IBM Blockchain

nShield® HSM Integration Guide

27 Jul 2022

Contents

1. Introduction

IBM Blockchain Platform integrates with the Entrust nShield® Hardware Security Module (HSM) to generate and store the private keys used by its Certificate Authority (CA), Peer, and Orderer nodes. This guide demonstrates using an HSM On Demand service's PKCS #11 API to securely store Blockchain CA, Peer, and Orderer private keys. When an HSM generates the signing keys for Blockchain Identities, the cryptographic operations are offloaded to the HSM. This provides protects and manages the keys with its FIPS 140-2 Level 3 certified hardware.

> This guide describes how to perform and validate the integration. It does not necessarily describe the best practices for the implementation.

1.1. Product configurations

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

Product	Version
IBM Blockchain Platform	2.5.2-132, 2.5.3-11
OpenShift Container Platform (Client)	4.8.43
OpenShift Container Platform (Server)	4.8.43
Kubernetes Version	v1.21.11+6b3cbdd
Base OS (Image Building machine)	Red Hat Enterprise Linux release 8.5 (Ootpa)
OS for NFS server	CentOS Linux release 7.9.2009 (Core)
nShield HSM	Connect XC (12.50.11) - Image 12.80.4 Connect + (12.50.8) - Image 12.80.4
HSM Protection Methods Used.	Module, Softcard. OCS not tested.
nShield Security World	12.80.4
nShield Container Option Pack (nSCOP)	1.1.1
VMware	ESXi 7.0.1 on a Dell PowerEdge R740

Product	Version
Docker	Docker version 20.10.17, build 100c701
Podman	3.2.3

1.2. Requirements

Ensure that you have supported versions and entitlements of the Entrust nShield, IBM Blockchain Platform, and third-party products. See Product configurations.

To perform the integration tasks, you must have:

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

Before starting the integration process, familiarize yourself with:

- The documentation for the HSM.
- The documentation and setup process for Docker or Podman.

Before using the nShield software, you need to know:

- Whether the application keys are protected by the module, or a Softcard with or without a pass phrase.
- Whether the Security World should be compliant with FIPS 140-2 Level 3.

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

2. Procedures

Before deploying the nShield nSCOP image onto OpenShift to be integrated with IBM Blockchain Platform, complete the following integration procedures:

- Install and configure the nShield Connect XC, see the *Installation Guide* for your HSM.
- Configure the HSM(s) to have the IP address of your container host machine as a client, as well as the IP addresses of the nodes that make up your OpenShift cluster.
- Create a new Security World or load an existing one on the HSM. The Security World and module files will need to be copied to a specific directory on the NFS server that will be accessed by the IBM Blockchain Platform nodes. This will use either OpenShift or a Kubernetes persistent volume. Instructions for this are included later in this guide.
- Docker or Podman installed to build the image.
- Install nSCOP on the image-building machine. 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).

> This guide uses a Red Hat OpenShift cluster for deployment of the IBM Blockchain Platform, so many commands in this guide are `oc` commands. These `oc` commands can be directly replaced with `kubectl` commands if you are deploying onto a Kubernetes cluster.

2.1. OpenShift container platform

For this integration a Red Hat OpenShift cluster running OpenShift 4.8.43 is used. For example:

```
% oc version

Client Version: 4.8.43
Server Version: 4.8.43
Kubernetes Version: v1.21.11+6b3cbdd
```

The cluster is composed of three master nodes and three worker nodes:

```
% oc get nodes

NAME                        STATUS   ROLES    AGE   VERSION
ocp4843-w9lph-master-0      Ready    master   79m   v1.21.11+6b3cbdd
ocp4843-w9lph-master-1      Ready    master   79m   v1.21.11+6b3cbdd
ocp4843-w9lph-master-2      Ready    master   79m   v1.21.11+6b3cbdd
ocp4843-w9lph-worker-925wf  Ready    worker   58m   v1.21.11+6b3cbdd
ocp4843-w9lph-worker-dzpmk  Ready    worker   58m   v1.21.11+6b3cbdd
ocp4843-w9lph-worker-gl8gr  Ready    worker   58m   v1.21.11+6b3cbdd
```

For more information on how to install and setup Red Hat OpenShift, see Installing and configuring OpenShift Container Platform clusters in the Red Hat online documentation.

2.2. Deploy IBM Blockchain

Perform the IBM Blockchain 2.5.3 deployment guide process in the IBM online documentation to deploy and install IBM Blockchain on your Red Hat OpenShift Cluster.

> Some sample YAML files are provided in this guide to help with the integration. You may need to make changes to these to support your specific integration.

2.3. Create the NFS storage class for the cluster

If you have already created a storage class for your cluster, you can skip this section. All that will be needed in later sections of this guide is the storage class name (*<storage-class-name>*) and the NFS share directory (*<nfs-directory>*).

To serve as persistent storage for the IBM Blockchain Platform deployment onto OpenShift, a separate CentOS 7 virtual machine was created with 2 vCPUs, 8 GB Memory, and 400 GB storage (to meet storage requirements for the IBM Blockchain console and nodes). This VM hosts an NFS server that is connected to the cluster.

- For steps on deploying your own NFS server, see https://dev.to/prajwalmithun/setup-nfs-server-client-in-linux-and-unix-27id.
- For steps on connecting it to OpenShift, see https://levelup.gitconnected.com/how-to-use-nfs-in-kubernetes-cluster-storage-class-ed1179a83817.
- Also see How do I create a storage class for NFS dynamic storage provisioning in an OpenShift environment? the IBM online documentation.

The following YAML files may be useful to setup NFS with OpenShift:

- `nfs-rbac.yaml`
- `nfs-deployment.yaml`

  The important fields on this file are:

```
- name: NFS_SERVER
  value: xx.xxx.xxx.xxx
- name: NFS_PATH
  value: /your/nfs/path
```

Make sure they match the NFS_SERVER and NFS_PATH you are using.

- `storage-class.yaml`

  The important fields on this file are:

  ```
  pathPattern: "ibmblockchain"
  ```

  This field will be used by IBM Blockchain. During the deployment process a folder named `ibmblockchain` will be created in the NFS_PATH where IBM Blockchain will store its files.

See *Sample YAML files* for YAML files you can adapt to your system.

If you need to connect your NFS server to OpenShift and make it the default storage class for your cluster, run the following commands:

```
% oc apply -f nfs-rbac.yaml -n ibm-blockchain-proj
serviceaccount/nfs-client-provisioner created
clusterrole.rbac.authorization.k8s.io/nfs-client-provisioner-runner unchanged
clusterrolebinding.rbac.authorization.k8s.io/run-nfs-client-provisioner unchanged
role.rbac.authorization.k8s.io/leader-locking-nfs-client-provisioner created
rolebinding.rbac.authorization.k8s.io/leader-locking-nfs-client-provisioner created

% oc adm policy add-scc-to-user hostmount-anyuid system:serviceaccount:ibm-blockchain-proj:nfs-client-provisioner
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:hostmount-anyuid added: "nfs-client-provisioner"

% oc apply -f nfs-deployment.yaml -n ibm-blockchain-proj
deployment.apps/nfs-client-provisioner created

% oc apply -f storage-class.yaml -n ibm-blockchain-proj
storageclass.storage.k8s.io/nfs-storage created
```

This setup needs to be performed after you create your `ibm-blockchain-proj` and before the deployment of the `bp-operator` during the deployment of IBM Blockchain.

Throughout the rest of this guide, for clarity, the storage class `nfs-storage`, is referred to as *<storage-class-name>*.

```
% oc get serviceaccount
NAME                     SECRETS   AGE
nfs-client-provisioner   2         3m

% oc get storageclasses -n ibm-blockchain-proj
NAME            PROVISIONER                                   RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION
AGE
nfs-storage     k8s-sigs.io/nfs-subdir-external-provisioner   Delete          Immediate           false
0s

% oc get deployment
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
nfs-client-provisioner   1/1     1            1           2m28s
```

2.4. Generate the HSM config files and copying files to the persistent volume

This section describes how to generate the HSM config file after installing nSCOP, and copy the required files to the persistent volume that will be accessed by the IBM Blockchain Platform nodes.

1. Change to the directory where nSCOP is installed.
2. Run the following command to create the `config` file in `/opt/nfast/kmdata/local`.

```
% ./make-nshield-hwsp-config --output /opt/nfast/kmdata/local/config <HSM-IP>
```

3. Verify your `config` file looks like the following.

```
syntax-version=1

[nethsm_imports]
local_module=1
remote_esn=<HSM-ESN>
remote_ip=<HSM-IP>
remote_port=9004
keyhash=...
privileged=0
```

4. Copy `config` to `<nfs-directory>`.
5. Additionally, copy the `world` and `module` files to `<nfs-directory>`. Both files are found at `/opt/nfast/kmdata/local` where your Security World is installed, and the module file is named `module_<HSM-ESN>`.
6. Finally, create the `cknfastrc` file in `<nfs-directory>`. The next section will describe how to populate this file to change the mode of key protection.

```
% touch <nfs-directory>/cknfastrc
```

7. Ensure all the files in the `<nfs-directory>` have proper users/groups and permissions.

```
% ls -l <nfs-directory>

-rw-r--r--. 1 nfast    nfast       33 Jun 14 14:04 cknfastrc
-rw-r--r--. 1 nfast    nfast      179 Jun 14 14:00 config
-rwxr-xr-x. 1 nfast    nfast     3488 Jun 14 14:02 module_BD10-03E0-D947
-rwxr-xr-x. 1 nfast    nfast    39968 Jun 14 14:02 world
```

The `pkcs11.log` is described in Generate keys using different key protection methods.

2.5. Generate keys using different key protection methods

This section describes how to populate the `<nfs-directory>/cknfastrc` file based on what method of key protection you choose.

1. If you want to generate keys with module protection, add the following line to the 'cknfastrc' file:

```
CKNFAST_FAKE_ACCELERATOR_LOGIN=1
```

2. To protect keys with softcard protection, add the following lines to `cknfastrc`:

```
CKNFAST_LOADSHARING=1
CKNFAST_FAKE_ACCELERATOR_LOGIN=1
```

   To generate a softcard, run the following command on a valid Security World installation:

```
% /opt/nfast/bin/ppmk --new <softcard-name>
```

   This generates a softcard file in `/opt/nfast/kmdata/local` which must be copied to `<nfs-directory>` to be accessed by both containers that are eventually created.

3. If you want to add PKCS11 debugging, add the following two lines to the `cknfastrc` file:

```
CKNFAST_DEBUG=10
CKNFAST_DEBUGFILE=/opt/nfast/kmdata/local/pkcs11.log
```

   Then create the `pkcs11.log` file. `/opt/nfast/kmdata/local` will be mounted to `<nfs-directory>`. This is why the `touch` command path and the debug file path differ.

```
% touch <nfs-directory>/pkcs11.log
```

2.6. Build the custom HSM image

Successful completion of this section requires the following files on your image-building machine:

- `make-nshield-ibmibp`.
- Security World ISO. This guide uses version `SecWorld_Lin64-12.80.4.iso`.

> The customer receives the `make-nshield-ibmibp` script with their purchase of nSCOP.

1. Ensure the `make-nshield-ibmibp` script has executable permissions:

```
% chmod +x make-nshield-ibmibp
```

2. Mount the appropriate security world ISO file to `/mnt`:

```
% mount -t iso9660 -o loop SecWorld_Lin64-12.80.4.iso /mnt
mount: /dev/loop0 is write-protected, mounting read-only
```

3. Run the script:

```
% ./make-nshield-ibmibp --from registry.access.redhat.com/ubi8/ubi:latest --tag rh8nshieldibm /mnt
Detecting nShield software version
Version is 12.80.4
Unpacking hwsp...
...
Building image...
Sending build context to Docker daemon  702.4MB
...
Successfully built 9764c359f3cc
Successfully tagged rh8nshieldibm:latest
```

4. Verify the image was built successfully:

```
% docker images
REPOSITORY                              TAG       IMAGE ID       CREATED         SIZE
rh8nshieldibm                           latest    9764c359f3cc   2 minutes ago   929MB
```

5. Umount the ISO:

```
% umount /mnt
```

2.7. Getting the rh8nshieldibm image into the OpenShift container image registry

This section details the steps needed to get the `rh8nshieldibm` image into the OpenShift image registry. Docker or Podman can be used. OpenShift supports pulling a container image for deployment from an external docker registry. In this guide, the external registry is *<external-docker-registry-IP-address>*.

To deploy the `rh8nshieldibm` image for use with OpenShift:

1. Retag the image:

```
% docker tag rh8nshieldibm <external-docker-registry-IP-address>/<image-name>
```

2. Log in to the external registry and enter the password when prompted:

```
% docker login -u <registry-username> <external-docker-registry-IP-address>
```

3. Push the docker image to the external registry:

```
% docker push <external-docker-registry-IP-address>/<image-name>
```

4. List the nodes to view the compute/worker nodes:

```
% oc get nodes
NAME                          STATUS   ROLES    AGE     VERSION
ocp4843-w9lph-master-0        Ready    master   4d20h   v1.21.11+6b3cbdd
ocp4843-w9lph-master-1        Ready    master   4d20h   v1.21.11+6b3cbdd
ocp4843-w9lph-master-2        Ready    master   4d20h   v1.21.11+6b3cbdd
ocp4843-w9lph-worker-925wf    Ready    worker   4d20h   v1.21.11+6b3cbdd
ocp4843-w9lph-worker-dzpmk    Ready    worker   4d20h   v1.21.11+6b3cbdd
ocp4843-w9lph-worker-gl8gr    Ready    worker   4d20h   v1.21.11+6b3cbdd
```

5. Create a shell for one of the compute nodes listed:

```
% oc debug nodes/ocp4843-w9lph-master-0
Starting pod/ocp4843-w9lph-master-0-debug ...
To use host binaries, run `chroot /host`
Pod IP: 10.194.148.228
If you don't see a command prompt, try pressing enter.
sh-4.4#
```

6. Create access to tools such as `oc` and Podman on the node:

```
% chroot /host
```

7. Log into the container platform from the node:

```
% oc login -u kubeadmin -p <cluster-password> <console-url>
```

8. Ensure you are using the same project on your cluster that the pods running the IBM Blockchain operator and console:

```
% oc project <ibp-project-name>
Now using project <ibp-project-name> on server <console-url>
```

9. Log in to the container image registry:

```
% podman login -u kubeadmin -p $(oc whoami -t) image-registry.openshift-image-registry.svc:5000
```

10. Log in to the remote registry and enter the password when prompted:

```
% podman login -u <registry-username> <external-docker-registry-IP-address>
```

11. Pull the image from the remote registry:

```
% podman image pull <external-docker-registry-IP-address>/<image-name>
```

12. List the downloaded image:

```
% podman images
<external-docker-registry-IP-address>/<image-name>   latest   6af59fac9600   21 hours ago    734 MB
```

13. Retag the image:

```
% podman tag <external-docker-registry-IP-address>/<image-name> image-registry.openshift-image-
registry.svc:5000/openshift/rh8nshieldibm
```

14. Push the image to the container platform registry:

```
% podman push image-registry.openshift-image-registry.svc:5000/openshift/rh8nshieldibm
Getting image source signatures
Copying blob ... ... done
...
Writing manifest to image destination
Storing signatures
```

15. Remove the debug pod:

```
% exit
exit
% exit
exit

Removing debug pod ...
```

2.8. Create the rh8nshieldibm image pull secret

This section details the steps needed to create an image pull secret for the `rh8nshieldibm` image. This pull secret is needed so that the IBM Blockchain Platform nodes can pull the image from the OpenShift container image registry. The pull secret uses a token as part of the credentials to pull images.

1. Ensure you have a valid token:

```
% oc whoami -t
a-Jj-mALh...
```

2. Create the secret:

```
% oc create secret docker-registry hsm-docker-secret --docker-server=image-registry.openshift-image
-registry.svc:5000 --docker-username=kubeadmin --docker-password=$(oc whoami -t) --docker-email=<email> -n <ibp-
project-name>
```

In this example:

   - Replace *<email>* with any email address.
   - Replace *<ibp-project-name>* with the namespace where your IBM Blockchain Platform operator and console are deployed.
   - Note how the Docker password is simply the token.

2.9. Create the persistent volume, persistent volume claim, and configmap for the HSM

This section describes how to deploy the persistent volume and persistent volume claims on your cluster so that the IBM Blockchain Platform nodes can access their data on the NFS server. It also covers how to deploy the HSM ConfigMap, which is pulled by the platform to store the HSM configuration.

> See *Sample YAML files* for YAML files you can adapt to your system.

1. Edit the `hsm-pv.yaml` file.

   Note the prefix being used in the name of the PV as this will will be used later in the integration. In this file, use the same nfs path and storage class as when IBM Blockchain was deployed.

```
  metadata:
    name: ibmblockchain-pv

  nfs:
    path: /your/nfs/path
    server: xx.xxx.xxx.xxx
  storageClassName: nfs-storage
```

2. Edit the `hsm-pvc.yaml` file.

   Note the prefix being used in the name of the PVC as this will will be used later in the integration. It should match the same prefix on the PV.

```
metadata:
  name: ibmblockchain-pvc
```

3. Edit the `hsm-cm.yaml` file.

   Note that the ConfigMap uses the image built earlier. This is required in the internal OpenShift registry.

```
      image: >-
        image-registry.openshift-image-registry.svc:5000/openshift/rh8nshieldibm
```

   If using softcard protection, change the `hsm-cm.yaml` file and add the following environment variable to the **envs** section:

```
    envs:
      - name: CKNFAST_LOADSHARING
        value: 1
```

4. Create the PV, PVC, and ConfigMap on the cluster:

```
% oc apply -f hsm-pv.yaml
persistentvolume/test1ca-pv created

% oc apply -f hsm-pvc.yaml
persistentvolumeclaim/test1ca-pvc created

% oc apply -f hsm-cm.yaml
configmap/ibp-hsm-config created
```

5. Verify everything was deployed successfully:

```
% oc get cm
NAME            DATA   AGE
ibp-hsm-config  1      40s

% oc get pv
NAME             CAPACITY   ACCESS RECLAIM P. STATUS   CLAIM                                  STORAGECLASS
REASON   AGE
ibmblockchain-pv  100Gi      RWX    Retain     Bound    ibm-blockchain-proj/ibmblockchain-pvc   nfs-storage
20s

% oc get pvc
NAME              STATUS   VOLUME            CAPACITY   ACCESS MODES   STORAGECLASS   AGE
ibmblockchain-pvc  Bound    ibmblockchain-pv   100Gi      RWX            nfs-storage    65s
```

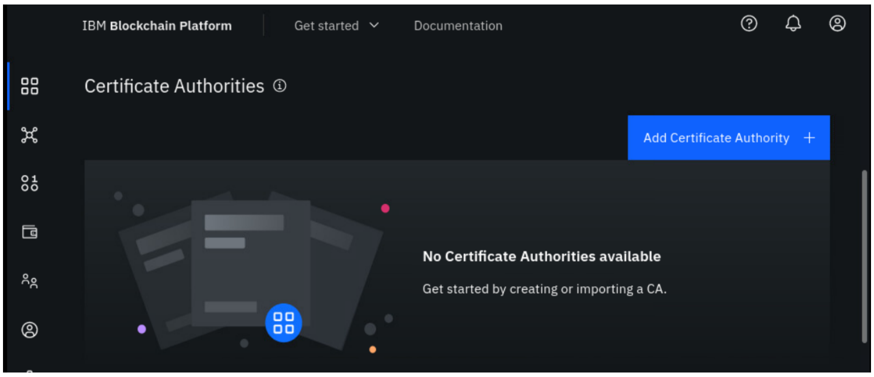
The prefixes of the PV and PVC are important. In the above example, the prefix is `ibmblockchain`. This must exactly match the name of the certificate authority node you will later create from the IBM Blockchain Platform console.

2.10. Deploy the IBM Blockchain certificate authority node

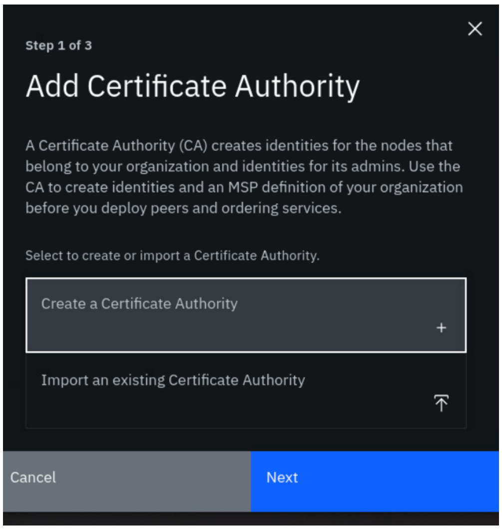
This section describes how to deploy a certificate authority (CA) node from the IBM Blockchain console and configure it with an Entrust HSM.

To deploy the CA node from the IBM Blockchain console:

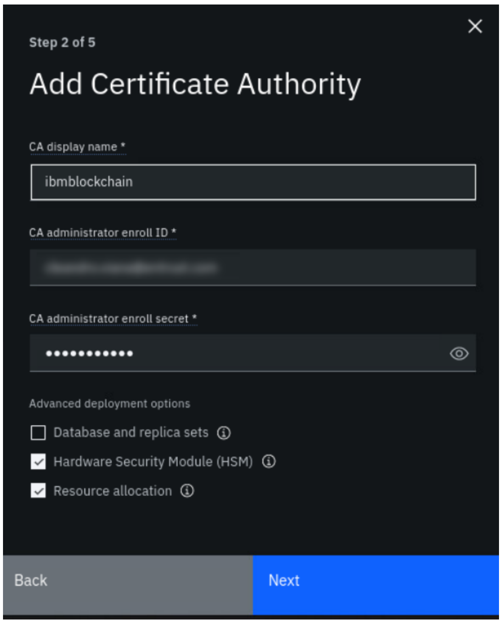
1. Browse to your IBM Blockchain console and log in using your credentials.
2. After logging in, from the **Nodes** page, scroll down and select **Add Certificate Authority +**.



3. Select **Create a Certificate Authority** and select **Next**.



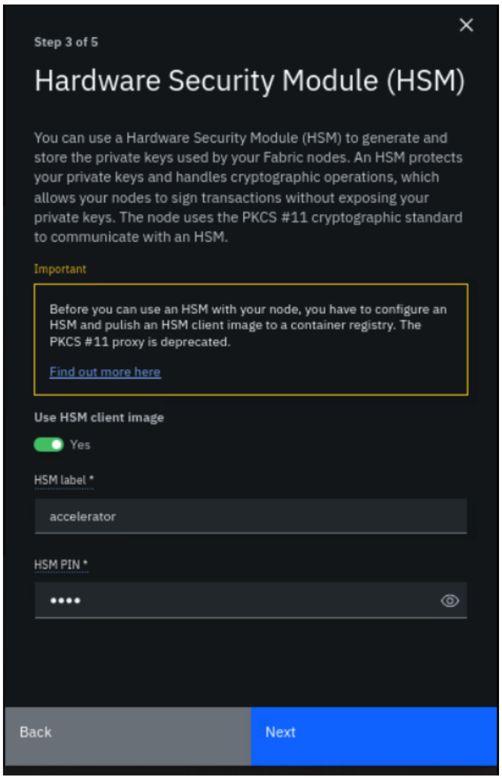
4. Enter a CA display name. This must match the prefix of the name of the Persistent Volume and Persistent Volume Claim, which can be found in either `hsm-pv.yaml` or `hsm-pvc.yaml`. In this example:
   - The name of the PVC is `ibmblockchain-pvc`.
   - The CA display name is `ibmblockchain`.
5. Enter a CA administrator enroll ID. Ideally, this should match the email used to sign into the console.
6. Enter a CA administrator enroll secret. Remember your enroll ID and secret as these will be needed later to associate an admin identity with the root CA.
7. Under **Advanced deployment options**, select the checkboxes for **Hardware Security Module (HSM)** and **Resource allocation**.



8. Select **Next**.
9. The next configuration page should pertain to the HSM. If the HSM ConfigMap (`hsm-cm.yaml`) was deployed and the custom HSM image was created and imported into

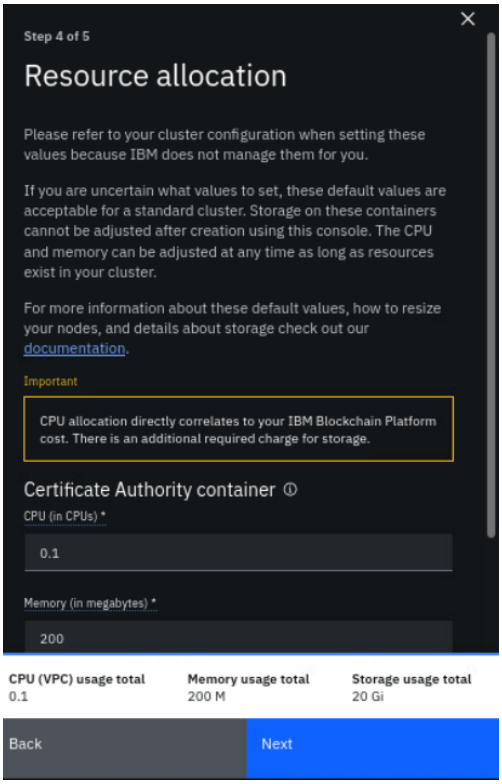
the OpenShift container image registry correctly, you should see a toggle for **Use HSM client image**. Make sure this toggle is present and toggled on.

10. For **HSM label**, enter `accelerator`.
11. For **HSM PIN**, anything can be entered (remember this number). For example, a valid PIN is `1234`.

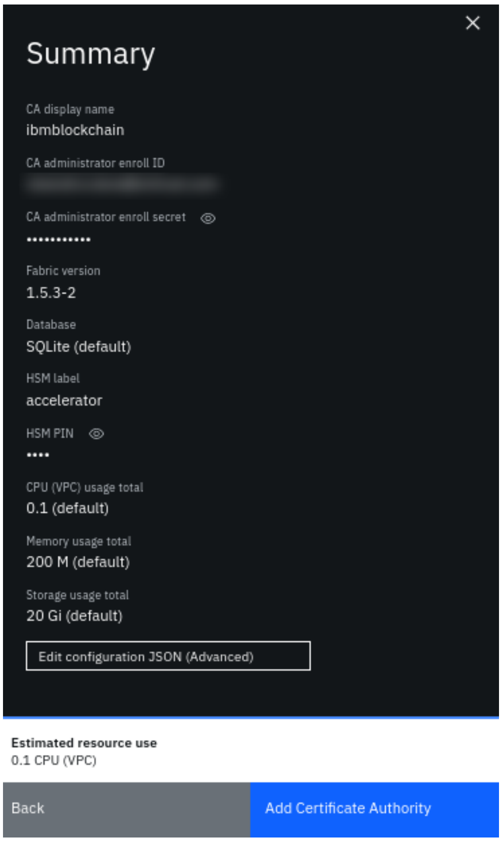


12. Select **Next** when finished.
13. The next configuration page should pertain to resource allocation.

    The default resource allocation is 0.1 CPU, 200 MB memory, and 20 GB storage. Change these numbers if needed, but make sure enough disk storage exists on the NFS server if you decide to increase the storage capacity.

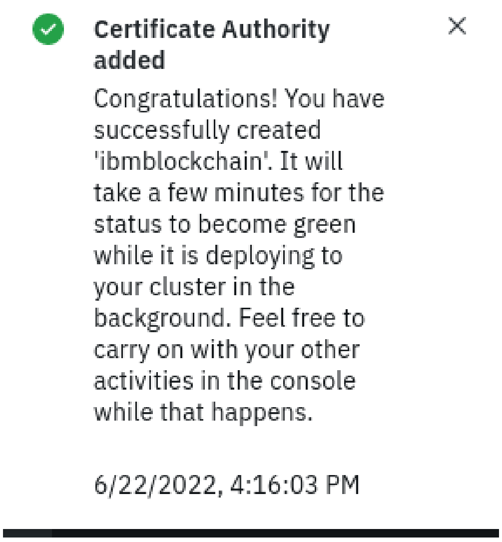


14. Select **Next** when finished.
15. Review the CA configuration.



16. When finished, select **Add Certificate Authority**.

    You should see a success message that the CA was added and the CA should show on the console dashboard under **Certificate Authorities**. For example:



There may be issues with the HSM and troubleshooting may be required at this stage, see Troubleshooting the CA node deployment. If there are no issues, continue from Check if deployment is successful.

2.11. Troubleshooting the CA node deployment

Here are some examples of things you can do to troubleshoot the CA node deployment:

1. Look at the logs for the `certgen` container:

```
% oc logs <ca-pod-name> certgen
```

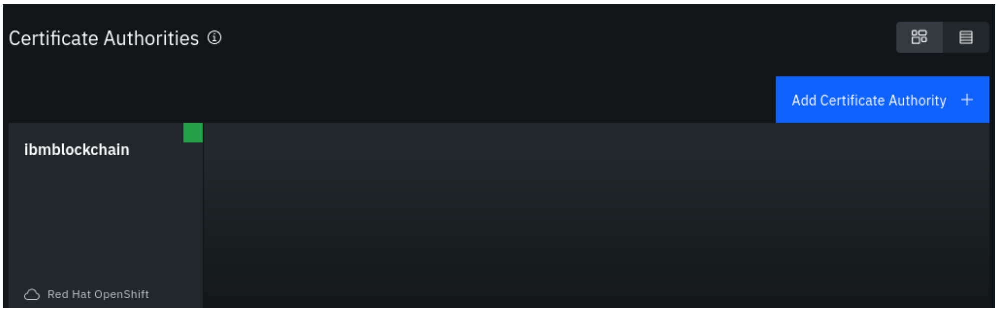
2. Look at the logs for the `hsm-daemon` container:

```
% oc logs <ca-pod-name> hsm-daemon
```

2.12. Check if deployment is successful

To check if a deployment is successful:

1. Wait for the CA to become operational. This may take a few minutes. When the CA is operational, a green box will show inside the CA square on the console dashboard. For example:



2. Verify that the CA files are present on the NFS server at `<nfs-directory>`:

```
% cd <nfs-directory>; ls
...
fabric-ca-server
...

% cd fabric-ca-server; ls
ca  db  tlsca
```

3. Verify that the `key_pkcs11` files are present on the NFS server at `<nfs-directory>`:

```
-rw-r--r--. 1 root  root    8008 Jun 21 15:37 key_pkcs11_ua4ef5b903573703fc63055eb1579d217f6a4bf6df
-rw-r--r--. 1 root  root    8008 Jun 21 15:37 key_pkcs11_ua641a8aa31ee1b3f9c61353d322a95d3f28fe9720
```

4. If the CA is successfully deployed, you should see the following pod listing:

```
% oc get pods
NAME                                      READY   STATUS    RESTARTS   AGE
ibmblockchain-789d7689ff-r7ctp            2/2     Running   0          20h
ibp-operator-6cc567667c-nqrlw             1/1     Running   0          20h
ibpconsole-58b57bb947-jc26w               4/4     Running   0          20h
nfs-client-provisioner-564f7cb984-bc9zh   1/1     Running   0          20h
```

5. The CA key that was generated via PKCS #11 by the HSM is now stored and protected within the HSM.

Appendix A: Sample YAML files

A.1. hsm-pv.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ibmblockchain-pv
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 100Gi
  nfs:
    path: <nfs-directory>
    server: <nfs-server-IP>
  persistentVolumeReclaimPolicy: Retain
  storageClassName: <storage-class-name>
  volumeMode: Filesystem
```

A.2. hsm-pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ibmblockchain-pvc
  namespace: ibm-blockchain-proj
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: <storage-class-name>
  volumeMode: Filesystem
  volumeName: ibmblockchain-pv
```

A.3. hsm-cm.yaml

```yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: ibp-hsm-config
data:
  ibp-hsm-config.yaml: |
    library:
      filepath: /opt/nfast/toolkits/pkcs11/libcknfast.so
      image: >-
        image-registry.openshift-image-registry.svc:5000/openshift/rh8nshieldibm
      auth:
        imagePullSecret: hsm-docker-secret
    daemon:
      image: >-
        image-registry.openshift-image-registry.svc:5000/openshift/rh8nshieldibm
      auth:
        imagePullSecret: hsm-docker-secret
    envs:
      - name: LD_LIBRARY_PATH
        value: /stdll
      - name: CKNFAST_FAKE_ACCELERATOR_LOGIN
        value: 1
      - name: CKNFAST_DEBUG
        value: 10
      - name: CKNFAST_DEBUGFILE
        value: /opt/nfast/kmdata/local/pkcs11.log
      - name: NFAST_SERVER
        value: /shared/sockets/nserver
      - name: NFAST_PRIVSERVER
        value: /shared/sockets/privnserver
    mountpaths:
      - mountpath: /opt/nfast/kmdata/local
        name: tokeninfo
        usePVC: true
    type: hsm
    version: v1
```

A.4. nfs-rbac.yaml

```yaml
kind: ServiceAccount
apiVersion: v1
metadata:
  name: nfs-client-provisioner
  namespace: ibm-blockchain-proj
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: nfs-client-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "update", "patch"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: run-nfs-client-provisioner
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    namespace: ibm-blockchain-proj
roleRef:
  kind: ClusterRole
  name: nfs-client-provisioner-runner
  apiGroup: rbac.authorization.k8s.io
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  namespace: ibm-blockchain-proj
rules:
  - apiGroups: [""]
    resources: ["endpoints"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  namespace: ibm-blockchain-proj
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    namespace: ibm-blockchain-proj
roleRef:
  kind: Role
  name: leader-locking-nfs-client-provisioner
  apiGroup: rbac.authorization.k8s.io
```

A.5. storage-class.yaml

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name>
provisioner: k8s-sigs.io/nfs-subdir-external-provisioner
parameters:
  pathPattern: "ibmblockchain"
  archiveOnDelete: "false"
```

A.6. nfs-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nfs-client-provisioner
  namespace: ibm-blockchain-proj
  labels:
    app: nfs-client-provisioner
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: nfs-client-provisioner
  template:
    metadata:
      labels:
        app: nfs-client-provisioner
    spec:
      serviceAccountName: nfs-client-provisioner
      containers:
        - name: nfs-client-provisioner
          image: k8s.gcr.io/sig-storage/nfs-subdir-external-provisioner:v4.0.2
          volumeMounts:
            - name: nfs-client-root
              mountPath: /persistentvolumes
          env:
            - name: PROVISIONER_NAME
              value: k8s-sigs.io/nfs-subdir-external-provisioner
            - name: NFS_SERVER
              value: <nfs-server-IP>
            - name: NFS_PATH
              value: <nfs-directory>
      volumes:
        - name: nfs-client-root
          nfs:
            server: <nfs-server-IP>
            path: <nfs-directory>
```