IBM Blockchain

nShield® HSM Integration Guide
Contents

1. Introduction ......................................................... 4
   1.1. Product configurations ......................................... 4
   1.2. Requirements .................................................. 5
2. Procedures .......................................................... 6
   2.1. Create the NFS storage class for the cluster .................. 6
   2.2. Generate the HSM config files and copying files to the persistent volume ................. 7
   2.3. Generate keys using different key protection methods ......................... 8
   2.4. Build the custom HSM image .................................... 9
   2.5. Getting the rh8nshieldibm image into the OpenShift container image registry .......... 10
   2.6. Create the rh8nshieldibm image pull secret .................... 12
   2.7. Create the persistent volume, persistent volume claim, and configmap for the HSM .................................................. 12
   2.8. Deploy the IBM Blockchain certificate authority node ............................... 13
 Appendix A: Sample YAML files ....................................... 17
   A.1. hsm-pv.yaml ................................................ 17
   A.2. hsm-pvc.yaml .............................................. 17
   A.3. hsm-cm.yaml ............................................. 17
   A.4. nfs-rbac.yaml ............................................... 18
   A.5. storage-class.yaml .......................................... 20
   A.6. nfs-client-prov-deployment.yaml ................................ 20
 Contact Us ..................................................................... 21
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 significant performance improvements and extra security by protecting and managing the keys with its FIPS 140-2 level 3 certified hardware.

This integration guide does not describe best practices for implementation. It only validates the integration.

The benefits of using an nShield Hardware Security Module (HSM) include:

- Secure storage of the private key.
- FIPS 140-2 level 3 validated hardware.
- Improved server performance by offloading the cryptographic processing.
- Full life cycle management of the keys.
- Failover support.
- Load balancing between HSMs.

You will use the Public-Key Cryptography Standards (PKCS #11) interface to integrate the HSM and IBM Blockchain Platform.

1.1. Product configurations

We have successfully tested nShield HSM integration in the following configurations:

<table>
<thead>
<tr>
<th>Product</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Blockchain Platform</td>
<td>2.5.2</td>
</tr>
<tr>
<td>OpenShift Container Platform</td>
<td>4.5.40</td>
</tr>
<tr>
<td>Base OS</td>
<td>RHEL 8.2</td>
</tr>
<tr>
<td>OS for the image-building machine and NFS server</td>
<td>CentOS 7</td>
</tr>
<tr>
<td>Product</td>
<td>Version</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>nShield HSM</td>
<td>Connect XC</td>
</tr>
<tr>
<td></td>
<td>Connect +</td>
</tr>
<tr>
<td>nShield Security World</td>
<td>12.60.11-TAC765</td>
</tr>
<tr>
<td>nShield Container Option Pack (nSCOP)</td>
<td>1.1</td>
</tr>
<tr>
<td>VMware</td>
<td>ESXi 6.7.0 on a Dell PowerEdge R740</td>
</tr>
<tr>
<td>Docker CE</td>
<td>20.10.6</td>
</tr>
<tr>
<td>Podman</td>
<td>3.0.2-dev</td>
</tr>
</tbody>
</table>

1.2. Requirements

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

To perform the integration tasks, you must have:

- \texttt{root} access on the operating system.
- Access to \texttt{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:

- The number and quorum of Administrator Cards in the Administrator Card Set (ACS), and the policy for managing these cards.
- Whether the application keys are protected by the module, an Operator Card Set (OCS) or a Softcard with or without a pass phrase.
- The number and quorum of Operator Cards in the OCS, and the policy for managing these cards.
- Whether the Security World should be compliant with FIPS 140-2 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 to be accessed by the IBM Blockchain Platform nodes via an OpenShift or Kubernetes persistent volume. Instructions for this are detailed 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).

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

2.1. 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, denoted <storage-class-name>, and the NFS share directory, denoted <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.

The following yaml files may be useful to setup NFS with OpenShift:
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
% oc apply -f storage-class.yaml -n ibm-blockchain-proj
% oc apply -f nfs-client-prov-deployment.yaml -n ibm-blockchain-proj
```

If you are using an NFS server that is already setup with your cluster confirm the following.

```
% oc get storageclasses
nfs-storage (default) ...
```

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

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

This section describes steps on how to generate the HSM config file after installing nSCOP, and copying the required files to the persistent volume to 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/kmdatalocal.

```
% ./make-nshield-hwsp-config --output /opt/nfast/kmdatalocal/config <HSM-IP>
```
3. Verify your `config` file looks like the following.

```plaintext
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/kmddata/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.

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

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

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

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

### 2.3. 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, populate the `cknfastrc` file with the following line.

   ```plaintext
   CKNFAST_FAKE_ACCELERATOR_LOGIN=1
   ```

2. To protect keys with OCS or softcard protection, populate `cknfastrc` with these lines.

   ```plaintext
   CKNFAST_LOADSHARING=1
   CKNFAST_FAKE_ACCELERATOR_LOGIN=1
   ```
a. To generate the OCS, run the following command where you have a valid Security World installation.

```bash
% /opt/nfast/bin/createocs -m<module_number> -Q k/N -n <ocs-name>
```

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

b. To generate a softcard, run the following command where you have a valid Security World installation.

```bash
% /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. Then create the `pkcs11.log` file. Note that `/opt/nfast/kmdata/local` will be mounted to `<nfs-directory>`, so this is why the `touch` command path and debug file path differ.

```bash
% vi cknfastrc
CKNFAST_DEBUG=10
CKNFAST_DEBUGFILE=/opt/nfast/kmdata/local/pkcs11.log
% touch <nfs-directory>/pkcs11.log
```

### 2.4. Build the custom HSM image

Completing this section requires the following files on your image-building machine:

- `make-nshield-ibmibp`
- Security World ISO
  - We will use version `SecWorld_Lin64-12.60.11-TAC-765.iso`

A customer will receive the `make-nshield-ibmibp` script with their purchase of nSCOP.

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

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

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

```bash
% mount -t iso9660 -o loop SecWorld_Lin64-12.60.11-TAC-765.iso /mnt
mount: /dev/loop0 is write-protected, mounting read-only
```
3. Run the script with the following command.

```
% ./make-nshield-ibmibp --from registry.access.redhat.com/ubi8/ubi:latest --tag rh8nshieldibm /mnt
Detecting nShield software version
Version is 12.60.11
... Building image
Sending build context to Docker daemon
... Successfully tagged rh8nshieldibm:latest
```

4. Verify the image was built successfully.

```
% docker images
REPOSITORY     TAG     IMAGE ID  CREATED        SIZE
rh8nshieldibm  latest  ...       3 minutes ago  710MB
```

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

This section details the steps needed to get the `rh8nshieldibm` image that was just built 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
cptnod-0.openshift.interop.com   Ready    worker   8d    v1.18.3+64fc02b
cptnod-1.openshift.interop.com   Ready    worker   8d    v1.18.3+64fc02b
master-0.openshift.interop.com  Ready    master   8d    v1.18.3+64fc02b
master-1.openshift.interop.com  Ready    master   8d    v1.18.3+64fc02b
master-2.openshift.interop.com  Ready    master   8d    v1.18.3+64fc02b
```
5. Create a shell for one of the compute nodes listed.
   ```bash
   % oc debug nodes/cptnod-0.openshift.interop.com
   ```

6. Run the following command to have access to tools such as `oc` and Podman on the node.
   ```bash
   % chroot /host
   ```

7. Login to the container platform from the node.
   ```bash
   % 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 belong to.
   ```bash
   % oc project <ibp-project-name>
   Now using project <ibp-project-name> on server <console-url>
   ```

9. Log in to the container image registry.
   ```bash
   % 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.
    ```bash
    % podman login -u <registry-username> <external-docker-registry-IP-address>
    ```

11. Pull the image from the remote registry.
    ```bash
    % podman image pull <external-docker-registry-IP-address>/<image-name>
    ```

12. List the downloaded image.
    ```bash
    % podman images
    <external-docker-registry-IP-address>/<image-name> latest 6af59fac9600 21 hours ago 734 MB
    ```

13. Retag the image.
    ```bash
    % 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.
15. Remove the debug pod.

```
% exit
exit
% exit
exit
Removing debug pod ...
```

### 2.6. 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 by running the following command.

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

2. Create the secret. 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.

```
% 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>
```

### 2.7. 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.

Completing this section requires the following yaml files:
• hsm-pv.yaml
• hsm-pvc.yaml
• hsm-cm.yaml

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

1. Run the following commands to create the PV, PVC, and ConfigMap on the cluster.

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

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

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

2. Verify everything was deployed successfully.

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

% oc get pv
NAME        CAPACITY   ACCESS MODES  RECLAIM POLICY  STATUS  STORAGECLASS
test1ca-pv  100Gi      RWX           Retain          Bound   nfs-storage

% oc get pvc
NAME          STATUS    VOLUME      CAPACITY   ACCESS MODES  STORAGECLASS
test1ca-pvc   Bound     test1ca-pv  100Gi      RWX           nfs-storage
```

The prefixes of the pv and pvc are very important. In the above example, the prefix is test1ca. This must exactly match the name of the certificate authority node you will later create from the IBM Blockchain Platform console.

2.8. 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 our case, the name of the pvc was `test1ca-pvc`, so the CA display name should be `test1ca`.

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. Select Next.

8. 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.

9. For HSM label, enter accelerator.

10. For HSM PIN, anything can be entered (remember this number). For example, a valid PIN is 1234. Select Next when finished.

11. 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. Select Next when finished.

12. Review the CA configuration. When finished, select Add Certificate Authority.

13. Either from the OpenShift console UI, or from the command line, open a shell into the `hsm-daemon` container once the container is successfully created and run the following command to signal to the IBM container, `certgen`, that the HSM client is ready.

    ```
    % touch /shared/daemon-launched
    ```

14. You should now see the following in the `certgen` container logs, describing that the CA key was generated via PKCS11.
15. You should now see the following in the hsm-daemon container logs.

```
% oc logs <ca-pod-name> hsm-daemon
Hardserver INIT: Information: overridding setting unix_privsocket_name with environment variable NFAST_PRIVSERVER
with the value: /opt/nfast/sockets/priv/privserver

2021-06-17 21:53: t009729bdf77f0000: Hardserver [FP]: Notice: Hardserver using priority class queueing algorithm: 1 classes and 1 modules total.
Hardserver Config Spawner service: spawning command-line "/opt/nfast/sbin/server.conf module 1 operational"
2021-06-17 21:54: t009729bdf77f0000: Hardserver [FP]: Information: New client #13 connected
```

16. In <nfs-directory>, verify a key_pkcs11 file was created.

```
% ls -l <nfs-directory>
total 52
-rw-rw-r--. 1 nfast nfast 103 Jun 14 16:02 cknfastrc
-rw-rw-r--. 1 nfast nfast 179 Jun 14 16:02 config
-rw-rw-r--. 1 nfast nfast 7976 Jun 16 16:21 key_pkcs11...
-rw-rw-r--. 1 nfast nfast 3488 Jun 15 17:46 module_530E-02E0-D947
-rw-rw-r--. 1 root root 0 Jun 16 16:21 pkcs11.log
-rw-rw-r--. 1 nfast nfast 40632 Jun 14 16:02 world
```

17. At this point, you should see a success message that the CA was added and the CA should show on the console dashboard under Certificate Authorities.

18. Wait for the CA to become operational. This may take some time. When the CA is operational, a green box will show inside the CA square on the console dashboard.

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

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibp-operator-649c4f7db9-dkclf</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>9d</td>
</tr>
<tr>
<td>ibpconsole-7f89d64c7d-nj7fr</td>
<td>4/4</td>
<td>Running</td>
<td>1</td>
<td>9d</td>
</tr>
<tr>
<td>nfs-client-provisioner-c5f67557d-dxf86</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>138m</td>
</tr>
<tr>
<td>test1ca-86857f6fbf8-n9901c</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>5m36s</td>
</tr>
</tbody>
</table>

21. The CA key that was generated via PKCS11 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: test1ca-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: test1ca-pvc
  namespace: <ibp-project-name>
spec:
  accessModes:
  -ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: <storage-class-name>
  volumeMode: Filesystem
  volumeName: test1ca-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/privserver
mountpaths:
  - mountpath: /opt/nfast/kmdata/local
    name: tokeninfo
    usePVC: true
    type: hsm
    version: v1
```

A.4. nfs-rbac.yaml
kind: ServiceAccount
apiVersion: v1
metadata:
  name: nfs-client-provisioner
  namespace: <ibp-project-name>

---
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: ["persistentvolumeclasses"]
    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
  namespace: <ibp-project-name>
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    namespace: <ibp-project-name>
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: <ibp-project-name>
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: <ibp-project-name>
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    namespace: <ibp-project-name>
roleRef:
  kind: Role
  name: leader-locking-nfs-client-provisioner
  apiGroup: rbac.authorization.k8s.io

---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  namespace: <ibp-project-name>
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: <ibp-project-name>
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    namespace: <ibp-project-name>
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>
  annotations:
    storageclass.kubernetes.io/is-default-class: 'true'
    kubernetes.io/description: Storage Class for NFS Storage for IBM Blockchain
  provisioner: <cluster-domain>/nfs
parameters:
  archiveOnDelete: "false"
```

A.6. nfs-client-prov-deployment.yaml

```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nfs-client-provisioner
  namespace: <ibp-project-name>
  labels:
    apps: nfs-client-provisioner
spec:
  selector:
    matchLabels:
      app: nfs-client-provisioner
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nfs-client-provisioner
    spec:
      serviceAccountName: nfs-client-provisioner
      containers:
      - name: nfs-client-provisioner
        image: quay.io/external_storage/nfs-client-provisioner:latest
        resources:
          limits:
            cpu: 1
            memory: 16i
        volumeMounts:
        - name: nfs-client-root
          mountPath: /persistentvolumes
        env:
        - name: PROVISIONER_NAME
          value: <cluster-domain>/nfs
        - 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>
```
Contact Us

Web site
https://www.entrust.com

Support
https://nshieldsupport.entrust.com

Email Support
nShield.support@entrust.com

Online documentation:
Available from the Support site listed above.

You can also contact our Support teams by telephone, using the following numbers:

Europe, Middle East, and Africa
United Kingdom:
+44 1223 622444
One Station Square
Cambridge, UK CB1 2GA

Americas
Toll Free:
+1 833 425 1990
Fort Lauderdale:
+1 954 953 5229
Sawgrass Commerce Center – A Suite 130
13800 NW 14 Street
Sunrise, FL 33323 USA

Asia Pacific
Australia:
+61 8 9126 9070
World Trade Centre Northbank Wharf Siddeley St
Melbourne VIC 3005 Australia

Japan:
+81 50 3196 4994

Hong Kong:
+852 3008 3188
31/F, Hysan Place,
500 Hennessy Road,
Causeway Bay
ABOUT ENTRUST CORPORATION

Entrust keeps the world moving safely by enabling trusted identities, payments, and data protection. Today more than ever, people demand seamless, secure experiences, whether they’re crossing borders, making a purchase, accessing e-government services, or logging into corporate networks. Entrust offers an unmatched breadth of digital security and credential issuance solutions at the very heart of all these interactions. With more than 2,500 colleagues, a network of global partners, and customers in over 150 countries, it’s no wonder the world’s most entrusted organizations trust us.