# Red Hat OpenShift

## nShield® HSM Integration Guide

2026-01-28

© Entrust Corporation. All rights reserved.

# Table of Contents


1. Introduction . . . 1
    1.1. Integration architecture . . . 1
    1.2. Product configurations . . . 2
    1.3. Requirements . . . 3
    1.4. What changed since the last integration test . . . 3
2. Procedures . . . 4
    2.1. Prerequisites . . . 4
    2.2. Install nCOP . . . 4
    2.3. Build the nCOP containers . . . 4
    2.4. Register the nCOP containers to an external registry . . . 6
    2.5. Deploying the nCOP images . . . 7
    2.6. FIPS Level 3 recommendations . . . 28
3. 5c 10G Configuration using KeySafe 5 . . . 29
    3.1. Add the OpenShift nodes and deployment server as clients of the HSM. . . . 29
    3.2. TVD - Remote Administration Client . . . 31
4. Sample YAML files . . . 32
    4.1. project.yaml . . . 32
    4.2. cm.yaml . . . 32
    4.3. pv_nfast_kmdata_definition.yaml . . . 32
    4.4. pv_nfast_sockets_definition.yaml . . . 33
    4.5. pv_nfast_kmdata_claim.yaml . . . 33
    4.6. pv_nfast_sockets_claim.yaml . . . 33
    4.7. pod_dummy.yaml . . . 34
    4.8. pod_hwsp.yaml . . . 35
    4.9. pod_enquiry_container.yaml . . . 35
    4.10. pod_nfkminfo_container.yaml . . . 36
    4.11. pod_generatekeymodule_container.yaml . . . 37
    4.12. pod_generatekeysoftcard_container.yaml . . . 38
    4.13. pod_generatekeyocs_container.yaml . . . 39
    4.14. pod_rocs_container.yaml . . . 40
    4.15. pod_all_container.yaml . . . 41
    4.16. ocsexpect.sh . . . 42
    4.17. softcardexpect.sh . . . 42
5. Additional resources and related products . . . 44
    5.1. nShield HSMs . . . 44
    5.2. nShield as a Service . . . 44
    5.3. nShield Container Option Pack . . . 44

5.4. Entrust products . . . . . 44
5.5. nShield product documentation . . . . . 44

# Chapter 1. Introduction

This guide describes how to integrate a Red Hat OpenShift cluster with an Entrust nShield Hardware Security Module (HSM), using the nShield Container Option Pack (nCOP).

OpenShift is an application hosting platform by Red Hat. It makes it easy for developers to quickly build, launch, and scale container-based web applications in a public cloud environment. nCOP allows application developers, in the container-based environment of OpenShift, to access the cryptographic functionality of an HSM.

## 1.1. Integration architecture

### 1.1.1. OpenShift cluster and HSM

In this integration, a Red Hat OpenShift cluster is deployed on a VMware vSphere instance. Container images are used from a third-party cloud registry.

### 1.1.2. Container images

Two container images were created for the purpose of this integration: a hardserver container, and one application container. These images are stored in an external registry to be deployed to OpenShift:

- `cv-nshield-hwsp-container`

  A hardserver container image that controls communication between the HSM(s) and the application containers. One or more hardserver containers are required per deployment, depending on the number of HSMs and the number and types of application containers.

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

- `cv-nshield-app-container`

  Application container image to run nShield commands. It is an Ubuntu Universal Base Image container, in which Security World software is installed.

You can also create containers that contain your application. For instructions, see the *nShield Container Option Pack User Guide*.

## 1.2. Product configurations

Entrust has successfully tested the integration of an nShield HSM with Red Hat OpenShift in the following configurations:

| Product | Version |
|---|---|
| Base OS | RHEL 9.7 |
| OpenShift | 4.20.6 |
| VMware | vSphere 8.0.0.10200 |
| Security World Software | 13.6.14 |
| nCOP | 1.1.3 |

### 1.2.1. Supported nShield hardware and software versions

Entrust tested with the following nShield hardware and software versions:

#### 1.2.1.1. Connect XC

| Security World Software | Firmware | Image | OCS | Softcard | Module | FIPS Level 3 |
|---|---|---|---|---|---|---|
| 13.6.14 | 12.72.4 (FIPS 140-2 certified) | 13.6.14 | ✓ | ✓ | ✓ | No |

#### 1.2.1.2. nShield 5c

| Security World Software | Firmware | Image | OCS | Softcard | Module | FIPS Level 3 |
|---|---|---|---|---|---|---|
| 13.6.14 | 13.4.5 (FIPS 140-3 certified) | 13.6.14 | ✓ | ✓ | ✓ | Yes |

#### 1.2.1.3. nShield 5c 10G

| Security World Software | Firmware | Image | OCS | Softcard | Module | FIPS Level 3 |
|---|---|---|---|---|---|---|
| 13.6.14 | 13.4.5 (FIPS 140-3 certified) | 14.0.4 | ✓ | ✓ | ✓ | Yes |

## 1.3. Requirements

### 1.3.1. Before starting the integration process

Familiarize yourself with:

- The documentation for the nShield HSM.
- The *nShield Container Option Pack User Guide*.
- The documentation and setup process for Red Hat OpenShift.

## 1.4. What changed since the last integration test

- The deployment server changed from Red Hat 8 to a Red Hat 9.
- The OpenShift version changed from 4.14.7 to 4.20.6.
- The Security World Client software version changed from 13.4.4 to 13.6.14.
- The nShield Container Option Pack (nCOP) version changed from 1.1.2 to 1.1.3.
- yaml attributes:
  - Affinity attributes
    These attributes were added to the yaml files because OpenShift can deploy pods in different worker nodes. These attributes were added to the yaml files so the application pods deploy on the same node as the `hardserver` pod.
  - `ServiceAccountName` attribute
    This attribute was added to resolve security warnings. It contains the service account name that is new and created during the deployment process. All pods run under this service account. The service account name is `ncop-sa`.
  - Name changes
    The yaml files used the `nscop` string on some attributes. This string changed to `ncop` to match the nCOP name.
- HSMs changes: Connect XC and the nShield 5c continue to be supported, nShield 5c 10G is also supported now.

# Chapter 2. Procedures

## 2.1. Prerequisites

Before you can use nCOP container images in OpenShift, you must complete the following steps:

1. Install OpenShift on a supported configuration. See the documentation provided by Red Hat.
2. Set up the HSM. See the *Installation Guide* for your HSM.
3. Configure the HSM(s) to have the IP address of your container host machine as a client. These are the OpenShift cluster nodes deployed in vSphere in this integration.
4. Load an existing Security World or create a new one on the HSM. Copy the Security World and module files to your container host machine at a directory of your choice. Instructions on how to copy these files into an OpenShift persistent volume accessible by the application containers are given in [Copy files to the cluster persistent volume].

   For more information on configuring and managing nShield HSMs, Security Worlds, and Remote File Systems, see the *User Guide* for your HSM(s).

If you are using the 5c 10G, please refer to the 5c 10G Configuration using KeySafe 5 section in this guide for details on its configuration in the KeySafe 5 WebUI. Also refer to the KeySafe 5 Installation Guide for installation instructions for KeySafe 5.

## 2.2. Install nCOP

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
```

## 2.3. Build the nCOP containers

This process will build nCOP containers for the hardserver and application. Please note that:

- This guide uses the "ubuntu" 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.
- The example below uses version 13.6.14 of the Security World client.

To build the nCOP containers:

1. Mount the Security World Software ISO file:

```
% sudo mount -t iso9660 -o loop ISOFILE.iso /mnt/iso1
```

2. Build the nShield container for the hardserver and application (Ubuntu):

If you are using softcard or OCS protection in your integration, edit the `/opt/ncop/make-nshield-application` script and add the `expect` package so it gets installed in the image. This package is not available by default on the default image used by the script (Red-Hat image), so we will use the Ubuntu image as the basis for the containers. The `expect` package is available for install in the Ubuntu image. Modify `/opt/ncop/make-nshield-application` and add `expect`.

Current line:

```
RUN if [ -x /usr/bin/apt-get ]; then apt-get -y update && apt-get -y upgrade && apt-get -y install socat;
fi
```

Change it to:

```
RUN if [ -x /usr/bin/apt-get ]; then apt-get -y update && apt-get -y upgrade && apt-get -y install socat
expect; fi
```

Now build the containers:

```
% cd /opt/ncop
% sudo ./make-nshield-hwsp --tag nshield-hwsp-container:13.6.14 --from docker.io/library/ubuntu:focal
/mnt/iso1
% sudo ./make-nshield-application --tag nshield-app-container:13.6.14 --from docker.io/library/ubuntu:focal
/mnt/iso1
```

3. Validate that the images have been built:

```
% sudo docker images
```

You should see the 2 images listed.

4. Build the `nshield-hwsp` configuration. You will need the HSM IP address during this process.

```
% cd /opt/ncop
% sudo mkdir -p /opt/ncop/config1
% sudo ./make-nshield-hwsp-config --output /opt/ncop/config1/config HSM_IP_ADDRESS
% cat /opt/ncop/config1/config
```

5. Build the nShield Application Container Security World. You will need the HSM world and module file during this process.

```
% sudo mkdir -p /opt/ncop/app1/kmdata/local
% sudo cp world /opt/ncop/app1/kmdata/local/.
% sudo cp module_<ESN> /opt/ncop/app1/kmdata/local/.
% ls /opt/ncop/app1/kmdata/*
```

6. Create a Docker socket:

```
% sudo docker volume create socket1
```

7. Check if the hardserver container can access the HSM using sockets:

```
% sudo docker run -v /opt/ncop/config1:/opt/nfast/kmdata/config:ro -v socket1:/opt/nfast/sockets nshield-
hwsp-container:13.6.14 &
% dmountpoint=`sudo docker volume inspect --format '{{ .Mountpoint }}' socket1`
% export NFAST_SERVER=$dmountpoint/nserver
% /opt/nfast/bin/enquiry -m0
```

8. Check if the Container Application can access using the Security World:

```
% sudo docker run --rm -it -v /opt/ncop/app1/kmdata:/opt/nfast/kmdata:ro -v socket1:/opt/nfast/sockets -it
nshield-app-container:13.6.14 /opt/nfast/bin/enquiry
```

## 2.4. Register the nCOP containers to an external registry

In this guide, the external registry is *<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 OpenShift cluster.

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 nshield-hwsp-container:13.6.14" <external-docker-registry-IP-address>/nshield-hwsp-container
% sudo docker tag nshield-app-container:13.6.14" <external-docker-registry-IP-address>/nshield-app-container
```

3. Push images:

```
% sudo docker push <external-docker-registry-IP-address>/cv-nshield-hwsp-container
% sudo docker push <external-docker-registry-IP-address>/cv-nshield-app-container
```

4. Remove local images:

```
% sudo docker rmi <external-docker-registry-IP-address>/cv-nshield-hwsp-container
% sudo docker rmi <external-docker-registry-IP-address>/cv-nshield-app-container
```

5. Show images:

```
% sudo docker images
```

6. Pull images from the registry:

```
% sudo docker pull <external-docker-registry-IP-address>/cv-nshield-hwsp-container
% sudo docker pull <external-docker-registry-IP-address>/cv-nshield-app-container
```

7. Show images:

```
% sudo docker images
```

## 2.5. Deploying the nCOP images

This section describes how to deploy nCOP images to call `nfast` binaries using `hwsp` and application container images. The deployment consists of pods, each of which contains a hardserver and an application container. Each application container executes nShield command(s).

A persistent volume is set up in the OpenShift cluster file system. This persistent volume contains the Security World and module files. The hardserver container and application containers will have access to these files.

### 2.5.1. Create the project

This section describes how to deploy the nCOP image. Sample YAML files contains example files. Entrust does not provide support for using these files. You must adapt them to your system and use case:

- project.yaml

  Contains the container project name. You can change the project name, but then that change will also need to be propagated across the namespace entry in the other YAML files, as well as the command examples in the instructions in this guide.

- cm.yaml

  Configuration map that contains the ESN and the IP address of the HSM. Edit this file to match your system.

This guide uses VMware vSphere to deploy the OpenShift cluster. Once deployed, by default in our case, it uses https://api.ocp4.interop.local:6443 for the cluster API address.

1. Log in to the server and launch a terminal window.
2. Copy the `project.yaml` and `cm.yaml` files to a local directory on the server. Edit the files to match your system.
3. Log in to the container platform:

```
% oc login -u kubeadmin -p <container-password> https://api.ocp4.interop.local:6443

Authentication required for https://api.ocp4.interop.local:6443 (openshift)
Username: kubeadmin
Password:
Login successful.

You have access to 65 projects, the list has been suppressed. You can list all projects with 'oc projects'

Using project "default".
Welcome! See 'oc help' to get started.
```

4. Create a container project:

```
% oc create -f project.yaml

project.project.openshift.io/ncop-test created
```

5. Change from the current container project to the new one:

```
% oc project ncop-test

Now using project "ncop-test" on server "https://api.ocp4.interop.local:6443".
```

6. Create the configuration map for the HSM details:

```
% oc create -f cm.yaml

configmap/config created
```

7. Verify the HSM configuration:

```
% oc get configmap

NAME                      DATA   AGE
config                    1      6s
kube-root-ca.crt          1      11s
openshift-service-ca.crt  1      11s

% oc describe configmap/config

Name:         config
Namespace:    ncop-test
Labels:       <none>
Annotations:  <none>

Data
====
config:
----
syntax-version=1

[nethsm_imports]
local_module=1
remote_esn=7852-268D-3BF9
remote_ip=XX.XXX.XX.XXX
remote_port=9004
keyhash=ed28cc6bb5dfef39ff327002006a55d90be0758d
privileged=0

BinaryData
====

Events:  <none>
```

8. If you have not yet enrolled the Openshift cluster nodes as clients of the HSM, enroll it now. For instructions, see the *User Guide* for your HSM.

### 2.5.2. Create the registry secrets inside the cluster

At the beginning of this process, you created nCOP Docker containers and pushed them to the Docker registry. You must now configure the OpenShift cluster to authenticate to the

Docker registry.

1. Create the secret in the cluster:

```
% oc create secret generic regcred --from-file=.dockerconfigjson=$HOME/.docker/config.json
--type=kubernetes.io/dockerconfigjson
```

2. Confirm that the secret was created:

```
% oc get secret regcred
```

### 2.5.3. Create the OCS card and softcard secrets inside the cluster (Optional)

If using OCS card or softcard protection, the secrets for these cards need to be stored in the cluster. The password and card information for OCS and softcard will be stored. This guide demonstrates OCS card and softcard protection. These will be used by the `generatekey` examples when generating a key in the OCS card and softcard. They will be passed to the environment and used by expect scripts whenever the OCS and/or softcard requires the passphrase during key generation.

```
% oc create secret generic cardcred --from-literal=CARDPP=ncipher --from-literal=CARDMODULE=1 --from
-literal=OCS=testOCS --from-literal=OCSKEY=ocskey --from-literal=SOFTCARD=testSC --from
-literal=SOFTCARDKEY=softcardkey

secret/cardcred created

% oc get secret cardcred

NAME       TYPE     DATA   AGE
cardcred   Opaque   6      0s

% oc get secret cardcred -o YAML

apiVersion: v1
data:
  CARDMODULE: MQ==
  CARDPP: MTIz
  OCS: b3BlbnNoaWZ0b2Nz
  OCSKEY: b2Nza2V5
  SOFTCARD: b3BlbnNoaWZ0c29mdGNhcmQ=
  SOFTCARDKEY: c29mdGNhcmRrZXk=
kind: Secret
metadata:
  creationTimestamp: "2025-12-05T14:30:12Z"
  name: cardcred
  namespace: ncop-test
  resourceVersion: "745591"
  uid: 0ad193ba-d470-45ba-9df3-a4358cdd626e
type: Opaque
```

### 2.5.4. Create the cluster persistent volumes

This section describes how the persistent volume is created in the OpenShift cluster.

The following folder will contain the files you need for the integration.

- `/opt/nfast/kmdata`

The following YAML files are used to create and claim the persistent volume. Sample YAML files contains example files. Entrust does not provide support for using these files. You must adapt them to your system and use case:

- pv_nfast_sockets_definition.yaml
- pv_nfast_sockets_claim.yaml
- pv_nfast_kmdata_definition.yaml
- pv_nfast_kmdata_claim.yaml

1. Log in to the container platform and create the persistent volume and claims:

```
% oc create -f pv_nfast_sockets_definition.yaml

persistentvolume/nfast-sockets created

% oc create -f pv_nfast_kmdata_definition.yaml

persistentvolume/nfast-kmdata created

% oc create -f pv_nfast_sockets_claim.yaml

persistentvolumeclaim/nfast-sockets created

% oc create -f pv_nfast_kmdata_claim.yaml

persistentvolumeclaim/nfast-kmdata created
```

2. Verify that the persistent volume has been created:

```
% oc get pv

NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                    STORAGECLASS
VOLUMEATTRIBUTESCLASS   REASON   AGE
nfast-kmdata    1G         RWO            Retain           Bound    ncop-test/nfast-kmdata   manual
<unset>                          10s
nfast-sockets   1G         RWO            Retain           Bound    ncop-test/nfast-sockets  manual
<unset>                          10s
```

3. Verify that the claim has been created:

```
oc get pvc

NAME            STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS
AGE
nfast-kmdata    Bound    nfast-kmdata    1G         RWO            manual         <unset>
```

```
5s
nfast-sockets   Bound    nfast-sockets   1G         RWO            manual         <unset>
5s
```

### 2.5.5. Create the service account that will be used by the pods

1. Create the service account

```
oc -n ncop-test create sa ncop-sa

serviceaccount/ncop-sa created
```

2. Give the correct privileges to the account.

```
oc adm policy add-scc-to-user privileged -z ncop-sa -n ncop-test

clusterrole.rbac.authorization.k8s.io/system:openshift:scc:privileged added: "ncop-sa"
```

### 2.5.6. Copy needed files to the cluster persistent volume

At a minimum the Security World and module files are needed in the persistent volume. If using a FIPS Level 3 World file or OCS protection, the OCS card files are also needed, together with the cardlist file. If using soft card protection, the softcard files are needed.

If any custom scripts used by the application container were created, they can also be put in the persistent volume. In this guide, two scripts were created to demonstrate how to pass the passphrase for the OCS card and softcard when generating a key.

This section describes how to populate the `nfast-kmdata` persistent volume with these files:

- `/opt/nfast/kmdata/local/world`
- `/opt/nfast/kmdata/local/module_<ESN>`
- `/opt/nfast/kmdata/local/card*`
- `/opt/nfast/kmdata/local/softcard*`
- `/opt/nfast/kmdata/config/cardlist`
- Application scripts

A dummy application container provides access to the persistent volume. This enables you to copy these files from the host server to the OpenShift cluster.

The following files are required to perform these steps. Sample YAML files contains example files. Entrust does not provide support for using these files. You must adapt them to your system and use case:

- pod_hwsp.yaml
- pod_dummy.yaml

The container running the hardserver needs to run at this time.

1. Log in to the container platform and create the hardserver container and dummy application container:

```
% oc create -f pod_hwsp.yaml

pod/ncop-hwsp-trxgd created

% oc create -f pod_dummy.yaml

pod/ncop-test-dummy-48b6w created

% oc get pods

NAME                    READY   STATUS    RESTARTS   AGE
ncop-hwsp-trxgd         1/1     Running   0          11s
ncop-test-dummy-48b6w   1/1     Running   0          11s
```

2. Create the directory structure needed in the cluster `nfast-kmdata` persistent volume:

```
% oc debug pod/ncop-test-dummy-48b6w -- mkdir -p /opt/nfast/kmdata/local

% oc debug pod/ncop-test-dummy-48b6w -- mkdir -p /opt/nfast/kmdata/config

% oc debug pod/ncop-test-dummy-48b6w -- mkdir -p /opt/nfast/kmdata/bin
```

3. Copy the Security World and module files from the host directory to the cluster `nfast-kmdata` persistent volume:

```
% oc cp /opt/nfast/kmdata/local/world ncop-test/ncop-test-dummy-48b6w:/opt/nfast/kmdata/local/world

% oc cp /opt/nfast/kmdata/local/module_<ESN> ncop-test/ncop-test-dummy-48b6w:/opt/nfast/kmdata/local/.
```

4. Copy the card files associated with the OCS card.

   For a FIPS Level 3 World, these will be used to provide FIPS Authorization. They also will be used if OCS protection is in place.

```
% oc cp /opt/nfast/kmdata/local/card* ncop-test/ncop-test-dummy-48b6w:/opt/nfast/kmdata/local/.
```

5. Copy the softcard files if using softcard protection.

```
% oc cp /opt/nfast/kmdata/local/softcard* ncop-test/ncop-test-dummy-48b6w:/opt/nfast/kmdata/local/.
```

6. Copy the `config/cardlist` file.

```
% oc cp /opt/nfast/kmdata/config/cardlist ncop-test/ncop-test-dummy-48b6w:/opt/nfast/kmdata/config/cardlist
```

7. Verify that the files have been copied:

```
% oc debug pod/ncop-test-dummy-48b6w -- ls -al /opt/nfast/kmdata/local

Starting pod/ncop-test-dummy-48b6w-debug-bmc9g, command was: sh -c sleep 3600
total 104
drwxr-xr-x. 2 root root  4096 Dec  5 14:30 .
drwxr-xr-x. 5 root root    44 Dec  5 14:30 ..
-rw-r--r--. 1 root  975   104 Dec  5 14:30 card_7aaf758bc6790206198ea5218040d4faa09f035f_1
-rw-r--r--. 1 root  975   104 Dec  5 14:30 card_7aaf758bc6790206198ea5218040d4faa09f035f_2
-rw-r--r--. 1 root  975  1364 Dec  5 14:30 cards_7aaf758bc6790206198ea5218040d4faa09f035f
-rwxrwxrwx. 1 root  975  3716 Dec  5 14:30 module_7852-268D-3BF9
-rw-r--r--. 1 root  975  3716 Dec  5 14:30 module_7852-268D-3BF9_c8a766e280f9fc2b9e6a2ff2a0dcf8b5d37af725
-rw-r--r--. 1 root  975   628 Dec  5 14:30 softcard_925f67e72ea3c354cae4e6797bde3753d24e7744
-rwxrwxrwx. 1 root  975 40860 Dec  5 14:30 world

Removing debug pod ...
```

### 2.5.7. Handling passphrases when using OCS card protection or softcards

Part of the integration testing is to generate keys using OCS card production and softcard protections. The OCS cards and a softcard will require a passphrase when any key material gets generated inside the container. A containerized environment has no console to be able to type the passphrase when required. This guide provides a way in which this can take place inside the container. Two scripts have been created as examples to show how this can be performed: One for the OCS scenario and one for the softcard scenario. You must copy these scripts to the `nfast-kmdata` persistent volume so the pods that will use them have access.

Sample YAML files contains example files. Entrust does not provide support for using these files. You must adapt them to your system and use case:

- ocsexpect.sh
- softcardexpect.sh

1. Copy the `expect` scripts to the bin folder in the `nfast-kmdata` persistent volume.

```
% oc cp ocsexpect.sh ncop-test/ncop-test-dummy-48b6w:/opt/nfast/kmdata/bin/.

% oc cp softcardexpect.sh ncop-test/ncop-test-dummy-48b6w:/opt/nfast/kmdata/bin/.
```

2. Set the execute permissions on the files.

```
% oc debug pod/ncop-test-dummy-48b6w -- chmod +x /opt/nfast/kmdata/bin/ocsexpect.sh

% oc debug pod/ncop-test-dummy-48b6w -- chmod +x /opt/nfast/kmdata/bin/softcardexpect.sh
```

## 2.5.8. Create the application containers

This section describes how to create the application containers. The guide provides application containers examples for the following:

- `enquiry`
- `nfkminfo`
- Generating a key using module protection
- Generating a key using softcard protection
- Generating a key using OCS protection
- Running all the commands in a single container

The following YAML files are used to create the application containers:

- `pod_enquiry_container.yaml`
- `pod_nfkminfo_container.yaml`
- `pod_generatekeymodule_container.yaml`
- `pod_generatekeysoftcard_container.yaml`
- `pod_generatekeyocs_container.yaml`
- `pod_all_container.yaml`

Sample YAML files contains example files. Entrust does not provide support for using these files. You must adapt them to your system and use case.

The assumption is that you signed in to the container platform.

If using FIPS Level 3 world file, it is necessary to have the OCS cards available for FIPS authorization.

### 2.5.8.1. enquiry

Executes the `enquiry` command.

1. Create the application container with the image:

```
% oc create -f pod_enquiry_container.yaml

pod/ncop-test-enquiry-l6r5x created
```

2. Wait a short period of time, then verify that the pods are running:

```
% oc get pods

NAME                      READY   STATUS    RESTARTS   AGE
ncop-hwsp-kj299           1/1     Running   0          8m15s
ncop-test-dummy-ld8hl     1/1     Running   0          8m15s
ncop-test-enquiry-wlz5d   1/1     Running   0          6s
```

3. Run the application.

   A correct response confirms that both the hardserver and the HSM are up and running, and that the HSM is available. Retrieve the log output of the `enquiry` container:

```
% oc logs pod/ncop-test-enquiry-wlz5d

CONTAINER SCRIPT STARTED
Server:
 enquiry reply flags  none
 enquiry reply level  Six
 serial number        7852-268D-3BF9
 mode                 operational
 version              13.6.14
 speed index          20000
 rec. queue           514..812
 level one flags      Hardware HasTokens SupportsCommandState
 version string       13.6.14-390-0ff980a9, 13.4.5-751-56c6f1db, 13.6.14-388-95a16b77
 checked in           00000000691c7531 Tue Nov 18 13:31:29 2025
 level two flags      none
 max. write size      8192
 level three flags    KeyStorage
 level four flags     HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList HasSEE HasShareACL
HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds Type2Smartcard
ServerHasCreateClient HasInitialiseUnitEx AlwaysUseStrongPrimes Type3Smartcard HasKLF2
 module type code     0
 module type          nShield Hardserver
 product name         nFast server
 device name
 EnquirySix version   8
 impath kx groups     DHPrime1024 DHPrime3072 DHPrime3072Ex DHPrimeMODP3072 DHPrimeMODP3072mGCM
 feature ctrl flags   none
 features enabled     none
 version serial       0
 level six flags      none
 remote port (IPv4)   9004
 kneti hash           7ddfe541b1f026a32234246e3d52ddefd8ecfd4c
 rec. LongJobs queue  0
 SEE machine type     None
 supported KML types
 active modes         none
 remote port (IPv6)   9004

Module #1:
 enquiry reply flags  UnprivOnly
 enquiry reply level  Six
 serial number        7852-268D-3BF9
 mode                 operational
 version              13.4.5
 speed index          20000
 rec. queue           120..250
 level one flags      Hardware HasTokens SupportsCommandState SupportsHotReset
 version string       13.4.5-751-56c6f1db, 13.6.14-388-95a16b77
```

```
 checked in           00000000661e5e68 Tue Apr 16 11:18:00 2024
 level two flags      none
 max. write size      262152
 level three flags    KeyStorage
 level four flags     HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList HasSEE HasShareACL
HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds Type2Smartcard
ServerHasCreateClient HasInitialiseUnitEx AlwaysUseStrongPrimes Type3Smartcard HasKLF2
 module type code     14
 module type          nShield 5c
 product name         NH2096-0F
 device name          Rt1
 EnquirySix version   7
 impath kx groups     DHPrime1024 DHPrime3072 DHPrime3072Ex DHPrimeMODP3072
 feature ctrl flags   LongTerm
 features enabled     ForeignTokenOpen RemoteShare KISAAlgorithms StandardKM EllipticCurve ECCMQV
AcceleratedECC HSMSpeed2
 version serial       4
 connection status    OK
 connection info      esn = 7852-268D-3BF9; addr = INET/XX.XXX.XXX.XX/9004; ku hash =
ed28cc6bb5dfef39ff327002006a55d90be0758d, mech = Any
 image version        13.6.14-340-95a16b77
 level six flags      SerialConsoleAvailable Type3SmartcardRevB
 max exported modules 100
 rec. LongJobs queue  36
 SEE machine type     PowerPC64ELF
 supported KML types  DSAp1024s160 DSAp3072s256
 using impath kx grp  DHPrimeMODP3072mGCM
 active modes         UseFIPSApprovedInternalMechanisms AlwaysUseStrongPrimes FIPSLevel3Enforcedv2
 physical serial      46-U50625
 hardware part no     PCA10005-01 revision 03
 hardware status      OK
CONTAINER SCRIPT DONE
```

4. Delete the pod container:

```
% oc delete pod ncop-test-enquiry-wlz5d

pod "ncop-test-enquiry-wlz5d" deleted
```

### 2.5.8.2. nfkminfo

Executes the `nfkminfo` command.

1. Create the application container with the image:

```
% oc create -f pod_nfkminfo_container.yaml

pod/ncop-test-nfkminfo-nlntj created
```

2. Wait a short period of time, then verify that the pods are running:

```
% oc get pods

NAME                      READY   STATUS    RESTARTS   AGE
ncop-hwsp-kj299           1/1     Running   0          10m
ncop-test-dummy-ld8hl     1/1     Running   0          10m
```

```
ncop-test-nfkminfo-nlntj   1/1     Running   0          5s
```

3. Run the application.

   A correct response confirms that both the hardserver and the HSM are up and running, and that the HSM is available. Retrieve the log output of the `nfkminfo` container:

```
% oc logs pod/ncop-test-nfkminfo-nlntj

CONTAINER SCRIPT STARTED
World
 generation  2
 state       0x3737000c Initialised Usable Recovery !PINRecovery !ExistingClient RTC NVRAM FTO
AlwaysUseStrongPrimes !DisablePKCS1Padding !PpStrengthCheck !AuditLogging SEEDebug AdminAuthRequired
 n_modules   1
 hknso       0e4134b032886e6c2315086a386f6dabb54515e5
 hkm         b01a7d6ac910b720bf4319f5067a4569f087f81b (type Rijndael)
 hkmwk       c2be99fe1c77f1b75d48e2fd2df8dffc0c969bcb
 hkre        d00f8956fcda01bd4c7f539ee042ef6b5ac75917
 hkra        09e1980620bb94bb5501fee852dd83f1e148ba48
 hkfips      003e04e3c07fb5791f651c992da5527779159f87
 hkmc        f3341d182fb32c7aac75127f1c705da1414299e5
 hkrtc       da0fae6a6bd547644fce9368ab377b07f2ef164a
 hknv        e31db152d26f59fa47d8c18cddf0d502ecc7fda2
 hkdsee      7d28d99d3d6d9eccf555aed5a285af94a0eba7f1
 hkfto       990b794cf94cada7f56bd27c0f3e5fc4100d46c3
 hkmnull     0100000000000000000000000000000000000000
 ex.client   none
 k-out-of-n  1/15
 other quora m=1 r=1 nv=1 rtc=1 dsee=1 fto=1
 createtime  2023-07-20 18:00:03
 nso timeout 45 min
 ciphersuite DLf3072s256mAEScSP800131Ar1
 min pp      0 chars
 mode        fips1402level3

Module #1
 generation 2
 state      0x2 Usable
 flags      0x0 !ShareTarget
 n_slots    6
 esn        7852-268D-3BF9
 hkml       c8a766e280f9fc2b9e6a2ff2a0dcf8b5d37af725

Module #1 Slot #0 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x2 SupportsAuthentication
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

Module #1 Slot #1 IC 0
 generation    1
 phystype      SoftToken
 slotlistflags 0x0
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
```

```
 error         OK
No Cardset

Module #1 Slot #2 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x80002 SupportsAuthentication DynamicSlot
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

Module #1 Slot #3 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x80002 SupportsAuthentication DynamicSlot
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

Module #1 Slot #4 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x80002 SupportsAuthentication DynamicSlot
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

Module #1 Slot #5 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x80002 SupportsAuthentication DynamicSlot
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

No Pre-Loaded Objects
CONTAINER SCRIPT DONE
```

4. Delete the pod container:

```
% oc delete pod pod/ncop-test-nfkminfo-nlntj

pod "ncop-test-nfkminfo-nlntj" deleted
```

### 2.5.8.3. Generate a key using module protection

Executes the `generatekey` command using the module as the protection mechanism.

1. Create the application container with the image:

```
% oc create -f pod_generatekeymodule_container.yaml

pod/ncop-test-generatekeymodule-cph5d created
```

2. Wait a short period of time, then verify that the pods are running:

```
% oc get pods

NAME                                READY   STATUS    RESTARTS      AGE
ncop-hwsp-kj299                     1/1     Running   0             3h29m
ncop-test-dummy-ld8hl               1/1     Running   3 (29m ago)   3h29m
ncop-test-generatekeymodule-cph5d   1/1     Running   0             5s
```

3. Run the application.

   A correct response confirms that both the hardserver and the HSM are up and running, and that the HSM is available. Retrieve the log output of the container:

```
% oc logs pod/ncop-test-generatekeymodule-cph5d

CONTAINER SCRIPT STARTED
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)  65537
 plainname    Key name                          modulekey-6906e550-06d2-4b47-beec-ac851777101a
 nvram        Blob in NVRAM (needs ACS)         no
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ua8f0de1af47a70cd448e86aebccd379e888ed5842
`rocs' key recovery tool
Useful commands: `help', `help intro', `quit'.
rocs>   No. Name                    App        Protected by
    1 modulekey-6906e550-06d2- pkcs11     module
rocs>
CONTAINER SCRIPT DONE
```

4. Delete the pod container:

```
% oc delete pod pod/ncop-test-generatekeymodule-cph5d

pod "ncop-test-generatekeymodule-cph5d" deleted
```

### 2.5.8.4. Generate a key using softcard protection

Executes the `generatekey` command using the softcard as the protection mechanism.

1. Create the application container with the image:

```
% oc create -f pod_generatekeysoftcard_container.yaml

pod/ncop-test-generatekeysoftcard-bj8dl created
```

2. Wait a short period of time, then verify that the pods are running:

```
% oc get pods

NAME                                 READY   STATUS    RESTARTS      AGE
ncop-hwsp-kj299                      1/1     Running   0             3h34m
ncop-test-dummy-ld8hl                1/1     Running   3 (34m ago)   3h34m
ncop-test-generatekeysoftcard-bj8dl  1/1     Running   0             6s
```

3. Run the application.

   A correct response confirms that both the hardserver and the HSM are up and running, and that the HSM is available. Retrieve the log output of the container:

```
% oc logs pod/ncop-test-generatekeysoftcard-bj8dl

CONTAINER SCRIPT STARTED
spawn /opt/nfast/bin/generatekey -b -g -m1 pkcs11 plainname=softcardkey-c8a5575d-a10b-44fb-bc2b-
e77a5892027b type=rsa protect=softcard recovery=no size=2048 softcard=testSC
key generation parameters:
 operation    Operation to perform              generate
 application  Application                        pkcs11
 protect      Protected by                      softcard
 softcard     Soft card to protect key          testSC
 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-c8a5575d-a10b-44fb-bc2b-e77a5892027b
 nvram        Blob in NVRAM (needs ACS)         no
Please enter the pass phrase for softcard `testSC':

Please wait........

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uc925f67e72ea3c354cae4e6797bde3753d24e7744-
c4015b6c84e9f2325a2cf8dfb031d70abe7431a3
`rocs' key recovery tool
Useful commands: `help', `help intro', `quit'.
rocs>   No. Name                    App        Protected by
    1 modulekey-6906e550-06d2- pkcs11     module
-   2 softcardkey-c8a5575d-a10 pkcs11     testSC (testSC)
rocs>
CONTAINER SCRIPT DONE
```

4. Delete the pod container:

```
% oc delete pod pod/ncop-test-generatekeysoftcard-bj8dl
```

```
pod "ncop-test-generatekeysoftcard-bj8dl" deleted
```

### 2.5.8.5. Generate a key using OCS protection

Executes the `generatekey` command using OCS as the protection mechanism.

1. Create the application container with the image:

```
% oc create -f pod_generatekeyocs_container.yaml

pod/ncop-test-generatekeyocs-l6vjv created
```

2. Wait a short period of time, then verify that the pods are running:

```
% oc get pods

NAME                            READY   STATUS    RESTARTS      AGE
ncop-hwsp-kj299                 1/1     Running   0             3h38m
ncop-test-dummy-ld8hl           1/1     Running   3 (38m ago)   3h38m
ncop-test-generatekeyocs-l6vjv  1/1     Running   0             6s
```

3. Run the application.

   A correct response confirms that both the hardserver and the HSM are up and running, and that the HSM is available. Retrieve the log output of the container:

```
% oc logs pod/ncop-test-generatekeyocs-l6vjv

CONTAINER SCRIPT STARTED
spawn /opt/nfast/bin/generatekey -b -g -m1 pkcs11 plainname=ocskey-c45379cd-17fd-45b1-bfde-335d772f9f5c
type=rsa protect=token recovery=no size=2048 cardset=testOCS
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                      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                          ocskey-c45379cd-17fd-45b1-bfde-335d772f9f5c
 nvram        Blob in NVRAM (needs ACS)         no

Loading `testOCS':
 Module 1: 0 cards of 1 read
 Module 1 slot 2: `testOCS' #3
 Module 1 slot 0: empty
 Module 1 slot 3: empty
 Module 1 slot 4: empty
 Module 1 slot 5: empty
 Module 1 slot 2:- passphrase supplied - reading card
Card reading complete.

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ucedb3d45a28e5a6b22b033684ce589d9e198272c2-
```

```
ef32b426594e3285ea47b275e5839751fc1e719b
`rocs' key recovery tool
Useful commands: `help', `help intro', `quit'.
rocs>   No. Name                    App        Protected by
    1 modulekey-6906e550-06d2- pkcs11     module
-   2 softcardkey-c8a5575d-a10 pkcs11     testSC (testSC)
-   3 ocskey-c45379cd-17fd-45b pkcs11     testOCS
rocs>
CONTAINER SCRIPT DONE
```

4. Delete the pod container:

```
% oc delete pod pod/ncop-test-generatekeyocs-l6vjv

pod "ncop-test-generatekeyocs-l6vjv" deleted
```

#### 2.5.8.6. Run all commands in a single container

This example runs all commands inside a single container.

1. Create the application container with the image:

```
% oc create -f pod_all_container.yaml

pod/ncop-test-all-w9gpg created
```

2. Wait a short period of time, then verify that the pods are running:

```
% oc get pods

NAME                    READY   STATUS    RESTARTS      AGE
ncop-hwsp-kj299         1/1     Running   0             3h45m
ncop-test-all-w9gpg     1/1     Running   0             6s
ncop-test-dummy-ld8hl   1/1     Running   3 (44m ago)   3h45m
```

3. Run the application.

   A correct response confirms that both the hardserver and the HSM are up and running, and that the HSM is available. Retrieve the log output of the container:

```
% oc logs pod/ncop-test-all-w9gpg

CONTAINER SCRIPT STARTED

------------------ enquiry
Server:
 enquiry reply flags  none
 enquiry reply level  Six
 serial number        7852-268D-3BF9
 mode                 operational
 version              13.6.14
 speed index          20000
 rec. queue           514..812
```

```
 level one flags      Hardware HasTokens SupportsCommandState
 version string       13.6.14-390-0ff980a9, 13.4.5-751-56c6f1db, 13.6.14-388-95a16b77
 checked in           00000000691c7531 Tue Nov 18 13:31:29 2025
 level two flags      none
 max. write size      8192
 level three flags    KeyStorage
 level four flags     HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList HasSEE HasShareACL
HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds Type2Smartcard
ServerHasCreateClient HasInitialiseUnitEx AlwaysUseStrongPrimes Type3Smartcard HasKLF2
 module type code     0
 module type          nShield Hardserver
 product name         nFast server
 device name
 EnquirySix version   8
 impath kx groups     DHPrime1024 DHPrime3072 DHPrime3072Ex DHPrimeMODP3072 DHPrimeMODP3072mGCM
 feature ctrl flags   none
 features enabled     none
 version serial       0
 level six flags      none
 remote port (IPv4)   9004
 kneti hash           7ddfe541b1f026a32234246e3d52ddefd8ecfd4c
 rec. LongJobs queue  0
 SEE machine type     None
 supported KML types
 active modes         none
 remote port (IPv6)   9004

Module #1:
 enquiry reply flags  UnprivOnly
 enquiry reply level  Six
 serial number        7852-268D-3BF9
 mode                 operational
 version              13.4.5
 speed index          20000
 rec. queue           120..250
 level one flags      Hardware HasTokens SupportsCommandState SupportsHotReset
 version string       13.4.5-751-56c6f1db, 13.6.14-388-95a16b77
 checked in           00000000661e5e68 Tue Apr 16 11:18:00 2024
 level two flags      none
 max. write size      262152
 level three flags    KeyStorage
 level four flags     HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList HasSEE HasShareACL
HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds Type2Smartcard
ServerHasCreateClient HasInitialiseUnitEx AlwaysUseStrongPrimes Type3Smartcard HasKLF2
 module type code     14
 module type          nShield 5c
 product name         NH2096-0F
 device name          Rt1
 EnquirySix version   7
 impath kx groups     DHPrime1024 DHPrime3072 DHPrime3072Ex DHPrimeMODP3072
 feature ctrl flags   LongTerm
 features enabled     ForeignTokenOpen RemoteShare KISAAlgorithms StandardKM EllipticCurve ECCMQV
AcceleratedECC HSMSpeed2
 version serial       4
 connection status    OK
 connection info      esn = 7852-268D-3BF9; addr = INET/XX.XXX.XXX.XX/9004; ku hash =
ed28cc6bb5dfef39ff327002006a55d90be0758d, mech = Any
 image version        13.6.14-340-95a16b77
 level six flags      SerialConsoleAvailable Type3SmartcardRevB
 max exported modules 100
 rec. LongJobs queue  36
 SEE machine type     PowerPC64ELF
 supported KML types  DSAp1024s160 DSAp3072s256
 using impath kx grp  DHPrimeMODP3072mGCM
 active modes         UseFIPSApprovedInternalMechanisms AlwaysUseStrongPrimes FIPSLevel3Enforcedv2
 physical serial      46-U50625
 hardware part no     PCA10005-01 revision 03
```

```
 hardware status      OK

------------------ nfkminfo
World
 generation  2
 state       0x3737000c Initialised Usable Recovery !PINRecovery !ExistingClient RTC NVRAM FTO
AlwaysUseStrongPrimes !DisablePKCS1Padding !PpStrengthCheck !AuditLogging SEEDebug AdminAuthRequired
 n_modules   1
 hknso       0e4134b032886e6c2315086a386f6dabb54515e5
 hkm         b01a7d6ac910b720bf4319f5067a4569f087f81b (type Rijndael)
 hkmwk       c2be99fe1c77f1b75d48e2fd2df8dffc0c969bcb
 hkre        d00f8956fcda01bd4c7f539ee042ef6b5ac75917
 hkra        09e1980620bb94bb5501fee852dd83f1e148ba48
 hkfips      003e04e3c07fb5791f651c992da5527779159f87
 hkmc        f3341d182fb32c7aac75127f1c705da1414299e5
 hkrtc       da0fae6a6bd547644fce9368ab377b07f2ef164a
 hknv        e31db152d26f59fa47d8c18cddf0d502ecc7fda2
 hkdsee      7d28d99d3d6d9eccf555aed5a285af94a0eba7f1
 hkfto       990b794cf94cada7f56bd27c0f3e5fc4100d46c3
 hkmnull     0100000000000000000000000000000000000000
 ex.client   none
 k-out-of-n  1/15
 other quora m=1 r=1 nv=1 rtc=1 dsee=1 fto=1
 createtime  2023-07-20 18:00:03
 nso timeout 45 min
 ciphersuite DLf3072s256mAEScSP800131Ar1
 min pp      0 chars
 mode        fips1402level3

Module #1
 generation 2
 state      0x2 Usable
 flags      0x0 !ShareTarget
 n_slots    6
 esn        7852-268D-3BF9
 hkml       c8a766e280f9fc2b9e6a2ff2a0dcf8b5d37af725

Module #1 Slot #0 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x2 SupportsAuthentication
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

Module #1 Slot #1 IC 0
 generation    1
 phystype      SoftToken
 slotlistflags 0x0
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

Module #1 Slot #2 IC 3
 generation    1
 phystype      SmartCard
 slotlistflags 0x180002 SupportsAuthentication DynamicSlot Associated
 state         0x5 Operator
 flags         0x10000
 shareno       3
 shares        LTU(PIN) LTFIPS
```

```
 error         OK
Cardset
 name          "testOCS"
 k-out-of-n    1/5
 flags         NotPersistent PINRecoveryForbidden(disabled) !RemoteEnabled
 timeout       none
 card names    "" "" "" "" ""
 hkltu         edb3d45a28e5a6b22b033684ce589d9e198272c2
 gentime       2023-07-20 18:50:48

Module #1 Slot #3 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x80002 SupportsAuthentication DynamicSlot
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

Module #1 Slot #4 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x80002 SupportsAuthentication DynamicSlot
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

Module #1 Slot #5 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x80002 SupportsAuthentication DynamicSlot
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

No Pre-Loaded Objects

------------------ Generating module key
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)  65537
 plainname    Key name                           modulekey-537f0b13-a486-43c2-9d85-76d9ef0649ea
 nvram        Blob in NVRAM (needs ACS)          no
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ua09366c5b228619885d0349eb0c448e56352a120f

------------------ Generating ocs key
spawn /opt/nfast/bin/generatekey -b -g -m1 pkcs11 plainname=ocskey-537f0b13-a486-43c2-9d85-76d9ef0649ea
type=rsa protect=token recovery=no size=2048 cardset=testOCS
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                       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                           ocskey-537f0b13-a486-43c2-9d85-76d9ef0649ea
 nvram        Blob in NVRAM (needs ACS)          no

Loading `testOCS':
 Module 1: 0 cards of 1 read
 Module 1 slot 2: `testOCS' #3
 Module 1 slot 0: empty
 Module 1 slot 3: empty
 Module 1 slot 4: empty
 Module 1 slot 5: empty
 Module 1 slot 2:- passphrase supplied - reading card
Card reading complete.

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ucedb3d45a28e5a6b22b033684ce589d9e198272c2-
96b0375b27eff8ebdcad43005734c1efe5d991d7

------------------ Generating softcard key
spawn /opt/nfast/bin/generatekey -b -g -m1 pkcs11 plainname=softcardkey-537f0b13-a486-43c2-9d85-
76d9ef0649ea type=rsa protect=softcard recovery=no size=2048 softcard=testSC
key generation parameters:
 operation    Operation to perform               generate
 application  Application                        pkcs11
 protect      Protected by                       softcard
 softcard     Soft card to protect key           testSC
 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-537f0b13-a486-43c2-9d85-76d9ef0649ea
 nvram        Blob in NVRAM (needs ACS)          no
Please enter the pass phrase for softcard `testSC':

Please wait........

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uc925f67e72ea3c354cae4e6797bde3753d24e7744-
9f2f1d691a7f2f3ada7834822fa665b66cc6c2cd

------------------ list keys
`rocs' key recovery tool
Useful commands: `help', `help intro', `quit'.
rocs>   No. Name                     App        Protected by
    1 modulekey-6906e550-06d2- pkcs11     module
-   2 softcardkey-c8a5575d-a10 pkcs11     testSC (testSC)
-   3 ocskey-c45379cd-17fd-45b pkcs11     testOCS
    4 modulekey-537f0b13-a486- pkcs11     module
-   5 ocskey-537f0b13-a486-43c pkcs11     testOCS
-   6 softcardkey-537f0b13-a48 pkcs11     testSC (testSC)
rocs>
CONTAINER SCRIPT DONE
```

4. Delete the pod container:

```
% oc delete pod pod/ncop-test-all-w9gpg

pod "ncop-test-all-w9gpg" deleted
```

## 2.6. FIPS Level 3 recommendations

Here are some recommendations when a FIPS Level 3 world file is used for the HSM configuration:

- Create an OCS card 1/N where N is the number of HSMs being used in the configuration.
- All HSMs in the configuration must use the same world file.
- Leave the OCS card inserted on each HSM used in the configuration.
- The persistent volume must have the world, module, card, cards, and cardlist file.
- The OCS card is used for FIPS authorization only if not using OCS card protection.
- The OCS card must be present any time new key material is created.
- Remove the Admin card if inserted on slot 0 and only use the OCS card for FIPS authorization.

# Chapter 3. 5c 10G Configuration using KeySafe 5

The 5c 10G HSM does not use an RFS for its configuration. Instead, KeySafe 5 is used for configuration and management of the HSM. This section describes what needs to take place in the KeySafe 5 WebUI so the 5c 10G can be used by this integration. The assumption is that KeySafe 5 is already installed and that you have already provisioned the 5c 10G according to the 5c 10G Quick Start Guide.

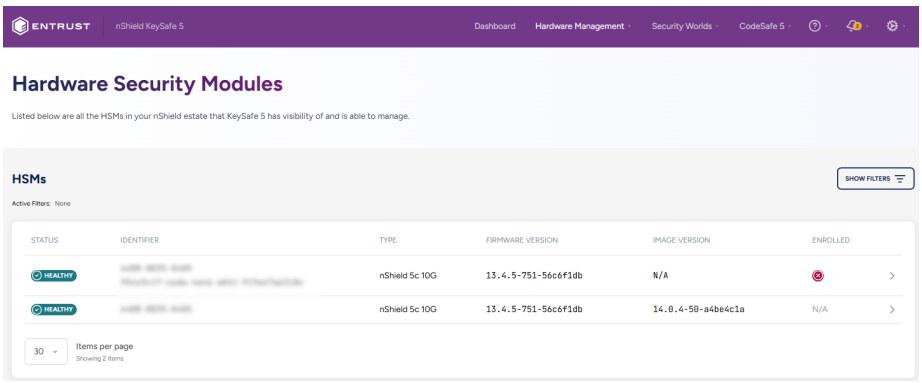
## 3.1. Add the OpenShift nodes and deployment server as clients of the HSM.

You will need to add all OpenShift nodes and the deployment server as clients of the HSM. Since you don't know what node will be used by Openshift to deploy the pods, all nodes must be added as clients.

1. Log in to the KeySafe 5 UI that you have installed:

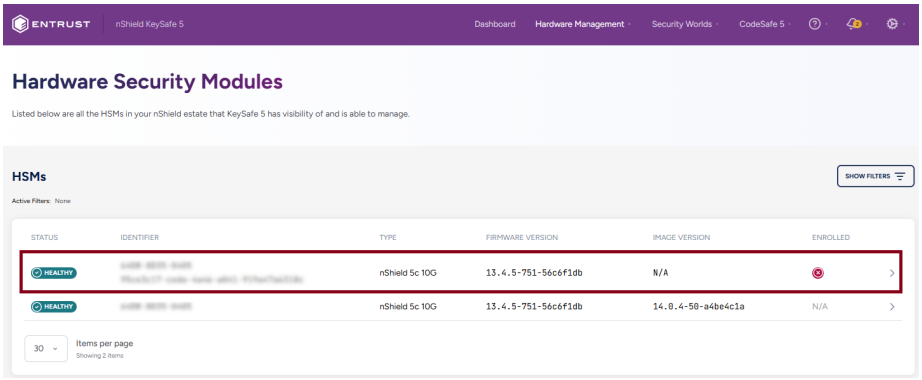
```
https://xx.xxx.xxx.xxx:18080
```

2. Select **Hardware Management** > **HSMs**.

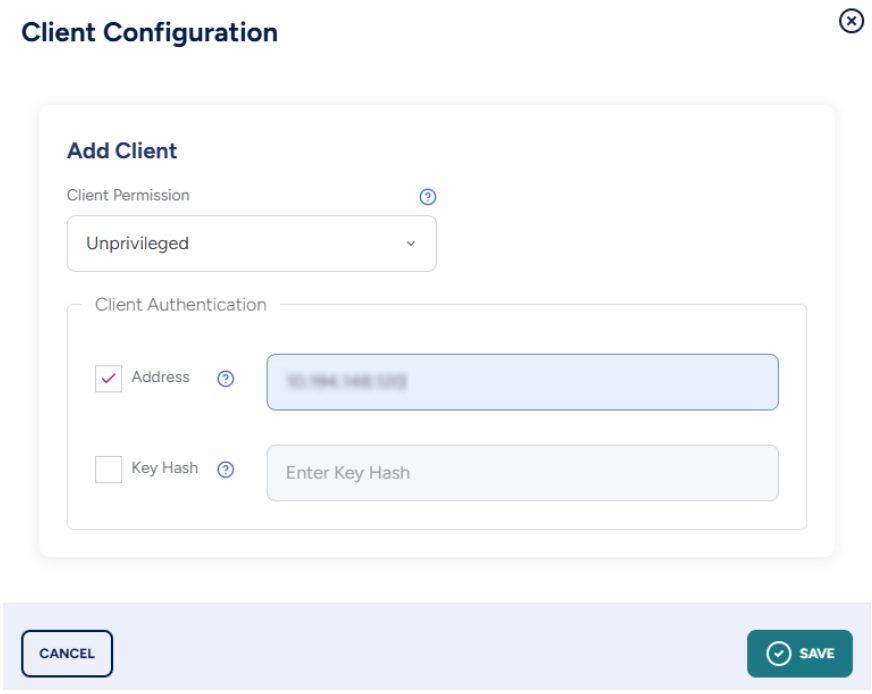



3. Select the **Tenant** HSM.

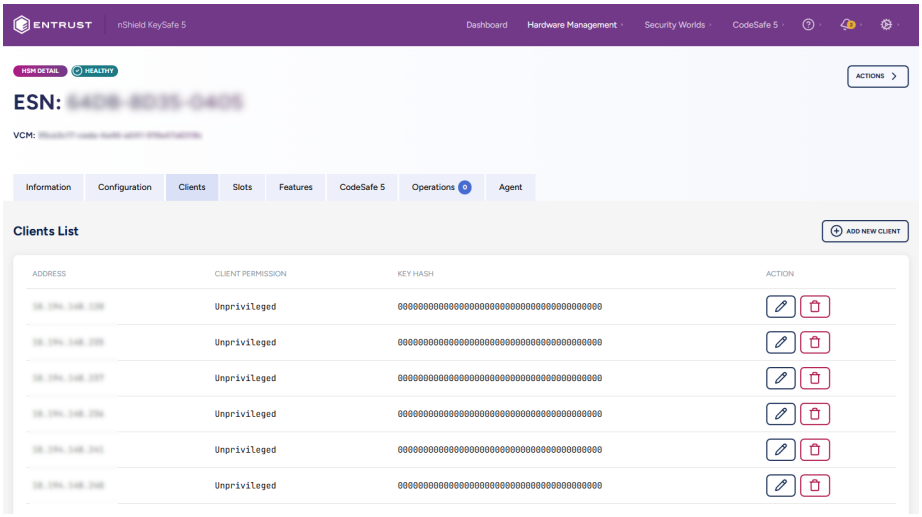
   The Tenant HSM is the HSM with the ESN displayed with the KeyHash.




4. Select the **Clients** tab.
5. Select **Add New Client**.
6. In the **Client Configuration** dialog:
    a. For **Client Permission**: select **unprivileged**.
    b. For **Client Authentication**: Select **Address** and enter the IP address.
    c. Select **Save**.




7. Once saved, the new client should be listed.




8. Do this for the deployment server and every OpenShift node.

   Now the HSM is ready to be enrolled at the client.

## 3.2. TVD - Remote Administration Client

If you have to present Cards to clients of the HSM using a TVD, you must use **RA Enable** cards. You do this by Enabling **Dynamic Slots** on the HSM.

1. In the KeySafe 5 UI, in the Tenant HSM, select the **Configuration** Tab.
2. Select **Dynamic Slots**.
3. Select **Edit**.
4. In the **Edit Dynamic Slots** Dialog:
   a. For **Number of Dynamic Slots**: Enter **4**.
   b. Select **Save**.
5. The KeySafe 5 UI will indicate the **Module must be cleared** for the configuration changes to take effect.
6. Select **Close**.
7. Select **Actions** > **Clear HSM** for the changes to take effect.

# Chapter 4. Sample YAML files

The yaml files here are examples that can be used to deploy the infrastructure needed and to perform the integration.

You will want to deploy some application pods in the same host as the `hardserver` pod. You will see an `affinity` section in some pod yaml files. This section describes that all pods with label `nshield` will be deployed together in the same node. Another important attribute that has been added is the `serviceAccountName`, that tells the service account the pods will run as.

## 4.1. project.yaml

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

## 4.2. cm.yaml

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

    [nethsm_imports]
    local_module=1
    remote_esn=5F08-02E0-D947
    remote_ip=xx.xxx.xxx.xx
    remote_port=9004
    keyhash=732523000c324c8a674236d1ad783a4dae0179fe
    privileged=0
```

## 4.3. pv_nfast_kmdata_definition.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfast-kmdata
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1G
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /opt/nfast/kmdata
```

## 4.4. pv_nfast_sockets_definition.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfast-sockets
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1G
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /opt/nfast/sockets
```

## 4.5. pv_nfast_kmdata_claim.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name : nfast-kmdata
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: local-storage
  resources:
    requests:
      storage: 1G
  storageClassName: manual
```

## 4.6. pv_nfast_sockets_claim.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
```

```
metadata:
  name : nfast-sockets
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: local-storage
  resources:
    requests:
      storage: 1G
  storageClassName: manual
```

## 4.7. pod_dummy.yaml

```
kind: Pod
apiVersion: v1
metadata:
  generateName: ncop-test-dummy-
  namespace: ncop-test
  labels:
    app: nshield
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: nshield
          topologyKey: "kubernetes.io/hostname"
  serviceAccountName: ncop-sa
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-container
      securityContext:
        privileged: true
      command:
        - sh
        - '-c'
        - sleep 3600
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-container
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - mountPath: /opt/nfast/sockets
          name: ncop-sockets
        - mountPath: /opt/nfast/kmdata
          name: ncop-kmdata
  securityContext: {}
  volumes:
    - name: ncop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: ncop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```

## 4.8. pod_hwsp.yaml

```yaml
kind: Pod
apiVersion: v1
metadata:
  generateName: ncop-hwsp-
  namespace: ncop-test
  labels:
    app: nshield
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: nshield
          topologyKey: "kubernetes.io/hostname"
  serviceAccountName: ncop-sa
  securityContext: {}
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-hwsp
      securityContext:
        privileged: true
      image: <external-docker-registry-IP-address>/cv-nshield-hwsp-container
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: ncop-config
          mountPath: /opt/nfast/kmdata/config
        - name: ncop-hardserver
          mountPath: /opt/nfast/kmdata/hardserver.d
        - name: ncop-sockets
          mountPath: /opt/nfast/sockets
  volumes:
    - name: ncop-config
      configMap:
        name: config
        defaultMode: 420
    - name: ncop-hardserver
      emptyDir: {}
    - name: ncop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
```

## 4.9. pod_enquiry_container.yaml

```yaml
kind: Pod
apiVersion: v1
metadata:
  generateName: ncop-test-enquiry-
  namespace: ncop-test
  labels:
    app: nshield
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
```

```
        - labelSelector:
            matchLabels:
              app: nshield
          topologyKey: "kubernetes.io/hostname"
  serviceAccountName: ncop-sa
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-container
      securityContext:
        privileged: true
      command: ["sh", "-c"]
      args:
        - echo CONTAINER SCRIPT STARTED;
          /opt/nfast/bin/enquiry;
          echo CONTAINER SCRIPT DONE && sleep 3600
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-container
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - mountPath: /opt/nfast/sockets
          name: ncop-sockets
        - mountPath: /opt/nfast/kmdata
          name: ncop-kmdata
  securityContext: {}
  volumes:
    - name: ncop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: ncop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```

## 4.10. pod_nfkminfo_container.yaml

```
kind: Pod
apiVersion: v1
metadata:
  generateName: ncop-test-nfkminfo-
  namespace: ncop-test
  labels:
    app: nshield
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: nshield
          topologyKey: "kubernetes.io/hostname"
  serviceAccountName: ncop-sa
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-container
      securityContext:
        privileged: true
      command: ["sh", "-c"]
      args:
        - echo CONTAINER SCRIPT STARTED;
```

```
          /opt/nfast/bin/nfkminfo;
          echo CONTAINER SCRIPT DONE && sleep 3600
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-container
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - mountPath: /opt/nfast/sockets
          name: ncop-sockets
        - mountPath: /opt/nfast/kmdata
          name: ncop-kmdata
  securityContext: {}
  volumes:
    - name: ncop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: ncop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```

## 4.11. pod_generatekeymodule_container.yaml

This example calls the `generatekey` command and uses some of the variables in the `cardcred` secret.

```
kind: Pod
apiVersion: v1
metadata:
  generateName: ncop-test-generatekeymodule-
  namespace: ncop-test
  labels:
    app: nshield
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: nshield
          topologyKey: "kubernetes.io/hostname"
  serviceAccountName: ncop-sa
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-container
      securityContext:
        privileged: true
      envFrom:
        - secretRef:
            name: cardcred
      env:
        - name: MY_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      command: ["sh", "-c"]
      args:
        - echo CONTAINER SCRIPT STARTED;
          /opt/nfast/bin/generatekey --generate --batch -m$CARDMODULE pkcs11 protect=module type=rsa size=2048
pubexp=65537 plainname=modulekey-$MY_POD_UID nvram=no recovery=yes;
```

```
          echo "list keys" | /opt/nfast/bin/rocs;
          echo CONTAINER SCRIPT DONE && sleep 3600
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-container
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - mountPath: /opt/nfast/sockets
          name: ncop-sockets
        - mountPath: /opt/nfast/kmdata
          name: ncop-kmdata
  securityContext: {}
  volumes:
    - name: ncop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: ncop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```

## 4.12. pod_generatekeysoftcard_container.yaml

This example calls the `sofcardexpect.sh` script, which does the call to the `generatekey` command. The script uses environment variables created in the pod, coming from the `cardcred` secret. One of the variables is the `CARDPP` variable which is the passphrase for the softcard.

```
kind: Pod
apiVersion: v1
metadata:
  generateName: ncop-test-generatekeysoftcard-
  namespace: ncop-test
  labels:
    app: nshield
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: nshield
          topologyKey: "kubernetes.io/hostname"
  serviceAccountName: ncop-sa
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-container
      securityContext:
        privileged: true
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-container
      envFrom:
        - secretRef:
            name: cardcred
      env:
        - name: MY_POD_UID
          valueFrom:
            fieldRef:
```

```
            fieldPath: metadata.uid
    command: ["sh", "-c"]
    args:
      - echo CONTAINER SCRIPT STARTED;
        /opt/nfast/kmdata/bin/softcardexpect.sh $CARDMODULE $SOFTCARD $SOFTCARDKEY-$MY_POD_UID;
        echo "list keys" | /opt/nfast/bin/rocs;
        echo CONTAINER SCRIPT DONE && sleep 3600
    ports:
      - containerPort: 8080
        protocol: TCP
    resources: {}
    volumeMounts:
      - mountPath: /opt/nfast/sockets
        name: ncop-sockets
      - mountPath: /opt/nfast/kmdata
        name: ncop-kmdata
  securityContext: {}
  volumes:
    - name: ncop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: ncop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```

## 4.13. pod_generatekeyocs_container.yaml

This example calls the `ocsexpect.sh` script, which does the call to the `generatekey` command. The script uses environment variables created in the pod, coming from the `cardcred` secret. One of the variables is the `CARDPP` variable which is the passphrase for the softcard.

```
kind: Pod
apiVersion: v1
metadata:
  generateName: ncop-test-generatekeyocs-
  namespace: ncop-test
  labels:
    app: nshield
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: nshield
          topologyKey: "kubernetes.io/hostname"
  serviceAccountName: ncop-sa
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-container
      securityContext:
        privileged: true
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-container
      envFrom:
        - secretRef:
            name: cardcred
      env:
```

```
        - name: MY_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      command: ["sh", "-c"]
      args:
        - echo CONTAINER SCRIPT STARTED;
          /opt/nfast/kmdata/bin/ocsexpect.sh $CARDMODULE $OCS $OCSKEY-$MY_POD_UID;
          echo "list keys" | /opt/nfast/bin/rocs;
          echo CONTAINER SCRIPT DONE && sleep 3600
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - mountPath: /opt/nfast/sockets
          name: ncop-sockets
        - mountPath: /opt/nfast/kmdata
          name: ncop-kmdata
  securityContext: {}
  volumes:
    - name: ncop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: ncop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```

## 4.14. pod_rocs_container.yaml

```
kind: Pod
apiVersion: v1
metadata:
  generateName: ncop-test-rocs-
  namespace: ncop-test
  labels:
    app: nshield
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: nshield
          topologyKey: "kubernetes.io/hostname"
  serviceAccountName: ncop-sa
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-container
      securityContext:
        privileged: true
      command: ["sh", "-c"]
      args:
        - echo CONTAINER SCRIPT STARTED;
          echo "list keys" | /opt/nfast/bin/rocs;
          echo CONTAINER SCRIPT DONE && sleep 3600
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-container
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
```

```
      volumeMounts:
        - mountPath: /opt/nfast/sockets
          name: ncop-sockets
        - mountPath: /opt/nfast/kmdata
          name: ncop-kmdata
  securityContext: {}
  volumes:
    - name: ncop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: ncop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```

## 4.15. pod_all_container.yaml

```
kind: Pod
apiVersion: v1
metadata:
  generateName: ncop-test-all-
  namespace: ncop-test
  labels:
    app: nshield
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: nshield
          topologyKey: "kubernetes.io/hostname"
  serviceAccountName: ncop-sa
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-container
      securityContext:
        privileged: true
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-container
      envFrom:
        - secretRef:
            name: cardcred
      env:
        - name: MY_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      command: ["sh", "-c"]
      args:
        - echo CONTAINER SCRIPT STARTED;
          echo;
          echo ------------------ enquiry;
          /opt/nfast/bin/enquiry;
          echo;
          echo ------------------ nfkminfo;
          /opt/nfast/bin/nfkminfo;
          echo;
          echo ------------------ Generating module key;
          /opt/nfast/bin/generatekey --generate --batch -m$CARDMODULE pkcs11 protect=module type=rsa size=2048
pubexp=65537 plainname=modulekey-$MY_POD_UID nvram=no recovery=yes;
          echo;
          echo ------------------ Generating ocs key;
```

```
          /opt/nfast/kmdata/bin/ocsexpect.sh $CARDMODULE $OCS $OCSKEY-$MY_POD_UID;
          echo;
          echo ------------------ Generating softcard key;
          /opt/nfast/kmdata/bin/softcardexpect.sh $CARDMODULE $SOFTCARD $SOFTCARDKEY-$MY_POD_UID;
          echo;
          echo ------------------ list keys;
          echo "list keys" | /opt/nfast/bin/rocs;
          echo CONTAINER SCRIPT DONE && sleep 3600
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - mountPath: /opt/nfast/sockets
          name: ncop-sockets
        - mountPath: /opt/nfast/kmdata
          name: ncop-kmdata
  securityContext: {}
  volumes:
    - name: ncop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: ncop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```

## 4.16. ocsexpect.sh

```
#!/usr/bin/expect
# Script to generate a key protected by an OCS card.
# You must pass the module, OCS name and the keyname to be created.
# The OCS Password is passed via the environment variable OCSPP
#
set MODULE [lindex $argv 0]
set OCS [lindex $argv 1]
set KEYNAME [lindex $argv 2]
sleep 2
spawn /opt/nfast/bin/generatekey -b -g -m$MODULE pkcs11 plainname=$KEYNAME type=rsa protect=token recovery=no
size=2048 cardset=$OCS
expect "Enter passphrase:"
sleep 1
send -- "$env(CARDPP)\r"
expect eof
```

## 4.17. softcardexpect.sh

```
#!/usr/bin/expect
# Script to generate a key protected by a Softcard card.
# You must pass the module, softcard name and the keyname to be created.
# The softcard Password is passed via the environment variable SOFTCARDPP
#
set MODULE [lindex $argv 0]
set SOFTCARD [lindex $argv 1]
set KEYNAME [lindex $argv 2]
sleep 2
spawn /opt/nfast/bin/generatekey -b -g -m$MODULE pkcs11 plainname=$KEYNAME type=rsa protect=softcard recovery=no
size=2048 softcard=$SOFTCARD
expect "pass phrase for softcard"
```

```
sleep 1
send -- "$env(CARDPP)\r"
expect eof
```

# Chapter 5. Additional resources and related products

## 5.1. nShield HSMs

## 5.2. nShield as a Service

## 5.3. nShield Container Option Pack

## 5.4. Entrust products

## 5.5. nShield product documentation