/tls/local
training@cmhost:~/tls/local$

```

```
$ cd ~/tls/local
```

### 3.2 Create the local keystore with only a private key. Use password <password>. Use the information provided.



```

training@cmhost:~/tls/local$ keytool -genkey -keystore local.jks -keyalg RSA -alias local.key -validity 365
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: Ned Kelly
What is the name of your organizational unit?
[Unknown]: Edu
What is the name of your organization?
[Unknown]: Cloudride
What is the name of your City or Locality?
[Unknown]: Santa Clara
What is the name of your State or Province?
[Unknown]: CA
What is the two-letter country code for this unit?
[Unknown]: US
Is CN=Ned Kelly, OU=Edu, O=Cloudride, L=Santa Clara, ST=CA, C=US correct?
[no]: yes

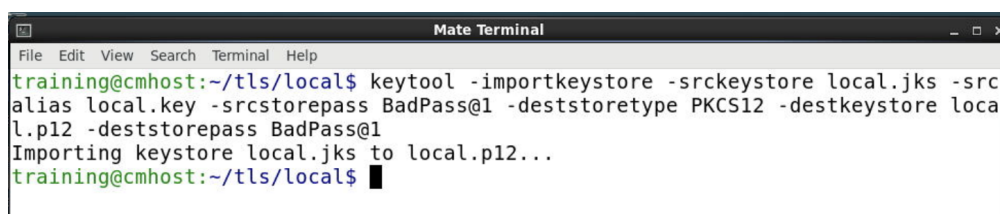
Enter key password for <local.key>
(RETURN if same as keystore password):

```

```
$ keytool -genkey -keystore local.jks -alias local.key -keyalg RSA -validity 365
```

## 4. Export the private key from the local keystore.

### 4.1 Convert the keystore into a PKCS12 format.



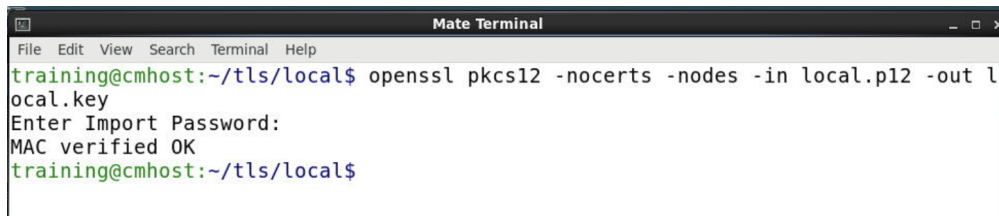
```

training@cmhost:~/tls/local$ keytool -importkeystore -srckeystore local.jks -srcalias local.key -srcstorepass BadPass@1 -deststoretype PKCS12 -destkeystore local.p12 -deststorepass BadPass@1
Importing keystore local.jks to local.p12...
training@cmhost:~/tls/local$

```

```
$ keytool -importkeystore -srckeystore local.jks -srcalias local.key -srcstorepass <password> -deststoretype PKCS12 -destkeystore local.p12 -deststorepass <password>
```

## 4.2 Convert PKCS12 format to PEM format.

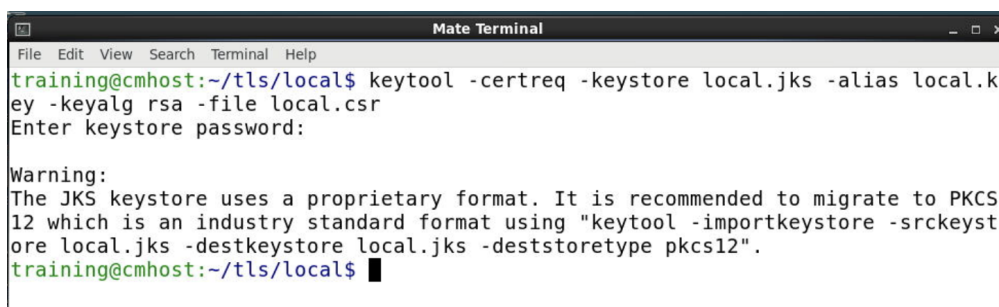


```
Mate Terminal
File Edit View Search Terminal Help
training@cmhost:~/tls/local$ openssl pkcs12 -nocerts -nodes -in local.p12 -out local.key
Enter Import Password:
MAC verified OK
training@cmhost:~/tls/local$
```

```
$ openssl pkcs12 -nocerts -nodes -in local.p12 -out local.key
```

## 5. Create a Certificate Signing Request.

### 5.1 Use the keytool to create a signing request.



```
Mate Terminal
File Edit View Search Terminal Help
training@cmhost:~/tls/local$ keytool -certreq -keystore local.jks -alias local.key -keyalg rsa -file local.csr
Enter keystore password:

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 local.jks -destkeystore local.jks -deststoretype pkcs12".
training@cmhost:~/tls/local$
```

```
$ keytool -certreq -keystore local.jks -alias local.key -keyalg rsa -file local.csr
```

### 5.2 Transfer the signing request to the CA.



```
Mate Terminal
File Edit View Search Terminal Help
training@cmhost:~/tls/local$ ls
global.jks global.p12 global.pem local.csr local.jks local.key local.p12
training@cmhost:~/tls/local$ cp local.csr ~/tls/ca/local.csr
training@cmhost:~/tls/local$
```

```
$ cp local.csr ~/tls/ca/local.csr
```

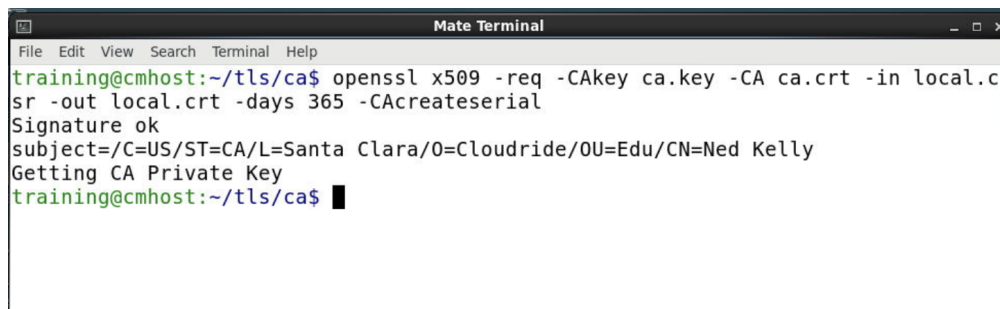
## 6. Sign a Certificate Signing Request.

## 6.1 Change directory to ca.

A terminal window titled 'Mate Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'training@cmhost:~/tls/local\$'. The command 'cd ~/tls/ca' is entered and executed, resulting in a new prompt 'training@cmhost:~/tls/ca\$'.

```
$ cd ~/tls/ca
```

## 6.2 Use the openssl command and the CA private key and CA public certificate to sign the certificate signing request.

A terminal window titled 'Mate Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'training@cmhost:~/tls/ca\$'. The command 'openssl x509 -req -CAkey ca.key -CA ca.crt -in local.csr -out local.crt -days 365 -CAcreateserial' is entered and executed. The output shows 'Signature ok', the subject information 'subject=/C=US/ST=CA/L=Santa Clara/O=Cloudride/OU=Edu/CN=Ned Kelly', and 'Getting CA Private Key'. The prompt returns to 'training@cmhost:~/tls/ca\$'.

```
$ openssl x509 -req -CAkey ca.key -CA ca.crt -in local.csr -out local.crt -days 365 -CAcreateserial
```

## 6.3 Send the signed certificates back to the client.

A terminal window titled 'Mate Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'training@cmhost:~/tls/ca\$'. The command 'ls' is entered, showing the output: 'ca.crt ca.key ca.srl local.crt local.csr web.crt web.csr'. Then, the command 'cp local.crt ~/tls/local/local.crt' is entered and executed. The prompt returns to 'training@cmhost:~/tls/ca\$'.

```
$ cp local.crt ~/tls/local/local.crt
```

## 7. Verify the signed certificate.

## 7.1 Review the contents of the keystore. Compare alias local to local.crt.

```
Mate Terminal
File Edit View Search Terminal Help
Alias name: local.key
Creation date: Feb 27, 2022
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Ned Kelly, OU=Edu, O=Cloudride, L=Santa Clara, ST=CA, C=US
Issuer: CN=Ned Kelly, OU=Edu, O=Cloudride, L=Santa Clara, ST=CA, C=US
Serial number: 2d63eeeb
Valid from: Sun Feb 27 11:12:16 PST 2022 until: Mon Feb 27 11:12:16 PST 2023
Certificate fingerprints:
    MD5:  D3:7D:BE:BC:F8:65:3E:19:25:30:E1:AF:33:F8:8B:2F
    SHA1: 3A:93:0F:5B:D6:1F:7C:F5:8A:D8:0E:D9:08:DB:C2:BE:88:41:57:D1
    SHA256: D8:D6:9B:42:0E:37:AE:3C:7B:5E:75:8E:B9:F2:35:73:17:C9:E9:37:5A:
E6:94:29:1D:F7:3F:AE:D1:85:11:1A
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
```

```
% keytool -list -v -keystore local.jks | less
```

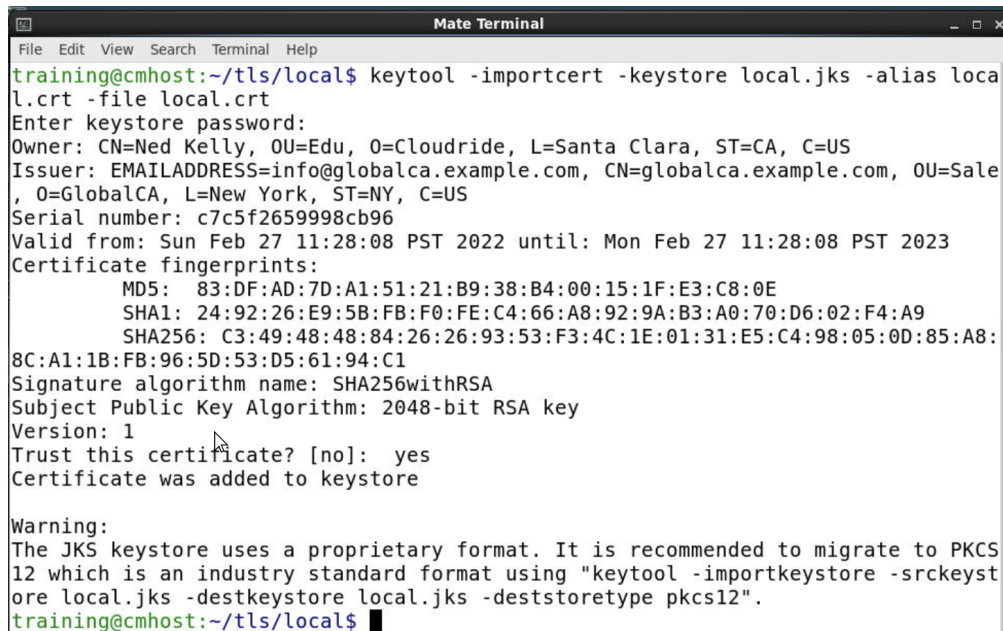
## 8. Import the signed certificate into the keystore.

### 8.1 Change directories to local.

```
Mate Terminal
File Edit View Search Terminal Help
training@cmhost:~/tls/ca$ cd ~/tls/local
training@cmhost:~/tls/local$
```

```
$ cd ~/tls/local
```

## 8.2 Import the signed certificate in the local keystore.



```
Mate Terminal
File Edit View Search Terminal Help
training@cmhost:~/tls/locals$ keytool -importcert -keystore local.jks -alias local.crt -file local.crt
Enter keystore password:
Owner: CN=Ned Kelly, OU=Edu, O=Cloudride, L=Santa Clara, ST=CA, C=US
Issuer: EMAILADDRESS=info@globalca.example.com, CN=globalca.example.com, OU=Sale, O=GlobalCA, L=New York, ST=NY, C=US
Serial number: c7c5f2659998cb96
Valid from: Sun Feb 27 11:28:08 PST 2022 until: Mon Feb 27 11:28:08 PST 2023
Certificate fingerprints:
    MD5: 83:DF:AD:7D:A1:51:21:B9:38:B4:00:15:1F:E3:C8:0E
    SHA1: 24:92:26:E9:5B:FB:F0:FE:C4:66:A8:92:9A:B3:A0:70:D6:02:F4:A9
    SHA256: C3:49:48:48:84:26:26:93:53:F3:4C:1E:01:31:E5:C4:98:05:0D:85:A8:8C:A1:1B:FB:96:5D:53:D5:61:94:C1
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 1
Trust this certificate? [no]: yes
Certificate was added to keystore

Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS 12 which is an industry standard format using "keytool -importkeystore -srckeystore local.jks -destkeystore local.jks -deststoretype pkcs12".
training@cmhost:~/tls/locals$
```

```
$ keytool -importcert -keystore local.jks -alias local.crt -file local.crt
```