HashiCorp Vault Enterprise and Kubernetes Support

nShield® HSM Integration Guide

2024-10-21

Table of Contents

Chapter 1. Introduction

HashiCorp Vault (referred to as Vault in this guide) protects your organization's credentials and confidential assets and provides secure access control to them through a process of secret leasing, renewal, and revocation. Entrust nShield Hardware Security Modules (HSMs) provide FIPS or Common Criteria certified solutions to securely generate, encrypt, and decrypt the keys which form the foundation of the HashiCorp Vault protection mechanism. The nShield HSM secures the key used to seal or unseal a Vault instance.

This guide describes how to integrate the HashiCorp Vault Enterprise with an nShield HSM on a server, or on a Kubernetes environment.

1.1. Product configurations

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

Product	Version
HashiCorp Vault	1.9.2 Enterprise HSM
Docker	20.10.3
Red Hat OpenShift	Client: 4.9.5, Server: 4.9.5, Kubernetes: v1.22.0-rc.0+a44d0f0
Base OS	Red Hat Enterprise 8.3

1.1.1. Supported nShield features

Entrust has successfully tested nShield HSM integration with the following features:

Feature	Support
Softcards	Yes
Module Only Key	Yes
OCS cards	Yes
nSaaS	Supported but not tested

1.1.2. Supported nShield hardware and software versions

Entrust has successfully tested with the following nShield hardware and software versions:

1.1.2.1. Connect XC

Security World Software	Firmware	Netimage	OCS	Softcard	Module
12.80.4	12.50.11 (FIPS 140-2 certified)	12.80.4	✓	✓	✓
12.80.4	CC 12.60.15	12.80.4	✓	✓	✓

Supported nShield Container Option Pack:

Product	Version
nCOP	1.1.1

1.2. Requirements

Before installing these products, read the associated nShield HSM *Installation Guide*, *User Guide*, and the HashiCorp Vault documentation. This guide assumes familiarity with the following:

- The importance of a correct quorum for the Administrator Card Set (ACS).
- Whether Operator Card Set (OCS) protection or Softcard protection is required.
- If OCS protection is to be used, a 1-of-N quorum must be used.
- Whether your Security World must comply with FIPS 140 Level 3 or Common Criteria standards. If using FIPS 140 Level 3, it is advisable to create an OCS for FIPS authorization. The OCS can also provide key protection for the Vault master key. For information about limitations on FIPS authorization, see the *Installation Guide* of the nShield HSM.

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

- Whether to instantiate the Security World as recoverable or not.
- Network environment setup, via correct firewall configuration with usable ports: 9004 for the HSM and 8200 for Vault.
- HashiCorp Enterprise Modules license, which is required for using Vault with Hardware Security Modules.

1.3. More information

For more information about OS support, contact your HashiCorp Vault sales representative or Entrust nShield Support, https://nshieldsupport.entrust.com.

Chapter 2. Integration Procedures

A dedicated Linux server is needed for the installation of HashiCorp Vault.

Follow these steps to install and configure the Vault with a single HSM:

1. System preparation
2. Install the HSM
3. Install the Security World software and create a Security World
4. Generate the keys with OCS and Softcard protection
5. Verify the PKCS#11 library is available
6. Find the slot value for each protection method
7. Create Vault user and group
8. Install Vault
9. Install the Vault license
10. Create a configuration file
11. Create and configure Vault directories
12. Enable Vault
13. Start Vault
14. Log in from the command line
15. Log in from the web user interface
16. Examine Vault secrets
17. Enable the KV engine
18. Write secret data
19. Retrieve secret data
20. Seal Vault
21. Rotate keys stored in the HSM and used by Vault
22. Change protection method

2.1. System preparation

1. Open the firewall for HSM incoming connections:

```
# sudo firewall-cmd --permanent --add-port=9004/tcp
```

2. Open the firewall for HashiCorp Vault incoming connections:

```
# sudo firewall-cmd --permanent --add-port=8200/tcp
# sudo firewall-cmd --permanent --add-port=8201/tcp
```

3. Apply the firewall changes above:

```
# sudo firewall-cmd --reload
```

4. Install `open-vm-tools`:

```
# sudo yum install open-vm-tools unzip opensc
```

2.2. Install the HSM

Install the nShield Connect HSM locally, remotely, or remotely via the serial console. See the following nShield Support articles, and the *Installation Guide* for the HSM:

- How to locally set up a new or replacement nShield Connect
- How to remotely set up a new or replacement nShield Connect
- How to remotely set up a new or replacement nShield Connect XC Serial Console model

Access to the Entrust nShield Support Portal is available to customers under maintenance. To request an account, contact nshield.support@entrust.com.

2.3. Install the Security World software and create a Security World

1. Install and configure the Security World software. For instructions, see the *Installation Guide* and the *User Guide* for the HSM.
2. When the Software has been installed, ensure that `$NFAST_HOME` is on the path:

```
# sudo vi /etc/profile.d/nfast.sh
```

Add the following info to `nfast.sh` and save:

```
# Entrust Security World path variable
export PATH="$PATH:/opt/nfast/bin"
```

3. Restart the server.
4. Confirm that the HSM is available:

```
# enquiry
```

5. Create your Security World if one does not already exist.
6. Confirm that the Security World is **operational** and **usable** in the output of the `nfkminfo` command:

```
# nfkminfo
```

2.4. Generate the keys with OCS and Softcard protection

The Vault seal key can be protected with an OCS, Softcard or Module:

- Operator Cards Set (OCS) are smartcards that are presented to the physical smartcard reader of a HSM. If an OCS is used, k must = 1 whereas N can be up to, but not exceed, 64. For more information on OCS use, properties, and k-of-N values, see the *User Guide* for your HSM.
- Softcards are logical tokens (passphrases) that protect they key and authorize its use.
- Module protection has no passphrase.

2.4.1. Generate key using an OCS protection

1. Create the OCS with example name and attributes, for 1-of-N (**1/2** for a single-HSM configuration):

```
# createocs -m1 -s2 -N HashiCorp -Q 1/1
```

2. Create an encryption key `vault_v1`:

```
# generatekey --generate --batch -m1 -s2 pkcs11 protect=token cardset=HashiCorp plainname=vault_v1 type=AES
size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               token
 slot         Slot to read cards from    2
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   AES
```

```
 size         Key size                  256
 plainname    Key name                  vault_v1
 nvram        Blob in NVRAM (needs ACS)  no

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

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ucde4fa7e3ad8d66b6c652d18140b37bab9fe9d106-
782f624f6eceb64a7515316d80a1c939b6e596e3
```

3. Create the HMAC key `vault_hmac_v1`:

```
# generatekey --generate --batch -m1 -s2 pkcs11 protect=token cardset=HashiCorp plainname=vault_hmac_v1
type=HMACSHA256 size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               token
 slot         Slot to read cards from    2
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   HMACSHA256
 size         Key size                   256
 plainname    Key name                   vault_hmac_v1
 nvram        Blob in NVRAM (needs ACS)  no

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

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ucde4fa7e3ad8d66b6c652d18140b37bab9fe9d106-
e0cd52a5b1eebb695a102fadf1ddb50e375ddcaf
```

2.4.2. Generate key using Softcard and Module protection

1. Create a `/opt/nfast/cknfastrc` file with the following content:

```
# cat /opt/nfast/cknfastrc
CKNFAST_LOADSHARING=1
CKNFAST_FAKE_ACCELERATOR_LOGIN=1
CKNFAST_DEBUG=10
CKNFAST_DEBUGFILE=/opt/nfast/log/pkcs11.log
```

The first line enables Softcard protection support. The second enables Module protection support. Third and fourth enable PKCS11 log files which you will need to find the `slot` to configure the Vault.

2. Create the Softcard token using the `ppmk` command. Enter a passphrase or password at the prompt.

```
# ppmk -n vaultsc
Enter new pass phrase:
Enter new pass phrase again:
New softcard created: HKLTU Softcard_ID
```

3. Create an encryption key `vault_v1_sc` using Softcard protection:

```
# generatekey --generate --batch -m1 pkcs11 protect=softcard softcard=vaultsc plainname=vault_v1_sc
type=AES size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               softcard
 softcard     Soft card to protect key   vaultsc
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   AES
 size         Key size                   256
 plainname    Key name                   vault_v1_sc
 nvram        Blob in NVRAM (needs ACS)  no
Please enter the pass phrase for softcard `vaultsc':

Please wait........
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ucb5df6e12703825562ce731e3286a4fb9f46e767a-
ebc7da3d8e2f9aa86377dc4e5269157557cddd1c
```

4. Create the HMAC key `vault_hmac_v1_sc` using Softcard protection:

```
# generatekey --generate --batch -m1 pkcs11 protect=softcard softcard=vaultsc plainname=vault_hmac_v1_sc
type=HMACSHA256 size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               softcard
 softcard     Soft card to protect key   vaultsc
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   HMACSHA256
 size         Key size                   256
 plainname    Key name                   vault_hmac_v1_sc
 nvram        Blob in NVRAM (needs ACS)  no
Please enter the pass phrase for softcard `vaultsc':

Please wait........
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ucb5df6e12703825562ce731e3286a4fb9f46e767a-
376268c6c89c1657fb22ca1f08fe4f20b58b1c07
```

5. Verify the keys created:

```
# rocs
`rocs' key recovery tool
Useful commands: `help', `help intro', `quit'.
```

```
rocs> list keys
  No. Name                    App        Protected by
    1 vault_v1                pkcs11     HashiCorp
    2 vault_hmac_v1           pkcs11     HashiCorp
    3 vault_v1_sc             pkcs11     vaultsc (vaultsc)
    4 vault_hmac_v1_sc        pkcs11     vaultsc (vaultsc)
rocs> exit
```

6. Create an encryption key `vault_v1` and HMAC key `vault_hmac_v1` using Module protection:

```
# generatekey --generate --batch -m1 -s2 pkcs11 protect=module softcard=vaultsc plainname=vault_v1_m
type=AES size=256
...
# generatekey --generate --batch -m1 -s2 pkcs11 protect=module softcard=vaultsc plainname=vault_hmac_v1_m
type=HMACSHA256 size=256
...
```

2.5. Verify the PKCS#11 library is available

In the example below an Operator Card Set has been created with the name **HashiCorp**. The card is present in the physical slot (card reader) of the HSM, and is loaded to slot **#1**.

1. Execute the `ckcheckinst` command to test the library:

```
# ckcheckinst
PKCS#11 library interface version 2.01
                           flags 0
                  manufacturerID "nCipher Corp. Ltd               "
               libraryDescription "nCipher PKCS#11 12.60.11-837-510"
          implementation version 12.60
        Loadsharing and Failover enabled

Slot  Status            Label
====  ======            =====
   0  Fixed token       "loadshared accelerator          "
   1  Operator card     "HashiCorp                       "
   2  Soft token        "vaultsc                         "
```

2. Select the slot number of the Operator card and press the **Enter** key:

```
Select slot number to run library test or 'R'etry or to 'E'xit: 1
Using slot number 1.
```

3. Enter the passphrase for the OCS:

```
Please enter the passphrase for this token (No echo set).
Passphrase:

Test                     Pass/Failed
----                     -----------
```

```
1 Generate RSA key pair   Pass
2 Generate DSA key pair   Pass
3 Encryption/Decryption   Pass
4 Signing/Verification    Pass

Deleting test keys         ok

PKCS#11 library test successful.
```

2.6. Find the slot value for each protection method

Each protection method is loaded to a virtual slot. The decimal value of this slot will be needed further down to configure the Vault.

1. Run the `cklist` command. Notice the lines below.

```
# cklist
Listing contents of slot 0
 (token label "loadshared accelerator          ")
...
Listing contents of slot 1
 (token label "HashiCorp                       ")
...
Listing contents of slot 2
 (token label "vaultsc                         ")
```

`loadshared accelerator`	Module protection, that is slot 0.
`HashiCorp`	The name given to the OCS created above, slot 1.
`vaultsc`	The name given to the Softcard token created above, slot 2.

2. Search file `/opt/nfast/log/pkcs11.log` for `pSlotList`. Notice the hex value for each slot.

```
...
2021-02-05 12:53:36 [4866]: pkcs11: 00000000 <    pSlotList[0] 0x2D622495
2021-02-05 12:53:36 [4866]: pkcs11: 00000000 <    pSlotList[1] 0x2D622496
2021-02-05 12:53:36 [4866]: pkcs11: 00000000 <    pSlotList[2] 0x2D622497
...
```

3. Convert to decimal:

Protection Method	Slot Number	Value (Hex)	Value (Decimal)
Module	0	0x2D622495	761406613
OCS	1	0x2D622496	761406614

Protection Method	Slot Number	Value (Hex)	Value (Decimal)
Softcards	2	0x2D622497	761406615

Note or save the decimal values.

Adding or deleting Softcard tokens, or adding or deleting OCS, or adding or deleting Modules keys will change the values above. Redo the step to find the new values if necessary.

2.7. Create Vault user and group

1. Create the Vault group:

```
# sudo groupadd --system vault
```

2. Create the Vault user:

```
# sudo useradd --system --shell /sbin/nologin --gid vault vault
```

3. Add the Vault user to the nShield **nfast** group:

```
# sudo usermod --append --groups nfast vault
```

2.8. Install Vault

1. Download the Vault package from HashiCorp at https://releases.hashicorp.com/vault/, ensuring that it is the binary file for Enterprise with HSM support. For example:

```
# wget https://releases.hashicorp.com/vault/1.9.2+ent.hsm/vault_1.9.2+ent.hsm_linux_amd64.zip
--2021-12-23 15:30:53--
https://releases.hashicorp.com/vault/1.9.2+ent.hsm/vault_1.9.2+ent.hsm_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 151.101.1.183, 151.101.65.183,
151.101.129.183, ...
Connecting to releases.hashicorp.com (releases.hashicorp.com)|151.101.1.183|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 69760875 (67M) [application/zip]
Saving to: 'vault_1.9.2+ent.hsm_linux_amd64.zip'

vault_1.9.2+ent.hsm_lin 100%[============================>]  66.53M  10.7MB/s    in 6.2s

2021-12-23 15:30:59 (10.7 MB/s) - 'vault_1.9.2+ent.hsm_linux_amd64.zip' saved [69760875/69760875]
```

2. Unzip the binary file and extract it to the working directory on the host machine, for example `/usr/local/bin`. There should only be a single binary file named `vault_`.

```
# unzip vault_1.9.2+ent.hsm_linux_amd64.zip -d /usr/local/bin
```

3. Set Vault permissions:

```
# chmod 755 /usr/local/bin/vault
# setcap cap_ipc_lock=+ep /usr/local/bin/vault
# ls -la /usr/local/bin/vault
-rwxr-xr-x. 1 root root 137112792 Dec 18 09:06 /usr/local/bin/vault
```

4. Add the Vault binary file to the path:

```
# sudo vi /etc/profile.d/vault.sh
```

   Add the following information to `vault.sh` and save it.

```
# HashiCorp Vault path variable
export PATH="$PATH:/usr/local/bin"
export VAULT_ADDR=http://127.0.0.1:8200
```

5. Create the Vault data directories:

```
# sudo mkdir --parents /opt/vault/data
# sudo mkdir --parents /opt/vault/logs
# sudo chmod --recursive 750 /opt/vault
# sudo chown --recursive vault:vault /opt/vault
```

6. Restart the server.
7. Confirm that the binary file is available:

```
# vault version
Vault v1.9.2+ent.hsm (f7be55269a69543aedae108588e63688e6490b44) (cgo)
```

2.9. Install the Vault license

1. Open a new terminal and create a directory for the Vault license and configuration files:

```
# sudo mkdir /etc/vault
```

2. Three options are given in https://learn.hashicorp.com/tutorials/nomad/hashicorp-enterprise-license?in=vault/enterprise for enabling an enterprise license, as well as a procedure to request a trail license. For this guide, create a file containing the enterprise license key.

```
# cat /etc/vault/license.hclic
02MV4UU43BK5HGYYTOJZWFQMTMNNEWU33JLJDVCMKOGJDGQWSUIV2E6VCNGVMVGMBT...
```

2.10. Create a configuration file

Set up a `/etc/vault/config.hcl` configuration file to enable Vault to be run as a service. See also Vault commands.

An example configuration file for using Vault with OCS protection is shown below.

- Refer to Find the slot value for each protection method for the slot value.
- The entropy seal mode is set to augmentation. This leverages the HSM for augmenting system entropy via the PKCS#11 protocol.
- The seal wrap is enabled. By enabling seal wrap, Vault wraps your secrets with an extra layer of encryption leveraging the HSM encryption and decryption.
- Notice the path to the license file.

```
# PKCS#11 Seal, Entrust nShield Integration
seal "pkcs11" {
lib = "/opt/nfast/toolkits/pkcs11/libcknfast.so"
slot = "761406614"
pin = "ncipher"
key_label = "vault_v1"
hmac_key_label = "vault_hmac_v1"
# Vault should not generate the keys, the default
#generate_key = false
}
# Vault uses nCipher nShield for entropy augmentation
# This is an optional configuration
entropy "seal" {
mode = "augmentation"
}
# Vault listener with TLS disabled
listener "tcp" {
address = "0.0.0.0:8200"
tls_disable = true
}
storage "file" {
path = "/opt/vault/data"
}
# Seal Wrap is enabled, the Default,
# Entrust nShield is used to wrap the CSPs
#disable_sealwrap=false
ui = true
# License file
license_path = "/etc/vault/license.hclic"
```

The following is an example configuration file for using Vault with Softcard protection. Notice that the slot value changed from the OCS protection example above.

```
# PKCS#11 Seal, Entrust nShield Integration
seal "pkcs11" {
lib = "/opt/nfast/toolkits/pkcs11/libcknfast.so"
slot = "761406615"
pin = "ncipher"
key_label = "vault_v1_sc"
hmac_key_label = "vault_hmac_v1_sc"
# Vault should not generate the keys, the default
#generate_key = false
}
# Vault uses nCipher nShield for entropy augmentation
# This is an optional configuration
entropy "seal" {
mode = "augmentation"
}
# Vault listener with TLS disabled
listener "tcp" {
address = "0.0.0.0:8200"
tls_disable = true
}
storage "file" {
path = "/opt/vault/data"
}
# Seal Wrap is enabled, the Default,
# Entrust nShield is used to wrap the CSPs
#disable_sealwrap=false
ui = true
# License file
license_path = "/etc/vault/license.hclic"
```

2.11. Create and configure Vault directories

1. Create a vault file in `sysconfig`:

   ```
   # touch /etc/sysconfig/vault
   ```

2. Create a service file:

   ```
   # vi /etc/systemd/system/vault.service
   ```

3. Add the following information to the file.

   If deploying on a server with more than two CPUs, you may increase the value of `Environment=GOMAXPROCS` accordingly.

   ```
   [Unit]
   Description="HashiCorp Vault"
   Requires=network-online.target
   After=network-online.target nc_hardserver.service
   ```

```
ConditionFileNotEmpty=/etc/vault/config.hcl
[Service]
User=vault
Group=vault
EnvironmentFile=/etc/sysconfig/vault
ExecStart=/usr/local/bin/vault server -config=/etc/vault/config.hcl
StandardOutput=/opt/vault/logs/output.log
StandardError=/opt/vault/logs/error.log
ExecReload=/bin/kill --signal -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=5
TimeoutStopSec=30
StartLimitInterval=60
StartLimitBurst=3
AmbientCapabilities=CAP_IPC_LOCK
LimitNOFILE=65536
LimitMEMLOCK=infinity
[Install]
WantedBy=multi-user.target
```

4. If you are setting paths different from the default, you must edit the following lines as well in the configuration file:

```
ConditionFileNotEmpty=/etc/vault/config.hcl
EnvironmentFile=-/etc/sysconfig/vault
ExecStart=/opt/vault/bin/vault server -config=/etc/vault/config.hcl
StandardOutput=/opt/vault/logs/output.log
StandardError=/opt/vault/logs/error.log
```

2.12. Enable Vault

1. Set the following environment variable to allow Vault to be accessed from a web browser via the web user interface (web UI). Append the following line to the `/etc/profile.d/vault.sh` file created above, and restart the system.

```
export VAULT_ADDR=http://127.0.0.1:8200
```

2. Enable Vault:

```
# systemctl enable vault.service
```

2.13. Start Vault

The HSM will be accessed as part of starting Vault. Therefore, the OCS or Softcard is needed.

1. Start the Vault service:

The OCS card created in section `Generate key using an OCS protection` must be inserted in the HSM slot if the protection method defined in `/etc/vault/config.hcl` is OCS protection. Otherwise the Vault will fail to start. The OCS card is not required for the Vault to start if the protection method is Softcard on Module.

```
# systemctl start vault.service
```

2. Check that the Vault service is running:

```
# systemctl status vault.service
● vault.service - "HashiCorp Vault"
   Loaded: loaded (/etc/systemd/system/vault.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2022-01-05 09:48:56 EST; 58min ago
 Main PID: 16120 (vault)
    Tasks: 7 (limit: 49193)
   Memory: 108.2M
   CGroup: /system.slice/vault.service
           └─16120 /usr/local/bin/vault server -config=/etc/vault/config.hcl

Jan 05 10:47:08 Red_Hat_8.3_HashiCorp_Containers vault[16120]: 2022-01-05T10:47:08.045-0500 [INFO] >
Jan 05 10:47:08 Red_Hat_8.3_HashiCorp_Containers vault[16120]: 2022-01-05T10:47:08.045-0500 [WARN] >
...
```

3. Check the Vault status:

```
# vault status
Key                      Value
---                      -----
Recovery Seal Type       pkcs11
Initialized              false
Sealed                   true
Total Recovery Shares    0
Threshold                0
Unseal Progress          0/0
Unseal Nonce             n/a
Version                  1.9.2+ent.hsm
Storage Type             file
HA Enabled               false
```

4. Initialize the Vault:

   When using an HSM for auto unseal, Vault requires the number of key shares, threshold, and stored shares to be set to one (**1**).

   Use the following command options to initialize the Vault: `recovery-shares=1` and `recovery-threshold=1`.

   The `vault operator init` command returns the Recovery Key(s) and Initial Root Token. Note or save these.

```
# vault operator init -recovery-shares=1 -recovery-threshold=1
Recovery Key 1: 3zs1c7LEAs/hq8pvVYkcLX3wZ3hpqavtvNZvkjFhh7E=

Initial Root Token: s.Yl5MiQBmWC5JVpnL5fsRMkeM

Success! Vault is initialized

Recovery key initialized with 1 key shares and a key threshold of 1. Please
securely distribute the key shares printed above.
```

2.14. Log in from the command line

Log in to Vault using the Initial Root Token saved above and save the token below.

```
# vault login s.Yl5MiQBmWC5JVpnL5fsRMkeM
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                  Value
---                  -----
token                s.Yl5MiQBmWC5JVpnL5fsRMkeM
token_accessor       9opm7hJmNyTdMhFcttrww0u8
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

2.15. Log in from the web user interface

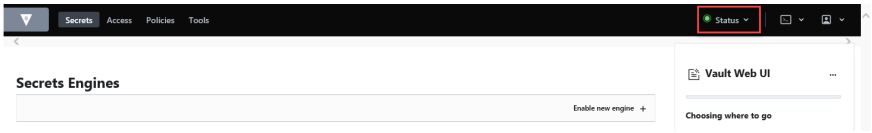
1. Open a browser and enter the IP address of the Vault Host Server. For example:

   ```
   http://127.0.0.1:8200/ui/vault/auth
   ```

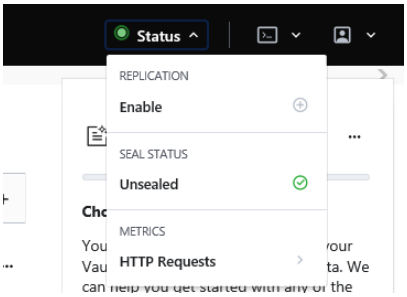
   The Vault sign-in page loads.

2. Enter the sign-in credentials:
   a. For the **Method** drop-down list, select **Token**.
   b. In the **Token** field, enter the Initial Root Token that you saved after you started Vault.
   c. Select **Sign In**.

   The **Secrets Engines** page appears. The **Status** light should be green. For example:



3. Select **Status** to check the details. The Vault should be **Unsealed**. For example:



2.16. Examine Vault secrets

Use the Secrets Engine operations of the Vault to check whether the integration works as intended. You can use these commands to generate, encrypt, and store any secret in a secure storage area, for example in a cubbyhole or in a key/value (KV) storage location.

View the current secrets and default locations.

```
vault secrets list
Path          Type         Accessor              Description
----          ----         --------              -----------
cubbyhole/    cubbyhole    cubbyhole_ddaf9cb7    per-token private secret storage
identity/     identity     identity_62a1922c     identity store
sys/          system       system_fbaaf933       system endpoints used for control, policy and debugging
```

2.17. Enable the KV engine

To enable the KV engine:

```
$ vault secrets enable -version=1 kv
Success! Enabled the kv secrets engine at: kv/
```

2.18. Write secret data

Add data to the key/value storage area of Vault:

```
$ vault kv put kv/opt/vault/secret key=test_secret
Success! Data written to: kv/opt/vault/secret
```

2.19. Retrieve secret data

To retrieve secret data:

```
$ vault kv get kv/opt/vault/secret
=== Data ===
Key    Value
---    -----
key    test_secret
```

2.20. Seal Vault

When you are done, seal Vault:

```
# vault operator seal
Success! Vault is sealed.
```

2.21. Rotate keys stored in the HSM and used by Vault

1. Create new HSM-protected Vault keys:

   Using OCS protection

   a. Create a new encryption key `vault_v2`:

```
# generatekey --generate --batch -m1 -s2 pkcs11 protect=token cardset=HashiCorp plainname=vault_v2
type=AES size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               token
 slot         Slot to read cards from    2
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   AES
 size         Key size                   256
 plainname    Key name                   vault_v2
 nvram        Blob in NVRAM (needs ACS)  no

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

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ucde4fa7e3ad8d66b6c652d18140b37bab9fe9d106-
ea26282514f5701358c1c90c983f39e1e86d9292
```

b. Create a new HMAC key `vault_hmac_v2`:

```
# generatekey --generate --batch -m1 -s2 pkcs11 protect=token cardset=HashiCorp
plainname=vault_hmac_v2 type=HMACSHA256 size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               token
 slot         Slot to read cards from    2
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   HMACSHA256
 size         Key size                   256
 plainname    Key name                   vault_hmac_v2
 nvram        Blob in NVRAM (needs ACS)  no

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

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ucde4fa7e3ad8d66b6c652d18140b37bab9fe9d106-
569f841f569cafa49abcc0bb5308aef67f62df19
```

Using Softcard protection

a. Create a new encryption key `vault_v2`:

```
# generatekey --generate --batch -m1 pkcs11 protect=softcard softcard=vaultsc plainname=vault_v2
type=AES size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               softcard
 softcard     Soft card to protect key   vaultsc
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   AES
 size         Key size                   256
 plainname    Key name                   vault_v2
 nvram        Blob in NVRAM (needs ACS)  no
Please enter the pass phrase for softcard `vaultsc':

Please wait........
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uca4c30a628024194c2c44b579ab54088f2baa5964-
6274a79faa4cb7ce8f7f9bade9fb1bd4240caee1
```

b. Create a new HMAC key `vault_hmac_v2`:

```
# generatekey --generate --batch -m1 pkcs11 protect=softcard softcard=vaultsc plainname=vault_hmac_v2
type=HMACSHA256 size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               softcard
```

```
 softcard     Soft card to protect key   vaultsc
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   HMACSHA256
 size         Key size                   256
 plainname    Key name                   vault_hmac_v2
 nvram        Blob in NVRAM (needs ACS)  no
Please enter the pass phrase for softcard `vaultsc':

Please wait........
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uca4c30a628024194c2c44b579ab54088f2baa5964-
5ae8f9fd29813f900eb9602558677d6ed9bbdcd2
```

2. Update the Vault configuration:

```
# sudo vi /etc/vault/config.hcl
```

   Update the `_v1` version in the following lines to `_v2`:

   - `key_label = "vault_v2"`
   - `hmac_key_label = "vault_hmac_v2"`

3. Restart Vault:

```
# systemctl restart vault.service
```

4. Log in to Vault, as a user with rewrap permission:

```
# vault login
Token (will be hidden):
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                  Value
---                  -----
token                s.Yu94RNC8nHtGD4MUT0REa12G
token_accessor       dGjokkWequ2IFU260Jwanw7b
token_duration       â^ž
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

5. Rewrap all.

   Optionally, use Seal Wrapped Data. Lazy rewrap is the default.

```
# vault write -f /sys/sealwrap/rewrap
Success! Data written to: sys/sealwrap/rewrap
```

6. Check the rewrap status:

```
# vault read /sys/sealwrap/rewrap
Key           Value
---           -----
entries       map[failed:0 processed:24 succeeded:24]
is_running    false
```

2.22. Change protection method

The protection method cannot be changed after Vault has been initialized. You can change keys within the same protection method without re-initializing the Vault, but not the protection method.

Changing the appropriate parameters in `/etc/vault/config.hcl` and re-starting the Vault causes a `CKR_KEY_HANDLE_INVALID` error.

The following steps were done for the purpose of integration testing. This will allow re-initialization of Vault, but will destroy all secrets in a previously initialized Vault.

1. Prepare the Vault for re-initialization:

```
# systemctl stop vault.service
# userdel vault
# groupdel -f vault
# rm -r -f /opt/vault/data/*
# rm -r -f /opt/vault/logs/*
# touch /etc/sysconfig/vault
# touch /etc/systemd/system/vault.service
```

2. Edit the `/etc/vault/config.hcl` file per the new protection method. Change the parameters `slot, key_label, hmac_key_label` accordingly.
3. Reboot the client.
4. Create new vault user and group, re-initialize the Vault, and do the integration as described above.

Chapter 3. Kubernetes Deployment Procedures

Prerequisites:

- Dedicated Linux server with a working HashiCorp installation to build the container images. This is the same server above after completing all the steps previously outlined.
- Container platform. Any Kubernetes platform is supported. The example process in this guide uses an OpenShift Kubernetes installation on a single machine.

Follow these steps to create a HashiCorp image which supports the HSM, generate the containers, and test the Kubernetes integration with the HSM.

1. Install nShield nCOP
2. Install Docker
3. Build the nShield hardserver container
4. Build the HashiCorp Vault container
5. Build the nShield Vault application container
6. Run the containers locally
7. Test containers running locally
8. Push the container images to your registry
9. Create the project in the container platform
10. Create the persistent volumes
11. Claim the persistent volumes
12. Copy the configuration files to the cluster persistent volumes
13. Deploy the HashiCorp Vault nShield application

3.1. Install nShield nCOP

1. Create the installation directory:

```
# sudo mkdir -p /opt/ncop
```

2. Extract the nShield Container Option Pack tarball:

```
# sudo tar -xvf /root/Downloads/ncop-1.1.1.tar -C /opt/ncop
extend-nshield-application
```

```
make-nshield-application
make-nshield-hwsp
make-nshield-hwsp-config
examples/
examples/javaenquiry/
examples/javaenquiry/Dockerfile
examples/javaenquiry/README.md
examples/javaenquiry/cmd
examples/nfkminfo/
examples/nfkminfo/Dockerfile
examples/nfkminfo/README.md
examples/nfkmverify/
examples/nfkmverify/Dockerfile
examples/nfkmverify/README.md
examples/nfweb/
examples/nfweb/Dockerfile
examples/nfweb/README.md
examples/nfweb/nfweb.py
README.md
images/architecture.png
images/java-architecture.png
license.rtf
rnotes.pdf
version.json
```

3.2. Install Docker

1. Add the Docker CE repository:

```
# yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
Updating Subscription Management repositories.

This system is registered to Red Hat Subscription Management, but is not receiving updates
You can use subscription-manager to assign subscriptions.

Adding repo from: https://download.docker.com/linux/rhel/docker-ce.repo
```

2. Verify the repo contains the stable version of Docker:

```
# yum repolist
Updating Subscription Management repositories.

This system is registered to Red Hat Subscription Management, but is not receiving updates.
You can use subscription-manager to assign subscriptions.

repo id                               repo name
docker-ce-stable                      Docker CE Stable - x86_64
rhel-8-for-x86_64-appstream-rpms      Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
rhel-8-for-x86_64-baseos-rpms         Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
```

3. Install Docker:

```
yum install docker-ce
Updating Subscription Management repositories.

This system is registered to Red Hat Subscription Management, but is not receiving updates.
You can use subscription-manager to assign subscriptions.
```

```
Docker CE Stable - x86_64        48 kB/s | 9.5 kB     00:00
```

See troubleshoot section for issues with the installation.

4. Start Docker manually, start the `docker` service on startup, and verify it is running:

```
# systemctl start docker

# systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service →
/usr/lib/systemd/system/docker.service.

# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2022-01-06 10:02:30 EST; 37s ago
     Docs: https://docs.docker.com
 Main PID: 30667 (dockerd)
    Tasks: 13
   Memory: 172.9M
   CGroup: /system.slice/docker.service
           └─30667 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Jan 06 10:02:29 Red_Hat_8.3_HashiCorp_Containers dockerd[30667]: time="2022-01-06T10:02:29.01936988>
Jan 06 10:02:29 Red_Hat_8.3_HashiCorp_Containers dockerd[30667]: time="2022-01-06T10:02:29.41964695>
...

# docker --version
Docker version 20.10.3, build 48d30b5
```

3.3. Build the nShield hardserver container

1. Stop the hardserver:

```
# /opt/nfast/sbin/init.d-ncipher stop
 -- Running shutdown script 90ncsnmpd
 -- Running shutdown script 60raserv
 -- Running shutdown script 50hardserver
 -- Running shutdown script 46exard
 -- Running shutdown script 45drivers
```

2. Mount the Security World ISO file:

```
# mount -t iso9660 -o loop /root/Downloads/SecWorld_Lin64-12.80.4.iso /mnt/iso
mount: /mnt/iso: WARNING: device write-protected, mounted read-only.
```

3. Change directory:

```
# cd /opt/ncop
```

4. Build the nShield hardserver container:

```
# ./make-nshield-hwsp --from registry.access.redhat.com/ubi8/ubi --tag nshield-hwsp-pkcs11-redhat /mnt/iso
Detecting nShield software version
Version is 12.80.4
Unpacking hwsp...
Extracting tools from ctls...
Removing redundant files...
Creating files...
Building image...
Sending build context to Docker daemon  234.6MB
Step 1/24 : FROM registry.access.redhat.com/ubi8/ubi
latest: Pulling from ubi8/ubi
26f1167feaf7: Pull complete
adffa6963146: Pull complete
Digest: sha256:228824aa581f3b31bf79411f8448b798291c667a37155bdea61cfa128b2833f2
Status: Downloaded newer image for registry.access.redhat.com/ubi8/ubi:latest
 ---> fca12da1dc30
Step 2/24 : RUN if [ -x /usr/bin/microdnf ]; then microdnf update && microdnf install shadow-utils libcap
findutils && microdnf clean all; fi
 ---> Running in 8caca115e9f9
Removing intermediate container 8caca115e9f9
 ---> 61a560b15baf

...

Successfully built 21fc2835b68e
Successfully tagged nshield-hwsp-pkcs11-redhat:latest
```

5. Unmount the Security World ISO:

```
umount /mnt/iso
```

6. Verify the built container:

```
# docker images
REPOSITORY                          TAG       IMAGE ID       CREATED          SIZE
nshield-hwsp-pkcs11-redhat          latest    21fc2835b68e   30 minutes ago   484MB
...
```

3.4. Build the HashiCorp Vault container

1. Create the `working` directory for Docker:

```
# mkdir ~/working
```

2. Create a `/root/working/Dockerfile`. Notice the nShield files copied.

```
FROM registry.access.redhat.com/ubi8

# Working directory.
WORKDIR /root/working
```

```
# nShield files.
RUN mkdir -p /opt/nfast
COPY cknfastrc /opt/nfast/

# Create Vault user and group.
RUN groupadd --system nfast && \
    groupadd --system vault && \
    useradd --system --shell /sbin/nologin --gid vault vault && \
    usermod --append --groups nfast vault

# Download the Vault package from HashiCorp at https://releases.hashicorp.com/vault/.
# Unzip the binary file and extract it to the working directory.
RUN yum install -y wget && \
    yum install -y unzip && \
    wget https://releases.hashicorp.com/vault/1.9.2+ent.hsm/vault_1.9.2+ent.hsm_linux_amd64.zip && \
    unzip vault_1.9.2+ent.hsm_linux_amd64.zip -d /usr/local/bin && \
    yum remove -y wget unzip && \
    rm -r *.zip

# Set Vault permissions.
RUN chmod 755 /usr/local/bin/vault && \
    setcap cap_ipc_lock=+ep /usr/local/bin/vault

# Create the Vault data directories.
RUN mkdir --parents /opt/vault/data && \
    mkdir --parents /opt/vault/logs && \
    chmod --recursive 750 /opt/vault && \
    chown --recursive vault:vault /opt/vault

# Create a vault file in sysconfig.
RUN touch /etc/sysconfig/vault

# Expose the data and logs directory as a volume.
VOLUME /opt/vault/data
VOLUME /opt/vault/logs

# 8200/tcp is the primary interface that applications use to interact with Vault.
EXPOSE 8200

# Enable Vault.
RUN export VAULT_ADDR=http://127.0.0.1:8200

# Starting Vault as follows fails with error "System has not been booted with systemd as init system (PID
1)#.
# ENTRYPOINT  systemctl start vault.service
# Instead use the parameter ExecStart at /etc/systemd/system/vault.service.
ENTRYPOINT /usr/local/bin/vault server -config=/etc/vault/config.hcl
```

3. Build the container:

```
# cd /root/working

# docker build . --no-cache -t hashicorp-vault-enterprise-hsm
Sending build context to Docker daemon  121.3kB
Step 1/12 : FROM registry.access.redhat.com/ubi8
 ---> fca12da1dc30
Step 2/12 : WORKDIR /root/working
 ---> Running in 247a31ee2196
Removing intermediate container 247a31ee2196
 ---> ef51adc3a657
Step 3/12 : RUN groupadd --system nfast &&     groupadd --system vault &&     useradd --system --shell
/sbin/nologin --gid vault vault &&     usermod --append --groups nfast vault

...
```

```
Successfully built 58a728b21930
Successfully tagged hashicorp-vault-enterprise-hsm:latest
```

4. Verify the built container:

```
# docker images
REPOSITORY                                      TAG       IMAGE ID       CREATED         SIZE
hashicorp-vault-enterprise-hsm                  latest    58a728b21930   2 minutes ago   635MB
nshield-hwsp-pkcs11-redhat                      latest    21fc2835b68e   2 weeks ago     484MB
...
```

3.5. Build the nShield Vault application container

1. Stop the hardserver:

```
# /opt/nfast/sbin/init.d-ncipher stop
 -- Running shutdown script 90ncsnmpd
 -- Running shutdown script 60raserv
 -- Running shutdown script 50hardserver
 -- Running shutdown script 46exard
 -- Running shutdown script 45drivers
```

2. Mount the Security World ISO file:

```
# mount -t iso9660 -o loop /root/Downloads/SecWorld_Lin64-12.80.4.iso /mnt/iso
mount: /mnt/iso: WARNING: device write-protected, mounted read-only.
```

3. Build the container:

```
# cd /opt/ncop

# ./extend-nshield-application --from hashicorp-vault-enterprise-hsm --pkcs11 --tag nshield-vault-app-
pkcs11-redhat /mnt/iso
Detecting nShield software version
Version is 12.80.4
NOTICE: --pkcs11 included by default with 12.60 ISO. Flag ignored
Unpacking /mnt/iso/linux/amd64/hwsp.tar.gz ...
Unpacking /mnt/iso/linux/amd64/ctls.tar.gz ...
Adding files...
Building image...
Sending build context to Docker daemon  702.7MB
Step 1/4 : FROM hashicorp-vault-enterprise-hsm
 ---> 58a728b21930
Step 2/4 : COPY opt /opt
 ---> 47a71ac5f1b0
Step 3/4 : RUN mkdir -p /opt/nfast/kmdata /opt/nfast/sockets && mkdir -m 1755 /opt/nfast/kmdata/tmp
 ---> Running in be7ad7b82bb5
Removing intermediate container be7ad7b82bb5
 ---> 147827a9fc16
Step 4/4 : VOLUME [ "/opt/nfast/kmdata", "/opt/nfast/sockets" ]
 ---> Running in 4b1d7f697f36
Removing intermediate container 4b1d7f697f36
 ---> 363024ec103d
Successfully built 363024ec103d
```

```
Successfully tagged nshield-vault-app-pkcs11-redhat:latest
```

4. Unmount the Security World ISO:

```
umount /mnt/iso
```

5. Verify the built container:

```
# docker images
REPOSITORY                                      TAG       IMAGE ID       CREATED          SIZE
nshield-vault-app-pkcs11-redhat                 latest    363024ec103d   2 minutes ago    1.33GB
hashicorp-vault-enterprise-hsm                  latest    58a728b21930   15 minutes ago   635MB
nshield-hwsp-pkcs11-redhat                      latest    21fc2835b68e   2 weeks ago      484MB
...
```

3.6. Run the containers locally

This test is performed in the Linux server with the HashiCorp Vault installation used so far in all the steps outlined above.

1. Create a `/root/working/nfast/kmdata-config` directory for the nShield configuration file and cardlist file (if using OCS protection). Populate the directory.

```
# mkdir /root/working/nfast/kmdata-config

# cp /opt/nfast/kmdata/config/* /root/working/nfast/kmdata-config/.
```

2. Create a `/root/working/nfast/kmdata-local` directory for the nShield world and module files, and the keys created for the Vault. Populate the directory.

```
# mkdir /root/working/nfast/kmdata-local

# cp /opt/nfast/kmdata/local/* /root/working/nfast/kmdata-local/.
```

3. Create a `/root/working/vault-config` directory for the license, config, and other Vault files. Populate the directory.

```
# mkdir /root/working/vault-config

# cp /etc/profile.d/vault.sh /root/working/vault-config/vault.sh
# cp /etc/vault/license.hclic /root/working/vault-config/license.hclic
# cp /etc/vault/config.hcl /root/working/vault-config/config.hcl
```

4. Stop the nShield Hardserver and the Vault service. Otherwise the test will pass

regardless of whether the container is running or not.

```
# /opt/nfast/sbin/init.d-ncipher stop

# systemctl stop vault.service
```

5. Open a new command window and run the `nshield-hwsp-pkcs11-redhat` image:

```
# cd /root/working

# docker volume create socket1
socket1

# docker run --rm -it -v socket1:/opt/nfast/sockets -v $PWD/nfast/kmdata-config:/opt/nfast/kmdata/config
nshield-hwsp-pkcs11-redhat
```

6. Open a new command window and run the `nshield-vault-app-pkcs11-redhat` image. Notice all the parameters passed.

```
# cd /root/working

# docker volume create socket1
socket1

# docker run --rm -it --privileged -v socket1:/opt/nfast/sockets -v $PWD/vault-
config/license.hclic:/etc/vault/license.hclic -v $PWD/vault-config/config.hcl:/etc/vault/config.hcl -v
$PWD/nfast/kmdata-local:/opt/nfast/kmdata/local --env VAULT_ADDR=http://127.0.0.1:8200 -p8200:8200 nshield-
vault-app-pkcs11-redhat
```

7. Verify the containers are running:

```
# docker ps
CONTAINER ID   IMAGE                                                COMMAND                  CREATED
STATUS          PORTS                    NAMES
b466aeabe786   nshield-vault-app-pkcs11-redhat                      "/bin/sh -c '/usr/lo…"   56 seconds ago
Up 56 seconds   0.0.0.0:8200->8200/tcp   nice_fermi
b1229bd35efb   nshield-hwsp-pkcs11-redhat                           "/opt/nfast/sbin/nsh…"   22 hours ago
Up 22 hours
```

3.7. Test containers running locally

1. Check the Vault status:

```
# docker exec -it b466aeabe786 vault status
Key                      Value
---                      -----
Recovery Seal Type       pkcs11
Initialized              false
Sealed                   true
Total Recovery Shares    0
Threshold                0
Unseal Progress          0/0
```

```
Unseal Nonce            n/a
Version                 1.9.2+ent.hsm
Storage Type            file
HA Enabled              false
```

2. Initialize the Vault:

```
# docker exec -it b466aeabe786 vault operator init -recovery-shares=1 -recovery-threshold=1
Recovery Key 1: 0ZJVfhRWIzFVU8aMqEe8I05yGVuV4SsgsdZu63fM0ts=

Initial Root Token: s.Yy4pFr03KdfuV9ZKxUB9AZDv

Success! Vault is initialized

Recovery key initialized with 1 key shares and a key threshold of 1. Please
securely distribute the key shares printed above.
```

3. Log in to the Vault:

```
# docker exec -it b466aeabe786 vault login s.Yy4pFr03KdfuV9ZKxUB9AZDv
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                  Value
---                  -----
token                s.Yy4pFr03KdfuV9ZKxUB9AZDv
token_accessor       fu745qP8XmKeuN1OeMEGgXW8
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

4. Examine Vault secrets:

```
# docker exec -it b466aeabe786 vault list secrets
No value found at secrets
```

3.8. Push the container images to your registry

1. Log into your remote registry:

```
# docker swarm init

# docker login -u <your_user_id> https://registry.eselab.net
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

2. Tag and push the images:

```
# docker tag nshield-hwsp-pkcs11-redhat:latest registry.eselab.net/hashicorp-vault-nshield-hwsp:latest

# docker push registry.eselab.net/hashicorp-vault-nshield-hwsp:latest
The push refers to repository [registry.eselab.net/hashicorp-vault-nshield-hwsp]
7124856b02e0: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
d04672a1fe0b: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
2336267e5c4c: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
25965ddea3b0: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
a6a76cb66da3: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
3c94a72beb86: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
945bd9297bdc: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
1d3ad63a37b6: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
1d212b6142a1: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
e4612491fc7d: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
5202f468f3c9: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
2a8d12e1343e: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
07728d479b42: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
7175fef03a4b: Mounted from hashicorp-nshield-hwsp-pkcs11-redhat
3ba8c926eef9: Mounted from hashicorp-nshield-app-pkcs11-redhat
352ba846236b: Mounted from hashicorp-nshield-app-pkcs11-redhat
latest: digest: sha256:21f5cd82310012e6a16c6989425569ba2c2707485342e470210f6e0958c0d39d size: 3650
```

```
# docker tag nshield-vault-app-pkcs11-redhat:latest registry.eselab.net/hashicorp-vault-nshield-app:latest

# docker push registry.eselab.net/hashicorp-vault-nshield-app:latest
The push refers to repository [registry.eselab.net/hashicorp-vault-nshield-app]
11436f7c7418: Pushed
7516f7bce24a: Pushed
efc7c4a8cc00: Pushed
20d2b70c4819: Pushed
f7064b7a4790: Pushed
424bf5904064: Pushed
fdb2287e8ad0: Pushed
144ac96160c8: Pushed
3ba8c926eef9: Layer already exists
352ba846236b: Layer already exists
latest: digest: sha256:05bdba96ddc9409a9695ad0e864ede1f3c7b780d4916a7c20fb869064e399a0a size: 2410
```

3. Log out from registry:

```
# docker logout https://registry.eselab.net
Removing login credentials for registry.eselab.net
```

4. Notice the `config.json` file that was created during the logging in process:

```
# cat /root/.docker/config.json
{
        "auths": {
                "registry.eselab.net": {
                        "auth": "..."
                }
        }
```

3.9. Create the project in the container platform

Red Hat OpenShift is the container platform used in this integration.

1. Log into the OpenShift container platform server. If logging in as **root**, change to another user.
2. Add the nShield HSM as a client on the OpenShift server. Refer to section *Install the HSM* above.
3. Create the `pull-secret` file with the pull secret copied from https://cloud.redhat.com/openshift/create/local:

```
# cat /home/testuser/Documents/pull-secret
{"auths":{"cloud.openshift.com":{"auth":"b3Blbn...
```

4. Copy the `config.json` file created above in the dedicated Linux server to this OpenShift environment:

```
$ ls -al /home/testuser/Documents/config.json
-rw-rw-r--. 1 testuser testuser 147 Jan 21 10:49 /home/testuser/Documents/config.json
```

5. Start the Red Hat CodeReady environment:

```
$ crc start
```

6. Log into the OpenShift environment:

```
$ eval $(crc oc-env)

$ oc login -u kubeadmin https://api.crc.testing:6443
Logged into "https://api.crc.testing:6443" as "kubeadmin" using existing credentials.

You have access to 64 projects, the list has been suppressed. You can list all projects with 'oc projects'

Using project "default".
```

7. Create the project:

```
$ oc create -f /home/testuser/Documents/project.yaml
project.project.openshift.io/hashicorpvault created
```

8. Change from the current project to the newly created project:

```
$ oc project hashicorpvault
Now using project "hashicorpvault" on server "https://api.crc.testing:6443".
```

```
$ oc get namespaces
NAME                                               STATUS   AGE
default                                            Active   78d
hashicorpvault                                     Active   7m55s
...
```

9. Create and retrieve the secret:

```
$ oc create secret generic hashicorpvault --from
-file=.dockerconfigjson=/home/testuser/Documents/config.json --type=kubernetes.io/dockerconfigjson
secret/hashicorpvault created

Create secret - 1/2 steps
```

```
$ oc get secret
NAME                       TYPE                                DATA   AGE
...
hashicorpvault             kubernetes.io/dockerconfigjson      1      0s
```

10. Create the config map with the Connect details:

```
$ oc create -f /home/testuser/Documents/cm.yaml
configmap/config created
```

11. Verify the nShield Connect configuration:

```
$ oc get configmap
NAME                       DATA   AGE
config                     1      0s
kube-root-ca.crt           1      1s
openshift-service-ca.crt   1      1s
```

```
$ oc describe configmap/config
Name:         config
Namespace:    hashicorpvault
Labels:       <none>
Annotations:  <none>

Data
====
config:

syntax-version=1

[nethsm_imports]
local_module=0
remote_ip=10.194.148.33
remote_port=9004
remote_esn=201E-03E0-D947
keyhash=84800d1bfff6515ed5806fe443bbaca812d73733
privileged=0


BinaryData
====
```

```
Events:  <none>
```

3.10. Create the persistent volumes

The following persistent volumes will be created. The first is for the communication between the Vault application and the nShield hardserver. The others are for configuration, keys, and license files.

- `/opt/nfast/socket`
- `/opt/nfast/kmdata/`
- `/etc/vault`
- `/opt/vault/data`

To create the persistent volumes:

1. Create the `/opt/nfast/sockets` persistent volume for the nShield hardserver communication with the Vault application. See the Sample YAML files appendix for YAML files.

```
$ oc create -f /home/testuser/Documents
pv_nfast_sockets_definition.yaml
persistentvolume/nfast-sockets created
```

2. Create the `/opt/nfast/kmdata` persistent volume for the nShield configuration files:

```
$ oc create -f /home/testuser/Documents/pv_nfast_kmdata_definition.yaml
persistentvolume/nfast-kmdata created
```

3. Create the `/etc/vault` persistent volume for the Vault configuration files:

```
$ oc create -f /home/testuser/Documents/pv_vault_config_definition.yaml
persistentvolume/vault-config created
```

4. Create the `/opt/vault/data` persistent volume for the Vault storage backend:

```
$ oc create -f /home/testuser/Documents/pv_vault_data_definition.yaml
persistentvolume/vault-data created
```

5. Verify the persistent volumes created:

```
$ oc get pv
```

```
NAME           CAPACITY ACCESS MODES RECLAIM POLICY STATUS    CLAIM STORAGECLASS REASON AGE
nfast-kmdata   1G       RWO          Retain         Available       manual              0s
nfast-sockets  1G       RWO          Retain         Available       manual              0s
vault-config   10M      RWO          Retain         Available       manual              0s
vault-data     10M      RWO          Retain         Available       manual              0s
```

3.11. Claim the persistent volumes

1. Create the `/opt/nfast/sockets` volume claim:

```
$ oc create -f /home/testuser/Documents
pv_nfast_sockets_claim.yaml
persistentvolumeclaim/nfast-sockets created
```

2. Create the `/opt/nfast/kmdata` volume claim:

```
$ oc create -f /home/testuser/Documents
pv_nfast_kmdata_claim.yaml
persistentvolumeclaim/nfast-kmdata created
```

3. Create the `/etc/vault` volume claim:

```
$ oc create -f /home/testuser/Documents/pv_vault_config_claim.yaml
persistentvolumeclaim/vault-config created
```

4. Create the `/opt/vault/data` volume claim:

```
$ oc create -f /home/testuser/Documents/pv_vault_data_claim.yaml
persistentvolumeclaim/vault-data created
```

5. Verify the persistent volumes claimed:

```
$ oc get pvc
NAME           STATUS   VOLUME         CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfast-kmdata   Bound    nfast-kmdata   1G         RWO            manual         1s
nfast-sockets  Bound    nfast-sockets  1G         RWO            manual         1s
vault-config   Bound    vault-config   10M        RWO            manual         1s
vault-data     Bound    vault-data     10M        RWO            manual         0s
```

3.12. Copy the configuration files to the cluster persistent volumes

1. Copy the `/root/working/nfast`` and `/root/working/vaultconfig` required files from the dedicated Linux server above to OpenShift server. Then perform a `chmod 775 <all copied files>`.

```
$ $ ls -al /home/testuser/Documents/nfast/kmdata/local
total 96
...
-rwxrwxr-x. 1 testuser testuser  7180 Jan 18 21:27 key_pkcs11_ucb5df6e12703825562ce731e3286a4fb9f46e767a-
ebc7da3d8e2f9aa86377dc4e5269157557cddd1c
-rwxrwxr-x. 1 testuser testuser  5000 Jan 27 13:48 module_201E-03E0-D947
-rwxrwxr-x. 1 testuser testuser  1428 Jan 18 21:27 softcard_b5df6e12703825562ce731e3286a4fb9f46e767a
-rwxrwxr-x. 1 testuser testuser 39968 Jan 18 21:27 world



$ ls -al /home/testuser/Documents/vaultconfig
total 4
...
drwxrwxr-x. 2 testuser testuser    6 Jan 21 12:53 config.hcl
-rw-rw-r--. 1 testuser testuser 1202 Jan 21 12:50 license.hclic
```

2. Show the nodes:

```
$ oc get nodes --show-labels
NAME                 STATUS   ROLES           AGE   VERSION                LABELS
crc-ktfxm-master-0   Ready    master,worker   78d   v1.22.0-rc.0+a44d0f0
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=c
rc-ktfxm-master-0,kubernetes.io/os=linux,node-role.kubernetes.io/master=,node-
role.kubernetes.io/worker=,node.openshift.io/os_id=rhcos
```

3. Label the nodes:

```
$ oc label node crc-ktfxm-master-0 nodeName=master-0
node/crc-ktfxm-master-0 labeled

$ oc get nodes --show-labels
NAME                 STATUS   ROLES           AGE   VERSION                LABELS
crc-ktfxm-master-0   Ready    master,worker   78d   v1.22.0-rc.0+a44d0f0
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=c
rc-ktfxm-master-0,kubernetes.io/os=linux,node-role.kubernetes.io/master=,node-
role.kubernetes.io/worker=,node.openshift.io/os_id=rhcos,nodeName=master-0
```

4. Create a `pod_dummy.yaml` application container for the purpose of populating the persistent volumes.

```
$ oc create -f /home/testuser/Documents/pod_dummy.yaml
pod/ncop-test-dummy-svhnn created
```

5. Verify the pods are running. This might take several minutes to download form the remote registry:

```
$ oc get pods
NAME                          READY   STATUS    RESTARTS   AGE
ncop-test-dummy-kw5mv         2/2     Running   0          3m
```

6. Populate the persistent storage with configuration files:

```
$ oc cp /home/testuser/Documents/nfast/kmdata-config/config ncop-test-dummy-
kw5mv:/opt/nfast/kmdata/config/config

$ oc cp /home/testuser/Documents/nfast/kmdata-config/cardlist ncop-test-dummy-
kw5mv:/opt/nfast/kmdata/config/cardlist

$ oc cp /home/testuser/Documents/nfast/kmdata-local/world ncop-test-dummy-
kw5mv:/opt/nfast/kmdata/local/world

$ oc cp /home/testuser/Documents/nfast/kmdata-local/module_201E-03E0-D947 ncop-test-dummy-
kw5mv:/opt/nfast/kmdata/local/module_201E-03E0-D947

$ oc cp /home/testuser/Documents/vaultconfig/config.hcl ncop-test-dummy-kw5mv:/etc/vault/config.hcl

$ oc cp /home/testuser/Documents/vaultconfig/license.hclic ncop-test-dummy-kw5mv:/etc/vault/license.hclic
```

7. Populate the persistent storage with keys:

```
$ oc cp /home/testuser/Documents/nfast/kmdata-local/card_124902ced9e45399cfa993eabd5d5d9e5c7b5a7f_1 ncop-
test-dummy-kw5mv:/opt/nfast/kmdata/local/card_124902ced9e45399cfa993eabd5d5d9e5c7b5a7f_1

...
```

8. Spot check the copied files:

```
$ oc debug pod/ncop-test-dummy-kw5mv                        Starting pod/ncop-test-dummy-kw5mv-debug,
command was: sh -c sleep 3600
Pod IP: 10.217.1.13
If you don't see a command prompt, try pressing enter.
sh-4.4# ls -al /opt/nfast/kmdata/local
total 92
drwxr-xr-x. 2 root root  4096 Jan 26 02:16 .
drwxr-xr-x. 4 root root    33 Jan 25 04:37 ..
-rwxrwxr-x. 1 1004 1004   904 Jan 26 02:07 card_124902ced9e45399cfa993eabd5d5d9e5c7b5a7f_1
-rwxrwxr-x. 1 1004 1004   112 Jan 26 02:07 cards_124902ced9e45399cfa993eabd5d5d9e5c7b5a7f
-rwxrwxr-x. 1 1004 1004  7176 Jan 26 02:13 key_pkcs11_uc124902ced9e45399cfa993eabd5d5d9e5c7b5a7f-
0ee55ef5b9b6c3d42cee681e3b8c056f2df00a8f
-rwxrwxr-x. 1 1004 1004  7216 Jan 26 02:13 key_pkcs11_uc124902ced9e45399cfa993eabd5d5d9e5c7b5a7f-
bfa1988aed796d05cbf852abccf5380ff90f4f91
-rwxrwxr-x. 1 1004 1004  7216 Jan 26 02:14 key_pkcs11_ucb5df6e12703825562ce731e3286a4fb9f46e767a-
376268c6c89c1657fb22ca1f08fe4f20b58b1c07
-rwxrwxr-x. 1 1004 1004  7180 Jan 26 02:14 key_pkcs11_ucb5df6e12703825562ce731e3286a4fb9f46e767a-
ebc7da3d8e2f9aa86377dc4e5269157557cddd1c
-rwxrwxr-x. 1 1004 1004  3488 Jan 26 01:27 module_201E-03E0-D947
-rwxrwxr-x. 1 1004 1004  1428 Jan 26 02:16 softcard_b5df6e12703825562ce731e3286a4fb9f46e767a
-rwxrwxr-x. 1 1004 1004 39968 Jan 26 01:27 world
sh-4.4# exit
exit

Removing debug pod ...
```

3.13. Deploy the HashiCorp Vault nShield application

1. Create the `pod_hashicorpvault_nshield.yaml` pod running the HashiCorp Vault and nShield application.

```
$ oc create -f pod_hashicorpvault_nshield.yaml
pod/hashicorpvault-status-fglgv created
```

2. Verify the Vault is available:

```
$ oc logs -f pod/hashicorpvault-status-fglgv hashicorp-app
Vault v1.9.2+ent.hsm (f7be55269a69543aedae108588e63688e6490b44) (cgo)
```

3. Start the Vault server:

```
$ oc debug pod/hashicorpvault-nshield-fglgv -c hashicorp-app
Starting pod/hashicorpvault-nshield-fglgv-debug ...
Pod IP: 10.217.0.79
If you don't see a command prompt, try pressing enter.

sh-4.4# /usr/local/bin/vault server -config=/etc/vault/config.hcl
```

4. Verify the Vault status:
   a. Open a second window and log into the OpenShift environment.
   b. Set the project, and execute the following command:

```
$ oc exec hashicorpvault-nshield-fglgv -c hashicorp-app -- /usr/local/bin/vault status
Key                      Value
---                      -----
Recovery Seal Type       pkcs11
Initialized              false
Sealed                   true
Total Recovery Shares    0
Threshold                0
Unseal Progress          0/0
Unseal Nonce             n/a
Version                  1.9.2+ent.hsm
Storage Type             file
HA Enabled               false
command terminated with exit code 2
```

Chapter 4. Troubleshooting

Error Message	Resolution
Vault fails to start. There may not be a log file created if the vault fails to start upon executing `# systemctl start vault.service`.	Execute the following instead to get some debugging information. `# vault server -config=/etc/vault/config.hcl`.
Error: **failed to decrypt encrypted stored keys: error initializing session for decryption: error logging in to HSM: pkcs11: 0xE0: CKR_TOKEN_NOT_PRESENT**	Ensure that the Operator card is inserted in the physical slot of the nShield HSM.
Docker installation fails with errors during downloading metadata for repository `docker-ce-stable`.	1) Edit `/etc/yum.repos.d/docker-ce.repo`. 2) Change `baseurl=https://download.docker.com/linux/rhel/$releasever/$basearch/stable` in section `[docker-ce-stable]` accordingly. 3) Change `baseurl=https://download.docker.com/linux/rhel/$releasever/s390x/stable` for release 8.9.

Chapter 5. Vault commands

5.1. Vault commands

Task	Command
Log into Vault	`# vault login s.InitialRootToken`
Check Vault status	`# vault status`
Unseal Vault	`# vault operator unseal -address=http://127.0.0.1:8200`
Seal Vault	`# vault operator seal`

5.2. vault.service commands

Task	Command
Enable Vault Service	`#systemctl enable vault.service`
Disable Vault service	`#systemctl disable vault.service`
Start Vault service	`# systemclt start vault.service`
Stop Vault service	`# systemclt stop vault.service`
Restart Vault service	`#systemctl restart vault.service`

Chapter 6. Sample YAML files

6.1. project.yaml

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

6.2. cm.yaml

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

    [nethsm_imports]
    local_module=0
    remote_ip=10.194.148.33
    remote_port=9004
    remote_esn=201E-03E0-D947
    keyhash=84800d1bfff6515ed5806fe443bbaca812d73733
    privileged=0
```

6.3. pv_nfast_sockets_definition.yaml

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

```
    path: /opt/nfast/sockets
```

6.4. pv_nfast_sockets_claim.yaml

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

6.5. pv_nfast_kmdata_definition.yaml

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

6.6. pv_nfast_kmdata_claim.yaml

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

6.7. pv_vault_config_definition.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: vault-config
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10M
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /etc/vault
```

6.8. pv_vault_config_claim.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name : vault-config
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: local-storage
  resources:
    requests:
      storage: 10M
  storageClassName: manual
```

6.9. pv_vault_data_definition.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: vault-data
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10M
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /etc/vault/data
```

6.10. pv_vault_data_claim.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name : vault-data
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: local-storage
  resources:
    requests:
      storage: 10M
  storageClassName: manual
```

6.11. pod_dummy.yaml

```yaml
kind: Pod
apiVersion: v1
metadata:
  generateName: ncop-test-dummy-
  namespace: hashicorpvault
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: hashicorpvault
  containers:
    - name: ncop-app
      securityContext:
        privileged: true
      command:
        - sh
        - '-c'
        - sleep 3600
      image: >-
        registry.eselab.net/hashicorp-vault-nshield-app
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: nfast-sockets
          mountPath: /opt/nfast/sockets
        - name: nfast-kmdata
          mountPath: /opt/nfast/kmdata
        - name: vault-config
          mountPath: /etc/vault
        - name: vault-data
          mountPath: /opt/vault/data
  securityContext: {}
  volumes:
    - name: nfast-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: nfast-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
    - name: vault-config
      persistentVolumeClaim:
        claimName: vault-config
    - name: vault-data
      persistentVolumeClaim:
```

```
        claimName: vault-data
```

6.12. pod_hashicorpvault_nshield.yaml

```
kind: Pod
apiVersion: v1
metadata:
  generateName: hashicorpvault-nshield-
  namespace: hashicorpvault
  labels:
    app: nshield
spec:
  imagePullSecrets:
  containers:
    - name: ncop-hwsp
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: true
      image: >-
        registry.eselab.net/hashicorp-vault-nshield-hwsp
      ports:
        - containerPort: 8080
          protocol: TCP
      volumeMounts:
        - name: ncop-config
          mountPath: /opt/nfast/kmdata/config
        - name: ncop-hardserver
          mountPath: /opt/nfast/kmdata/hardserver.d
        - name: nfast-sockets
          mountPath: /opt/nfast/sockets
    - name: hashicorp-app
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: true
      image: >-
        registry.eselab.net/hashicorp-vault-nshield-app
      ports:
        - containerPort: 8080
          protocol: TCP
        - containerPort: 8200
          protocol: TCP
      env:
        - name: VAULT_ADDR
          value: "http://127.0.0.1:8200"
      resources: {}
      volumeMounts:
        - name: nfast-sockets
          mountPath: /opt/nfast/sockets
        - name: nfast-kmdata
          mountPath: /opt/nfast/kmdata
        - name: vault-config
          mountPath: /etc/vault
        - name: vault-data
          mountPath: /opt/vault/data
  securityContext: {}
  volumes:
    - name: ncop-config
      configMap:
        name: config
        defaultMode: 420
    - name: ncop-hardserver
      emptyDir: {}
    - name: nfast-sockets
```

```
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: nfast-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
    - name: vault-config
      persistentVolumeClaim:
        claimName: vault-config
    - name: vault-data
      persistentVolumeClaim:
        claimName: vault-data
```

Chapter 7. Additional resources and related products

7.1. nShield Connect

7.2. nShield as a Service

7.3. Entrust products

7.4. nShield product documentation