VMware Tanzu Kubernetes Grid

nShield® HSM Integration Guide

2026-01-28

© Entrust Corporation. All rights reserved.

Table of Contents

1. Introduction . . . 1
    1.1. Integration architecture overview . . . 1
    1.2. Product configurations . . . 2
    1.3. Requirements . . . 2
2. Procedures . . . 4
    2.1. Prerequisites . . . 4
    2.2. Push the nCOP container images to an internal Docker registry . . . 4
    2.3. Deploy a TKG cluster . . . 5
    2.4. Deploy nCOP into the TKG cluster . . . 11
3. Additional resources and related products . . . 23
    3.1. nShield HSMs . . . 23
    3.2. nShield as a Service . . . 23
    3.3. nShield Container Option Pack . . . 23
    3.4. Entrust products . . . 23
    3.5. nShield product documentation . . . 23

Chapter 1. Introduction

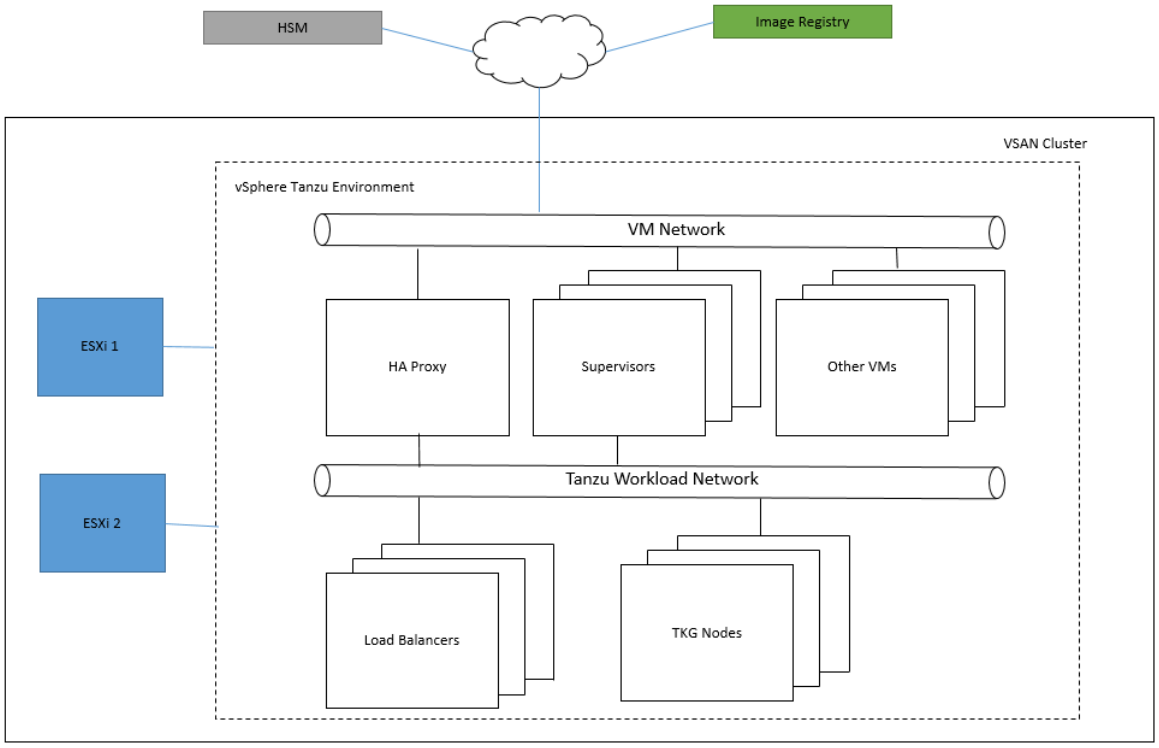
VMware vSphere with Tanzu Kubernetes cluster integrates with an nShield Hardware Security Module (HSM), using the nShield Container Option Pack (nCOP).

VMware vSphere with Tanzu uses your existing vSphere environment to deliver Kubernetes clusters at a rapid pace. Developers can build, launch, and scale container-based web applications in the vSphere environment. nCOP allows application developers, in the container-based environment of vSphere with Tanzu, to access the cryptographic functionality of an HSM.

1.1. Integration architecture overview

1.1.1. vSphere Tanzu Cluster and HSM

In this integration, a vSphere Tanzu Kubernetes cluster is deployed in a VMware vSAN cluster. Container images are downloaded from a Docker registry.



The following hosts are created:

- HA-Proxy virtual appliance for provisioning Load balancers.
- Supervisors virtual machines.

- Load balancers.
- TKG nodes.

For more information on how to deploy a VMware Tanzu Kubernetes cluster see your VMware documentation.

1.1.2. Container images

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

- nshield-hwsp

  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.

- nshield-app

  Application container images to run nShield commands. They are Red Hat Universal Base Image containers, 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

We have successfully tested the integration of an nShield HSM with VMware vSphere Tanzu in the following configurations:

Container Base OS	vSphere Tanzu	VMware	nShield HSM	nShield Image	nShield Firmware	nShield Software	nCOP
RHEL, CentOS, Ubuntu	7.0.2	7.0.2	Connect XC	12.60.10		12.71.0	1.1.1

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 vSphere with Tanzu.

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

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 VMware vSphere Tanzu in a VMware vCenter environment. See the documentation provided by VMware.
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. This can be a VM running Red Hat.
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 two files into a persistent volume accessible by the application containers are given when you create the persistent volume during the deployment of the TKG cluster.
5. Installed nCOP and created 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 we will refer to them as `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).

2.2. 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 Kubernetes pods into the Tanzu Kubernetes Cluster you will create 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 VMware vSphere Tanzu:

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:12.71` and `nshield-app:12.71`.

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:12.71" <docker-registry-address>/nshield-hwsp

% sudo docker tag nshield-app:12.71" <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
```

2.3. Deploy a TKG cluster

You will use the namespace you created when you deployed vSphere Tanzu to deploy out a TKG - a Tanzu Kubernetes cluster. You will do this from the container host machine. Make sure the `kubectl` and `kubectl-vsphere` commands are downloaded and available to be used in that machine. We will call the namespace `tanzu-ns`.

2.3.1. Log in to the namespace context

Use the `kubectl-vsphere` login command and set the server to the Supervisor Control Plane API Server IP address. This is the first IP address you use for the load balancers when you deployed the HA proxy server during the vSphere Tanzu setup.

```
% kubectl-vsphere login --insecure-skip-tls-verify --vsphere-username administrator@vsphere.local
--server=https://<load_balancer_ip>

Password:********
Logged in successfully.

You have access to the following contexts:
    * xxx.xxx.xxx.xxx
    * tanzu-ns

If the context you wish to use is not in this list, you may need to try
logging in again later, or contact your cluster administrator.

To change context, use `kubectl config use-context <workload name>`
```

```
% kubectl config use-context tanzu-ns

Switched to context "tanzu-ns".
```

2.3.2. Verify the control plane nodes and the storage class

Check that the control plane nodes are in a ready state, and that the storage policy assigned to the namespace earlier appears as a storage class. It is also useful to verify that the TKG virtual machine images are visible and that the content library used for storing the images has synchronized successfully.

```
% kubectl get nodes

NAME                                STATUS   ROLES    AGE    VERSION
4237574fb12b83b63026e6cc20abb152   Ready    master   28m    v1.18.2-6+38ac483e736488
4237c4c2761f4810a7794fc2ccb7433d   Ready    master   28m    v1.18.2-6+38ac483e736488
4237eb7fea870a051ae56eb564083bcb   Ready    master   28m    v1.18.2-6+38ac483e736488
```

```
% kubectl get sc

NAME                        PROVISIONER            RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE
vsan-default-storage-policy csi.vsphere.vmware.com Delete        Immediate         true                 28m
```

```
% kubectl get virtualmachineimages

NAME                                                  VERSION                         OSTYPE
ob-15957779-photon-3-k8s-v1.16.8---vmware.1-tkg.3.60d2ffd  v1.16.8+vmware.1-tkg.3.60d2ffd  vmwarePhoton64Guest
ob-16466772-photon-3-k8s-v1.17.7---vmware.1-tkg.1.154236c  v1.17.7+vmware.1-tkg.1.154236c  vmwarePhoton64Guest
ob-16545581-photon-3-k8s-v1.16.12---vmware.1-tkg.1.da7afe7 v1.16.12+vmware.1-tkg.1.da7afe7 vmwarePhoton64Guest
```

2.3.3. Create a manifest file for the TKG deployment

You create a TKG cluster in vSphere with Tanzu using a `yaml` manifest file, which contains the following information:

- The name of the cluster.
- The number of control plane nodes.
- The number of worker nodes.
- The size of the nodes from a resource perspective (`class`).
- Which storage class to use (`storageClass`).
- Which image to use for the nodes (`version`).

The following example specifies:

- A single control plane node.
- 2 worker nodes.
- Uses image version 1.17.7.

This use of the version number is a shorthand to specify which Photon OS image to use. Note the indentation. It needs to be just right for the manifest to work.

cluster.yaml

```
apiVersion: run.tanzu.vmware.com/v1alpha1
kind: TanzuKubernetesCluster
metadata:
  name: tkg-cluster
spec:
  topology:
    controlPlane:
      count: 1
      class: best-effort-small
      storageClass: management-storage-policy-single-node
    workers:
      count: 2
      class: best-effort-small
      storageClass: management-storage-policy-single-node
  distribution:
    version: v1.17.7
```

To learn more about the resources assigned to the various classes, list them by using the following command:

```
% kubectl get virtualmachineclass

NAME                 AGE
best-effort-2xlarge  2d16h
best-effort-4xlarge  2d16h
best-effort-8xlarge  2d16h
best-effort-large    2d16h
best-effort-medium   2d16h
```

```
best-effort-small     2d16h
best-effort-xlarge    2d16h
best-effort-xsmall    2d16h
guaranteed-2xlarge    2d16h
guaranteed-4xlarge    2d16h
guaranteed-8xlarge    2d16h
guaranteed-large      2d16h
guaranteed-medium     2d16h
guaranteed-small      2d16h
guaranteed-xlarge     2d16h
guaranteed-xsmall     2d16h
```

To get more information about a specific class, use the `describe` command:

```
% kubectl describe virtualmachineclass guaranteed-small
```

2.3.4. Apply the TKG manifest and monitor the deployment

1. Deploy the TKG cluster by applying the manifest:

   ```
   % kubectl apply -f cluster.yaml

   tanzukubernetescluster.run.tanzu.vmware.com/tkg-cluster created
   ```

2. Check if cluster has been provisioned:

   ```
   % kubectl get cluster

   NAME             PHASE
   tkg-cluster   Provisioned
   ```

3. Check the deployment of the cluster:

   ```
   % kubectl get tanzukubernetescluster

   NAME             CONTROL PLANE   WORKER   DISTRIBUTION                    AGE    PHASE
   tkg-cluster   1              2        v1.17.7+vmware.1-tkg.1.154236c   3m7s   creating
   ```

You can then use a variety of commands to monitor the deployment. The `describe` command can be long and you can use it repeatedly to monitor the TKG cluster deployment status. It contains useful information, such as the use of Antrea as the CNI, node and VM status, Cluster API endpoint from our load balancer, frontend network range of IP addresses.

```
% kubectl describe tanzukubernetescluster
```

You should see the cluster in the `running` phase if it was deployed successfully.

```
% kubectl get tanzukubernetescluster

NAME             CONTROL PLANE   WORKER   DISTRIBUTION                    AGE    PHASE
tkg-cluster   1              2        v1.17.7+vmware.1-tkg.1.154236c   11m    running
```

2.3.5. Log out then log in to TKG cluster context

The previous stages were carried out in the context of a namespace in the vSphere with Tanzu Supervisor cluster. The next step is to log out from the namespace context, and log in to the TKG guest cluster context. This allows you to direct `kubectl` commands at the TKG cluster API server.

1. Log out of the namespace:

```
% kubectl-vsphere logout

Your KUBECONFIG context has changed.
The current KUBECONFIG context is unset.
To change context, use `kubectl config use-context <workload name>`
Logged out of all vSphere namespaces.
```

2. Log in to the cluster:

```
% kubectl-vsphere login --insecure-skip-tls-verify --vsphere-username administrator@vsphere.local
--server=https://<load_balancer_ip> --tanzu-kubernetes-cluster-namespace tanzu-ns --tanzu-kubernetes
-cluster-name tkg-cluster

Password: ********
Logged in successfully.

You have access to the following contexts:
   xxx.xxx.xxx.xxx
   tanzu-ns
   tkg-cluster

If the context you wish to use is not in this list, you may need to try logging in again later, or contact
your cluster administrator.

To change context, use `kubectl config use-context <workload name>`
```

3. Set the context:

```
% kubectl config use-context tkg-cluster

Switched to context "tkg-cluster".
```

2.3.6. Validate the TKG cluster context

You can run some `kubectl` commands to verify that you are now working in the correct

context. If you display the nodes, you should see the 1 x control plane node and the 2 x worker nodes specified in the manifest when you deployed the cluster. You should also be able to display all the pods deployed in the cluster and observe both the Antrea agents for networking and CSI node agents for storage. This is also a good opportunity to check that everything has entered a Ready/Running state in the TKG cluster.

```
% kubectl get nodes

NAME                                        STATUS   ROLES    AGE    VERSION
tkg-cluster-control-plane-m62jn             Ready    master   28m    v1.17.7+vmware.1
tkg-cluster-workers-djqql-b45c55588-59wz2   Ready    <none>   28m    v1.17.7+vmware.1
tkg-cluster-workers-djqql-b45c55588-c52h7   Ready    <none>   28m    v1.17.7+vmware.1
```

```
% kubectl get pods -A

NAMESPACE                     NAME                                                     READY   STATUS
RESTARTS   AGE
kube-system                   antrea-agent-7zphw                                       2/2     Running   0
2m23s
kube-system                   antrea-agent-mvczg                                       2/2     Running   0
2m23s
kube-system                   antrea-agent-t6qgc                                       2/2     Running   0
8m11s
kube-system                   antrea-controller-76c76c7b7c-4cwtm                       1/1     Running   0
8m11s
kube-system                   coredns-6c78df586f-6d77q                                 1/1     Running   0
7m55s
kube-system                   coredns-6c78df586f-c8rtj                                 1/1     Running   0
8m14s
kube-system                   etcd-tkg-cluster-control-plane-sn85m                     1/1     Running   0
7m24s
kube-system                   kube-apiserver-tkg-cluster-control-plane-sn85m           1/1     Running   0
7m12s
kube-system                   kube-controller-manager-tkg-cluster-control-plane-sn85m  1/1     Running   0
7m30s
kube-system                   kube-proxy-frpgn                                         1/1     Running   0
2m23s
kube-system                   kube-proxy-lchkq                                         1/1     Running   0
2m23s
kube-system                   kube-proxy-m2xjn                                         1/1     Running   0
8m14s
kube-system                   kube-scheduler-tkg-cluster-control-plane-sn85m           1/1     Running   0
7m27s
vmware-system-auth            guest-cluster-auth-svc-lf2nf                             1/1     Running   0
8m2s
vmware-system-cloud-provider  guest-cluster-cloud-provider-7788f74548-cfng4            1/1     Running   0
8m13s
vmware-system-csi             vsphere-csi-controller-574cfd4569-kfdz8                  6/6     Running   0
8m12s
vmware-system-csi             vsphere-csi-node-hvzpg                                   3/3     Running   0
8m12s
vmware-system-csi             vsphere-csi-node-t4w8p                                   3/3     Running   0
2m23s
vmware-system-csi             vsphere-csi-node-t6rdw                                   3/3     Running   0
2m23s
```

This output shows that the TKG cluster is deployed in vSphere with Tanzu.

2.4. Deploy nCOP into the TKG cluster

When the Tanzu Kubernetes Cluster has been deployed, it is time to deploy the nCOP containers as pods into the cluster and test the nCOP functionality inside the cluster.

2.4.1. Create the registry secrets inside the TKG cluster

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 the TKG cluster know how to authenticate to that registry.

1. Log in to the TKG cluster:

```
% kubectl-vsphere login --insecure-skip-tls-verify --vsphere-username administrator@vsphere.local
--server=https://<load_balancer_ip> --tanzu-kubernetes-cluster-namespace tanzu-ns --tanzu-kubernetes
-cluster-name tkg-cluster
```

<load_balancer_ip> is the IP address used by the cluster namespace.

2. Set the context to the TKG cluster:

```
% kubectl config use-context tkg-cluster
```

3. Create the secret in the cluster:

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

4. Check if the secret was created

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

2.4.2. Create the configuration map for the HSM details

1. Edit the following template `yaml` file for your HSM:

configmap yaml file

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config
data:
  config: |
```

```
    syntax-version=1

    [nethsm_imports]
    local_module=1
    remote_esn=BD10-03E0-D947
    remote_ip=10.194.148.36
    remote_port=9004
    keyhash=2dd7c10c73a3c5346d1246e6a8cf6766a7088e41
    privileged=0
```

2. Create the configuration map:

```
% kubectl apply -f configmap.yaml

configmap/config created
```

3. Verify that the configuration map was created:

```
% kubectl describe configmap/config

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

config:

syntax-version=1

[nethsm_imports]
local_module=1
remote_esn=BD10-03E0-D947
remote_ip=10.194.148.36
remote_port=9004
keyhash=2dd7c10c73a3c5346d1246e6a8cf6766a7088e41
privileged=0

Events:  <none>
```

2.4.3. Create the cluster persistent volumes in the TKG cluster

1. Create the `/opt/nfast/kmdata/local` directory in your host machine and copy the Security World and module files to it.

   Do this before you proceed with the creation of the persistent volume.

2. Edit the following example `yaml` file to create and claim the persistent volume:

   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
```

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
```

3. Apply the definition file to the cluster:

```
% kubectl apply -f persistent_volume_kmdata_definition.yaml

persistentvolume/nfast-kmdata created
```

4. Verify the persistent volume has been created:

```
% kubectl get pv

NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM                 STORAGECLASS
REASON   AGE
nfast-kmdata    1G         RWO            Retain           Available                         manual
43m
```

5. Create the claim:

```
% kubectl apply -f persistent_volume_kmdata_claim.yaml

persistentvolumeclaim/nfast-kmdata created
```

6. Verify the claim has been created:

```
% kubectl get pvc

NAME            STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfast-kmdata    Bound    nfast-kmdata    1G         RWO            manual         61m
```

```
% kubectl get pv

NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS
REASON   AGE
nfast-kmdata    1G         RWO            Retain           Bound    default/nfast-kmdata    manual
67m
```

2.4.4. Deploy the nCOP pod with your application

Create a `yaml` file that defines how to launch the hardserver and your application container into the cluster.

The examples below were created to show how you can talk to the HSM from inside the Kubernetes pod. Each example shows how to execute the following commands:

- `enquiry`
- `nfkminfo`
- `sigtest`

2.4.4.1. Populate the persistent volume with the world and modules file

1. Before running any of the applications, update `/opt/nfast/kmdata/local` in the persistent volume with the latest Security World and module files.

   To do this, create a `yaml` file to run a pod that gives access to the persistent volume so these files can be copied.

   The following example shows how to get access to the persistent volume.

   persistent_volume_kmdata_populate.yaml

```
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: {}
```

2. Deploy the pod:

```
% kubectl apply -f persistent_volume_kmdata_populate.yaml
```

3. Check if the pod is running:

```
% kubectl get pods
```

   You should see that the deployment is taking place. Wait 10 seconds and run the command again until the status is Running.

   If there are errors, run the following command to check what went wrong:

```
% kubectl describe pod ncop-populate-kmdata
```

4. Copy the module file to `/opt/nfast/kmdata/local` in the pod:

```
% kubectl cp /opt/nfast/kmdata/local/module_BD10-03E0-D947 ncop-populate-kmdata:/opt/nfast/kmdata/local/.
```

5. Copy the Security World file to `/opt/nfast/kmdata/local` in the pod:

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

6. Check if the files are in the persistent volume

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

total 68
drwxr-xr-x 2 root root  4096 Sep 20 18:40 .
drwxr-xr-x 3 root root  4096 Dec 16  2020 ..
-rwxrwxrwx 1 root 1001  3488 Sep 20 18:40 module_BD10-03E0-D947
-rwxrwxrwx 1 root 1001 39968 Sep 20 18:40 world
```

2.4.4.2. enquiry

This example shows how to run the `enquiry` command.

pod_enquiry_app.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: ncop-test-enquiry
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop
      command:
        - sh
        - '-c'
        - /opt/nfast/bin/enquiry && 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: {}
```

<docker_registry-address> is the address of your internal Docker registry server.

1. Deploy the pod

```
% kubectl apply -f pod_enquiry_app.yaml
```

2. Check that the pod is running:

```
% kubectl get pods
```

You should see that the deployment is taking place. Wait 10 seconds and run the command again until the status is Running.

If there are errors, run the following command to check what went wrong:

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

3. Check if the `enquiry` command runs:

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

Server:
 enquiry reply flags  none
 enquiry reply level  Six
 serial number        BD10-03E0-D947
 mode                 operational
 version              12.71.0
 speed index          15843
 rec. queue           368..566
 level one flags      Hardware HasTokens SupportsCommandState
 version string       12.71.0-353-f63c551, 12.50.11-270-fb3b87dd465b6f6e53d9f829fc034f8be2dafd13 2019/05/16
22:02:33 BST, Bootloader: 1.2.3, Security Processor: 12.50.11 , 12.60.10-708-ea4dc41d
 checked in           000000006053229a Thu Mar 18 09:51:22 2021
 level two flags      none
 max. write size      8192
 level three flags    KeyStorage
 level four flags     OrderlyClearUnit HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList
HasSEE HasKLF HasShareACL HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds
JobFragmentation LongJobsPreferred Type2Smartcard ServerHasCreateClient HasInitialiseUnitEx
AlwaysUseStrongPrimes Type3Smartcard HasKLF2
 module type code     0
 product name         nFast server
 device name
 EnquirySix version   4
 impath kx groups
 feature ctrl flags   none
 features enabled     none
 version serial       0
 level six flags      none
 remote server port   9004
 kneti hash           8a16e4e8c5069ac16b7ba03334e463b11a15a400

Module #1:
 enquiry reply flags  UnprivOnly
 enquiry reply level  Six
 serial number        BD10-03E0-D947
 mode                 operational
 version              12.50.11
 speed index          15843
 rec. queue           43..150
 level one flags      Hardware HasTokens SupportsCommandState
```

```
 version string       12.50.11-270-fb3b87dd465b6f6e53d9f829fc034f8be2dafd13 2019/05/16 22:02:33 BST,
Bootloader: 1.2.3, Security Processor: 12.50.11 , 12.60.10-708-ea4dc41d
 checked in           000000005cddcfe9 Thu May 16 21:02:33 2019
 level two flags      none
 max. write size      8192
 level three flags    KeyStorage
 level four flags     OrderlyClearUnit HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList
HasSEE HasKLF HasShareACL HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds
JobFragmentation LongJobsPreferred Type2Smartcard ServerHasCreateClient HasInitialiseUnitEx
AlwaysUseStrongPrimes Type3Smartcard HasKLF2
 module type code     12
 product name         nC3025E/nC4035E/nC4335N
 device name          Rt1
 EnquirySix version   7
 impath kx groups     DHPrime1024 DHPrime3072 DHPrime3072Ex
 feature ctrl flags   LongTerm
 features enabled     GeneralSEE StandardKM EllipticCurve ECCMQV AcceleratedECC HSMHighSpeed
 version serial       37
 connection status    OK
 connection info      esn = BD10-03E0-D947; addr = INET/10.194.148.36/9004; ku hash =
2dd7c10c73a3c5346d1246e6a8cf6766a7088e41, mech = Any
 image version        12.60.10-507-ea4dc41d
 level six flags      none
 max exported modules 100
 rec. LongJobs queue  42
 SEE machine type     PowerPCELF
 supported KML types  DSAp1024s160 DSAp3072s256
 using impath kx grp  DHPrime3072Ex
 active modes         UseFIPSApprovedInternalMechanisms AlwaysUseStrongPrimes
 hardware status      OK
```

2.4.4.3. nfkminfo

This example shows how to run the `nfkminfo` command.

pod_nfkminfo_app.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: ncop-test-nfkminfo
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop
      command:
        - sh
        - '-c'
        - /opt/nfast/bin/nfkminfo && 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: {}
```

<docker_registry-address> is the address of your internal Docker registry server.

1. Deploy the pod:

   ```
   % kubectl apply -f pod_nfkminfo_app.yaml
   ```

2. Check if the pod is running:

   ```
   % kubectl get pods
   ```

   You should see that the deployment is taking place. Wait 10 seconds and run the command again until the status is Running.

   If there are errors, run the following command to check what went wrong:

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

3. Check if the `nfkminfo` command runs:

   ```
   % kubectl logs pod/ncop-test-nfkminfo ncop

   World
    generation  2
    state       0x37a70008 Initialised Usable Recovery PINRecovery !ExistingClient RTC NVRAM FTO
   AlwaysUseStrongPrimes !DisablePKCS1Padding !PpStrengthCheck !AuditLogging SEEDebug
    n_modules   1
    hknso       c9fb9e4cc5ec99fed1b92a766d90faccb639c7da
    hkm         235a046a49e6361470ba5a76dc1b3f745521dbd3 (type Rijndael)
    hkmwk       c2be99fe1c77f1b75d48e2fd2df8dffc0c969bcb
    hkre        4c8a5db06af0f51ab29a5ca5dacc72929f9f1b87
   ```

```
 hkra        6a1691c6d9a447ed90379fa49ebb6808a5b1851f
 hkmc        77bdd4664d681c9211b0fca71ced36351dfafc72
 hkp         2e1db243c305c1b0b9ff9a13b5039cce10119413
 hkrtc       6e100c78fd6e200e6fffd6419089d5dd34320097
 hknv        7d64bf068d30e0283219665b10aee498b061f85d
 hkdsee      059485679ff0a96048891bb6041cc11e4b0e9236
 hkfto       8aa8a2a902ffe4a6c83beaab3984aea1626b90d0
 hkmnull     0100000000000000000000000000000000000000
 ex.client   none
 k-out-of-n  1/1
 other quora m=1 r=1 p=1 nv=1 rtc=1 dsee=1 fto=1
 createtime  2021-06-21 20:55:08
 nso timeout 10 min
 ciphersuite DLf3072s256mAEScSP800131Ar1
 min pp      0 chars
 mode        none

Module #1
 generation 2
 state      0x2 Usable
 flags      0x0 !ShareTarget
 n_slots    4
 esn        BD10-03E0-D947
 hkml       7f6ee17f7d9c0c26297ee788a1e1a5e698040b5d

Module #1 Slot #0 IC 1
 generation    1
 phystype      SmartCard
 slotlistflags 0x2 SupportsAuthentication
 state         0x7 Error
 flags         0x0
 shareno       0
 shares
 error         UnlistedCard
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

No Pre-Loaded Objects
```

2.4.4.4. sigtest

This example shows how to run the `sigtest` command.

pod_sigtest_app.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: ncop-test-sigtest
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ncop
      command:
        - sh
        - '-c'
        - /opt/nfast/bin/sigtest && 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: {}
```

<docker_registry-address> is the address of your internal Docker registry server.

1. Deploy the pod:

```
% kubectl apply -f pod_sigtest_app.yaml
```

2. Check if the pod is running:

```
% kubectl get pods
```

   You should see that the deployment is taking place. Wait 10 seconds and run the command again until the status is Running.

   If there are errors, run the following command to check what went wrong:

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

3. Check if the `sigtest` command runs:

```
% kubectl logs pod/ncop-test-sigtest ncop

 Hardware module #1 speed index 15843 recommended minimum queue 43
Found 1 module; using 43 jobs
Making 1024-bit RSAPrivate key on module #1;
  using Mech_RSApPKCS1 and PlainTextType_Bignum.
Generated and exported key from module #1.
Imported keys on module #1
 1, 9106 3642.4, 9106 overall
 2, 19814 6468.64, 9907 overall
 3, 29238 7650.78, 9746 overall
 4, 38331 8227.67, 9582.75 overall
 5, 46616 8250.6, 9323.2 overall
 6, 56538 8919.16, 9423 overall
 7, 66632 9389.1, 9518.86 overall
 8, 77314 9906.26, 9664.25 overall
 9, 87932 10191, 9770.22 overall
 10, 98555 10363.8, 9855.5 overall
 11, 108923 10365.5, 9902.09 overall
 12, 118720 10138.1, 9893.33 overall
```

Chapter 3. Additional resources and related products

3.1. nShield HSMs

3.2. nShield as a Service

3.3. nShield Container Option Pack

3.4. Entrust products

3.5. nShield product documentation