



Apache HTTP Server

nShield® HSM Integration Guide - PKCS #11

2024-02-19

Table of Contents

1. Introduction	1
1.1. Product configurations	1
1.2. Requirements	2
1.3. More information	3
2. Procedures	4
2.1. Installing and configuring the Apache HTTP server	4
2.2. Install the HSM	5
2.3. Install the Security World software and create the Security World	5
2.4. Set up the PKCS11 engine	6
2.5. Configure the Apache HTTP Server to use the PKCS #11 engine	9
2.6. Test the PKCS #11 integration with the Apache HTTP Server and the HSM	10
3. Apache nShield on Kubernetes	18
3.1. Prerequisites	18
3.2. Generate the key and certificate and save Security World files	19
3.3. Install nCOP 1.1.1	20
3.4. Build the nCOP containers	21
3.5. Build the Apache container image	22
3.6. Build the test image container	23
3.7. Test the configuration locally	24
3.8. Register the Apache and nCOP containers to an external registry	28
3.9. Deploy on Kubernetes	29
3.10. Deploy the application	32
4. Additional resources and related products	38
4.1. nShield Connect	38
4.2. nShield as a Service	38
4.3. Entrust digital security solutions	38
4.4. nShield product documentation	38

Chapter 1. Introduction

This guide describes how to integrate an nShield HSM with the Apache HTTP Server using `mod_ssl` to serve HTTPS websites. The integration process uses the Public-Key Cryptography Standards (PKCS #11) interface. The HSM stores the Apache Server's SSL private key within its FIPS 140 Level 3 validated hardware. After the integration is complete, you can deploy this integration in a Kubernetes environment.

Throughout this guide, the term HSM refers to nShield Solo and nShield Connect units.

1.1. Product configurations

Entrust has successfully tested nShield HSM integration with the Apache server in the following configurations:

Product	Version
Operating System	Red Hat Enterprise Linux 8 X86-64
Apache version	2.4.37
OpenSSL version	openssl-1:1.1.1k-7
OpenSSL PKCS #11 version	openssl-pkcs11-0.4.10-2
RedHat Openshift (for Kubernetes Integration only)	4.11.20
nCOP (for Kubernetes Integration only)	1.1.1

1.1.1. Supported nShield features

Entrust has successfully tested nShield HSM integration with the following features:

Feature	Support
Softcards	Yes

Feature	Support
Module-only key	Yes
OCS cards	Yes
nSaaS	Not tested

1.1.2. Supported nShield hardware and software versions

Entrust has successfully tested with the following nShield hardware and software versions:

1.1.2.1. Connect XC

Security World Software	Firmware	Image	OCS	Softcard	Module	FIPS Level 3
12.80.4 & 13.4.4	12.50.11 (FIPS Certified)	12.80.4	✓	✓	✓	
12.80.4 & 13.4.4	12.72.1 (FIPS Certified)	12.80.5	✓	✓	✓	✓

1.1.2.2. nShield 5

Security World Software	Firmware	Image	OCS	Softcard	Module	FIPS Level 3
13.2.2	13.2.2 (FIPS Pending)	13.2.2	✓	✓	✓	
13.4.4	13.2.2 (FIPS Pending)	13.3.2	✓	✓	✓	

1.2. Requirements

Ensure that you have supported versions of the nShield, Apache, and third-party

products. See [Product configurations](#).

Consult the security team in your organization for a suitable setting of the SE Linux policy to allow the web server read access to the files in `/opt/nfast`.

To perform the integration tasks, you must have:

- `root` access on the operating system.
- Access to `nfast` and `httpd` accounts.

Before starting the integration process, familiarize yourself with:

- The documentation for the HSM.
- The documentation and setup process for the Apache HTTP Server.

Before using the nShield software, you need to know:

- The number and quorum of Administrator Cards in the Administrator Card Set (ACS), and the policy for managing these cards.
- Whether the application keys are protected by the module, an Operator Card Set (OCS) or a Softcard with or without a pass phrase.
- The number and quorum of Operator Cards in the OCS, and the policy for managing these cards.
- Whether the Security World should be compliant with FIPS 140 Level 3.



Entrust recommends that you allow only unprivileged connections unless you are performing administrative tasks.

For more information, refer to the *User Guide* and *Installation Guide* for the HSM.

1.3. More information

For more information about OS support, contact your Apache HTTP Server sales representative or Entrust nShield Support, <https://nshieldsupport.entrust.com>.



Access to the Entrust nShield Support Portal is available to customers under maintenance. To request an account, contact nshield.support@entrust.com.

Chapter 2. Procedures

Integration procedures include:

- [Installing and configuring the Apache HTTP server.](#)
- [Install the HSM.](#)
- [Install the Security World software and create the Security World.](#)
- [Set up the PKCS11 engine.](#)
- [Configure the Apache HTTP Server to use the PKCS #11 engine.](#)
- [Test the PKCS #11 integration with the Apache HTTP Server and the HSM.](#)

This chapter describes these procedures.

2.1. Installing and configuring the Apache HTTP server

To install the Apache HTTP Server on your Red Hat server:

```
% sudo yum install -y openssl-pkcs11 httpd httpd-tools openssl-libs mod_ssl openssl
```

2.1.1. Open the firewall

If the firewall is active, this might prevent Apache from loading the library. To open the firewall:

```
% sudo firewall-cmd --zone=public --permanent --add-service=http  
% sudo firewall-cmd --zone=public --permanent --add-service=https  
% sudo firewall-cmd --reload
```



Consult the security team in your organization for suitable setting of the firewall.

2.1.2. Switch off SE Linux

If SE Linux is active, this might prevent Apache from loading the library. To switch it off:

```
% sudo setenforce 0
```



Consult the security team in your organization for a suitable setting of the SE Linux policy to allow the web server read access to the files in `/opt/nfast`.

2.1.3. Restart the httpd service

```
% sudo service httpd restart
```

2.1.4. Check if Apache is running

Open a browser and access the following URL: `http://<YOUR_IP_ADDRESS>`. You should see a page similar to this:

Red Hat Enterprise Linux Test Page

This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page, it means that the Apache HTTP server installed at this site is working properly.

If you are a member of the general public:

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems, or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.


For example, if you experienced problems while visiting `www.example.com`, you should send e-mail to "webmaster@example.com".

For information on Red Hat Enterprise Linux, please visit the [Red Hat, Inc. website](#). The documentation for Red Hat Enterprise Linux is [available on the Red Hat, Inc. website](#).

If you are the website administrator:

You may now add content to the directory `/var/www/html/`. Note that until you do so, people visiting your website will see this page, and not your content. To prevent this page from ever being used, follow the instructions in the file `/etc/httpd/conf.d/welcome.conf`.

You are free to use the image below on web sites powered by the Apache HTTP Server:

Powered by  2.4

2.2. Install the HSM

Install the HSM by following the instructions in the *Installation Guide* for the HSM.

Entrust recommends that you install the HSM before configuring the Security World software with your Apache HTTP Server.

2.3. Install the Security World software and create the Security World

To install the Security World Software and create the Security World:

1. On the computer that you want to make the Apache HTTP Server, install the latest version of the Security World Software as described in the *Installation Guide* for the HSM.



Entrust recommends that you uninstall any existing nShield software before installing the new nShield software.

2. Create the Security World as described in the *User Guide*, creating the ACS and OCS that you require.

2.4. Set up the PKCS11 engine

To avoid problems associated with the Entrust supplied OpenSSL, which is used internally by `generatekey` to make certificates, ensure that `/opt/nfast/bin` is not at the front of your `$PATH`.

You can confirm that the right binary is being run with the following command:

```
% which openssl
/usr/bin/openssl
```

If this command returns anything inside `/opt/nfast`, check your `$PATH` variable.

2.4.1. Configuration

Find out where your OpenSSL configuration file is located:

```
% openssl version -d
OPENSSLDIR: "/etc/pki/tls"
```

The minimum configuration is similar to the following:

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#
# Note that you can include other files from the main configuration
# file using the .include directive.
#.include filename
# This definition stops the following lines generating an error if HOME isn't
# defined.
HOME = .
RANDFILE = $ENV::HOME/.rnd
# nShield PKCS11
openssl_conf = openssl_def
[openssl_def]
engines = engine_section
[engine_section]
```



```
pkcs11 = pkcs11_section
[pkcs11_section]
engine_id = pkcs11
dynamic_path = /usr/lib64/engines-1.1/pkcs11.so
MODULE_PATH = /opt/nfast/toolkits/pkcs11/libcknfast.so
init = 0
#!
```

The following message can appear when creating certificates:

```
unable to find 'distinguished_name' in config
problems making Certificate Request
140493626791824:error:0E06D06C:configuration file routines:NCONF_get_string:no value:conf_lib.c:324:group=req
name=distinguished_name
```

If it does, you need to add the following to your OpenSSL configuration, adjusted to your organization's values:

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no
[req_distinguished_name]
C = US
ST = FL
L = Sunrise
O = Entrust
OU = nShield
CN = www.entrust.com
[v3_req]
subjectAltName = @alt_names
[alt_names]
DNS.1 = www.entrust.com
DNS.2 = entrust.com
```

Make sure the server's hostname matches the CN in the certificate.

Create a file called `openssl.pkcs11.cnf` with the settings above, and save it where your OpenSSL configuration settings are located:

1. Create/edit the `/etc/pki/tls/openssl.pkcs11.cnf` file:

```
% sudo vi /etc/pki/tls/openssl.pkcs11.cnf
```

2. Enter the settings above and save the file.

2.4.2. Set up `/opt/nfast/cknfustrc`

The following variables may need to be added to the `/opt/nfast/cknfustrc` file. They are referenced in this guide to address specific situations, and their use will

depend on your current environment.

```
CKNFAST_DEBUG=10
CKNFAST_DEBUGFILE=/path/to/debug/file
CKNFAST_FAKE_ACCELERATOR_LOGIN=1
CKNFAST_LOADSHARING=1
```

If you omit `CKNFAST_DEBUGFILE`, log entries will be added to Apache's `error_log`.



Turn debug off in a production environment.

2.4.3. Test the configuration

You must now test the new configuration file using OpenSSL:

1. Exporting the `OPENSSL_CONF` environment variable:

```
% export OPENSSL_CONF=/etc/pki/tls/openssl.pcks11.cnf
```

2. Test the configuration:

```
% openssl engine -tt -c -v

There may be other output, but you should see this included:

(pkcs11) pkcs11 engine
[RSA, rsaEncryption, id-ecPublicKey]
[ available ]
SO_PATH, MODULE_PATH, PIN, VERBOSE, QUIET, INIT_ARGS, FORCE_LOGIN
```

2.4.3.1. Debugging notes

1. Security World permissions.

The following message can appear:

```
Unable to load module /opt/nfast/toolkits/pkcs11/libcknfast.so
```

If it does, it indicates that there is no Security World. Make sure you create a Security world first.

2. Debugging variables.

These variables can be used for debugging purpose. They can be set in `/opt/nfast/cknfastr` or as environment variables.

```
CKNFAST_DEBUG=10
CKNFAST_DEBUGFILE=/path
```

3. Missing PKCS #11 engine in the output.

If you don't see the PKCS #11 engine in the output, check the `dynamic_path` line in the `openssl.pkcs11.cnf` configuration file. This may vary on other platforms and other operating system versions.

```
dynamic_path = /usr/lib64/engines-1.1/pkcs11.so
```

2.4.3.2. List available slots

First generate and insert your OCS in the normal way:

```
% pkcs11-tool --module /opt/nfast/toolkits/pkcs11/libcknfast.so -L
Available slots:
Slot 0 (0x1d622495): 6308-03E0-D947 Rt1
  token label      : accelerator
  token manufacturer : nCipher Corp. Ltd
  token model      :
  token flags      : rng, token initialized, PIN initialized, other flags=0x200
  hardware version : 0.0
  firmware version : 12.50
  serial num       : 6308-03E0-D947
  pin min/max      : 0/256
Slot 1 (0x1d622496): 6308-03E0-D947 Rt1 slot 0
  (empty)
Slot 2 (0x1d622497): 6308-03E0-D947 Rt1 slot 2
  (token not recognized)
Slot 3 (0x1d622498): 6308-03E0-D947 Rt1 slot 3
  (empty)
```

2.5. Configure the Apache HTTP Server to use the PKCS #11 engine

You need to update the Apache start-up file to tell it to use the new Open SSL configuration file, and to pass the necessary environment variables. These environment variables allow PKCS #11 engine to work.

1. Edit the file `/usr/lib/systemd/system/httpd.service` and add the environment variable under the "Service" section:

```
[Service]
Type=notify
Environment=LANG=C
Environment="OPENSSL_CONF=/etc/pki/tls/openssl.pkcs11.cnf"
```

- Restart the daemon units:

```
% sudo systemctl daemon-reload
```

- Restart the Apache service:

```
% sudo service httpd restart
```

- Set the environment variable so that OpenSSL commands use the PKCS #11 engine:

```
% export OPENSSL_CONF=/etc/pki/tls/openssl.pkcs11.cnf
```

2.6. Test the PKCS #11 integration with the Apache HTTP Server and the HSM

This section describes the following scenarios that can be used by your organization according to the security guidelines that you follow:

- Functionality test with non-HSM keys.
- Module-only protection.
- Softcard protection.
- OCS protection.



A self-signed certificate is required for tests. In a production environment exposed to the internet, create the certificate request and sign it by the Trusted Certificate Authority.

2.6.1. Functionality test with non-HSM keys

You must now test that the Apache HTTP Server installation is operational and capable of serving https content. To do this, you must first create a software-based key and certificate.

- Create a directory to hold the keys:

```
% mkdir keys; cd keys
```

- Create a private key:

```
% openssl genrsa -engine pkcs11 2048 > pkcs11localhost.key
```

3. Create a self-signed certificate using this private key:

```
% openssl req -engine pkcs11 -new -x509 -days 365 -key pkcs11localhost.key -out pkcs11localhost.crt
```

4. Configure the Apache HTTP Server for SSL:

a. Copy the .key and .crt files:

```
% sudo cp pkcs11localhost.key /etc/pki/tls/private/.  
% sudo cp pkcs11localhost.crt /etc/pki/tls/certs/.
```

b. Edit `/etc/httpd/conf.d/ssl.conf` and change the following lines to use the new .key and .crt files:

```
SSLCertificateFile /etc/pki/tls/certs/pkcs11localhost.crt  
SSLCertificateKeyFile /etc/pki/tls/private/pkcs11localhost.key  
SSLCryptoDevice pkcs11
```

c. Restart the Apache service:

```
% sudo service httpd restart
```

5. Test the connection:

```
% openssl s_client -crLf -connect localhost:443 -CAfile pkcs11localhost.crt  
  
CONNECTED(00000003)  
depth=0 C = US, ST = FL, L = Sunrise, O = Entrust, OU = nShield, CN = www.entrust.com  
verify return:1  
---  
Certificate chain  
0 s:/C=US/ST=FL/L=Sunrise/O=Entrust/OU=nShield/CN=www.entrust.com  
i:/C=US/ST=FL/L=Sunrise/O=Entrust/OU=nShield/CN=www.entrust.com  
---  
Server certificate  
-----BEGIN CERTIFICATE-----  
MIIDUDCCAjgCCQCx9G3D5F5XITANBgkqhkiG9w0BAQsFAADBgMQswCQYDVQQGEwJV  
UzELMAkGA1UECAwCRkwxEDAOBgNVBAcMB1N1bnJpc2UxEDAoBgNVBAoMB0VudHJ1  
c3QxEDAoBgNVBAsMB25TaG1lbGQxGDAWBgNVBAMMD3d3dy51bnRydXN0LmNvbTAE  
Fw0yMTAxMTEwOTI1NDBaFw0yMjAxMTEwOTI1NDBaGoxCzAJBgNVBAYTAlVTMQsw  
CQYDVQQIDAJGTDEQMA4GA1UEBwwHU3VucmlzZTEQMA4GA1UECgwHRW50cnVzdDEQ  
MA4GA1UECwwHb1NoaWVsZDEYMBYGA1UEAwwPd3d3LmVudHJ1c3QuY29tMIIBjAN  
BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0/eL/YjuScdeGwMEJicEbMMU4LdM  
KRjj9Kp66YvjRThrdValJLcMkC5Gu+BRDg020hPr11w6z3TfUm0+Aa/BAkhUcec  
V04+6YZf6LXB3cgrkACr/5mxsthTkU218tT2TD6RlM0phorQqsv3T2tA37URgMYm  
h5G6wRuBWrOCbKjncjEz475NYw0I6oet197+gZKerKz0YjRvZ2p3IGcV6bSI/u7  
r/T10XF9q8sc04kzNd4ssAOMzlkwmN7XKMeayatDSD6HxNAGXMyCaDW52PTLtvcy  
rI0nI fufx0db/322DkUB6LDjQQkq5v9smE9Hw7CeIt0CadJj dkePmiAVwIDAQAB  
MA0GCsGSIb3DQEBCwUAA4IBAQBgbzge0Gekt8xyjAn4v+0nuVm5k2CfQGN0Fl+m  
w1KeqsVAYrFwBNjRf7ce0fZTQrn1o44e0tTlMfLLf0yiQ02yYky6uJ28Lnv2Ju3A  
nZj08FnyN/AoaPFqAqAJbkffSesLdw8Qf1d9Wcr5mbJcD0iQQK0z/xHE1a9saf5X
```

```

7HfP1eN0no10yjEs03hgmaTPzB9EK26QpQPzZD99Z0a309qbcRboTIUYUF725c1
V1gzCxLABK/k6ahquEPKNxqpb0B+ccrfIhgVMe0x6HsAPi3/epAGj8HQybrG1C6q
A7ghHyWDR8pVrLmuuJmNtilN5woybmD13mqFBjZ3MbGy5zeK
-----END CERTIFICATE-----
subject=/C=US/ST=FL/L=Sunrise/O=Entrust/OU=nShield/CN=www.entrust.com
issuer=/C=US/ST=FL/L=Sunrise/O=Entrust/OU=nShield/CN=www.entrust.com
---
No client certificate CA names sent
Peer signing digest: SHA512
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 1543 bytes and written 415 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol : TLSv1.2
    Cipher   : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: 381AF3AA705C0769B1E8118A3268621E1AD86176CFEE339D5CE966892D84EF66
    Session-ID-ctx:
    Master-Key:
5B787BFAC32C85916C85FB57BBE6DC43E36C7D6E4C6F095DD84F07DB642A8A3FE56F9FD789726FE79D9382D647862246
    Key-Arg  : None
    Krb5 Principal: None
    PSK identity: None
    PSK identity hint: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
0000 - 3f 75 0d eb 59 32 90 56-17 6a 86 a5 28 2d ed 9b   ?u..Y2.V.j..(-..
0010 - a8 a1 d7 2b ae 6d 4e d8-8d f9 7f d5 15 8c 2b d0   ...+mN.....+.
0020 - e0 9d d7 fc bb 61 4e 1f-63 61 41 03 5a 98 bf ed   ....aN.caA.Z...
0030 - 50 b1 d0 36 30 a5 48 6c-d2 da 1a df 34 a5 57 02   P..60.Hl....4.W.
0040 - d3 43 49 b5 27 dc 53 90-23 9e 1c 0d c2 54 d5 d3   .CI.'.S.#....T..
0050 - e4 ca 32 e4 33 da 28 70-d3 1d 34 57 b1 10 c2 28   ..2.3.(p..4W...(
0060 - 63 c5 5c 9a 9c ef d9 17-9c 72 65 41 35 eb 98 15   c.\.....reA5...
0070 - 2c 73 8a b1 99 4d 73 8f-7a b6 9b 4f 17 3c 91 4e   ,s...Ms.z..O.<.N
0080 - 71 ab ad f8 ad a9 99 4d-fd 7b f2 db 12 1a ea 28   q.....M.{.....(
0090 - e2 1d b8 fe 67 9c 36 a3-e3 6a 81 5f 3c a1 a8 a6   ....g.6..j.<...
00a0 - a9 7d bc 4c d6 a0 51 22-6d 51 87 70 bb b1 e0 4b   }.L..Q"mQ.p...K
00b0 - 95 94 60 0b e9 2e 49 8e-2d e5 4c 66 de 8f cc 1e   ..`.I.-.Lf....

Start Time: 1610394029
Timeout    : 300 (sec)
Verify return code: 0 (ok)
---
closed

```

6. Check the following messages and fields in the output:

- CONNECTED(00000003)
- depth
- Certificate chain information
- Server certificate information
- Session-ID
- Master-Key

- TLS session ticket:
- Verify return code: 0 (ok)

2.6.2. Module protection

1. Create a key:

```
% generatekey -b -g -m1 pkcs11 plainname=modulersa type=rsa protect=module size=2048

key generation parameters:
operation      Operation to perform      generate
application    Application                pkcs11
protect        Protected by              module
verify         Verify security of key    yes
type           Key type                  rsa
size           Key size                  2048
pubexp         Public exponent for RSA key (hex)
plainname      Key name                  modulersa
nvr            Blob in NVRAM (needs ACS) no
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uae95fb7af0294b94f742b22c62812fd0f18e0cf24
```

2. Get the certificate using this key:

```
% openssl req -engine pkcs11 -x509 -out modulersa.crt -days 365 -key
"pkcs11:token=accelerator;object=modulersa" -keyform engine -subj "/CN=modulersa"
```

3. Configure the Apache HTTP Server for SSL:

a. Copy the .crt file:

```
% sudo cp modulersa.crt /etc/pki/tls/certs/.
```

b. Edit `/etc/httpd/conf.d/ssl.conf` and change the following lines to use the new .key and .crt files:

```
SSLCertificateFile /etc/pki/tls/certs/modulersa.crt
SSLCertificateKeyFile "pkcs11:object=modulersa;token=accelerator"
SSLCryptoDevice pkcs11
```

c. Restart the Apache service:

```
% sudo service httpd restart
```

4. Test the connections:

```
% openssl s_client -crLf -connect localhost:443 -CAfile modulersa.crt
```

5. Check the following messages and fields in the output:

- CONNECTED(00000003)
- depth
- Certificate chain information
- Server certificate information
- Session-ID
- Master-Key
- TLS session ticket:
- Verify return code: 0 (ok)

2.6.3. Set up Softcard protection

1. To expose Softcards, the `cknfast` library has to be in load sharing mode (`CKNFAST_LOADSHARING`).

Edit the `/opt/nfast/cknfast.rc` file, and add the following information before proceeding to set up Softcard protection:

```
CKNFAST_LOADSHARING=1
```

2. Create a Softcard:

```
% ppmk -n apachesoftcard
```



Use "123456" as the passphrase for the Softcard.

3. Create a key:

```
% generatekey -b -g -m1 pkcs11 plainname=softcardkey type=rsa protect=softcard recovery=no size=2048  
softcard=apachesoftcard
```

```
key generation parameters:  
operation      Operation to perform      generate  
application    Application                pkcs11  
protect        Protected by              softcard  
softcard       Soft card to protect key  apachesoftcard  
recovery        Key recovery              no  
verify         Verify security of key    yes  
type           Key type                  rsa  
size           Key size                  2048  
pubexp         Public exponent for RSA key (hex)  
plainname      Key name                  softcardkey  
nvram          Blob in NVRAM (needs ACS) no  
Please enter the pass phrase for softcard 'apachesoftcard':
```

```
Please wait.....  
Key successfully generated.
```

```
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ucb87f22b0df8d3b72a2f4c654ae1d3b0973b93de8-
ddd20b997d276f3304e0011fc79971344c630b0f
```

4. Get the certificate using this key:

```
% openssl req -engine pkcs11 -x509 -out softcard.crt -days 365 -key
"pkcs11:model=;token=apachesoftcard;pin-value=123456;object=softcardkey" -keyform ENGINE -subj
"/CN=softcardkey"
```

The following error can appear:

```
engine "pkcs11" set.
Specified object not found
PKCS11_get_private_key returned NULL
cannot load Private Key from engine
139939575797568:error:80067065:pkcs11 engine:ctx_load_privkey:object not found:eng_back.c:975:
139939575797568:error:26096080:engine routines:ENGINE_load_private_key:failed loading private
key:crypto/engine/eng_pkey.c:78:
```

If it does, make sure you expose the Softcards as described in this section, and run the command again.

5. Configure the Apache HTTP Server for SSL:

a. Copy the .crt file:

```
% sudo cp softcard.crt /etc/pki/tls/certs/.
```

b. Edit `/etc/httpd/conf.d/ssl.conf` and change the following lines to use the new .key and .crt files:

```
SSLCertificateFile /etc/pki/tls/certs/softcard.crt
SSLCertificateKeyFile "pkcs11:object=softcardkey;token=apachesoftcard;type=private?pin-value=123456"
SSLCryptoDevice pkcs11
```

c. Restart the Apache service:

```
% sudo service httpd restart
```

6. Test the connections:

```
% openssl s_client -crLf -connect localhost:443 -CAfile softcard.crt
```

7. Check the following messages and fields in the output:

- CONNECTED(00000003)

- depth
- Certificate chain information
- Server certificate information
- Session-ID
- Master-Key
- TLS session ticket:
- Verify return code: 0 (ok)

2.6.4. Set up OCS protection

1. Create an OCS:

```
% /opt/nfast/bin/createocs -m1 -s0 --persist -Q 1/1 -N apacheocs
```



Use "123456" as the passphrase.

2. Create a key:

```
% generatekey --cardset=apacheocs pkcs11 protect=token type=rsa size=2048 pubexp=65537 plainname=ocskey  
nvrnm=no recovery=yes
```

```
slot: Slot to read cards from? (0-3) [0] > 0
```

```
key generation parameters:
```

operation	Operation to perform	generate
application	Application	pkcs11
protect	Protected by	token
slot	Slot to read cards from	0
recovery	Key recovery	yes
verify	Verify security of key	yes
type	Key type	rsa
size	Key size	2048
pubexp	Public exponent for RSA key (hex)	65537
plainname	Key name	ocskey
nvrnm	Blob in NVRAM (needs ACS)	no

```
Loading 'apacheocs':
```

```
Module 1: 0 cards of 1 read  
Module 1 slot 0: 'apacheocs' #1  
Module 1 slot 2: Admin Card #1  
Module 1 slot 3: empty  
Module 1 slot 0:- passphrase supplied - reading card  
Card reading complete.
```

```
Key successfully generated.
```

```
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uc547fb435172da4280cc771eb3c2ad8b86ab06d0a-  
8d6a4394b07fc70148ff9c1f9960d279bf4a1d6b
```

3. Get the certificate using this key:

```
% openssl req -engine pkcs11 -x509 -out ocskey.crt -days 365 -key
```

```
"pkcs11:token=apacheocs;object=ocskey;type=private?pin-value=123456" -keyform engine -subj "/CN=ocskey"
```

4. Configure the Apache HTTP Server for SSL:

- a. Copy the .crt file:

```
% sudo cp ocskey.crt /etc/pki/tls/certs/.
```

- b. Edit `/etc/httpd/conf.d/ssl.conf` and change the following lines to use the new .key and .crt files:

```
SSLCertificateFile /etc/pki/tls/certs/ocskey.crt  
SSLCertificateKeyFile "pkcs11:object=ocskey;token=apacheocs;type=private?pin-value=123456"  
SSLCryptoDevice pkcs11
```

- c. Restart the Apache service:

```
% sudo service httpd restart
```

5. Test the connections:

```
% openssl s_client -crLf -connect localhost:443 -CAfile ocskey.crt
```

6. Check the following messages and fields in the output:

- CONNECTED(00000003)
- depth
- Certificate chain information
- Server certificate information
- Session-ID
- Master-Key
- TLS session ticket:
- Verify return code: 0 (ok)

Chapter 3. Apache nShield on Kubernetes

This section describes how to build, deploy and confirm an instance of the Apache WebServer running with PKCS #11 with a nShield HSM on Kubernetes. This guide uses Red Hat OpenShift for the Kubernetes environment of choice. This section of the guide has only been tested with the Connect XC.

3.1. Prerequisites

1. This integration procedure uses two servers:

- Server #1 will be used to build the Apache Docker image.

This server will be referred to as "the build machine". It should be the same server on which you did the Apache PKCS #11 integration.

- Server #2 is the server from which you have access to your Kubernetes environment.

This guide has been tested using a server that has Red Hat CodeReady OpenShift installed and also with a OpenShift Cluster deployed in a VMware vSphere environment. This will be used to deploy the image. This server will be referred to as the "the Kubernetes server".



It is possible to perform this procedure using a single server, but Entrust strongly suggests the use of two servers.

2. Security World software is already installed on the build machine. This was performed during the initial integration, refer to [Install the Security World software and create the Security World](#).
3. OpenSSL configured on the build machine.
4. nCOP v1.1.1 installed on the build machine with Docker.
5. Access to a Kubernetes cluster in the Kubernetes server.



Entrust strongly suggests you familiarise with the procedures in the *Red Hat OpenShift nShield HSM Integration Guide*. This chapter includes many of the same procedures from that guide, the key difference being the creation of the Apache Container.

3.2. Generate the key and certificate and save Security World files

This procedure is performed on the build machine.

1. Edit `/opt/nfast/cknfastrc` and make sure it contains the following:

```
CKNFAST_FAKE_ACCELERATOR_LOGIN=1
```



Make sure `CKNFAST_LOADSHARING` is commented out.

2. Create a **working** directory:

```
% mkdir ~/working; cd ~/working
```

3. Configure your OpenSSL configuration to point to the OpenSSL configuration file used in this integration, see [Set up the PKCS11 engine](#).

```
% export OPENSSL_CONF=/etc/pki/tls/openssl.pkcs11.cnf
```

4. Check the engine:

```
% openssl engine -tt -c -v
```

The following should be included in the output:

```
(pkcs11) pkcs11 engine
[RSA, rsaEncryption, id-ecPublicKey]
[ available ]
SO_PATH, MODULE_PATH, PIN, VERBOSE, QUIET, INIT_ARGS, FORCE_LOGIN
```

5. Generate the key, specifying module protection:

```
% /opt/nfast/bin/generatekey pkcs11 type=RSA size=2048 pubexp= plainname=apachecontainerkey nvram=no protect=module
```

```
key generation parameters:
operation      Operation to perform      generate
application    Application                pkcs11
protect        Protected by              module
verify         Verify security of key    yes
type           Key type                  RSA
size           Key size                  2048
pubexp         Public exponent for RSA key (hex)
plainname      Key name                  apachecontainerkey
nvram          Blob in NVRAM (needs ACS)  no
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uaa522cc0b4c329ab1138a1ff90b49a408e46f630d
```

6. Copy the key file from `/opt/nfast/kmdata/local` into the `kmdl` directory in your `working` directory:

```
% mkdir -p ~/working/kmdl; cd ~/work/kmdl
% cp /opt/nfast/kmdata/local/key_pkcs11_uaa522cc0b4c329ab1138a1ff90b49a408e46f630d ~/working/kmdl/.
```

7. Generate the certificate:

```
% /usr/bin/openssl req -engine pkcs11 -x509 -out certificate.pem -days 365 -key
"pkcs11:token=accelerator;object=apachecontainerkey;" -keyform ENGINE -subj "/CN=apachecontainerkey"
```

8. Copy the world and module files from `/opt/nfast/kmdata/local` into the `working/kmdl` directory:

```
% cp /opt/nfast/kmdata/local/world ~/working/kmdl/.
% cp /opt/nfast/kmdata/local/module_XXXXXX ~/working/kmdl/.
```

The `working/kmdl` directory should contain the world file, the module file, the key and the `certificate.pem` file. For example:

```
% ls -al1

-rw-rw-r--. 1 cviana cviana 1017 Dec 20 11:40 certificate.pem
-rw-rw-r--. 1 cviana cviana 10052 Dec 20 11:39 key_pkcs11_uaa522cc0b4c329ab1138a1ff90b49a408e46f630d
-rwxrwxr-x. 1 cviana cviana 5000 Dec 20 11:41 module_201E-03E0-D947
-rwxrwxr-x. 1 cviana cviana 39968 Dec 20 11:40 world
```

9. Copy `certificate.pem` to `/etc/pki/tls/certs/`:

```
% cp certificate.pem /etc/pki/tls/certs/.
```

10. Edit `/etc/httpd/conf.d/ssl.conf` and set the following lines:

```
SSLCertificateFile /etc/pki/tls/certs/certificate.pem
SSLCertificateKeyFile "pkcs11:object=apachecontainerkey;token=accelerator"
SSLCryptoDevice pkcs11
```

3.3. Install nCOP 1.1.1

The installation process involves extracting the nCOP tarball into `/opt/ncop`.

1. Make the installation directory:

```
% sudo mkdir -p /opt/ncop
```

2. Extract the tarball:

```
% sudo tar -xvf NCOPTARFILE -C /opt/ncop
```

3.4. Build the nCOP containers

This process will build nCOP containers for the hardserver and application Note that:

- This guide uses the Red Hat flavor of the container.
- Docker needs to be installed for this process to be successful.
- You will also need the Security World ISO file to be able to build nCOP.
- To configure the containers, you will need the HSM IP address, world and module files.
- In this example, version 12.80.4 of the Security World client is in use.
- The steps below are performed on the build machine.

To build the nCOP containers:

1. Before building the images, stop the local hardserver:

```
% sudo /opt/nfast/sbin/init.d-ncipher stop
```

2. Mount the Security World Software ISO file:

```
% sudo mount -t iso9660 -o loop ISOFILE.iso /mnt/iso
```

3. Build the nShield container for the hardserver and application (Red Hat):

```
% cd /opt/ncop
% sudo ./make-nshield-hwsp --from registry.access.redhat.com/ubi8/ubi --tag nshield-hwsp-pkcs11-redhat /mnt/iso
% sudo ./make-nshield-application --from registry.access.redhat.com/ubi8/ubi --tag nshield-app-pkcs11-redhat /mnt/iso
```

4. Unmount the Security World file:

```
% sudo umount /mnt/iso
```

5. Validate that the images have been built:

```
% sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nshield-app-pkcs11-redhat	latest	27cc224fe5f7	2 minutes ago	950MB
nshield-hwsp-pkcs11-redhat	latest	bfe9eafa3197	2 minutes ago	486MB
registry.access.redhat.com/ubi8/ubi	latest	cc0656847854	6 weeks ago	216MB

3.5. Build the Apache container image

The steps below are performed on the build machine.

1. Copy the required files to your **working** directory:

```
% cd ~/working
% cp /etc/pki/tls/openssl.pkcs11.cnf openssl.cnf
% cp /etc/httpd/conf.d/ssl.conf .
% cp /etc/httpd/conf/httpd.conf httpd.conf
```

The required files are:

- **openssl.cnf** is already set, based on the Apache PKCS #11 integration.
- **ssl.conf** needs to be edited with the keys and certificates generated previously. Also change the SSL port to 9443.
- **httpd.conf**.

2. Create a file in your **working** directory to build your Apache container:

```
% vi Dockerfile
```

The file should contain the following:

```
FROM nshield-app-pkcs11-redhat
RUN yum -y install httpd mod_ssl openssl-pkcs11
COPY openssl.cnf /etc/pki/tls/openssl.cnf
COPY ssl.conf /etc/httpd/conf.d/ssl.conf
COPY httpd.conf /etc/httpd/conf/httpd.conf
EXPOSE 9443/tcp
COPY kmdl /opt/nfast/kmdata/local
COPY kmdl/certificate.pem /opt/nfast/kmdata/local/
COPY kmdl/certificate.pem /etc/pki/tls/certs/
#(for openssl)
RUN chmod 777 /etc/httpd/run
ENTRYPOINT ["/usr/sbin/httpd", "-e", "info", "-D", "FOREGROUND"]
```

3. Build the Apache container image:

```
% sudo docker build . -t nshield-apache-pkcs11-redhat
```

4. Once the build is complete, the new image will be included in the list of

Docker images:

```
% sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nshield-apache-pkcs11-redhat	latest	80cd2a476deb	18 seconds ago	982MB
nshield-app-pkcs11-redhat	latest	27cc224fe5f7	19 minutes ago	950MB
nshield-hwsp-pkcs11-redhat	latest	bfe9eafa3197	20 minutes ago	486MB
registry.access.redhat.com/ubi8/ubi	latest	cc0656847854	6 weeks ago	216MB

3.6. Build the test image container

The steps below are performed on the build machine. The test image will be the base image when testing the container integration inside the Kubernetes cluster.

1. In the **working** directory, create a **testimage** directory and copy the needed files to build the test image:

```
% cd ~/working
% mkdir testimage
% cd testimage
% cp ../kmdl/certificate.pem .
```

2. In the **testimage** directory create the **abi.sh** script. This will be used to test the performance of the Apache web server when integrated.

```
vi abi.sh
```

The content of the files should be the following:

```
ab -c50 -n10000 https://localhost:9443/icons/apache_pb2.gif
```

3. In the **testimage**, directory create the **osc.sh** script. This will be used to connect to the Apache webserver when integrated, to test if the server is using the key and certificates generated previously.

```
vi osc.sh
```

The content of the files should be the following:

```
openssl s_client -crlf -connect localhost:9443 -CAfile certificate.pem
```

4. Change the permissions of the scripts:

```
% chmod 777 abi.sh
% chmod 777 osc.sh
```

5. Create a Docker file to build the test image:

```
% vi Dockerfile
```

The contents of the file should be the following:

```
FROM registry.access.redhat.com/ubi8/ubi
RUN yum -y install net-tools httpd-tools openssl nano
COPY certificate.pem /opt/test/
COPY abi.sh /opt/test/
COPY osc.sh /opt/test/
RUN chmod 777 /opt/test/*.sh
```

6. Build the test image:

```
% sudo docker build . -t test-nshield-apache-pkcs11-redhat
```

7. List the Docker images:

```
% sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
test-nshield-apache-pkcs11-redhat	latest	7dbedf1c97d9	About an hour ago	242MB
nshield-apache-pkcs11-redhat	latest	80cd2a476deb	2 hours ago	982MB
nshield-app-pkcs11-redhat	latest	27cc224fe5f7	2 hours ago	950MB
nshield-hwsp-pkcs11-redhat	latest	bfe9eafa3197	2 hours ago	486MB
registry.access.redhat.com/ubi8/ubi	latest	cc0656847854	6 weeks ago	216MB

3.7. Test the configuration locally

The steps below are performed on the build machine.

1. Edit the `/opt/ncop/config1/config` file. Change HSM settings according to the HSM that is being used and loaded with the Security World that matches your files in `/opt/nfast/kmdata/local`.

```
% sudo vi /opt/ncop/config1/config
```

The contents of the file should be the following:

```
syntax-version=1

[nethsm_imports]
```



```

No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 1265 bytes and written 369 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
---
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol : TLSv1.3
    Cipher   : TLS_AES_256_GCM_SHA384
    Session-ID: 9EE98D136ABC1E77DC175D590226602E009BFFD76DDD3D2E67E610F10C1504B7
    Session-ID-ctx:
    Resumption PSK:
32F4D1F4A4BD0CE70A037EB5F773AFF3B19BD88F8127E03E4DB1ED3079917099004F1583C80D8C6F15CA6E0573058AD5
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
0000 - ef e2 e0 5f 17 7c 72 3b-01 b7 9b 53 f9 73 d7 c9   ..._.|r;...S.s..
0010 - d5 fe b3 cc 03 b8 98 82-b8 3b df 92 fc 00 a1 f6   .....;.....
0020 - c3 3e 63 1b f3 27 84 fd-cb 3d 8c a5 89 61 18 1f   .>c..'...=...a..
0030 - dd 71 98 9e 19 dd 50 e3-e5 46 81 8b df a9 75 6b   .q....P..F....uk
0040 - 0b 18 cb 51 85 56 d1 1f-68 86 bb 41 c6 97 97 e7   ...Q.V..h..A....
0050 - 86 ea b1 66 14 c2 20 40-41 e4 e1 f5 72 b6 0a 00   ...f.. @A...r...
0060 - e0 6e 25 39 10 e8 c3 51-2b 41 47 6a 9e 7b fa f3   .n%9...Q+AGj.{..
0070 - 2d 51 8e b8 81 2d 86 51-74 89 8c ea 90 26 15 90   -Q....Qt....&..
0080 - 50 9e e8 23 54 0f 5f 5f-89 bf ca 92 cf 72 fc fc   P..#T.____.r...
0090 - 69 c0 61 e8 be b4 05 c9-da 87 c0 63 ad 8c e2 13   i.a.....c....
00a0 - 3d 88 84 4e 15 c2 0b 43-8e 6b 3a 15 f6 aa 86 8b   =.N...C.k:.....
00b0 - fa 57 3e 25 f7 63 11 19-36 e4 c1 c3 48 46 05 25   .W>%..c..6...HF.%
00c0 - ba 46 64 ee 86 2d 19 b9-0f 02 47 86 05 be c5 7c   .Fd.-...G....|
00d0 - f2 51 ba 7a 14 5b f5 1b-57 5c ad 8e 3e dc 29 64   .Q.z.[.W\...>.)d

    Start Time: 1640029791
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol : TLSv1.3
    Cipher   : TLS_AES_256_GCM_SHA384
    Session-ID: F3BE76408FB218FEABAB99F2172166FB0C97A0068BDCE2ECE37D4F842D68DBD
    Session-ID-ctx:
    Resumption PSK:
36B592796087A64B4B99FB18C83D8C824C08F5DF51A0F5BCADEAF286D24895B42ACF0698E79BC2459124F6B918D3392
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:

```

```

0000 - ef e2 e0 5f 17 7c 72 3b-01 b7 9b 53 f9 73 d7 c9 ..._.|r;...S.s..
0010 - f1 db 38 c3 49 50 9e e4-39 50 1a 81 ab 69 f8 85 ..8.IP..9P...i..
0020 - ea a7 27 f3 4e 7d 73 ee-2e 8b eb 2c 13 38 dc aa ..'.N}s.....,8..
0030 - 19 ff e7 c6 e2 b6 41 2d-a6 94 f4 cb 2b 66 20 09 .....A-....+f .
0040 - fd 01 4f 0a dc 6e 2e 3c-b1 e5 fe 96 61 3f e4 47 ..0..n.<....a?.G
0050 - 5d ec e3 6b 03 50 94 6a-c9 62 22 72 65 4a 86 69 ]..k.P.j.b"reJ.i
0060 - a2 7f ab 88 3c 77 03 c5-d9 64 5c 84 7b d4 41 5f ....<w...d\.{.A_
0070 - cb 44 2b 64 9e 3a e2 bb-56 5b 4d 2f c6 43 a9 8a .D+d:..V[M/.C..
0080 - 40 49 9b 39 07 ad 75 33-90 52 d1 e8 d5 d1 c7 50 @I.9..u3.R.....P
0090 - 50 42 e9 72 24 54 60 86-63 32 bb 25 1b 9e 83 99 PB.r$T`.c2.%....
00a0 - d3 4f c3 5a bb a8 d1 65-d9 f4 6c 48 d6 09 bd 1c .O.Z...e..lH....
00b0 - 83 d9 1b d6 28 b9 25 32-86 4a 2b c9 8c 5e 68 bc ....(.%2.J+..^h.
00c0 - c7 f7 8c f7 c3 5e fd e6-02 33 65 fa 51 e8 bb 19 .....^...3e.Q...
00d0 - e2 3b 54 56 e6 ee e9 2a-5c 46 d6 6e b6 76 61 91 .;TV...*\F.n.va.

```

```

Start Time: 1640029791
Timeout : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
Max Early Data: 0

```

```

---
read R BLOCK
closed

```

5. Run the `abi.sh` script.

```
% abi.sh
```

This script tests performance against the server running in the container.

```

This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

```

```
Benchmarking localhost (be patient)
```

```

Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests

```

```

Server Software:      Apache/2.4.37
Server Hostname:     localhost
Server Port:         9443
SSL/TLS Protocol:    TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,2048,256
Server Temp Key:     X25519 253 bits
TLS Server Name:     localhost

```

```

Document Path:       /icons/apache_pb2.gif
Document Length:     4234 bytes

```

```

Concurrency Level:   50
Time taken for tests: 94.090 seconds
Complete requests:   10000
Failed requests:     0
Total transferred:   45170000 bytes

```

```

HTML transferred:      42340000 bytes
Requests per second:  106.28 [#/sec] (mean)
Time per request:     470.449 [ms] (mean)
Time per request:     9.409 [ms] (mean, across all concurrent requests)
Transfer rate:        468.82 [Kbytes/sec] received

Connection Times (ms)
                    min  mean[+/-sd] median  max
Connect:           49  465 457.3   333   5856
Processing:         0    1   1.1     1     44
Waiting:           0    1   1.0     1     41
Total:             49  466 457.4   334   5857

Percentage of the requests served within a certain time (ms)
 50%    334
 66%    480
 75%    603
 80%    660
 90%   1022
 95%   1347
 98%   1824
 99%   2236
100%   5857 (longest request)

```

3.8. Register the Apache and nCOP containers to an external registry

In this guide, the external registry is indicated as `<external-docker-registry-IP-address>`. Register the Docker container images to a Docker registry, so they can be used when you deploy Kubernetes pods into the cluster.



The distribution of the nShield container image is not permitted, because the software components are under strict export controls.

1. Log in to the Docker registry using your site credentials:

```
% sudo docker login -u YOURUSERID https://<external-docker-registry-IP-address>
```

2. Tag images:

```

% sudo docker tag test-nshield-apache-pkcs11-redhat <external-docker-registry-IP-address>/cv-test-nshield-
apache-pkcs11-redhat
% sudo docker tag nshield-apache-pkcs11-redhat <external-docker-registry-IP-address>/cv-nshield-apache-
pkcs11-redhat
% sudo docker tag nshield-app-pkcs11-redhat <external-docker-registry-IP-address>/cv-nshield-app-pkcs11-
redhat
% sudo docker tag nshield-hwsp-pkcs11-redhat <external-docker-registry-IP-address>/cv-nshield-hwsp-pkcs11-
redhat

```

3. Push images:

```
% sudo docker push <external-docker-registry-IP-address>/cv-test-nshield-apache-pkcs11-redhat
% sudo docker push <external-docker-registry-IP-address>/cv-nshield-apache-pkcs11-redhat
% sudo docker push <external-docker-registry-IP-address>/cv-nshield-app-pkcs11-redhat
% sudo docker push <external-docker-registry-IP-address>/cv-nshield-hwsp-pkcs11-redhat
```

4. Remove local images:

```
% sudo docker rmi <external-docker-registry-IP-address>/cv-test-nshield-apache-pkcs11-redhat
% sudo docker rmi <external-docker-registry-IP-address>/cv-nshield-apache-pkcs11-redhat
% sudo docker rmi <external-docker-registry-IP-address>/cv-nshield-app-pkcs11-redhat
% sudo docker rmi <external-docker-registry-IP-address>/cv-nshield-hwsp-pkcs11-redhat
```

5. Show images:

```
% sudo docker images
```

6. Pull images from the registry:

```
% sudo docker pull <external-docker-registry-IP-address>/cv-test-nshield-apache-pkcs11-redhat
% sudo docker pull <external-docker-registry-IP-address>/cv-nshield-apache-pkcs11-redhat
% sudo docker pull <external-docker-registry-IP-address>/cv-nshield-app-pkcs11-redhat
% sudo docker pull <external-docker-registry-IP-address>/cv-nshield-hwsp-pkcs11-redhat
```

7. Show images:

```
% sudo docker images
```

3.9. Deploy on Kubernetes

For this integration, the Kubernetes environment is Red Hat OpenShift. This can be applied to any other Kubernetes environment as required. The steps below are performed on the Kubernetes server. This procedure requires that the Kubernetes environment is already created.

1. Log in to the Kubernetes server.

This server will be used to deploy the Kubernetes pod that will run the nShield Apache application.

2. Install the Security World software.

Install the latest version of the Security World Software as described in the *Installation Guide* for the HSM.



Entrust recommends that you uninstall any existing nShield

software before installing the new nShield software.

3. Add the HSM used during the Apache container creation as a client to the OpenShift server.

Install the HSM by following the instructions in the *Installation Guide* for the HSM. Once the Kubernetes server has been added as a client in the HSM, enroll the HSM in the Kubernetes server:

```
% sudo /opt/nfast/bin/nethsmenroll --privileged 10.194.148.33
```

When using an OpenShift cluster it is important to add all master and worker nodes of the cluster as clients in the HSM.

4. Create YAML directory and files needed to deploy the application:
 - a. Create a directory named `yaml`. Use this directory when creating all required files.
 - b. Create the `project.yaml` project YAML file needed to deploy the application. For example:

```
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  annotations:
    openshift.io/description: ""
    openshift.io/display-name: apache
    openshift.io/requester: kube:admin
  name: apache
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
```

- c. Create the `cm.yaml` config map YAML file needed by the application. For example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: ncop-config-apache
  namespace: apache
data:
  config: |
    syntax-version=1

    [nethsm_imports]
    local_module=1
    remote_esn=201E-03E0-D947
    remote_ip=10.194.148.33
    remote_port=9004
    keyhash=84800d1bfff6515ed5806fe443bbaca812d73733
```



```
- name: ncop-config-apache
  mountPath: /opt/nfast/kmdata/config
- name: ncop-hardserver
  mountPath: /opt/nfast/kmdata/hardserver.d
- name: ncop-sockets
  mountPath: /opt/nfast/sockets
- name: ncop-sockets-priv
  mountPath: /opt/nfast/sockets-priv
- name: nshield-apache-pkcs11
  image: >-
    registry.eselab.net/cv-nshield-apache-pkcs11-redhat
  imagePullPolicy: Always
  terminationMessagePath: /dev/termination-log
  terminationMessagePolicy: File
  ports:
    - containerPort: 9443
      protocol: TCP
  resources: {}
  volumeMounts:
    - name: ncop-sockets
      mountPath: /opt/nfast/sockets
  securityContext: {}
  volumes:
    - configMap:
        name: ncop-config-apache
        defaultMode: 420
      name: ncop-config-apache
    - name: ncop-hardserver
      emptyDir: {}
    - name: ncop-sockets
      emptyDir: {}
    - name: ncop-sockets-priv
      emptyDir: {}
```

3.10. Deploy the application

The steps below are performed in the Kubernetes server. This procedure requires that the Kubernetes environment is already created.

1. Set the environment.
 - a. CodeReady OpenShift Environment

This step assumes a CodeReady OpenShift environment. Perform what is required from your Kubernetes environment. For example:

```
% eval $(crc oc-env)
```

Log into OpenShift:

```
% oc login -u kubeadmin https://api.crc.testing:6443
```

- b. Openshift Cluster

```
% export KUBECONFIG=/path/to/the/cluster/kubeconfig/file
```

2. Change directory to where the YAML files are located:

```
% cd yaml
```

3. Create the project:

```
% oc create -f project.yaml  
project.project.openshift.io/apache created
```

4. Change from the current project to the newly created project:

```
% oc project apache  
Now using project "apache" on server "https://api.crc.testing:6443".
```

5. Create the namespace (CodeReady Only)

```
% oc create namespace apache  
namespace/apache created
```

6. Create the registry secret:

```
% oc create -f secret.yaml  
secret/regcred created
```

7. Create the config map:

```
% oc create -f cm.yaml  
configmap/ncop-config-apache created
```

8. Deploy the Apache application containers:

```
% oc create -f pod_apache.yaml  
pod/nshield-apache-example-9nl74 created
```

With the latest security policy changes on Openshift 4.11.20, you may see the following warning which does not affect the pod deployment:

```
Warning: would violate PodSecurity "restricted:v1.24": allowPrivilegeEscalation != false (container "test-nshield-apache-pkcs11" must set securityContext.allowPrivilegeEscalation=false), unrestricted capabilities (container "test-nshield-apache-pkcs11" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or containers "test-nshield-apache-pkcs11", "nshield-hwsp-pkcs11", "nshield-apache-pkcs11" must set securityContext.runAsNonRoot=true), seccompProfile (pod or containers "test-nshield-apache-pkcs11", "nshield-hwsp-pkcs11", "nshield-apache-pkcs11" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost")
```

9. List the containers in the pod:

```
% oc get pods nshield-apache-example-9nl74 -n apache -o jsonpath="{.spec.containers[*].image}"
registry.eselab.net/cv-test-nshield-apache-pkcs11-redhat registry.eselab.net/cv-nshield-hwsp-pkcs11-redhat
registry.eselab.net/cv-nshield-apache-pkcs11-redhat
```

10. Verify Apache is running:

```
% oc get pods --all-namespaces | grep apache
apache          nshield-apache-example-9nl74      3/3      Running    0          5m
```

11. Open a debug session on the pod so you can check the Apache application:

```
% oc debug pod/nshield-apache-example-9nl74

Defaulting container name to test-nshield-apache-pkcs11.
Use 'oc describe pod/nshield-apache-example-9nl74-debug -n apache' to see all of the containers in this pod.

Starting pod/nshield-apache-example-9nl74-debug, command was: sh -c sleep 7200
Pod IP: 10.217.0.118
If you don't see a command prompt, try pressing enter.
sh-4.4#
```

12. In the debug session, run the `osc.sh` script to check the connection with the Apache server that is running in the container.

You should see that it is using the key created when you created the Docker container. In this case, this is the `apachecontainerkey` key.

```
sh-4.4# cd /opt/test/osc.sh

CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 CN = apachecontainerkey
verify return:1
---
Certificate chain
 0 s:CN = apachecontainerkey
  i:CN = apachecontainerkey
  ---
Server certificate
```

```

-----BEGIN CERTIFICATE-----
MIICwTCCAakCFAmL778s/6aHXzKK84DsM6cv02AhMA0GCSqGSIb3DQEBCwUAMB0x
GzAZBgNVBAMTEmFwYWN0ZWNvbnRhaW5lcmtleTAeFw0yMTEyMjAxNjQwMjFhZjEw
MjE5MjAxNjQwMjFhZjEwYWN0ZWNvbnRhaW5lcmtleTCCASwDQYJKoZIhvcNAQEB
BQADggEPADCCAQoCggEBAJ48W0IR60D5wcMuQv/aYCWnYa8uhtz+Pq2UiXcmwhd9f
SSkeVGHVEOP5+dtJkI6DTMwWefg+u515HLM+rteeQCicDnoXEm/WV8HRffnknznHt
KMIOw75UNcs0xuavc0hKcdn/q/Yn5AsBQTbjmUSFQhDQZ4saNuN6PsgNjh/grjZiLE
huRyh4neAEyINMBIULXecbnBpoMnMtQLm6PT1IRZCQQ3L8yONTpXqiap/hcSdUxOZ
jaTWL7oDsnTaK42eX0x2MytFSMULw+KTwnCrKYnc+DLiRCMJsVbdHwFahjMsidnLghe
Bnc7BN5Lax5rH0a2J1G91DQv/5UFa+4sCAWEAATANBgkqhkiG9w0BAQsFAAOCAQEA
aTAWb/jaEicXcVv/80IyDnSyBHSyDT3yKMrdEn4061UgEwo1tU1MgNHPr7ripK12UQ4G
S7vrntWU7d9k7Rzq2QQB5NkhvC1xzszBRV/miBKTd4cNGMxfB+zmEwPv82nMKsWj
fQmbwPVXoIHcprpCNRJKMBO1emp0sGo9Y2xBxJst8HGZ027j1rB3yMmZHSBFMRHu
DRG1PMLFbDmzsc/6VmVb+inMA1fgsaz3bkQMYJ7Q3BzWV730tAyOcVVK79aRRQBL
pohjCBkjqsepuRMP2EP/apKNJ9SsgEfwyrSYZE45ImJQORATHxwDbYbc5cbTknOVLCLk
idg+Gfuw==
-----END CERTIFICATE-----
subject=CN = apachecontainerkey

issuer=CN = apachecontainerkey

---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 1265 bytes and written 369 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
---
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: 3FB4E1ACACEB5FC796DBB2EAC0BC92EBFC795AF7199BDAE680CAEBB606DE5D4F
    Session-ID-ctx:
    Resumption PSK:
A41DA3DEBDE08228F09051E6671BCC1D3C0F715AF538D5476342067CDC3BFA717F88A1D3F9AE3DAA8F4AE3B954ADBAD3
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
0000 - 29 42 88 f9 42 e6 a2 6c-5e c8 82 21 d1 4d 9d 8e )B..B..l^..!.M..
0010 - 58 0f 9d 98 7d 1d 9f cd-b7 1f 44 9f 69 7c b5 c3 X...}.....D.i|..
0020 - 0c 47 e4 18 db a5 7e d4-93 81 d2 46 a1 51 b8 97 .G.....~.....F.Q..
0030 - 50 cb d7 af ee 1a 4f 07-4c 20 5e 15 ca 7b ce e0 P.....0.L ^..{..
0040 - a8 9f fe 9e 6e 4c 21 21-43 e1 ca ce 0a 79 e0 f2 ....nL!!C....y..
0050 - 76 1c a6 e5 b9 ac eb 77-05 52 7a b0 1a 78 e8 4c v.....w.Rz..x.L
0060 - 28 67 6c ef 01 5a 29 04-c8 b6 eb 17 b7 2c a5 8b (gl..Z).....,..
0070 - a7 5b da 33 eb dd 95 3d-58 05 03 b8 43 72 c8 70 [.3...=X...Cr.p
0080 - cd b1 3c e4 fa 95 92 d4-6f a8 49 4f b3 03 02 53 ..<.....o.IO...S
0090 - 42 6c 7c d5 62 82 ee 01-80 09 cc 44 09 53 e0 25 B||.b.....D.S.%
00a0 - 98 7a cf 55 9f 2a fb 14-c1 9f ea 20 a3 cc 31 d2 .z.U.*..... ..1.
00b0 - 8c 8d bd 09 4c 30 c2 07-83 b5 c2 5a 34 18 ce 74 ....L0.....Z4..t
00c0 - b8 66 35 1f 6d 41 a3 8a-aa d4 cb e8 97 4e b4 09 .f5.mA.....N..
00d0 - 76 8e 38 5b 9b a4 76 fa-0c 6d 01 a3 dd ff 7a 83 v.8[.v..m....z.

```

```

Start Time: 1640110099
Timeout   : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
Max Early Data: 0
---
read R BLOCK
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol   : TLSv1.3
    Cipher     : TLS_AES_256_GCM_SHA384
    Session-ID: 65EEA38183B8D613B0E866114C0CF84CE8A8EFE0879D137083607FD28251DFC6
    Session-ID-ctx:
    Resumption PSK:
E5E878DCC69F627043889B7A2CCFDEEE753F16B4F2BDA9A28670E01D9459BBBEE10380F5304784D7D38981CF40F86353
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
0000 - 29 42 88 f9 42 e6 a2 6c-5e c8 82 21 d1 4d 9d 8e   )B..B..l^..!.M..
0010 - 99 eb 3f 7b 17 11 3b e6-3c 7c 83 44 80 dd 4d 1b   ..?{..;<|.D..M.
0020 - 75 fe bf 5a c0 34 5b ca-07 1a fa 54 1e 41 be 8c   u..Z.4[....T.A..
0030 - 85 ed 87 a4 c1 21 5b 88-fa f0 71 b3 ab d3 2d d1   .....![...q...-.
0040 - ec 69 99 6c 77 94 5e e9-11 8d 9c c6 9d 2f d7 c2   .i.lw.^...../..
0050 - a8 a1 5c 70 75 18 80 36-bd 08 27 79 98 57 94 91   ..\pu..6..'y.W..
0060 - 6e 4c f5 6c d5 aa aa 8e-57 a2 dd e6 ca 5e 43 76   nL.l...W....^Cv
0070 - f5 30 48 95 30 8b 1b 93-af 71 15 5f c2 7e ee 4e   .0H.0....q_..~.N
0080 - 10 83 f6 4a 1c b1 88 54-f8 d0 73 63 57 ed 63 37   ...J...T..scW.c7
0090 - 26 ed 48 4b b4 c9 cf ff-fc d6 2f de df 40 ee e6   8.HK...../..@..
00a0 - c7 41 1d 92 9f de fd a9-c0 a8 40 d0 69 22 cb 74   .A.....@.i".t
00b0 - 08 0b b2 60 70 d2 02 3c-8a a0 4c 49 48 bb c9 0a   ...'p..<..LIH...
00c0 - 52 cd 97 ea 69 3b 1d f8-65 56 f1 32 f0 5d df a1   R...i;..eV.2.]..
00d0 - 44 d7 f5 73 a5 a4 fa 9f-62 09 8b c5 c3 4a 26 cf   D..s....b....J&.

Start Time: 1640110099
Timeout   : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
Max Early Data: 0
---
read R BLOCK
closed

```

13. In the debug session, run the `abi.sh` script to check the performance of the Apache server running in the container:

```

sh-4.4# /opt/test/abi.sh

This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests

```

```

Completed 9000 requests
Completed 10000 requests
Finished 10000 requests

Server Software:      Apache/2.4.37
Server Hostname:      localhost
Server Port:          9443
SSL/TLS Protocol:     TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,2048,256
Server Temp Key:      X25519 253 bits
TLS Server Name:      localhost

Document Path:        /icons/apache_pb2.gif
Document Length:      4234 bytes

Concurrency Level:    50
Time taken for tests: 87.555 seconds
Complete requests:    10000
Failed requests:      0
Total transferred:    45170000 bytes
HTML transferred:     42340000 bytes
Requests per second: 114.21 [#/sec] (mean)
Time per request:     437.775 [ms] (mean)
Time per request:     8.755 [ms] (mean, across all concurrent requests)
Transfer rate:        503.81 [Kbytes/sec] received

Connection Times (ms)
      min  mean[+/-sd] median  max
Connect:    49  434 476.9   262  5149
Processing:   0   1   4.9     1   242
Waiting:     0   1   4.8     0   241
Total:       49  435 477.0   264  5149

Percentage of the requests served within a certain time (ms)
 50%    264
 66%    434
 75%    569
 80%    658
 90%   1036
 95%   1389
 98%   1895
 99%   2255
100%   5149 (longest request)

```

The integration of Apache using PKC #11 running inside a Kubernetes container is now complete.

Chapter 4. Additional resources and related products

4.1. nShield Connect

4.2. nShield as a Service

4.3. Entrust digital security solutions

4.4. nShield product documentation