Mirantis Kubernetes Engine

nShield® HSM Integration Guide

2024-10-21

Table of Contents

Chapter 1. Introduction

This guide describes the steps to integrate the nShield Container Option Pack (nCOP) with Mirantis Kubernetes Engine. The nCOP provides application developers, within a container-based Mirantis Kubernetes Engine environment, the ability to access the cryptographic functionality of an nShield Hardware Security Module (HSM).

1.1. Product configurations

We have successfully tested nShield HSM integration with Mirantis Kubernetes Engine in the following configurations:

Software	Version
nCOP	1.1.2
Operating System	Red Hat Enterprise Linux release 9.4 (Plow)
Mirantis Kubernetes Engine	3.7.4
Mirantis Container Runtime	23.0.14

1.2. Supported nShield hardware and software versions

We have successfully tested with the following nShield hardware and software versions:

1.2.1. Connect XC

Security World Software	Firmware	Image	OCS	Softcard	Module
13.6.3	12.72.1 (FIPS 140-2 certified)	13.4.5	✓	✓	✓

1.2.2. nShield 5C

Security World Software	Firmware	Image	OCS	Softcard	Module
13.6.3	13.2.4 (FIPS 140-3 certified)	13.6.1	✓	✓	✓

1.3. Supported nShield HSM functionality

Feature	Support
Module-only key	Yes
OCS cards	Yes
Softcards	Yes
nSaaS	Yes
FIPS 140 Level 3	Yes

1.4. Requirements

Before installing these products, read the associated documentation:

- For the nShield HSM: *Installation Guide* and *User Guide*.
- If nShield Remote Administration is to be used: *nShield Remote Administration User Guide*.
- *nShield Container Option Pack User Guide*.
- MCR documentation (https://docs.mirantis.com/mcr/23.0/overview.html)
- MKE documentation (https://docs.mirantis.com/mke/3.7/overview.html).
- kubectl documentation (https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/)

Furthermore, the following design decisions have an impact on how the HSM is installed and configured:

- Whether your Security World must comply with FIPS 140 Level 3 standards.
  - If using FIPS 140 Level 3, it is advisable to create an OCS for FIPS authorization. For information about limitations on FIPS authorization, see the *Installation Guide* of the nShield HSM.

- Whether to instantiate the Security World as recoverable or not.

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

1.5. More information

For more information about OS support, contact your Mirantis 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

2.1. Prerequisites

Before you can use nCOP and pull the nCOP container images to the external registry, complete the following steps:

1. Install the Mirantis Container Runtime on the host machine. This can be a VM running Red Hat 9 or other compatible Operating Systems.
2. Install the Mirantis Kubernetes Engine on the host machine.
3. Install kubectl on the host machine.
4. Set up the HSM. See the Installation Guide for your HSM.
5. Configure the HSM(s) to have the IP address of your container host machine as a client.
6. 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 a persistent volume accessible by the application containers are given when you create the persistent volume during the deployment of MKE.

2.2. Install nCOP

Install nCOP and create the containers that contain your application. For the purpose of this guide you will need the nCOP hardserver container and your application container. In this guide they are referred to as the *nshield-hwsp* and *nshield-app* containers. For instructions, see the *nShield Container Option Pack User Guide*.

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

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. Note the following items:

- 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 13.6.3 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):

   ```
   % cd /opt/ncop
   % sudo ./make-nshield-hwsp --tag nshield-hwsp-container:13.6.3 --from ubuntu /mnt/iso1
   % sudo ./make-nshield-application --tag nshield-app-container:13.6.3 --from ubuntu /mnt/iso1
   ```

3. Validate the images have been built:

   ```
   % sudo docker images
   ```

   > You should see the two 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:13.6.3 &
% dmountpoint=`sudo docker volume inspect --format '{{ .Mountpoint }}' socket1`
% export NFAST_SERVER=$dmountpoint/nserver
% /opt/nfast/bin/enquiry
```

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:13.6.3 /opt/nfast/bin/enquiry
```

2.4. Push the nCOP container images to an internal Docker registry

You will need to register the nCOP container images you created to a Docker registry so they can be used when you deploy the Kubernetes pods later. In this guide, the external registry is <docker-registry-address>. Distribution of the nCOP container image is not permitted because the software components are under strict export controls.

To deploy an nCOP container images for use with Mirantis Kubernetes Engine:

1. Log in to the container host machine server as root, and launch a terminal window. We assume that you have built the nCOP container images in this host and that they are available locally in Docker. They are: nshield-hwsp-container:13.6.3 and nshield-app-container:13.6.3.
2. Log in to the Docker registry.

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

3. Register the images:

a. Tag the images:

```
% sudo docker tag nshield-hwsp-container:13.6.3 <docker-registry-address>/nshield-hwsp
% sudo docker tag nshield-app-container:13.6.3 <docker-registry-address>/nshield-app
```

b. Push the images to the registry:

```
% sudo docker push <docker-registry-address>/nshield-hwsp
% sudo docker push <docker-registry-address>/nshield-app
```

c. Remove the local images:

```
% sudo docker rmi <docker-registry-address>/nshield-hwsp
% sudo docker rmi <docker-registry-address>/nshield-app
```

d. List the images:

```
% sudo docker images
```

e. Pull the images from the registry:

```
% sudo docker pull <docker-registry-address>/nshield-hwsp
% sudo docker pull <docker-registry-address>/nshield-app
```

f. List the images:

```
% sudo docker images
```

2.5. Create the registry secrets

At the beginning of our process, we created nCOP Docker containers and we pushed them to our internal Docker registry. Now it is necessary to let MKE know about how to authenticate to that registry.

1. Create the secret.

```
% kubectl create secret generic regcred --from-file= dockerconfigjson=/home/<YOUR USER
ID>/.docker/config.json --type=kubernetes.io/dockerconfigjson
```

2. Check if the secret was created.

```
% kubectl get secret regcred --output=yaml
```

2.6. 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.

```
% kubectl 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

% kubectl get secret cardcred

NAME       TYPE     DATA   AGE
cardcred   Opaque   6      0s

% kubectl get secret cardcred -o YAML

apiVersion: v1
data:
  CARDMODULE: MQ==
  CARDPP: MTIz
  OCS: dGVzdE9DUw==
  OCSKEY: b2Nza2V5
  SOFTCARD: dGVzdFND
  SOFTCARDKEY: c29mdGNhcmRrZXk=
kind: Secret
metadata:
  creationTimestamp: "2024-09-19T19:28:30Z"
  name: cardcred
  namespace: default
  resourceVersion: "19426"
  uid: 04cb64ce-9615-42e1-a002-bbf876d7aa55
type: Opaque
```

2.7. Create the Configuration Map for the HSM details

We have created a `configmap.yaml` file that can be modified according to the HSM you are using. Edit the file accordingly.

> This integration was tested using kubectl commands for generating kubernetes objects with yaml files. The MKE web ui provides an alternative interface that can be used to generate these objects, and view them. See MKE documentation for more information.

For example:

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

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

Make sure you update the following fields: **remote_esn**, **remote_ip** and **keyhash**. These must match the information from the HSM being used in the integration.

1. Create the Config Map.

```
% kubectl apply -f configmap.yaml

configmap/config created
```

2. Verify the config map was created successfully.

```
% kubectl describe configmap/config

Name:         config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
config:

syntax-version=1

[nethsm_imports]
local_module=1
remote_esn=7852-268D-3BF9
remote_ip=1X.19X.1XX.XX
remote_port=9004
keyhash=ed28cc6bb5dfef39ff327002006a55d90be0758d
privileged=0



BinaryData
====

Events:  <none>
```

2.8. Create the MKE persistent Volumes

This section describes how the persistent volumes is created in MKE. We will need to create the kmdata persistent volume and the socket persistent volume.

> Before you proceed with the creation of the persistent volume, you must create the directory `/opt/nfast/kmdata/local` in your host machine and copy the Security World and module files to it.

The example YAML files below are used to create and claim the persistent volume.

2.8.1. Create the kmdata persistent volume

- The `persistent_volume_kmdata_definition.yaml` file:

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

- The `persistent_volume_kmdata_claim.yaml` file:

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

1. Apply the definition file to MKE.

```
% kubectl apply -f persistent_volume_kmdata_definition.yaml

persistentvolume/nfast-kmdata created
```

2. Verify the persistent volume has been created.

```
% kubectl get pv

NAME          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON
AGE
nfast-kmdata   1G         RWX            Retain           Available           manual
18s
```

3. Create the claim.

```
% kubectl apply -f persistent_volume_kmdata_claim.yaml

persistentvolumeclaim/nfast-kmdata created
```

4. Verify the claim has been created.

```
% kubectl get pvc

NAME          STATUS   VOLUME         CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfast-kmdata   Bound    nfast-kmdata   1G         RWX            manual         19s

% kubectl get pv

NAME          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS   REASON   AGE
nfast-kmdata   1G         RWX            Retain           Bound    default/nfast-kmdata   manual
2m30s
```

2.8.2. Create the socket persistent volume

- The `persistent_volume_sockets_definition.yaml` file:

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

- The `persistent_volume_sockets_claim.yaml` file:

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

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

1. Apply the definition file to MKE.

```
% kubectl apply -f persistent_volume_sockets_definition.yaml

persistentvolume/ncop-sockets created
```

2. Verify the persistent volume has been created.

```
% kubectl get pv

NAME           CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
STORAGECLASS   REASON   AGE
ncop-sockets   1G         RWO            Retain           Available                          manual
66s
```

3. Create the claim.

```
% kubectl apply -f persistent_volume_sockets_claim.yaml

persistentvolumeclaim/ncop-sockets created
```

4. Verify the claim has been created.

```
% kubectl get pvc

NAME           STATUS   VOLUME         CAPACITY   ACCESS MODES   STORAGECLASS   AGE
ncop-sockets   Bound    ncop-sockets   1G         RWO            manual         7s

% kubectl get pv

NAME           CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS   REASON   AGE
ncop-sockets   1G         RWO            Retain           Bound    default/ncop-sockets   manual
3h18m
```

2.8.3. 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

We created an application container to provide access to the persistent volume. This enables you to copy these files from the host server to the kubernetes cluster.

- The `persistent_volume_kmdata_populate.yaml` file defines the application.

```
kind: Pod
apiVersion: v1
metadata:
  name: ncop-populate-kmdata
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-kmdata
      command:
        - sh
        - '-c'
        - sleep 3600
      image: <docker-registry-address>/nshield-app
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: ncop-kmdata
          mountPath: /opt/nfast/kmdata
        - name: ncop-sockets
          mountPath: /opt/nfast/sockets
  securityContext: {}
  volumes:
    - name: ncop-config
      configMap:
        name: config
        defaultMode: 420
    - name: ncop-hardserver
```

```
      emptyDir: {}
    - name: ncop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
    - name: ncop-sockets
      emptyDir: {}
```

The image name should match the name you gave it when you pushed it to the docker registry server.

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

```
% kubectl apply -f persistent_volume_kmdata_populate.yaml

pod/ncop-populate-kmdata created

% kubectl get pods

NAME                   READY   STATUS    RESTARTS   AGE
ncop-populate-kmdata   1/1     Running   0          66s
```

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

```
% kubectl exec ncop-populate-kmdata -- mkdir -p /opt/nfast/kmdata/local

% kubectl exec ncop-populate-kmdata -- mkdir -p /opt/nfast/kmdata/config

% kubectl exec ncop-populate-kmdata -- 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:

```
% kubectl cp /opt/nfast/kmdata/local/world ncop-populate-kmdata:/opt/nfast/kmdata/local/world

% kubectl cp /opt/nfast/kmdata/local/module_<ESN> ncop-populate-kmdata::/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.

```
% kubectl cp /opt/nfast/kmdata/local/card* ncop-populate-kmdata:/opt/nfast/kmdata/local/.
```

5. Copy the softcard files if using softcard protection.

```
% kubectl cp /opt/nfast/kmdata/local/softcard* ncop-populate-kmdata::/opt/nfast/kmdata/local/.
```

6. Copy the `config/cardlist` file.

```
% kubectl cp /opt/nfast/kmdata/config/cardlist ncop-populate-kmdata:/opt/nfast/kmdata/config/cardlist
```

7. Verify that the files have been copied:

```
% kubectl exec ncop-populate-kmdata -- ls -al /opt/nfast/kmdata/local

Starting pod/nscop-test-dummy-nzqpt-debug, command was: sh -c sleep 3600
total 104
drwxrwsrwx. 2  977 976  4096 Sep 19 20:03 .
drwxrwsr-x. 7  977 976    82 Sep 19 12:57 ..
-rw-r--r--. 1 root 976   104 Sep 19 20:02 card_7aaf758bc6790206198ea5218040d4faa09f035f_1
-rw-r--r--. 1 root 976   104 Sep 19 20:02 card_7aaf758bc6790206198ea5218040d4faa09f035f_2
-rw-r--r--. 1 root 976   104 Sep 19 20:02 card_7aaf758bc6790206198ea5218040d4faa09f035f_3
-rw-r--r--. 1 root 976   104 Sep 19 20:03 card_7aaf758bc6790206198ea5218040d4faa09f035f_4
-rw-r--r--. 1 root 976   104 Sep 19 20:03 card_7aaf758bc6790206198ea5218040d4faa09f035f_5
-rw-r--r--. 1 root 976   104 Sep 19 20:03 card_edb3d45a28e5a6b22b033684ce589d9e198272c2_1
-rw-r--r--. 1 root 976   104 Sep 19 20:03 card_edb3d45a28e5a6b22b033684ce589d9e198272c2_2
-rw-r--r--. 1 root 976   104 Sep 19 20:03 card_edb3d45a28e5a6b22b033684ce589d9e198272c2_3
-rw-r--r--. 1 root 976   104 Sep 19 20:03 card_edb3d45a28e5a6b22b033684ce589d9e198272c2_4
-rw-r--r--. 1 root 976   104 Sep 19 20:03 card_edb3d45a28e5a6b22b033684ce589d9e198272c2_5
-rw-r--r--. 1 root 976  1364 Sep 19 20:03 cards_7aaf758bc6790206198ea5218040d4faa09f035f
-rw-r--r--. 1 root 976  1352 Sep 19 20:03 cards_edb3d45a28e5a6b22b033684ce589d9e198272c2
-rwxrwxrwx. 1 root 976  5232 Sep 19 19:55 module_7852-268D-3BF9
-rw-r--r--. 1 root 976   628 Sep 19 20:03 softcard_925f67e72ea3c354cae4e6797bde3753d24e7744
-rwxrwxrwx. 1 root 976 40860 Sep 19 19:57 world
```

8. Check if the cardlist file is in the persistent volume.

```
% kubectl exec ncop-populate-kmdata -- cat /opt/nfast/kmdata/config/cardlist

# This is the cardlist file, which contains the serial numbers of any
# Remote Administration Ready Smartcards that a system administrator
# has permitted to be used. These serial numbers are printed on the
# face of the smartcards
# Examples of valid 16 digit serial numbers:
#     XXXXXXXX-XXXXXXXX
#     XXXXXXXXXXXXXXXX
#     XXXX-XXXX-XXXX-XXXX
# To permit any cards presented to be used:
#      *
# The default configuration file has no cards listed, this means
# that all cards will be rejected by default.
*
```

2.8.4. 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 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. These scripts need to be copied into the `nfast-kmdata` persistent volume so the pods that will use them have access.

1. Create `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 CARDPP.
#
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
sleep 1
expect "Enter passphrase:"
send -- "$env(CARDPP)\r"
expect eof
```

2. Create `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 CARDPP.
#
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
sleep 1
expect "pass phrase for softcard"
send -- "$env(CARDPP)\r"
expect eof
```

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

```
% kubectl cp ocsexpect.sh ncop-populate-kmdata:/opt/nfast/kmdata/bin/.

% kubectl cp softcardexpect.sh ncop-populate-kmdata:/opt/nfast/kmdata/bin/.
```

4. Set the execute permissions on the files.

```
% kubectl exec ncop-populate-kmdata -- chmod +x /opt/nfast/kmdata/bin/ocsexpect.sh

% kubectl exec ncop-populate-kmdata -- chmod +x /opt/nfast/kmdata/bin/softcardexpect.sh
```

2.9. Deploying and testing nCOP with your application

You will need to create a `.yaml` file that defines how to launch the hardserver and your application container into MKE. The examples below were created to show how you can talk to the HSM from inside the Kubernetes pod.

2.9.1. Running the enquiry command

To run the `enquiry` command, which prints enquiry data from the module, use the following `pod_enquiry_app.yaml` file.

```
kind: Pod
apiVersion: v1
metadata:
  name: ncop-test-enquiry
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-enquiry
      command: ["sh", "-c"]
      args:
        - echo CONTAINER SCRIPT STARTED;
          sleep 10;
          /opt/nfast/bin/enquiry;
          echo CONTAINER SCRIPT DONE && sleep 3600
      image: <docker-registry-address>/nshield-app
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: ncop-kmdata
          mountPath: /opt/nfast/kmdata
        - name: ncop-sockets
          mountPath: /opt/nfast/sockets
    - name: ncop-hwsp
      image: <docker-registry-address>/nshield-hwsp
      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-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
    - name: ncop-sockets
      emptyDir: {}
```

In this example, <docker_registry-address> is the address of your internal docker registry server. Make sure the name of the images match what was pushed into the docker registry.

- Deploy the pod.

```
% kubectl apply -f pod_enquiry_app.yaml

pod/ncop-test-enquiry created
```

- Check if the Pod is running.

```
% kubectl get pods

NAME                   READY   STATUS    RESTARTS       AGE
ncop-test-enquiry      2/2     Running   1 (38s ago)    40s
```

You should see the deployment taking place. Wait 10 seconds and run the command again until the status is Running. This will also let you know if there are any errors. If there are errors, run the following command:

```
% kubectl describe pod ncop-test-enquiry
```

- Check if the `enquiry` command ran successfully.

```
% kubectl logs pod/ncop-test-enquiry -c ncop-enquiry

Server:
 enquiry reply flags  none
 enquiry reply level  Six
 serial number        7852-268D-3BF9
 mode                 operational
 version              13.6.3
 speed index          20000
 rec. queue           514..812
 level one flags      Hardware HasTokens SupportsCommandState
 version string       13.6.3-90-86c7a396, 13.2.4-280-7f4f0c24, 13.6.1-61-6acd63f8
 checked in           000000006671e78b Tue Jun 18 20:01:15 2024
 level two flags      none
 max. write size      8192
 level three flags    KeyStorage
 level four flags     HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList HasShareACL
HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds Type2Smartcard
ServerHasCreateClient HasInitialiseUnitEx AlwaysUseStrongPrimes Type3Smartcard HasKLF2
 module type code     0
 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           133ce957334bab5ab9901eda116ef10307128221
 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              13.2.4
 speed index          20000
 rec. queue           120..250
 level one flags      Hardware HasTokens SupportsCommandState SupportsHotReset
 version string       13.2.4-280-7f4f0c24, 13.6.1-61-6acd63f8
 checked in           00000000651fceee Fri Oct  6 09:10:06 2023
 level two flags      none
 max. write size      262152
 level three flags    KeyStorage
 level four flags     HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList HasShareACL
HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds Type2Smartcard
ServerHasCreateClient HasInitialiseUnitEx AlwaysUseStrongPrimes Type3Smartcard HasKLF2
 module type code     14
 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       0
 connection status    OK
 connection info      esn = 7852-268D-3BF9; addr = INET/10.194.148.39/9004; ku hash =
ed28cc6bb5dfef39ff327002006a55d90be0758d, mech = Any
 image version        13.6.1-50-6acd63f8
 level six flags      SerialConsoleAvailable Type3SmartcardRevB
 max exported modules 100
 rec. LongJobs queue  36
 SEE machine type     None
 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
```

2.9.2. Running the nfkminfo command

The following `pod_nfkminfo_app.yaml` file shows how to run the `nfkminfo` command which shows information about the current Security World.

```
kind: Pod
apiVersion: v1
```

```
metadata:
  name: ncop-test-nfkminfo
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-nfkminfo
      command: ["sh", "-c"]
      args:
        - echo CONTAINER SCRIPT STARTED;
          sleep 10;
          /opt/nfast/bin/nfkminfo;
          echo CONTAINER SCRIPT DONE && sleep 3600
      image: <docker-registry-address>/nshield-app
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: ncop-kmdata
          mountPath: /opt/nfast/kmdata
        - name: ncop-sockets
          mountPath: /opt/nfast/sockets
    - name: ncop-hwsp
      image: <docker-registry-address>/nshield-hwsp
      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-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
    - name: ncop-sockets
      emptyDir: {}
```

In this example, <docker_registry-address> is the address of your internal docker registry server. Make sure the name of the images match what was pushed into the docker registry.

- Deploy the pod.

```
% kubectl apply -f pod_nfkminfo_app.yaml

pod/ncop-test-nfkminfo created
```

- Check if the Pod is running.

```
% kubectl get pods

NAME                   READY   STATUS    RESTARTS       AGE
ncop-test-nfkminfo     2/2     Running   1 (29s ago)    31s
```

You should see the deployment taking place. Wait 10 seconds and run the command again until the status is Running. This will also let you know if there are any errors. If there are errors, run the following command:

```
% kubectl describe pod ncop-test-nfkminfo
```

- Check if the `nfkminfo` command ran successfully.

```
% kubectl logs pod/ncop-test-nfkminfo -c ncop-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      0x10000 ShareTarget
 n_slots    6
 esn        7852-268D-3BF9
 hkml       644e05d8e379d0a4c47fa89bc55369d50db8b85f

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 23
 generation     1
 phystype       SmartCard
 slotlistflags 0x180002 SupportsAuthentication DynamicSlot Associated
 state          0x5 Operator
 flags          0x10000
 shareno        2
 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
```

2.9.3. Generating a key using module protection

The following `pod_genkey_module_app.yaml` file shows how to generate a key using module protection.

```
kind: Pod
apiVersion: v1
metadata:
  name: ncop-test-genkey-module
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-genkey-module
      envFrom:
        - secretRef:
            name: cardcred
      env:
        - name: MY_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      command: ["sh", "-c"]
      args:
        - echo CONTAINER SCRIPT STARTED;
          sleep 10;
          /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: <docker-registry-address>/nshield-app
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: ncop-kmdata
          mountPath: /opt/nfast/kmdata
        - name: ncop-sockets
          mountPath: /opt/nfast/sockets
    - name: ncop-hwsp
      image: <docker-registry-address>/nshield-hwsp
      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-kmdata
      persistentVolumeClaim:
```

```
      claimName: nfast-kmdata
  - name: ncop-sockets
    emptyDir: {}
```

In this example, <docker_registry-address> is the address of your internal docker registry server. Make sure the name of the images match what was pushed into the docker registry.

- Deploy the pod.

```
% kubectl apply -f pod_genkey_module_app.yaml

pod/ncop-test-genkey-module created
```

- Check if the Pod is running.

```
% kubectl get pods

NAME                     READY   STATUS    RESTARTS      AGE
ncop-test-genkey-module  2/2     Running   0             25s
```

You should see the deployment taking place. Wait 10 seconds and run the command again until the status is Running. This will also let you know if there are any errors. If there are errors, run the following command:

```
% kubectl describe pod ncop-test-genkey-module
```

- Check if the key was generated successfully.

```
% kubectl logs pod/ncop-test-genkey-module -c ncop-genkey-module

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-7db32b03-1728-4981-be11-ab5ad89477e6
 nvram        Blob in NVRAM (needs ACS)          no
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ua115ac48c5d3dc510c7e5abaefad244f2407de7ed
`rocs' key recovery tool
Useful commands: `help', `help intro', `quit'.
rocs>   No. Name                     App        Protected by
        1   modulekey-7db32b03-1728- pkcs11     module
rocs>
CONTAINER SCRIPT DONE
```

2.9.4. Generating a key using softcard protection

The following `pod_genkey_softcard_app.yaml` file shows how to generate a key using softcard protection.

```
kind: Pod
apiVersion: v1
metadata:
  name: ncop-test-genkey-softcard
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-genkey-softcard
      envFrom:
        - secretRef:
            name: cardcred
      env:
        - name: MY_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      command: ["sh", "-c"]
      args:
        - echo CONTAINER SCRIPT STARTED;
          apt-get install expect -y;
          sleep 10;
          /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
      image: <docker-registry-address>/nshield-app
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: ncop-kmdata
          mountPath: /opt/nfast/kmdata
        - name: ncop-sockets
          mountPath: /opt/nfast/sockets
    - name: ncop-hwsp
      image: <docker-registry-address>/nshield-hwsp
      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-kmdata
      persistentVolumeClaim:
```

```
      claimName: nfast-kmdata
  - name: ncop-sockets
    emptyDir: {}
```

In this example, <docker_registry-address> is the address of your internal docker registry server. Make sure the name of the images match what was pushed into the docker registry. Note also that in the command we added a 10 second sleep to give time for the hardserver to start. The pod also installs the expect package which is required by the `softcardexpect.sh` script. This script will be used to pass the softcard passphase stored in one of the secrets.

- Deploy the pod.

```
% kubectl apply -f pod_genkey_softcard_app.yaml

pod/ncop-test-genkey-softcard created
```

- Check if the pod is running.

```
% kubectl get pods

NAME                         READY   STATUS    RESTARTS      AGE
ncop-test-genkey-softcard    2/2     Running   0             20s
```

You should see the deployment taking place. Wait 10 seconds and run the command again until the status is Running. This will also let you know if there are any errors. If there are errors, run the following command:

```
% kubectl describe pod ncop-test-genkey-softcard
```

- Check if the key was generated successfully.

```
% kubectl logs pod/ncop-test-genkey-softcard -c ncop-genkey-softcard

CONTAINER SCRIPT STARTED
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  libtcl8.6 tcl-expect tcl8.6 tzdata
Suggested packages:
  tk8.6 tcl-tclreadline
The following NEW packages will be installed:
  expect libtcl8.6 tcl-expect tcl8.6 tzdata
0 upgraded, 5 newly installed, 0 to remove and 0 not upgraded.
Need to get 1523 kB of archives.
After this operation, 6178 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 tzdata all 2024a-3ubuntu1.1 [273 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble/main amd64 libtcl8.6 amd64 8.6.14+dfsg-1build1 [988 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble/main amd64 tcl8.6 amd64 8.6.14+dfsg-1build1 [14.7 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble/universe amd64 tcl-expect amd64 5.45.4-3 [110 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble/universe amd64 expect amd64 5.45.4-3 [137 kB]
```

```
debconf: delaying package configuration, since apt-utils is not installed
Fetched 1523 kB in 1s (2975 kB/s)
Selecting previously unselected package tzdata.
(Reading database ... 4433 files and directories currently installed.)
Preparing to unpack .../tzdata_2024a-3ubuntu1.1_all.deb ...
Unpacking tzdata (2024a-3ubuntu1.1) ...
Selecting previously unselected package libtcl8.6:amd64.
Preparing to unpack .../libtcl8.6_8.6.14+dfsg-1build1_amd64.deb ...
Unpacking libtcl8.6:amd64 (8.6.14+dfsg-1build1) ...
Selecting previously unselected package tcl8.6.
Preparing to unpack .../tcl8.6_8.6.14+dfsg-1build1_amd64.deb ...
Unpacking tcl8.6 (8.6.14+dfsg-1build1) ...
Selecting previously unselected package tcl-expect:amd64.
Preparing to unpack .../tcl-expect_5.45.4-3_amd64.deb ...
Unpacking tcl-expect:amd64 (5.45.4-3) ...
Selecting previously unselected package expect.
Preparing to unpack .../expect_5.45.4-3_amd64.deb ...
Unpacking expect (5.45.4-3) ...
Setting up tzdata (2024a-3ubuntu1.1) ...
debconf: unable to initialize frontend: Dialog
debconf: (TERM is not set, so the dialog frontend is not usable.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (Can't locate Term/ReadLine.pm in @INC (you may need to install the Term::ReadLine module) (@INC
entries checked: /etc/perl /usr/local/lib/x86_64-linux-gnu/perl/5.38.2 /usr/local/share/perl/5.38.2
/usr/lib/x86_64-linux-gnu/perl5/5.38 /usr/share/perl5 /usr/lib/x86_64-linux-gnu/perl-base /usr/lib/x86_64-
linux-gnu/perl/5.38 /usr/share/perl/5.38 /usr/local/lib/site_perl) at
/usr/share/perl5/Debconf/FrontEnd/Readline.pm line 8.)
debconf: falling back to frontend: Teletype
Configuring tzdata
------------------

Please select the geographic area in which you live. Subsequent configuration
questions will narrow this down by presenting a list of cities, representing
the time zones in which they are located.

  1. Africa   3. Antarctica  5. Asia      7. Australia  9. Indian    11. Etc
  2. America  4. Arctic      6. Atlantic  8. Europe     10. Pacific
Geographic area:
Use of uninitialized value $_[1] in join or string at /usr/share/perl5/Debconf/DbDriver/Stack.pm line 112.

Current default time zone: '/UTC'
Local time is now:      Mon Sep 23 18:51:46 UTC 2024.
Universal Time is now:  Mon Sep 23 18:51:46 UTC 2024.
Run 'dpkg-reconfigure tzdata' if you wish to change it.

Use of uninitialized value $val in substitution (s///) at /usr/share/perl5/Debconf/Format/822.pm line 84,
<GEN6> line 4.
Use of uninitialized value $val in concatenation (.) or string at /usr/share/perl5/Debconf/Format/822.pm
line 85, <GEN6> line 4.
Setting up libtcl8.6:amd64 (8.6.14+dfsg-1build1) ...
Setting up tcl8.6 (8.6.14+dfsg-1build1) ...
Setting up tcl-expect:amd64 (5.45.4-3) ...
Setting up expect (5.45.4-3) ...
Processing triggers for libc-bin (2.39-0ubuntu8.3) ...
spawn /opt/nfast/bin/generatekey -b -g -m1 pkcs11 plainname=softcardkey-3469824a-6456-44f7-8167-
5697bea86ded 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-3469824a-6456-44f7-8167-5697bea86ded
 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-
50b2300fd760069482e8b8ad4dfcfe126bca5162
`rocs' key recovery tool
Useful commands: `help', `help intro', `quit'.
rocs>   No. Name                     App       Protected by
        1   softcardkey-3469824a-645 pkcs11    testSC (testSC)
rocs>
CONTAINER SCRIPT DONE
```

2.9.5. Generating a key using OCS protection

The following `pod_genkey_ocs_app.yaml` file shows how to generate a key using OCS protection.

```
kind: Pod
apiVersion: v1
metadata:
  name: ncop-test-genkey-ocs
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop-genkey-ocs
      envFrom:
        - secretRef:
            name: cardcred
      env:
        - name: MY_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      command: ["sh", "-c"]
      args:
        - echo CONTAINER SCRIPT STARTED;
          apt-get install expect -y;
          sleep 10;
          /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
      image: <docker-registry-address>/nshield-app
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: ncop-kmdata
          mountPath: /opt/nfast/kmdata
        - name: ncop-sockets
          mountPath: /opt/nfast/sockets
    - name: ncop-hwsp
      image: <docker-registry-address>/nshield-hwsp
      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-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
    - name: ncop-sockets
      emptyDir: {}
```

In this example, <docker_registry-address> is the address of your internal docker registry server. Make sure the name of the images match what was pushed into the docker registry. Note also that in the command we added a 10 second sleep to give time for the hardserver to start. The pod also installs the expect package which is required by the `ocsexpect.sh` script. This script will be used to pass the ocs card passphase stored in one of the secrets.

- Deploy the pod.

```
% kubectl apply -f pod_genkey_ocs_app.yaml

pod/ncop-test-genkey-ocs created
```

- Check if the pod is running.

```
% kubectl get pods

NAME                          READY   STATUS    RESTARTS      AGE
ncop-test-genkey-ocs          2/2     Running   0             23s
```

You should see the deployment taking place. Wait 10 seconds and run the command again until the status is Running. This will also let you know if there are any errors. If there are errors, run the following command:

```
% kubectl describe pod ncop-test-genkey-ocs
```

- Check if the key was generated successfully.

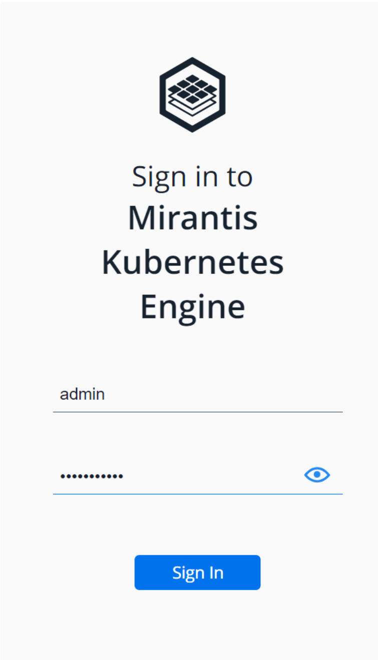
```
% kubectl logs pod/ncop-test-genkey-ocs -c ncop-genkey-ocs
```

```
CONTAINER SCRIPT STARTED
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  libtcl8.6 tcl-expect tcl8.6 tzdata
Suggested packages:
  tk8.6 tcl-tclreadline
The following NEW packages will be installed:
  expect libtcl8.6 tcl-expect tcl8.6 tzdata
0 upgraded, 5 newly installed, 0 to remove and 0 not upgraded.
Need to get 1523 kB of archives.
After this operation, 6178 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 tzdata all 2024a-3ubuntu1.1 [273 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble/main amd64 libtcl8.6 amd64 8.6.14+dfsg-1build1 [988 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble/main amd64 tcl8.6 amd64 8.6.14+dfsg-1build1 [14.7 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble/universe amd64 tcl-expect amd64 5.45.4-3 [110 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble/universe amd64 expect amd64 5.45.4-3 [137 kB]
debconf: delaying package configuration, since apt-utils is not installed
Fetched 1523 kB in 1s (1357 kB/s)
Selecting previously unselected package tzdata.
(Reading database ... 4433 files and directories currently installed.)
Preparing to unpack .../tzdata_2024a-3ubuntu1.1_all.deb ...
Unpacking tzdata (2024a-3ubuntu1.1) ...
Selecting previously unselected package libtcl8.6:amd64.
Preparing to unpack .../libtcl8.6_8.6.14+dfsg-1build1_amd64.deb ...
Unpacking libtcl8.6:amd64 (8.6.14+dfsg-1build1) ...
Selecting previously unselected package tcl8.6.
Preparing to unpack .../tcl8.6_8.6.14+dfsg-1build1_amd64.deb ...
Unpacking tcl8.6 (8.6.14+dfsg-1build1) ...
Selecting previously unselected package tcl-expect:amd64.
Preparing to unpack .../tcl-expect_5.45.4-3_amd64.deb ...
Unpacking tcl-expect:amd64 (5.45.4-3) ...
Selecting previously unselected package expect.
Preparing to unpack .../expect_5.45.4-3_amd64.deb ...
Unpacking expect (5.45.4-3) ...
Setting up tzdata (2024a-3ubuntu1.1) ...
debconf: unable to initialize frontend: Dialog
debconf: (TERM is not set, so the dialog frontend is not usable.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (Can't locate Term/ReadLine.pm in @INC (you may need to install the Term::ReadLine module) (@INC
entries checked: /etc/perl /usr/local/lib/x86_64-linux-gnu/perl/5.38.2 /usr/local/share/perl/5.38.2
/usr/lib/x86_64-linux-gnu/perl5/5.38 /usr/share/perl5 /usr/lib/x86_64-linux-gnu/perl-base /usr/lib/x86_64-
linux-gnu/perl/5.38 /usr/share/perl/5.38 /usr/local/lib/site_perl) at
/usr/share/perl5/Debconf/FrontEnd/Readline.pm line 8.)
debconf: falling back to frontend: Teletype
Configuring tzdata
------------------

Please select the geographic area in which you live. Subsequent configuration
questions will narrow this down by presenting a list of cities, representing
the time zones in which they are located.

  1. Africa   3. Antarctica  5. Asia      7. Australia  9. Indian    11. Etc
  2. America  4. Arctic      6. Atlantic  8. Europe     10. Pacific
Geographic area:
Use of uninitialized value $_[1] in join or string at /usr/share/perl5/Debconf/DbDriver/Stack.pm line 112.

Current default time zone: '/UTC'
Local time is now:      Mon Sep 23 15:24:09 UTC 2024.
Universal Time is now:  Mon Sep 23 15:24:09 UTC 2024.
Run 'dpkg-reconfigure tzdata' if you wish to change it.

Use of uninitialized value $val in substitution (s///) at /usr/share/perl5/Debconf/Format/822.pm line 84,
<GEN6> line 4.
Use of uninitialized value $val in concatenation (.) or string at /usr/share/perl5/Debconf/Format/822.pm
```

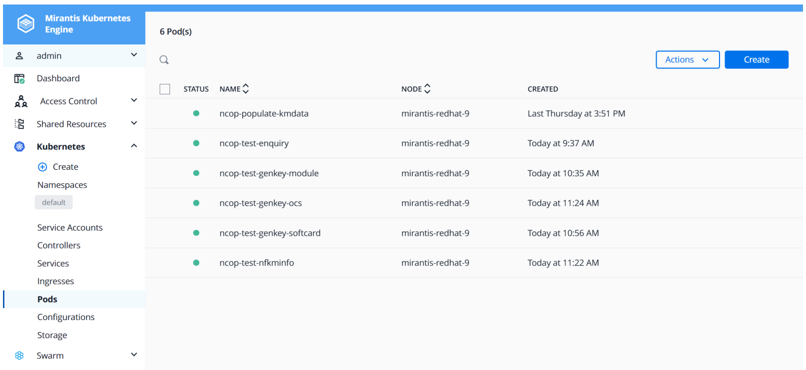
```
line 85, <GEN6> line 4.
Setting up libtcl8.6:amd64 (8.6.14+dfsg-1build1) ...
Setting up tcl8.6 (8.6.14+dfsg-1build1) ...
Setting up tcl-expect:amd64 (5.45.4-3) ...
Setting up expect (5.45.4-3) ...
Processing triggers for libc-bin (2.39-0ubuntu8.3) ...
spawn /opt/nfast/bin/generatekey -b -g -m1 pkcs11 plainname=ocskey-48b9d349-402c-4773-b41a-a637785bb976
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-48b9d349-402c-4773-b41a-a637785bb976
 nvram        Blob in NVRAM (needs ACS)          no

Loading `testOCS':
 Module 1: 0 cards of 1 read
 Module 1 slot 2: `testOCS' #2
 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-
0d0ec9d8e07ef8b5bbe82e3e0bc32245f51532ef
`rocs' key recovery tool
Useful commands: `help', `help intro', `quit'.
rocs>   No. Name                     App       Protected by
        1   ocskey-48b9d349-402c-477 pkcs11    testOCS
rocs>
CONTAINER SCRIPT DONE
```

2.10. Test MKE Web Interface

- Open a web browser and go to https://<host-node-ip-address>



- Log in with the account created during MKE installation.
- Navigate on the left pane to Kubernetes > Pods.
- The pods created should be shown running on this page.



- The other kubernetes objects generated in this integration can be viewed under the Kubernetes tab.

Chapter 3. Additional resources and related products

3.1. nShield Connect

3.2. nShield as a Service

3.3. nShield Container Option Pack

3.4. Entrust digital security solutions

3.5. nShield product documentation