Red Hat OpenStack Platform 16.1

Manage Secrets with OpenStack Key Manager

How to integrate OpenStack Key Manager (Barbican) with your OpenStack deployment.
How to integrate OpenStack Key Manager (Barbican) with your OpenStack deployment.

OpenStack Team
rhos-docs@redhat.com
Abstract

How to integrate OpenStack Key Manager (Barbican) with your OpenStack deployment.
Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE .......................................................... 4

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION .................................. 5

CHAPTER 1. OVERVIEW .................................................................................. 6

CHAPTER 2. CHOOSING A BACKEND .............................................................. 7
  2.1. MIGRATING BETWEEN BACKENDS .................................................... 7

CHAPTER 3. DEPLOYING BARBICAN ............................................................. 8
  3.1. ADD USERS TO THE CREATOR ROLE ON OVERCLOUD ....................... 9
    3.1.1. Test barbican functionality .......................................................... 9
  3.2. UNDERSTANDING POLICIES ........................................................... 10
    3.2.1. Viewing the default policy ......................................................... 10

CHAPTER 4. MANAGING SECRETS IN BARBICAN ........................................ 12
  4.1. LISTING SECRETS .............................................................................. 12
  4.2. ADDING NEW SECRETS ..................................................................... 12
  4.3. UPDATING SECRETS ......................................................................... 12
  4.4. DELETING SECRETS .......................................................................... 12
  4.5. GENERATE A SYMMETRIC KEY ....................................................... 13
  4.6. BACKUP AND RESTORE KEYS .......................................................... 14
    4.6.1. Backup and restore the simple crypto back end ......................... 14
    4.6.1.1. Backup and restore the KEK .................................................... 14
    4.6.1.2. Backup and restore the back end database ......................... 14
    4.6.1.2.1. Create the test secret .......................................................... 14
    4.6.1.2.2. Backup the barbican database ......................................... 15
    4.6.1.2.3. Delete the test secrets ....................................................... 16
    4.6.1.2.4. Restore the databases ....................................................... 16
    4.6.1.2.5. Verify the restore process ................................................. 17
  4.7. CHOOSING A BACKEND ..................................................................... 19
    4.7.1. Simple crypto ............................................................................. 20
    4.7.2. PKCS#11 .................................................................................... 20
  4.8. HARDWARE SECURITY MODULE (HSM) SUPPORT ............................ 20
  4.9. MIGRATING BETWEEN BACKENDS .................................................. 20

CHAPTER 5. INTEGRATING BARBICAN WITH AN HSM APPLIANCE ............... 21
  5.1. INTEGRATE BARBICAN WITH AN ATOS HSM .................................... 21
  5.2. INTEGRATING OPENSTACK KEY MANAGER (BARBICAN) WITH A THALES LUNA NETWORK HSM ................................. 24
  5.3. INTEGRATING BARBICAN WITH AN ENTRUST NSHIELD CONNECT XC .......................... 26
  5.4. REVIEWING TLS ACTIVITY BETWEEN BARBICAN AND THE HSM .......... 28
  5.5. KEY STORAGE CONSIDERATIONS .................................................... 29
  5.6. ROTATING THE KEYS ....................................................................... 29
  5.7. PLANNING BACKUP FOR BARBICAN AND THE HSM ....................... 30

CHAPTER 6. ENCRYPTING CINDER VOLUMES ............................................. 31
  6.1. MIGRATE EXISTING VOLUME KEYS TO BARBICAN ............................ 33
    6.1.1. Overview of the migration steps ............................................... 34
    6.1.2. Behavioral differences ............................................................... 34
    6.1.3. Reviewing the migration process ............................................. 34
    6.1.4. Troubleshooting the migration process .................................... 35
    6.1.4.1. Role assignment ................................................................. 35
    6.1.5. Clean up the fixed keys ............................................................. 35
MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Using the Direct Documentation Feedback (DDF) function

Use the Add Feedback DDF function for direct comments on specific sentences, paragraphs, or code blocks.

1. View the documentation in the Multi-page HTML format.
2. Ensure that you see the Feedback button in the upper right corner of the document.
3. Highlight the part of text that you want to comment on.
4. Click Add Feedback.
5. Complete the Add Feedback field with your comments.
6. Optional: Add your email address so that the documentation team can contact you for clarification on your issue.
7. Click Submit.
CHAPTER 1. OVERVIEW

OpenStack Key Manager (barbican) is the secrets manager for Red Hat OpenStack Platform. You can use the barbican API and command line to centrally manage the certificates, keys, and passwords used by OpenStack services. Barbican currently supports the following use cases described in this guide:

- **Symmetric encryption keys** - used for Block Storage (cinder) volume encryption, ephemeral disk encryption, and Object Storage (swift) encryption, among others.

- **Asymmetric keys and certificates** - used for glance image signing and verification, among others.

In this release, barbican offers integration with the Block Storage (cinder) and Compute (nova) components.
CHAPTER 2. CHOOSING A BACKEND

Secrets (such as certificates, API keys, and passwords) can either be stored as an encrypted blob in the barbican database, or directly in a secure storage system.

To store the secrets as an encrypted blob in the barbican database, the following options are available:

- **Simple crypto plugin** - The simple crypto plugin is enabled by default and uses a single symmetric key to encrypt all secret payloads. This key is stored in plain text in the barbican.conf file, so it is important to prevent unauthorized access to this file.

- **PKCS#11 crypto plugin** - The PKCS#11 crypto plugin encrypts secrets with project-specific key encryption keys (pKEK), which are stored in the barbican database. These project-specific pKEKs are encrypted by a main key-encryption-key (KEK), which is stored in a hardware security module (HSM). All encryption and decryption operations take place in the HSM, rather than in-process memory. The PKCS#11 plugin communicates with the HSM through the PKCS#11 API. Because the encryption is done in secure hardware, and a different pKEK is used per project, this option is more secure than the simple crypto plugin. Red Hat supports the PKCS#11 backend with any of the following HSMs.

<table>
<thead>
<tr>
<th>Device</th>
<th>Supported in release</th>
<th>High Availability (HA) support</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATOS Trustway Proteccio NetHSM</td>
<td>16.0+</td>
<td>16.1+</td>
</tr>
<tr>
<td>Entrust nShield Connect HSM</td>
<td>16.0+</td>
<td>Not supported</td>
</tr>
<tr>
<td>Thales Luna Network HSM</td>
<td>16.1 (Technology Preview)</td>
<td>16.1 (Technology Preview)</td>
</tr>
</tbody>
</table>

**NOTE**

Regarding high availability (HA) options: The barbican service runs within Apache and is configured by director to use HAProxy for high availability. HA options for the back end layer will depend on the back end being used. For example, for simple crypto, all the barbican instances have the same encryption key in the config file, resulting in a simple HA configuration.

2.1. MIGRATING BETWEEN BACKENDS

You can configure a single instance of Barbican to use more than one backend. When this is done, you must specify a backend as the **global default** backend. You can also specify a default backend per project. If no mapping exists for a project, the secrets for that project are stored using the global default backend.

For example, you can configure Barbican to use both the Simple crypto and PKCS#11 plugins. If you set Simple crypto as the global default, then all projects will use that backend. You can then specify which projects use the PKCS#11 backend by setting PKCS#11 as the preferred backend for that project.

If you decide to migrate to a new backend, you can keep the original available while enabling the new backend as the global default or as a project-specific backend. As a result, the old secrets remain available through the old backend, and new secrets are stored in the new global default backend.
CHAPTER 3. DEPLOYING BARBICAN

Barbican is not enabled by default in Red Hat OpenStack Platform. This procedure describes how you can deploy barbican in an existing OpenStack deployment. Barbican runs as a containerized service, so this procedure also describes how to prepare and upload the new container images:

NOTE

This procedure configures barbican to use the `simple_crypto` backend. Additional backends are available, such as `PKCS11` which requires a different configuration, and different heat template files depending on which HSM is used. Other backends such as KMIP, Hashicorp Vault and DogTag are not supported in this release.

1. On the undercloud node, create an environment file for barbican.

   ```bash
   $ cat /home/stack/templates/configure-barbican.yaml
   parameter_defaults:
     BarbicanSimpleCryptoGlobalDefault: true
   
   - BarbicanSimpleCryptoGlobalDefault - Sets this plugin as the global default plugin.
   
   Further options are also configurable:
   - BarbicanPassword - Sets a password for the barbican service account.
   - BarbicanWorkers - Sets the number of workers for `barbican::wsgi::apache`. Uses ‘% {::processorcount}’ by default.
   - BarbicanDebug - Enables debugging.
   - BarbicanPolicies - Defines policies to configure for barbican. Uses a hash value, for example: `{ barbican-context_is_admin: { key: context_is_admin, value: 'role:admin' } }`. This entry is then added to `/etc/barbican/policy.json`. Policies are described in detail in a later section.
   - BarbicanSimpleCryptoKek - The Key Encryption Key (KEK) is generated by director, if none is specified.

2. Include the following in files in the `openstack overcloud deploy` command, without removing previously added role, template or environment files from the script:

   ```bash
   - /usr/share/openstack-tripleo-heat-templates/environments/services/barbican.yaml
   - /usr/share/openstack-tripleo-heat-templates/environments/barbican-backend-simple-crypto.yaml
   - /home/stack/templates/configure-barbican.yaml
   
   3. Re-run the deployment script to apply changes to your deployment:

   ```bash
   $ openstack overcloud deploy \
   --timeout 100 \
   --templates /usr/share/openstack-tripleo-heat-templates \
   --stack overcloud \
   --libvirt-type kvm \
   --ntp-server clock.redhat.com \
   ```
3.1. ADD USERS TO THE CREATOR ROLE ON OVERCLOUD

Users must be members of the **creator** role in order to create and edit barbican secrets, or to create encrypted volumes that store their secret in barbican.

1. Retrieve the id of the **creator** role:

   openstack role show creator

   +-----------+----------------------------------+
   | Field     | Value                            |
   +-----------+----------------------------------+
   | domain_id | None                             |
   | id        | 4e9c560c6f104608948450fbf316f9d7 |
   | name      | creator                          |
   +-----------+----------------------------------+

   **NOTE**
   
   You will not see the **creator** role unless OpenStack Key Manager (barbican) is installed.

2. Assign a user to the **creator** role and specify the relevant project. In this example, a user named **user1** in the **project_a** project is added to the **creator** role:

   openstack role add --user user1 --project project_a 4e9c560c6f104608948450fbf316f9d7

3.1.1. Test barbican functionality

This section describes how to test that barbican is working correctly.

1. Create a test secret. For example:

   $ openstack secret store --name testSecret --payload 'TestPayload'

   +-----------+------------------------------------------------------------------------------------+
   | Field     | Value                                                                              |
   +-----------+------------------------------------------------------------------------------------+
   | Secret href| https://192.168.123.163/key-manager/v1/secrets/4cc5ffe0-eea2-449d-9e64-b664d574be53 |
   | Name      | testSecret                                                                       |
   | Created   | None                                                                              |
2. Retrieve the payload for the secret you just created:

```
openstack secret get https://192.168.123.163/key-manager/v1/secrets/4cc5ffe0-eea2-449d-9e64-b664d574be53 --payload
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payload</td>
<td>TestPayload</td>
</tr>
</tbody>
</table>

3.2. UNDERSTANDING POLICIES

Barbican uses policies to determine which users are allowed to perform actions against the secrets, such as adding or deleting keys. To implement these controls, keystone project roles (such as **creator** you created earlier) are mapped to barbican internal permissions. As a result, users assigned to those project roles receive the corresponding barbican permissions.

3.2.1. Viewing the default policy

The default policy is defined in code and typically does not require any amendments. If policy changes have not been made, you can view the default policy using the existing container in your environment. If changes have been made to the default policy, and you would like to see the defaults, use a separate system to pull the `openstack-barbican-api` container first:

1. Use your Red Hat credentials to log in to podman:

   ```
   podman login
   username: ********
   password: ********
   ```

2. Pull the `openstack-barbican-api` container:

   ```
   podman pull \n   registry.redhat.io/rhosp-rhel8/openstack-barbican-api:16.1
   ```

3. Run the `oslopolicy-policy-generator` command from inside the container:

   ```
   podman run -it \n   registry.redhat.io/rhosp-rhel8/openstack-barbican-api:16.1 \n   oslopolicy-policy-generator \n   --namespace barbican > barbican-policy.yaml
   ```

   This generates a policy file in your present working directory. The contents of this file are explained in the following step.
The `barbican-policy.yaml` file you generated describes the policies used by barbican. The policy is implemented by four different roles that define how a user interacts with secrets and secret metadata. A user receives these permissions by being assigned to a particular role:

- **admin** - Can delete, create/edit, and read secrets.
- **creator** - Can create/edit, and read secrets. Can not delete secrets.
- **observer** - Can only read data.
- **audit** - Can only read metadata. Can not read secrets.

For example, the following entries list the **admin**, **observer**, and **creator** keystone roles for each project. On the right, notice that they are assigned the `role:admin`, `role:observer`, and `role:creator` permissions:

```yaml
# "admin": "role:admin"
#
# "observer": "role:observer"
#
# "creator": "role:creator"
```

These roles can also be grouped together by barbican. For example, rules that specify `admin_or_creator` can apply to members of either `rule:admin` or `rule:creator`.

Further down in the file, there are `secret:put` and `secret:delete` actions. To their right, notice which roles have permissions to execute these actions. In the following example, `secret:delete` means that only **admin** and **creator** role members can delete secret entries. In addition, the rule states that users in the **admin** or **creator** role for that project can delete a secret in that project. The project match is defined by the `secret_project_match` rule, which is also defined in the policy.

```yaml
secret:delete": "rule:admin_or_creator and rule:secret_project_match"
```
CHAPTER 4. MANAGING SECRETS IN BARBICAN

4.1. LISTING SECRETS

Secrets are identified by their URI, indicated as a href value. This example shows the secret you created in the previous step:

```bash
$ openstack secret list
```

<table>
<thead>
<tr>
<th>Secret href</th>
<th>Name</th>
<th>Created</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content types</td>
<td>Algorithm</td>
<td>Bit length</td>
<td>Secret type</td>
</tr>
<tr>
<td>+--------+-------------------------------------------+-----------+------------+-------------+------+------------+------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://192.168.123.169:9311/v1/secrets/24845e6d-64a5-4071-ba99-0fdd1046172e">https://192.168.123.169:9311/v1/secrets/24845e6d-64a5-4071-ba99-0fdd1046172e</a></td>
<td>None</td>
<td>2018-01-22T02:15:00+00:00</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>+--------+-------------------------------------------+-----------+------------+-------------+------+------------</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.2. ADDING NEW SECRETS

Create a test secret. For example:

```bash
$ openstack secret store --name testSecret --payload 'TestPayload'
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secret href</td>
<td><a href="https://192.168.123.169:9311/v1/secrets/ecc7b2a4-f0b0-47ba-b451-0f7d42bc1746">https://192.168.123.169:9311/v1/secrets/ecc7b2a4-f0b0-47ba-b451-0f7d42bc1746</a></td>
</tr>
<tr>
<td>Name</td>
<td>testSecret</td>
</tr>
<tr>
<td>Created</td>
<td>None</td>
</tr>
<tr>
<td>Status</td>
<td>None</td>
</tr>
<tr>
<td>Content types</td>
<td>None</td>
</tr>
<tr>
<td>Algorithm</td>
<td>aes</td>
</tr>
<tr>
<td>Bit length</td>
<td>256</td>
</tr>
<tr>
<td>Secret type</td>
<td>opaque</td>
</tr>
<tr>
<td>Mode</td>
<td>cbc</td>
</tr>
<tr>
<td>Expiration</td>
<td>None</td>
</tr>
</tbody>
</table>

4.3. UPDATING SECRETS

You cannot change the payload of a secret (other than deleting the secret), but if you initially created a secret without specifying a payload, you can later add a payload to it by using the update function. For example:

```bash
$ openstack secret update https://192.168.123.169:9311/v1/secrets/ca34a264-fd09-44a1-8856-c6e7116c3b16 'TestPayload-updated'
```

4.4. DELETING SECRETS
You can delete a secret by specifying its URI. For example:

$ openstack secret delete https://192.168.123.163:9311/v1/secrets/ecc7b2a4-f0b0-47ba-b451-0f7d42bc1746
$

### 4.5. GENERATE A SYMMETRIC KEY

Symmetric keys are suitable for certain tasks, such as nova disk encryption and swift object encryption.

1. Generate a new 256-bit key using `order create` and store it in barbican. For example:

   ```bash
   $ openstack secret order create --name swift_key --algorithm aes --mode ctr --bit-length 256 --payload-content-type=application/octet-stream key
   +----------------+-----------------------------------------------------------------------------------+
   | Field          | Value                                                                             |
   +----------------+-----------------------------------------------------------------------------------+
   | Order href     | https://192.168.123.173:9311/v1/orders/043383fe-d504-42cf-a9b1-bc328d0b4832 |
   | Type           | Key                                                                               |
   | Container href | N/A                                                                               |
   | Secret href    | None                                                                              |
   | Created        | None                                                                              |
   | Status         | None                                                                              |
   | Error code     | None                                                                              |
   | Error message  | None                                                                              |
   +----------------+-----------------------------------------------------------------------------------+
   --mode - Generated keys can be configured to use a particular mode, such as `ctr` or `cbc`. For more information, see [NIST SP 800-38A](https://csrc.nist.gov/publications/detail/sp/800-38a/rev-4).

2. View the details of the order to identify the location of the generated key, shown here as the `Secret href` value:

   ```bash
   $ openstack secret order get https://192.168.123.173:9311/v1/orders/043383fe-d504-42cf-a9b1-bc328d0b4832
   +----------------+------------------------------------------------------------------------------------+
   | Field          | Value                                                                              |
   +----------------+------------------------------------------------------------------------------------+
   | Order href     | https://192.168.123.173:9311/v1/orders/043383fe-d504-42cf-a9b1-bc328d0b4832 |
   | Type           | Key                                                                                |
   | Container href | N/A                                                                                |
   | Created        | 2018-01-24T04:24:33+00:00                                                          |
   | Status         | ACTIVE                                                                             |
   | Error code     | None                                                                              |
   | Error message  | None                                                                              |
   +----------------+------------------------------------------------------------------------------------+
   ``

3. Retrieve the details of the secret:

   ```bash
   $ openstack secret get https://192.168.123.173:9311/v1/secrets/efcfec49-b9a3-4425-a9b6-5ba69cb18719
   ```
4.6. BACKUP AND RESTORE KEYS

The process for backup and restore of encryption keys will vary depending on the type of back end:

4.6.1. Backup and restore the simple crypto back end

Two separate components need to be backed up for *simple crypto* back end: the KEK and the database. It is recommended that you regularly test your backup and restore process.

4.6.1.1. Backup and restore the KEK

For the *simple crypto* back end, you need to backup the *barbican.conf* file that contains the main KEK. This file must be backed up to a security hardened location. The actual data is stored in the Barbican database in an encrypted state, described in the next section.

- To restore the key from a backup, you need to copy the restored *barbican.conf* over the existing *barbican.conf*.

4.6.1.2. Backup and restore the back end database

This procedure describes how to backup and restore a barbican database for the simple crypto back end. To demonstrate this, you will generate a key and upload the secrets to barbican. You will then backup the barbican database, and delete the secrets you created. You will then restore the database and confirm that the secrets you created earlier have been recovered.

**NOTE**

Be sure you are also backing up the KEK, as this is also an important requirement. This is described in the previous section.

4.6.1.2.1. Create the test secret

1. On the overcloud, generate a new 256-bit key using `order create` and store it in barbican. For example:

   ```bash
   (overcloud) [stack@undercloud-0 ~]$ openstack secret order create --name swift_key --algorithm aes --mode ctr --bit-length 256 --payload-content-type=application/octet-stream key
   ```
2. Create a test secret:

```
(overcloud) [stack@undercloud-0 ~]$ openstack secret store --name testSecret --payload 'TestPayload'
```


3. Confirm that the secrets were created:

```
(overcloud) [stack@undercloud-0 ~]$ openstack secret list
```

---
<table>
<thead>
<tr>
<th>Secret href</th>
<th>Name</th>
<th>Created</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://10.0.0.104:9311/v1/secrets/93f62cfd-e008-401f-be74-bf057c88b04a">http://10.0.0.104:9311/v1/secrets/93f62cfd-e008-401f-be74-bf057c88b04a</a></td>
<td>testSecret</td>
<td>2018-06-19T18:25:25+00:00</td>
<td>ACTIVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{u'default': u'text/plain'}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>opaque</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cbc</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>None</td>
</tr>
<tr>
<td><a href="http://10.0.0.104:9311/v1/secrets/f664b5cf-5221-47e5-9887-608972a5fe8f">http://10.0.0.104:9311/v1/secrets/f664b5cf-5221-47e5-9887-608972a5fe8f</a></td>
<td>swift_key</td>
<td>2018-06-19T18:24:40+00:00</td>
<td>ACTIVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{u'default': u'application/octet-stream'}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>symmetric</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ctr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>None</td>
</tr>
</tbody>
</table>

---

4.6.1.2.2. Backup the barbican database
Run these steps while logged in to the **controller-0** node.

**NOTE**

Only the user *barbican* has access to the *barbican* database. So the barbican user password is required to backup or restore the database.

1. Retrieve *barbican* user password. For example:

   ```bash
   [heat-admin@controller-0 ~]$ sudo grep -r "barbican::db::mysql::password" /etc/puppet/hieradata /etc/puppet/hieradata/service_configs.json:  "barbican::db::mysql::password": "seDJRsMNRrBdFryCmNUEFPPev",
   ```

2. Backup the *barbican* database:

   ```bash
   [heat-admin@controller-0 ~]$ mysqldump -u barbican -p"seDJRsMNRrBdFryCmNUEFPPev" barbican > barbican_db_backup.sql
   ```

3. Database backup is stored in `/home/heat-admin`

   ```bash
   [heat-admin@controller-0 ~]$ ll
   total 36
   -rw-rw-r--. 1 heat-admin heat-admin 36715 Jun 19 18:31 barbican_db_backup.sql
   ```

4.6.1.2.3. Delete the test secrets

1. On the overcloud, delete the secrets you created previously, and verify they no longer exist. For example:

   ```bash
   (overcloud) [stack@undercloud-0 ~]$ openstack secret delete http://10.0.0.104:9311/v1/secrets/93f62cfd-e008-401f-be74-bf057c88b04a
   (overcloud) [stack@undercloud-0 ~]$ openstack secret delete http://10.0.0.104:9311/v1/secrets/f664b5cf-5221-47e5-9887-608972a5fefb
   (overcloud) [stack@undercloud-0 ~]$ openstack secret list
   (overcloud) [stack@undercloud-0 ~]$ openstack secret list
   ```

4.6.1.2.4. Restore the databases

Run these steps while logged in to the **controller-0** node.

1. Make sure you have the *barbican* database on the controller which grants access to the *barbican* user for database restoration:

   ```bash
   [heat-admin@controller-0 ~]$ mysql -u barbican -p"seDJRsMNRrBdFryCmNUEFPPev" Welcome to the MariaDB monitor. Commands end with ; or 'g.
   Your MariaDB connection id is 3799
   Server version: 10.1.20-MariaDB MariaDB Server
   Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.
   Type 'help;' or '?h' for help. Type '\c' to clear the current input statement.
   ```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| barbican           |
| information_schema |
+--------------------+
2 rows in set (0.00 sec)

MariaDB [(none)]> exit
Bye
[heat-admin@controller-0 ~]$ 

9) Restore the backup file to the barbican database:

```
[heat-admin@controller-0 ~]$ sudo mysql -u barbican -p"seDJRsMNRrBdFryCmNUEFPPev" barbican < barbican_db_backup.sql
[heat-admin@controller-0 ~]$ 
```

4.6.1.2.5. Verify the restore process

1. On the overcloud, verify that the test secrets were restored successfully:

```
(overcloud) [stack@undercloud-0 ~]$ openstack secret list

+------------------------------------------------------------------------+------------+------------------------+-----------+------------+-------------+------+------+
| Secret href                                                            | Name       | Created                | Status    | Content types          | Algorithm | Bit length | Secret type | Mode | Expiration |
+------------------------------------------------------------------------+------------+------------------------+-----------+------------------------+------------+-----------+-------------+------+--------+
| http://10.0.0.104:9311/v1/secrets/93f62cfd-e008-401f-be74-bf057c88b04a | testSecret | 2018-06-19T18:25:25+00:00 | ACTIVE    | {u'default': u'text/plain'} | aes        | 256       | opaque      | cbc  | None       |
| http://10.0.0.104:9311/v1/secrets/f664b5cf-5221-47e5-9887-608972a5f6fb | swift_key  | 2018-06-19T18:24:40+00:00 | ACTIVE    | {u'default': u'application/octet-stream'} | aes        | 256       | symmetric   | ctr  | None       |
+------------------------------------------------------------------------+------------+------------------------+-----------+------------------------+------------+-----------+-------------+------+--------+
```

== Barbican Hardware Security Module (HSM) Integration

OpenStack Key Manager (Barbican) is the secrets manager for Red Hat OpenStack Platform. You can use the Barbican API and command line to centrally manage the certificates, keys, and passwords used by OpenStack services. Barbican currently supports the following use cases described in this guide:

- **Symmetric encryption keys** - used for Block Storage (cinder) volume encryption, ephemeral disk encryption, Object Storage (Swift) encryption, among others.
- **Asymmetric keys and certificates** - glance image signing and verification, octavia TLS load balancing, among others.

In this release, Barbican offers integration with the Block Storage (cinder), Networking (neutron), and Compute (nova) components.
4.7. CHOOSING A BACKEND

Secrets (such as certificates, API keys, and passwords) can either be stored as an encrypted blob in the Barbican Database or in a HSM (Hardware Security Module).

- **Admin/Deployer**
  - Create Cinder volume encryption type LUKS
  - Create MKEK through CLI

- **Tenant**
  - Give me a volume V of encryption type LUKS
  - Attach volume to VM
  - Get volume metadata
  - Read Kref from metadata
  - Get secret Kref

- **Nova (QEMU)**
  - Generate a key KI as needed for encryption type LUKS
  - Get MKEK(pKEK)

- **Cinder**
  - Store MKEK in HSM
  - Generate MKEK(pKEK)
  - Store MKEK(pKEK)

- **Barbican**
  - Generate KI
  - Generate KI, decrypt MKEK(pKEK), wrap KI with pKEK
  - Return pKEK(KI), MKEK(pKEK)

- **Barbican Database**
  - Store pKEK(KI), MKEK(pKEK) as metadata

- **Hardware Security Module**
  - Return reference to secret KI (Kref)
  - Return Kref

- **Ceph Cluster**
  - Entry contains pKEK(KI), MKEK(pKEK)
  - Unwrap pKEK from MKEK(pKEK), unwrap KI from pKEK(KI)

Data is encrypted (QEMU -> LUKS process) on the compute node with KI before entering the Ceph cluster.
Secrets (such as certificates, API keys, and passwords) can either be stored as an encrypted blob in the Barbican database, directly in a secure storage system, including a Hardware Security Module (HSM) appliance.

### 4.7.1. Simple crypto

The *simple crypto* plugin is enabled by default and uses a single symmetric key to encrypt all secret payloads. This key is stored in plain text in the `barbican.conf` file.

### 4.7.2. PKCS#11

You must plan the overcloud network topology so that the Controller nodes where the Barbican containers are run have network access to the HSM that will be used by the PKCS#11 backend. You also need a password-protected HTTPS server to host the client software provided by your HSM vendor. At deployment time, these client files are downloaded to the Controller nodes.

### 4.8. HARDWARE SECURITY MODULE (HSM) SUPPORT

This guide explains how to integrate Barbican with various HSM appliances.

You can use the PKCS#11 crypto plugin to store the secrets in a Hardware Security Module (HSM), which are physical rack-mounted appliances produced by third party vendors. These secrets are encrypted using the pKEK, which in turn is also stored in the Barbican database. The pKEK is encrypted and an HMAC operation is applied using the MKEK and HMAC keys, which are also stored in the HSM.

There are additional plugins that can be used, such as the [KMIP plugin](https://www.kslko.com/kmip/) and [Red Hat Certificate System (dogtag)](https://www.redhat.com/support/ods), however these are not supported at this time.

**NOTE**

Regarding high availability (HA) options: The Barbican service runs within Apache and is configured by director to use HAProxy for high availability. Your HA options for the backend layer will depend on which backend is used: For example, with simple crypto, all the Barbican instances have the same encryption key in the configuration file, resulting in a simple HA configuration.

### 4.9. MIGRATING BETWEEN BACKENDS

Barbican allows you to define a different backend for a project. If no mapping exists for a project, then secrets are stored in the global default backend. This means that multiple backends can be configured, but there must be only one global backend defined. The heat templates supplied for the different backends contain the parameters that set each backend as the default.

If you do store secrets in a certain backend and then decide to migrate to a new backend, you can keep the old backend available while enabling the new backend as the global default (or as a project-specific backend). As a result, the old secrets remain available through the old backend.
CHAPTER 5. INTEGRATING BARBICAN WITH AN HSM APPLIANCE

Integrate your Red Hat OpenStack Platform deployment with hardware security module (HSM) appliances to increase your security posture by using hardware based cryptographic processing.

5.1. INTEGRATE BARBICAN WITH AN ATOS HSM

You can integrate the PKCS#11 back end with your Trustway Proteccio Net HSM appliance. You can enable HA by listing two or more HSMs below the `atos_hsms` parameter.

**Prerequisites**

- A password-protected HTTPS server that provides vendor software for the Atos HSM

**Table 5.1. Files provided by the HTTPS server**

<table>
<thead>
<tr>
<th>File</th>
<th>Example</th>
<th>Provided by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proteccio Client Software IS</td>
<td>Proteccio1.09.05.iso</td>
<td>HSM Vendor</td>
</tr>
<tr>
<td>image file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL server certificate</td>
<td>proteccio.CRT</td>
<td>HSM administrator</td>
</tr>
<tr>
<td>SSL client certificate</td>
<td>client.CRT</td>
<td>HSM administrator</td>
</tr>
<tr>
<td>SSL Client key</td>
<td>client.KEY</td>
<td>HSM administrator</td>
</tr>
</tbody>
</table>

**Procedure**

1. Create a `configure-barbican.yaml` environment file for Barbican and add the following parameters:

   ```yaml
   parameter_defaults
   BarbicanSimpleCryptoGlobalDefault: false
   BarbicanPkcs11CryptoGlobalDefault: true
   BarbicanPkcs11CryptoLogin: ********
   BarbicanPkcs11CryptoSlotId: 1
   ATOSVars:
   atos_client_iso_name: Proteccio1.09.05.iso
   atos_client_iso_location: https://user@PASSWORD:example.com/Proteccio1.09.05.iso
   atos_client_cert_location: https://user@PASSWORD:example.com/client.CRT
   atos_client_key_location: https://user@PASSWORD:example.com/client.KEY
   atos_hsms:
   - name: myHsm1
     server_cert_location: https://user@PASSWORD:example.com/myHsm1.CRT
     ip: 192.168.1.101
   - name: myHsm2
     server_cert_location: https://user@PASSWORD:example.com/myHsm2.CRT
     ip: 192.168.1.102
   ```
The **atos_hsms** parameter supersedes the parameters **atos_hsm_ip_address** and **atos_server_cert_location** which have been deprecated and will be removed in a future release.

### Table 5.2. Heat parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarbicanSimpleCryptoGlobalDefault</td>
<td>This is a boolean that determines if <strong>simplecrypto</strong> will be the global default.</td>
</tr>
<tr>
<td>BarbicanPkcs11GlobalDefault</td>
<td>This is a boolean that determines if <strong>PKCS#11</strong> will be the global default.</td>
</tr>
<tr>
<td>BarbicanPkcs11CryptoSlotId</td>
<td>Slot ID for the Virtual HSM to be used by Barbican.</td>
</tr>
<tr>
<td>ATOSVars</td>
<td></td>
</tr>
<tr>
<td>atos_client_iso_name</td>
<td>The filename for the Atos client software ISO. This value must match the filename in the URL for the <strong>atos_client_iso_location</strong> parameter.</td>
</tr>
<tr>
<td>atos_client_iso_location</td>
<td>The URL, including the username and password, that specifies the HTTPS server location of the Proteccio Client Software ISO image.</td>
</tr>
<tr>
<td>atos_client_cert_location</td>
<td>The URL, including the username and password, that specifies the HTTPS server location of the SSL client certificate.</td>
</tr>
<tr>
<td>atos_client_key_location</td>
<td>The URL, including the username and password, that specifies the HTTPS server location of the SSL client key. This must be the matching key for the client certificate above.</td>
</tr>
<tr>
<td>atos_hsms</td>
<td>A list of one or more HSMs that specifies the name, certificate location and IP address of the HSM. When you include more than one HSM in this list, Barbican configures the HSMs for load balancing and high availability.</td>
</tr>
</tbody>
</table>
NOTE

By default, the HSM can have a maximum of 32 concurrent connections. If you exceed this number, you might experience a memory error from the PKCS#11 client. You can calculate the number of connections as follows:

- Each Controller has one **barbican-api** and one **barbican-worker** process.
- Each Barbican API process is executed with **N** Apache workers - (where **N** defaults to the number of CPUs).
- Each worker has one connection to the HSM.

Each **barbican-worker** process has one connection to the database. You can use the **BarbicanWorkers** heat parameter to define the number of Apache workers for each API process. By default, the number of Apache workers matches the CPU count.

For example, if you have three Controllers, each with 32 cores, then the Barbican API on each Controller uses 32 Apache workers. Consequently, one Controller consumes all 32 HSM connections available. To avoid this contention, limit the number of Barbican Apache workers configured for each node. In this example, set **BarbicanWorkers** to **10** so that all three Controllers can make ten concurrent connections each to the HSM.

2. Include the custom **configure-barbican.yaml**, **barbican.yaml** and ATOS specific **barbican-backend-pkcs11-atos.yaml** environment files in the deployment command, as well as any other environment files relevant to your deployment:

   ```
   $ openstack overcloud deploy \
   --timeout 100 \
   --templates /usr/share/openstack-tripleo-heat-templates \
   --stack overcloud \
   --libvirt-type kvm \
   --ntp-server clock.redhat.com \
   -e /home/stack/virt/config_lvm.yaml \
   -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
   -e /home/stack/virt/network/network-environment.yaml \
   -e /home/stack/virt/hostnames.yaml \
   -e /home/stack/virt/nodes_data.yaml \
   -e /home/stack/virt/extra_templates.yaml \
   -e /home/stack/container-parameters-with-barbican.yaml \
   -e /usr/share/openstack-tripleo-heat-templates/environments/services/barbican.yaml \
   -e /usr/share/openstack-tripleo-heat-templates/environments/barbican-backend-pkcs11-atos.yaml \
   -e /home/stack/configure-barbican.yaml \
   --log-file overcloud_deployment_with_atos.log
   ```

Verification

1. Create a test secret:

   ```
   $ openstack secret store --name testSecret --payload 'TestPayload'
   +---------------+------------------------------------------------------------------------------------+
   | Field         | Value                                                                              |
   +---------------+------------------------------------------------------------------------------------+
   | Field         | Value                                                                              |
   +---------------+------------------------------------------------------------------------------------+
   ```
2. Retrieve the payload for the secret that you just created:

```
openstack secret get https://192.168.123.163/key-manager/v1/secrets/4cc5ffe0-eea2-449d-9e64-b664d574be53 --payload
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payload</td>
<td>TestPayload</td>
</tr>
</tbody>
</table>

5.2. INTEGRATING OPENSTACK KEY MANAGER (BARBICAN) WITH A THALES LUNA NETWORK HSM

Integrate the PKCS#11 backend with your Thales Luna Network HSM appliance for hardware based cryptographic processing. Use an Ansible role to download and install the Thales Luna client software on the Controller, and a Key Manager configuration file to include the predefined HSM IP and credentials.

Prerequisites

- A password-protected HTTPS server that provides vendor software for the Thales Luna Network HSM.
- The vendor provided Luna Network HSM client software in a compressed zip archive.

Procedure

1. Install the `ansible-role-lunasa-hsm` role on the director:

```
sudo dnf install ansible-role-lunasa-hsm
```

2. Create a `configure-barbican.yaml` environment file for Key Manager (barbican) and add parameters specific to your environment.

```
parameter_defaults:
  BarbicanPkcs11CryptoMKEKLabel: "barbican_mkek_0"
  BarbicanPkcs11CryptoHMACLabel: "barbican_hmac_0"
  BarbicanPkcs11CryptoLogin: "$PKCS_11_USER_PIN"
  BarbicanPkcs11CryptoGlobalDefault: true
  LunasaVars:
    lunasa_client_tarball_name: 610-012382-014_SW_Client_HSM_6.2_RevA.tar.zip
```
lunasa_client_tarball_location: https://user:$PASSWORD@http-server.example.com/luna_software/610-012382-014_SW_Client_HSM_6.2_RevA.tar.zip
lunasa_client_installer_path: 610-012382-014_SW_Client_HSM_6.2_RevA/linux/64/install.sh
lunasa_hsms:
  - hostname: luna-hsm.example.com
    admin_password: "$HSM_ADMIN_PASSWORD"
    partition: myPartition1
    partition_serial: 123456789

Table 5.3. Heat parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarbicanSimpleCryptoGlobalDefault</td>
<td>This is a boolean that determines if simplecrypto will be the global default.</td>
</tr>
<tr>
<td>BarbicanPkcs11GlobalDefault</td>
<td>This is a boolean that determines if PKCS#11 will be the global default.</td>
</tr>
<tr>
<td>BarbicanPkcs11CryptoTokenLabel</td>
<td>If you have one HSM, then the value of the parameter will be the partition Label. If you are using HA between two or more partitions, then this is the label that you want to give to the HA group.</td>
</tr>
<tr>
<td>BarbicanPkcs11CryptoLogin</td>
<td>The PKCS#11 password used to log into the HSM, provided by the HSM administrator.</td>
</tr>
</tbody>
</table>

LunasaVar

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lunasa_client_tarball_name</td>
<td>The name of the Luna software tarball.</td>
</tr>
<tr>
<td>lunasa_client_tarball_location</td>
<td>The URL that specifies the HTTPS server location of the Luna Software tarball.</td>
</tr>
<tr>
<td>lunasa_client_installer_path</td>
<td>Path to the install.sh script in the zipped tarball.</td>
</tr>
<tr>
<td>lunasa_client_rotate_cert</td>
<td>(Optional) When set to true, new client certificates will be generated to replace any existing certificates. Default: false</td>
</tr>
<tr>
<td>lunasa_client_working_dir</td>
<td>(Optional) Working directory in the Controller nodes. Default: /tmp/lunasa_client_install</td>
</tr>
<tr>
<td>lunasa_hsms</td>
<td>A list of one or more HSMs that specifies the name, hostname, admin_password, partition, and partition serial number. When you include more than one HSM in this list, Barbican configures the HSMs for high availability.</td>
</tr>
</tbody>
</table>
3. Include the custom `configure-barbican.yaml` and Thales specific `barbican-backend-pkcs11-lunasa.yaml` environment files in the deployment command, as well as any other templates relevant for your deployment:

$$
$ openstack overcloud deploy --templates \\
.... \\
-e /usr/share/openstack-tripleo-heat-templates/environments/services/barbican.yaml \\
-e /usr/share/openstack-tripleo-heat-templates/environments/barbican-backend-pkcs11-lunasa.yaml \\
-e /home/stack/configure-barbican.yaml \\
--log-file overcloud_deployment_with_luna.log
$$

### 5.3. INTEGRATING BARBICAN WITH AN ENTRUST NSHIELD CONNECT XC

You can integrate the PKCS#11 backend with your Entrust nShield Connect XC HSM. Use an Ansible role to download and install the Entrust client software on the Controller, and a Barbican configuration file to include the predefined HSM IP and credentials.

**Prerequisites**

- A password-protected HTTPS server that provides vendor software for the Entrust nShield Connect XC.

**Procedure**

1. Create a `configure-barbican.yaml` environment file for Barbican and add parameters specific to your environment. Use the following snippet as an example:

```yaml
parameter_defaults:
  VerifyGlanceSignatures: true
  SwiftEncryptionEnabled: true
  ComputeExtraConfig:
    BarbicanPkcs11CryptoLogin: 'sample string'
    BarbicanPkcs11CryptoSlotId: '492971158'
    BarbicanPkcs11CryptoGlobalDefault: true
    BarbicanPkcs11CryptoLibraryPath: '/opt/nfast/toolkits/pkcs11/libcknfast.so'
    BarbicanPkcs11CryptoEncryptionMechanism: 'CKM_AES_CBC'
    BarbicanPkcs11CryptoHMACKeyType: 'CKK_SHA256_HMAC'
    BarbicanPkcs11CryptoHMACKeygenMechanism: 'CKM_NC_SHA256_HMAC_KEY_GEN'
    BarbicanPkcs11CryptoMKEKLabel: 'barbican_mkek_10'
    BarbicanPkcs11CryptoMKEKLength: '32'
    BarbicanPkcs11CryptoHMACLabel: 'barbican_hmac_10'
    BarbicanPkcs11CryptoThalesEnabled: true
    BarbicanPkcs11CryptoEnabled: true
  ThalesVars:
    thales_client_working_dir: /tmp/thales_client_install
    thales_client_tarball_location: https://your server/CipherTools-linux64-dev-12.40.2.tgz
    thales_client_tarball_name: CipherTools-linux64-dev-12.40.2.tgz
    thales_client_path: linux/libc6_11/amd64/nfast
    thales_client_uid: 42481
    thales_client_gid: 42481
    thales_km_data_location: https://your server/kmdata_post_card_creation.tar.gz
```
### thales_km_data_tarball_name: kmdata_post_card_creation.tar.gz
### thales_hsm_ip_address: 192.168.10.10
### thales_rfs_server_ip_address: 192.168.10.11
### thales_hsm_config_location: hsm-C90E-02E0-D947
### thales_rfs_user: root
### thales_rfs_key: |

-----BEGIN RSA PRIVATE KEY-----
Sample private key
-----END RSA PRIVATE KEY-----

resource_registry:
  OS::TripleO::Services::BarbicanBackendPkcs11Crypto: /home/stack/tripleo-heat-templates/puppet/services/barbican-backend-pkcs11-crypto.yaml

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarbicanSimpleCryptoGlobalDefault</td>
<td>This is a boolean that determines if simplecrypto will be the global default.</td>
</tr>
<tr>
<td>BarbicanPkcs11GlobalDefault</td>
<td>This is a boolean that determines if PKCS#11 will be the global default.</td>
</tr>
<tr>
<td>BarbicanPkcs11CryptoSlotId</td>
<td>Slot ID for the Virtual HSM to be used by Barbican.</td>
</tr>
<tr>
<td>BarbicanPkcs11CryptoMKEKLabel</td>
<td>This parameter defines the name of the mKEK generated in the HSM. Director creates this key in the HSM using this name.</td>
</tr>
<tr>
<td>BarbicanPkcs11CryptoHMACLabel</td>
<td>This parameter defines the name of the HMAC key generated in the HSM. Director creates this key in the HSM using this name.</td>
</tr>
</tbody>
</table>

**ThalesVars**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>thales_client_working_dir</td>
<td>A user-defined temporary working directory.</td>
</tr>
<tr>
<td>thales_client_tarball_location</td>
<td>The URL that specifies the HTTPS server location of the Entrust software.</td>
</tr>
<tr>
<td>thales_km_data_tarball_name</td>
<td>The name of the Entrust software tarball.</td>
</tr>
<tr>
<td>thales_rfs_key</td>
<td>A private key used to obtain an SSH connection to the RFS server. You must add this as an authorized key to the RFS server.</td>
</tr>
</tbody>
</table>

2. Include the custom `configure-barbican.yaml` environment file, along with the `barbican.yaml` and Thales specific `barbican-backend-pkcs11-thales.yaml` environment files, and any other templates needed for you deployment when running the `openstack overcloud deploy`
command:

```bash
$ openstack overcloud deploy \
  --timeout 100 \
  --templates /usr/share/openstack-tripleo-heat-templates \
  --stack overcloud \
  --libvirt-type kvm \
  --ntp-server clock.redhat.com \
  -e /home/stack/virt/config_lvm.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /home/stack/virt/network/network-environment.yaml \
  -e /home/stack/virt/hostnames.yaml \
  -e /home/stack/virt/nodes_data.yaml \
  -e /home/stack/virt/extra_templates.yaml \
  -e /home/stack/virt/container-parameters-with-barbican.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/barbican.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/barbican-backend-pkcs11-thales.yaml \
  -e /home/stack/configure-barbican.yaml \
  --log-file overcloud_deployment_with_atos.log
```

Verification

1. Create a test secret:

```bash
$ openstack secret store --name testSecret --payload 'TestPayload'
```

| Field         | Value                                                                 |
|---------------+-----------------------------------------------------------------------|
| Secret href   | https://192.168.123.163/key-manager/v1/secrets/4cc5ffe0-eea2-449d-9e64-b664d574be53 |
| Name          | testSecret                                                             |
| Created       | None                                                                   |
| Status        | None                                                                   |
| Content types | None                                                                   |
| Algorithm     | aes                                                                     |
| Bit length    | 256                                                                    |
| Secret type   | opaque                                                                 |
| Mode          | cbc                                                                    |
| Expiration    | None                                                                   |

2. Retrieve the payload for the secret that you just created:

```bash
openstack secret get https://192.168.123.163/key-manager/v1/secrets/4cc5ffe0-eea2-449d-9e64-b664d574be53 --payload
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payload</td>
<td>TestPayload</td>
</tr>
</tbody>
</table>

5.4. REVIEWING TLS ACTIVITY BETWEEN BARBICAN AND THE HSM
Barbican communicates with the HSM through the vendor-provided PKCS#11 library. For example, for an ATOS Proteccio HSM, you can configure the HSM client to communicate with the HSM using TLS by configuring the `proteccio.rc` file.

For the Atos HSM, the files containing the CA, server certificate, and key are located on the Controller, and are owned by the `barbican` user. Note that the `barbican` user does not exist on the Controller, and is the `barbican` user as defined in the Barbican container. As a result, file ownership is indicated through the `barbican` UID, which is 400; these files are then bind mounted by the Barbican container.

For the Entrust nShield Connect XC, to view additional logs on the pkcs#11 transactions between the HSM and the client software, add the following entries to `/opt/nfast/cknfastrc`:

```
CKNFAST_DEBUG=9
CKNFAST_DEBUGFILE=/tmp/hsm_log.txt
```

### 5.5. KEY STORAGE CONSIDERATIONS

The Barbican MKEK and HMAC keys are generated using Barbican utilities that communicate with the HSM using the vendor’s PKCS#11 library. Therefore the MKEK and HMAC keys are generated in the HSM and never leave the HSM.

In a director-based deployment, these utilities are executed within containers on the first Controller; the undercloud is never involved in this process.

### 5.6. ROTATING THE KEYS

You can rotate the MKEK and HMAC keys using a director update.

**NOTE**

The MKEK and HMAC have the same key type. This is a limitation in Barbican, and is currently expected to be addressed at a later time.

1. To rotate the keys, add the following parameter to your deployment environment files:

   ```
   BarbicanPkcs11CryptoRewrapKeys: true
   ```

2. Change the labels on the MKEK and HMAC keys. For example, if your labels are similar to these:

   ```
   BarbicanPkcs11CryptoMKEKLabel: 'barbican_mkek_10'
   BarbicanPkcs11CryptoHMACLabel: 'barbican_hmac_10'
   ```

   You can change the labels by incrementing the values:

   ```
   BarbicanPkcs11CryptoMKEKLabel: 'barbican_mkek_11'
   BarbicanPkcs11CryptoHMACLabel: 'barbican_hmac_11'
   ```

   **NOTE**

   Do not change the HMAC key type.

3. Re-deploy using director to apply the update. Director checks whether the keys that are labelled...
for the MKEK and HMAC exist, and then creates them. In addition, with the `BarbicanPkcs11CryptoRewrapKeys` parameter set to `True`, director calls `barbican-manage hsm pkek_rewrap` to rewrap all existing pKEKs.

### 5.7. PLANNING BACKUP FOR BARBICAN AND THE HSM

The section describes the components you will need to consider when planning your Barbican and HSM backup strategy.

- Barbican secrets - These are stored in the database, and must be backed up regularly.
- MKEK and HMAC keys - These are stored in the HSM. Check with your HSM vendor for recommended practices.
- HSM client certificates and keys - These are located on the Controller, and must be included in your Controller’s file backup procedure. Note that these files are sensitive credentials.
- Barbican configuration files
You can use barbican to manage your Block Storage (cinder) encryption keys. This configuration uses LUKS to encrypt the disks attached to your instances, including boot disks. Key management is transparent to the user; when you create a new volume using luks as the encryption type, cinder generates a symmetric key secret for the volume and stores it in barbican. When booting the instance (or attaching an encrypted volume), nova retrieves the key from barbican and stores the secret locally as a Libvirt secret on the Compute node.

**IMPORTANT**

Nova formats encrypted volumes during their first use if they are unencrypted. The resulting block device is then presented to the Compute node.

**NOTE**

If you intend to update any configuration files, be aware that certain OpenStack services now run within containers; this applies to keystone, nova, and cinder, among others. As a result, there are administration practices to consider:

- Do not update any configuration file you might find on the physical node’s host operating system, for example, `/etc/cinder/cinder.conf`. The containerized service does not reference this file.
- Do not update the configuration file running within the container. Changes are lost once you restart the container. Instead, if you must change containerized services, update the configuration file in `/var/lib/config-data/puppet-generated/`, which is used to generate the container.

For example:

- keystone: `/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf`
- cinder: `/var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf`
- nova: `/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf`

Changes are applied after you restart the container.

1. On nodes running the `cinder-volume` and `nova-compute` services, confirm that nova and cinder are both configured to use barbican for key management:

   ```bash
   $ crudini --get /var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf key_manager backend castellan.key_manager.barbican_key_manager.BarbicanKeyManager
   $ crudini --get /var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf key_manager backend castellan.key_manager.barbican_key_manager.BarbicanKeyManager
   ```

2. Create a volume template that uses encryption. When you create new volumes they can be modeled off the settings you define here:
3. Create a new volume and specify that it uses the **LuksEncryptor-Template-256** settings:

```bash
$ openstack volume create --size 1 --type LuksEncryptor-Template-256 'Encrypted-Test-Volume'
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachments</td>
<td>[]</td>
</tr>
<tr>
<td>availability_zone</td>
<td>nova</td>
</tr>
<tr>
<td>bootable</td>
<td>false</td>
</tr>
<tr>
<td>consistencygroup_id</td>
<td>None</td>
</tr>
<tr>
<td>created_at</td>
<td>2018-01-22T00:19:06.000000</td>
</tr>
<tr>
<td>description</td>
<td>None</td>
</tr>
<tr>
<td>encrypted</td>
<td>True</td>
</tr>
<tr>
<td>id</td>
<td>a361fd0b-882a-46cc-a669-c633630b5c93</td>
</tr>
<tr>
<td>migration_status</td>
<td>None</td>
</tr>
<tr>
<td>multattach</td>
<td>False</td>
</tr>
<tr>
<td>name</td>
<td>Encrypted-Test-Volume</td>
</tr>
<tr>
<td>properties</td>
<td></td>
</tr>
<tr>
<td>replication_status</td>
<td>None</td>
</tr>
<tr>
<td>size</td>
<td>1</td>
</tr>
<tr>
<td>snapshot_id</td>
<td>None</td>
</tr>
<tr>
<td>source_volid</td>
<td>None</td>
</tr>
<tr>
<td>status</td>
<td>creating</td>
</tr>
<tr>
<td>type</td>
<td>LuksEncryptor-Template-256</td>
</tr>
</tbody>
</table>

**NOTE**

Ensure that the user creating the encrypted volume has the **creator** barbican role on the project. For more information, see the **Grant user access to the creator role** section.
The resulting secret is automatically uploaded to the barbican backend.

4. Use barbican to confirm that the disk encryption key is present. In this example, the timestamp matches the LUKS volume creation time:

```
$ openstack secret list
```

5. Attach the new volume to an existing instance. For example:

```
$ openstack server add volume testInstance Encrypted-Test-Volume
```

The volume is then presented to the guest operating system and can be mounted using the built-in tools.

### 6.1. MIGRATE EXISTING VOLUME KEYS TO BARBICAN

Previously, deployments might have used **ConfKeyManager** to manage disk encryption keys. This meant that a fixed key was generated and then stored in the nova and cinder configuration files. The key IDs can be migrated to barbican using the following procedure. This utility works by scanning the databases for `encryption_key_id` entries within scope for migration to barbican. Each entry gets a new barbican key ID and the existing `ConfKeyManager` secret is retained.

**NOTE**

Previously, you could reassign ownership for volumes encrypted using **ConfKeyManager**. This is not possible for volumes that have their keys managed by barbican.

**NOTE**

Activating barbican will not break your existing `keymgr` volumes.

After it is enabled, the migration process runs automatically, but it requires some configuration, described in the next section. The actual migration runs in the **cinder-volume** and **cinder-backup** process, and you can track the progress in the cinder log files.

- **cinder-volume** - migrates keys stored in cinder’s Volumes and Snapshots tables.
6.1.1. Overview of the migration steps

1. Deploy the barbican service.

2. Add the creator role to the cinder service. For example:
   
   ```#openstack role create creator
   #openstack role add --user cinder creator --project service```

3. Restart the cinder-volume and cinder-backup services.

4. cinder-volume and cinder-backup automatically begin the migration process.

5. Monitor the logs for the message indicating migration has finished and check that no more volumes are using the ConfKeyManager all-zeros encryption key ID.

6. Remove the fixed_key option from cinder.conf and nova.conf. You must determine which nodes have this setting configured.

7. Remove the creator role from the cinder service.

6.1.2. Behavioral differences

Barbican-managed encrypted volumes behave differently than volumes that use ConfKeyManager:

- You cannot transfer ownership of encrypted volumes, because it is not currently possible to transfer ownership of the barbican secret.

- Barbican is more restrictive about who is allowed to read and delete secrets, which can affect some cinder volume operations. For example, a user cannot attach, detach, or delete a different user’s volumes.

6.1.3. Reviewing the migration process

This section describes how you can view the status of the migration tasks. After you start the process, one of these entries appears in the logs. This indicates whether the migration started correctly, or it identifies the issue it encountered:

- Not migrating encryption keys because the ConfKeyManager is still in use.

- Not migrating encryption keys because the ConfKeyManager’s fixed_key is not in use.

- Not migrating encryption keys because migration to the ‘XXX’ key_manager backend is not supported. - This message is unlikely to appear; it is a safety check to handle the code ever encountering another Key Manager backend other than barbican. This is because the code only supports one migration scenario: From ConfKeyManager to barbican.

- Not migrating encryption keys because there are no volumes associated with this host. - This may occur when cinder-volume is running on multiple hosts, and a particular host has no volumes associated with it. This arises because every host is responsible for handling its own volumes.

- Starting migration of ConfKeyManager keys.
Migrating volume <UUID> encryption key to Barbican - During migration, all of the host’s volumes are examined, and if a volume is still using the ConfKeyManager’s key ID (identified by the fact that it’s all zeros (00000000-0000-0000-0000-000000000000)), then this message appears.

- For cinder-backup, this message uses slightly different capitalization: Migrating Volume [...] or Migrating Backup [...]

After each host examines all of its volumes, the host displays a summary status message:

`No volumes are using the ConfKeyManager’s encryption_key_id.`

`No backups are known to be using the ConfKeyManager’s encryption_key_id.``

You may also see the following entries:

- There are still %d volume(s) using the ConfKeyManager’s all-zeros encryption key ID.
- There are still %d backup(s) using the ConfKeyManager’s all-zeros encryption key ID.

Note that both of these messages can appear in the cinder-volume and cinder-backup logs. Whereas each service only handles the migration of its own entries, the service is aware of the the other’s status. As a result, cinder-volume knows if cinder-backup still has backups to migrate, and cinder-backup knows if the cinder-volume service has volumes to migrate.

Although each host migrates only its own volumes, the summary message is based on a global assessment of whether any volume still requires migration. This allows you to confirm that migration for all volumes is complete. Once you receive confirmation, remove the fixed_key setting from cinder.conf and nova.conf. See the Clean up the fixed keys section below for more information.

6.1.4. Troubleshooting the migration process

6.1.4.1. Role assignment

The barbican secret can only be created when the requestor has the creator role. This means that the cinder service itself requires the creator role, otherwise a log sequence similar to this will occur:

1. Starting migration of ConfKeyManager keys.
2. Migrating volume <UUID> encryption key to Barbican
3. Error migrating encryption key: Forbidden: Secret creation attempt not allowed - please review your user/project privileges
4. There are still %d volume(s) using the ConfKeyManager’s all-zeros encryption key ID.

The key message is the third one: Secret creation attempt not allowed. To fix the problem, update the cinder account’s privileges:

1. Run openstack role add --project service --user cinder creator
2. Restart the cinder-volume and cinder-backup services.

As a result, the next attempt at migration should succeed.

6.1.5. Clean up the fixed keys
The encryption_key_id was only recently added to the Backup table, as part of the Queens release. As a result, pre-existing backups of encrypted volumes are likely to exist. The all-zeros encryption_key_id is stored on the backup itself, but it won’t appear in the Backup database. As such, it is impossible for the migration process to know for certain whether a backup of an encrypted volume exists that still relies on the all-zeros ConfKeyMgr key ID.

After migrating your key IDs into barbican, the fixed key remains in the configuration files. This may present a security concern to some users, because the fixed_key value is not encrypted in the .conf files. To address this, you can manually remove the fixed_key values from your nova and cinder configurations. However, first complete testing and review the output of the log file before you proceed, because disks that are still dependent on this value will not be accessible.

1. Review the existing fixed_key values. The values must match for both services.

   crudini --get /var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf keymgr fixed_key
   crudini --get /var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf keymgr fixed_key

   IMPORTANT

   Make a backup of the existing fixed_key values. This allows you to restore the value if something goes wrong, or if you need to restore a backup that uses the old encryption key.

2. Delete the fixed_key values:

   crudini --del /var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf keymgr fixed_key
   crudini --del /var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf keymgr fixed_key

6.2. AUTOMATIC DELETION OF VOLUME IMAGE ENCRYPTION KEY

The Block Storage service (cinder) creates an encryption key in the Key Management service (barbican) when it uploads an encrypted volume to the Image service (glance). This creates a 1:1 relationship between an encryption key and a stored image.

Encryption key deletion prevents unlimited resource consumption of the Key Management service. The Block Storage, Key Management, and Image services automatically manage the key for an encrypted volume, including the deletion of the key.

The Block Storage service automatically adds two properties to a volume image:

- **cinder_encryption_key_id** - The identifier of the encryption key that the Key Management service stores for a specific image.

- **cinder_encryption_key_deletion_policy** - The policy that tells the Image service to tell the Key Management service whether to delete the key associated with this image.
IMPORTANT

The values of these properties are automatically assigned. To avoid unintentional data loss, do not adjust these values.

When you create a volume image, the Block Storage service sets the `cinder_encryption_key_deletion_policy` property to `on_image_deletion`. When you delete a volume image, the Image service deletes the corresponding encryption key if the `cinder_encryption_key_deletion_policy` equals `on_image_deletion`.

IMPORTANT

Red Hat does not recommend manual manipulation of the `cinder_encryption_key_id` or `cinder_encryption_key_deletion_policy` properties. If you use the encryption key that is identified by the value of `cinder_encryption_key_id` for any other purpose, you risk data loss.
CHAPTER 7. ENCRYPT AT-REST SWIFT OBJECTS

By default, objects uploaded to Object Storage are stored unencrypted. Because of this, it is possible to access objects directly from the file system. This can present a security risk if disks are not properly erased before they are discarded. When you have barbican enabled, the Object Storage service (swift) can transparently encrypt and decrypt your stored (at-rest) objects. At-rest encryption is distinct from in-transit encryption in that it refers to the objects being encrypted while being stored on disk.

Swift performs these encryption tasks transparently, with the objects being automatically encrypted when uploaded to swift, then automatically decrypted when served to a user. This encryption and decryption is done using the same (symmetric) key, which is stored in barbican.

NOTE

You cannot disable encryption after you have enabled encryption and added data to the swift cluster, because the data is now stored in an encrypted state. Consequently, the data will not be readable if encryption is disabled, until you re-enable encryption with the same key.

7.1. ENABLE AT-REST ENCRYPTION FOR SWIFT

1. You can enable the swift encryption capabilities by including **SwiftEncryptionEnabled: True** in your environment file, then re-running `openstack overcloud deploy` using `./home/stack/overcloud_deploy.sh`. Note that you still need to enable barbican, as described in the Install Barbican chapter.

2. Confirm that swift is configured to use at-rest encryption:

   ```bash
   $ crudini --get /var/lib/config-data/puppet-generated/swift/etc/swift/proxy-server.conf pipeline-main pipeline
   
   pipeline = catch_errors healthcheck proxy-logging cache ratelimit bulk tempurl formpost authtoken keystone staticweb copy container_quotas account_quotas slo dlo versioned_writes kms_keymaster encryption proxy-logging proxy-server
   
   The result should include an entry for encryption.
CHAPTER 8. VALIDATE GLANCE IMAGES

After enabling Barbican, you can configure the Image Service (glance) to verify that an uploaded image has not been tampered with. In this implementation, the image is first signed with a key that is stored in barbican. The image is then uploaded to glance, along with the accompanying signing information. As a result, the image’s signature is verified before each use, with the instance build process failing if the signature does not match.

Barbican’s integration with glance means that you can use the openssl command with your private key to sign glance images before uploading them.

8.1. ENABLE GLANCE IMAGE VALIDATION

In your environment file, enable image verification with the `VerifyGlanceSignatures: True` setting. You must re-run the `openstack overcloud deploy` command for this setting to take effect.

To verify that glance image validation is enabled, run the following command on an overcloud Compute node:

```
$ sudo crudini --get /var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf glance verify_glance_signatures
```

NOTE
If you use Ceph as the back end for the Image and Compute services, a CoW clone is created. Therefore, Image signing verification cannot be performed.

8.2. VALIDATE AN IMAGE

To configure a glance image for validation, complete the following steps:

1. Confirm that glance is configured to use barbican:

   ```
   $ sudo crudini --get /var/lib/config-data/puppet-generated/glance_api/etc/glance/glance-api.conf key_manager backend castellan.key_manager.barbican_key_manager.BarbicanKeyManager
   ```

2. Generate a private key and convert it to the required format:

   ```
   openssl genrsa -out private_key.pem 1024
   openssl rsa -pubout -in private_key.pem -out public_key.pem
   openssl req -new -key private_key.pem -out cert_request.csr
   openssl x509 -req -days 14 -in cert_request.csr -signkey private_key.pem -out x509_signing_cert.crt
   ```

3. Add the key to the barbican secret store:

   ```
   $ source ~/overcloudrc
   $ openstack secret store --name signing-cert --algorithm RSA --secret-type certificate --payload-content-type "application/octet-stream" --payload-content-encoding base64 --payload "$\$(base64 x509_signing_cert.crt)" -c 'Secret href' -i value https://192.168.123.170:9311/v1/secrets/5df14c2b-f221-4a02-948e-48a61edd3f5b
   ```
NOTE

Record the resulting UUID for use in a later step. In this example, the certificate’s UUID is 5df14c2b-f221-4a02-948e-48a61edd3f5b.

4. Use `private_key.pem` to sign the image and generate the `.signature` file. For example:

   ```
   $ openssl dgst -sha256 -sign private_key.pem -sigopt rsa_padding_mode:pss -out cirros-0.4.0.signature cirros-0.4.0-x86_64-disk.img
   ```

5. Convert the resulting `.signature` file into base64 format:

   ```
   $ base64 -w 0 cirros-0.4.0.signature > cirros-0.4.0.signature.b64
   ```

6. Load the base64 value into a variable to use it in the subsequent command:

   ```
   $ cirros_signature_b64=$(cat cirros-0.4.0.signature.b64)
   ```

7. Upload the signed image to glance. For `img_signature_certificate_uuid`, you must specify the UUID of the signing key you previously uploaded to barbican:

   ```
   openstack image create \
   --container-format bare \n   --disk-format qcow2 \
   --property img_signature="$cirros_signature_b64" \
   --property img_signature_certificate_uuid="5df14c2b-f221-4a02-948e-48a61edd3f5b" \n   --property img_signature_hash_method="SHA-256" \n   --property img_signature_key_type="RSA-PSS" cirros_0_4_0_signed \n   --file cirros-0.4.0-x86_64-disk.img
   ```

   +--------------------------------+------------------------------------------------------------------------------+
   | Property                       | Value                                                                            |
   +--------------------------------+------------------------------------------------------------------------------+
   | checksum                       | None                                                                             |
   | container_format               | bare                                                                             |
   | created_at                     | 2018-01-23T05:37:31Z                                                           |
   | disk_format                    | qcow2                                                                            |
   | id                             | d3396fa0-2ea2-4832-8a77-d36fa3f2ab27                                           |
   | img_signature                  | lc17nGgoKxnCyOcsJ4abbEZEpzXByFPlgiPeiT+O7jz0yvW00KN3fl0AA6tn9EXrp7fb2xbDE4UaO3v |
   | img_signature_certificate_uuid | ba3641c2-6a3d-445a-8543-85168110eab                                            |
   | img_signature_hash_method      | SHA-256                                                                          |
   | img_signature_key_type         | RSA-PSS                                                                          |
   | min_disk                       | 0                                                                                |
   | min_ram                        | 0                                                                                |
   | name                           | cirros_0_4_0_signed                                                             |
   | owner                          | 9f812310d904e6ea01e1bacb84c9f1a                                                 |

Red Hat OpenStack Platform 16.1 Manage Secrets with OpenStack Key Manager
### CHAPTER 8. VALIDATE GLANCE IMAGES

| protected | False |
| size      | None  |
| status    | queued |
| tags      | []    |
| updated_at | 2018-01-23T05:37:31Z |
| virtual_size | None    |
| visibility | shared |

8. You can view glance’s image validation activities in the Compute log: `/var/log/containers/nova/nova-compute.log`. For example, you can expect the following entry when the instance is booted:

```
2018-05-24 12:48:35.256 1 INFO nova.image.glance [req-7c271904-4975-4771-9d26-cbea6c0ade31 b464b2fd2a2140e9a88bbdacf67bdd8c a3db2f2beaee454182c95b646fa7331f - default default] Image signature verification succeeded for image d3396fa0-2ea2-4832-8a77-d36fa3f2ab27
```
CHAPTER 9. VALIDATE IMAGES USED FOR VOLUME CREATION

The Block Storage Service (cinder) automatically validates the signature of any downloaded, signed image during volume from image creation. The signature is validated before the image is written to the volume.

To improve performance, you can use the Block Storage Image-Volume cache to store validated images for creating new volumes. For more information, see Configure and Enable the Image-Volume Cache in the Storage Guide.

NOTE

Cinder image signature validation does not work with Red Hat Ceph Storage or RBD volumes.

9.1. VALIDATE THE IMAGE SIGNATURE ON A NEW VOLUME

This procedure demonstrates how you can use validate a volume signature created from a signed image.

1. Log in to a controller node.

   For example, you can expect the following entry when the instance is booted:

```
2018-05-24 12:48:35.256 1 INFO cinder.image.image_utils [req-7c271904-4975-4771-9d26-cbea6c0ade31 b464b2fd2a2140e9a88bbdacf67bdd8c a3db2f2beaee454182c95b646fa7331f - default default] Image signature verification succeeded for image d3396fa0-2ea2-4832-8a77-d36fa3f2ab27
```

Alternatively, you can use the openstack volume list and cinder volume show commands.

1. Use the openstack volume list command to locate the volume ID.

2. Run the cinder volume show command on a compute node:

   ```
cinder volume show <VOLUME_ID>
   ```

3. Locate the volume_image_metadata section with the line signature verified : True.

   ```
   $ cinder show d0db26bb-449d-4111-a59a-6fbb080bb483
   +--------------------------------+-------------------------------------------------+
   | Property                       | Value                                           |
   +--------------------------------+-------------------------------------------------+
   | attached_servers               | []                                              |
   | attachment_ids                 | []                                              |
   | availability_zone              | nova                                            |
   | bootable                       | true                                            |
   | consistencygroup_id            | None                                            |
   | created_at                     | 2018-10-12T19:04:41.000000                      |
   | description                    | None                                            |
   | encrypted                      | True                                            |
   ```
| id | d0db26bb-449d-4111-a59a-6bb080bb483 |
| metadata |
| migration_status | None |
| multiattach | False |
| name | None |
| os-vol-host-attr:host | centstack.localdomain@nfs#nfs |
| os-vol-mig-status-attr:migstat | None |
| os-vol-mig-status-attr:name_id | None |
| os-vol-tenant-attr:tenant_id | 1a081dd2505547f5a8bb1a230f2295f4 |
| replication_status | None |
| size | 1 |
| snapshot_id | None |
| source_volid | None |
| status | available |
| updated_at | 2018-10-12T19:05:13.000000 |
| user_id | ad9fe430b3a6416f908c79e4de3bfa98 |
| volume_image_metadata | checksum : f8ab98ff5e73ebab884d80c9dc9c7290 |
| | container_format : bare |
| | disk_format : qcow2 |
| | image_id : 154d4d4b-12bf-41dc-b7c4-35e5a6a3482a |
| | image_name : cirros-0.3.5-x86_64-disk |
| | min_disk : 0 |
| | min_ram : 0 |
| | signature_verified : False |
| | size : 13267968 |
| volume_type | nfs |