Red Hat OpenStack Platform 17.1

使用密钥管理器服务管理 secret

将密钥管理器服务(barbican)与您的 OpenStack 部署集成。

Last Updated: 2024-05-23

Red Hat OpenStack Platform 17.1 使用密钥管理器服务管理 secret

将密钥管理器服务(barbican)与您的 OpenStack 部署集成。

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

如何将 OpenStack Key Manager (barbican)与您的 OpenStack 部署集成。

目录

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

对红帽文档提供反馈

我们感谢您对文档提供反馈信息。与我们分享您的成功秘诀。

在 JIRA 中提供文档反馈

使用 Create Issue 表单对文档提供反馈。JIRA 问题将在 Red Hat OpenStack Platform Jira 项目中创建，您可以在其中跟踪您的反馈进度。

1. 确保您已登录到 JIRA。如果您没有 JIRA 帐户，请创建一个帐户来提交反馈。
2. 点击以下链接打开 **Create Issue** 页面： Create Issue
3. 完成 **Summary** 和 **Description** 字段。在 **Description** 字段中，包含文档 URL、章节或章节号以及问题的详细描述。不要修改表单中的任何其他字段。
4. 点 **Create**。

第 1 章 部署和配置 OPENSTACK KEY MANAGER (BARBICAN)

OpenStack Key Manager (barbican)是 Red Hat OpenStack Platform 的 secret 管理器。您可以使用 barbican API 和命令行来集中管理 OpenStack 服务使用的证书、密钥和密码。默认情况下，Red Hat OpenStack Platform 中不启用 barbican。您可以在现有的 OpenStack 部署中部署 barbican。

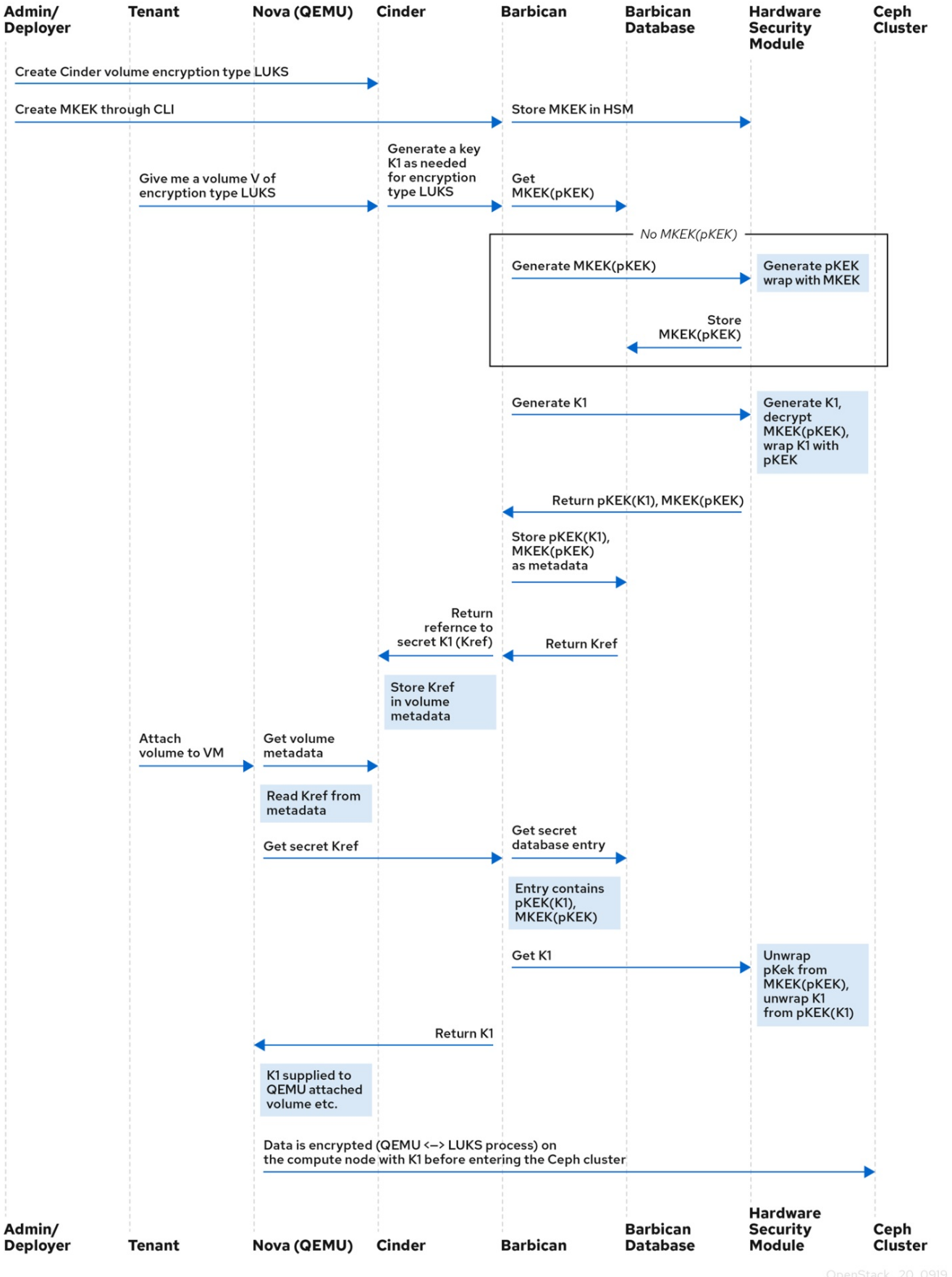
barbican 目前支持以下用例，如本指南所述：

- **对称加密密钥** - 用于 Block Storage (cinder)卷加密、临时磁盘加密和 Object Storage (swift)加密。
- **非对称密钥和证书** - 用于 glance 镜像签名和验证。

OpenStack Key Manager 与 Block Storage (cinder)、网络(neutron)和计算(nova)组件集成。

1.1. OPENSTACK KEY MANAGER 工作流

下图显示了 OpenStack Key Manager 用来管理环境的 secret 的工作流。



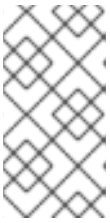
1.2. OPENSTACK KEY MANAGER 加密类型

证书、API 密钥和密码等机密可以存储在 barbican 数据库中加密的 blob 中，或者直接存储在安全存储系统中。您可以使用简单的加密插件或 PKCS#11 crypto 加密插件来加密 secret。

要将 secret 作为一个加密的 blob 存储在 barbican 数据库中，可以使用以下选项：

- **简单加密插件** - 默认启用简单加密插件，并使用单个对称密钥加密所有 secret 有效负载。此密钥以纯文本形式存储在 **barbican.conf** 文件中，因此防止未经授权的访问此文件非常重要。
- **PKCS#11 crypto plugin 插件** - PKCSBACKEND 加密插件使用特定于项目的密钥加密密钥(pKEK)加密 secret，该 secret 存储在 barbican 数据库中。特定于项目的 pKEK 由主密钥加密(MKEK)加密，该密钥存储在硬件安全模块(HSM)中。所有加密和解密操作都会在 HSM 中进行，而不是在进程内存中。PKCS#11 插件通过 PKCS#11 API 与 HSM 通信。由于加密是在安全硬件中完成的，并且每个项目使用不同的 pKEK，所以这个选项比简单的加密插件更安全。
  红帽支持带有以下 HSM 中的任何一个 PKCS#11 后端。

设备	发行版本支持	高可用性(HA)支持
ATOS Trustway Proteccio NetHSM	16.0+	16.1+
Entrust nShield Connect HSM	16.0+	不支持
Thales Luna Network HSM	16.1+（技术预览）	16.1+（技术预览）



注意

关于高可用性(HA)选项：barbican 服务在 Apache 中运行，并由 director 配置为使用 HAProxy 来实现高可用性。后端层的 HA 选项将取决于所使用的后端。例如，对于简单加密，所有 barbican 实例在配置文件中都有相同的加密密钥，从而导致简单的 HA 配置。

1.2.1. 配置多个加密机制

您可以将 Barbican 的一个实例配置为使用多个后端。完成后，您必须将后端指定为 **全局默认** 后端。您还可以指定每个项目的默认后端。如果项目不存在映射，则该项目的 secret 将存储为使用全局默认后端。

例如，您可以将 Barbican 配置为使用简单加密和 PKCS#11 插件。如果将 Simple crypto 设为全局默认值，则所有项目都将使用该后端。然后，您可以通过将 PKCS#11 设置为该项目的首选后端来指定哪些项目使用 PKCS#11 后端。

如果您决定迁移到新后端，您可以在启用新后端作为全局默认值或作为特定于项目的后端时保持原始可用。因此，旧的 secret 可以通过旧后端保持可用，新的 secret 则存储在新的全局默认后端中。

1.3. 部署密钥管理器

要部署 OpenStack Key Manager，首先为 barbican 服务创建一个环境文件，并使用额外的环境文件重新部署 overcloud。然后，您可以将用户添加到 **creator** 角色中，以创建和编辑 barbican secret，或者创建在 barbican 中保存 secret 的加密的卷。

注意

此流程将 barbican 配置为使用 **simple_crypto** 后端。提供了其他后端，如 **PKCS#11** 需要不同的配置，以及不同的 heat 模板文件，具体取决于使用 HSM。不支持 KMIP、Hashicorp Vault 和 DogTag 等其他后端。

前提条件

- overcloud 已部署和运行

流程

1. 在 undercloud 节点上，为 barbican 创建一个环境文件。

```
$ cat /home/stack/templates/configure-barbican.yaml
 parameter_defaults:
   BarbicanSimpleCryptoGlobalDefault: true
```

BarbicanSimpleCryptoGlobalDefault 将此插件设置为全局默认插件。

您还可以在环境文件中添加以下选项：

- **BarbicanPassword** - 为 barbican 服务帐户设置密码。
- **BarbicanWorkers** - 设置 **barbican::wsgi::apache** 的 worker 数量。默认使用 **'%{::processorcount}'**。
- **BarbicanDebug** - 启用调试。
- **BarbicanPolicies** - 定义要为 barbican 配置的策略。使用哈希值，例如： **{ barbican-context_is_admin: { key: context_is_admin, value: 'role:admin' } }**.然后，此条目被添加到 **/etc/barbican/policy.json** 中。后续小节中将详细介绍策略。
- **BarbicanSimpleCryptoKek** - 如果没有指定密钥加密密钥(KEK)，则由 director 生成密钥加密密钥(KEK)。

2. 在 **openstack overcloud deploy** 命令中添加以下文件，而不从脚本中删除之前添加的角色、模板或环境文件：
   - /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. 重新运行部署脚本，将更改应用到您的部署：

```
$ 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/containers-prepare-parameter.yaml \
   -e /home/stack/templates/config_lvm.yaml \
```

```
    -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
    -e /home/stack/templates/network/network-environment.yaml \
    -e /home/stack/templates/hostnames.yml \
    -e /home/stack/templates/nodes_data.yaml \
    -e /home/stack/templates/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-simple-
crypto.yaml \
    -e /home/stack/templates/configure-barbican.yaml \
    --log-file overcloud_deployment_38.log
```

4. 检索 **creator** 角色的 **id** ：

```
openstack role show creator
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | 4e9c560c6f104608948450fbf316f9d7 |
| name      | creator                          |
+-----------+----------------------------------+
```

注意

除非安装了 OpenStack Key Manager (barbican)，否则您不会看到 **创建者** 角色。

5. 将用户分配到 **创建者** 角色并指定相关项目。在本例中，**project_a** 项目中名为 **user1** 的用户添加到 **creator** 角色中：

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

验证

1. 创建 test secret。例如：

```
$ 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. 检索您刚才创建的 secret 的有效负载：

```
openstack secret get https://192.168.123.163/key-manager/v1/secrets/4cc5ffe0-eea2-449d-9e64-b664d574be53 --payload
+---------+-------------+
| Field   | Value       |
+---------+-------------+
| Payload | TestPayload |
+---------+-------------+
```

1.4. 查看密钥管理器策略

barbican 使用策略来确定允许哪些用户对 secret 执行操作，如添加或删除密钥。为实施这些控制，之前创建的 **创建者** 等 keystone 项目角色映射到 barbican 内部权限。因此，分配给这些项目角色的用户会收到对应的 barbican 权限。

默认策略在代码中定义，通常不需要任何修改。如果没有进行任何更改，您可以使用环境中的现有容器查看默认策略。如果对默认策略进行了更改，并且您希望查看默认值，请首先使用单独的系统拉取 **openstack-barbican-api** 容器。

前提条件

- OpenStack Key Manager 已部署并运行

流程

1. 使用您的红帽凭证登录到 podman：

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

2. 拉取 **openstack-barbican-api** 容器：

```
podman pull \
registry.redhat.io/rhosp-rhel8/openstack-barbican-api:17.1
```

3. 在当前工作目录中生成策略文件：

```
podman run -it \
registry.redhat.io/rhosp-rhel8/openstack-barbican-api:17.1 \
oslopolicy-policy-generator \
--namespace barbican > barbican-policy.yaml
```

验证

查看 **barbican-policy.yaml** 文件，以检查 barbican 使用的策略。该策略由四个不同的角色实施，它们定义用户如何与 secret 和 secret 元数据进行交互。用户通过分配给特定角色来接收这些权限：

admin

admin 角色可以在所有项目间读取、创建、编辑和删除 secret。

创建者

creator 角色可以读取、创建、编辑和删除位于创建者范围的项目中的 secret。

Observer

observer 角色只能读取 secret。

audit

audit 角色只能读取元数据。audit 角色无法读取 secret。

例如，下列条目列出了各个项目的 **admin**、**观察** 器和 **创建者** Keystone 角色：请注意，请注意，它们被分配了 **role:admin**、**role:observer** 和 **role:creator** 权限：

```
#
#"admin": "role:admin"

#
#"observer": "role:observer"

#
#"creator": "role:creator"
```

这些角色也可以通过 barbican 分组在一起。例如，指定 **admin_or_creator** 的规则可应用到 **rule:admin** 或 **rule:creator** 的成员。

在文件的下面，有 **secret:put** 和 **secret:delete** 操作。在其右侧，注意哪些角色具有执行这些操作的权限。在以下示例中，**secret:delete** 表示只有 **admin** 和 **creator** 角色成员才能删除 secret 条目。此外，该规则指出该项目的 **admin** 或 **creator** 角色中的用户可以删除该项目中的机密。项目匹配由 **secret_project_match** 规则定义，该规则也会在策略中定义。

```
secret:delete": "rule:admin_or_creator and rule:secret_project_match"
```

第 2 章 使用 OPENSTACK KEY MANAGER (BARBICAN)管理 SECRET 和密钥。

您可以使用 OpenStack Key Manager 创建、更新和删除 secret 和加密密钥。您还可以备份和恢复加密密钥和 barbican 数据库。建议您定期备份加密密钥和 barbican 数据库。

2.1. 查看 SECRET

要查看 secret 列表，请运行 **openstack secret list** 命令。列表中包括 URI、名称、类型和其他有关 secret 的信息。

流程

- 查看 secret 列表：

```
$ openstack secret list
+------------------------------------------------------------------------------+------+------------------
---------+--------+-----------------------------------------+----------+------------+-------------+------
+------------+
| Secret href                                                                  | Name | Created                  | Status
| Content types                        | Algorithm | Bit length | Secret type | Mode | Expiration |
+------------------------------------------------------------------------------+------+------------------
---------+--------+-----------------------------------------+----------+------------+-------------+------
+------------+
| https://192.168.123.169:9311/v1/secrets/24845e6d-64a5-4071-ba99-0fdd1046172e | None |
2018-01-22T02:23:15+00:00 | ACTIVE | {u'default': u'application/octet-stream'} | aes       |
256 | symmetric   | None | None       |
+------------------------------------------------------------------------------+------+------------------
---------+--------+-----------------------------------------+----------+------------+-------------+------
+------------+
```

2.2. 创建 SECRET

要创建 secret，请运行 **openstack secret store** 命令，并指定 secret 的名称以及 secret 的 payload （可选）。

流程

- 创建 secret。例如：

```
$ openstack secret store --name testSecret --payload 'TestPayload'
+---------------+------------------------------------------------------------------------------------+
| Field         | Value                                                                              |
+---------------+------------------------------------------------------------------------------------+
| Secret href   | https://192.168.123.163:9311/v1/secrets/ecc7b2a4-f0b0-47ba-b451-
0f7d42bc1746 |
| Name          | testSecret                                                                           |
| Created       | None                                                                               |
| Status        | None                                                                               |
| Content types | None                                                                                 |
| Algorithm     | aes                                                                                |
| Bit length    | 256                                                                                |
| Secret type   | opaque                                                                               |
```

```
| Mode          | cbc                                                                                  |
| Expiration    | None                                                                                 |
+---------------+--------------------------------------------------------------------------------------+
```

2.3. 在 SECRET 中添加有效负载

您无法更改 secret 的有效负载（除删除 secret 外），但是如果您在没有指定有效负载的情况下创建了 secret，则稍后您可以使用 **openstack secret update** 命令向其添加有效负载。

流程

- 在 secret 中添加有效负载：

```
$ openstack secret update https://192.168.123.163:9311/v1/secrets/ca34a264-fd09-44a1-8856-c6e7116c3b16 'TestPayload-updated'
$
```

2.4. 删除 SECRET

要删除 secret，请运行 **openstack secret delete** 命令并指定 secret URI。

流程

- 使用指定的 URI 删除 secret：

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

2.5. 生成对称密钥

要生成对称密钥，请使用 **order create** 命令，然后将密钥存储在 barbican 中。然后，您可以将对称密钥用于某些任务，如 nova 磁盘加密和 swift 对象加密。

前提条件

- OpenStack Key Manager 已安装并运行

流程

1. 使用 **顺序创建** 并将其保存在 barbican 中，生成 256 位密钥。例如：

```
$ 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                                                                                  |
+---------------+---------------------------------------------------------------------------------------+
```

您还可以使用 **--mode** 选项将生成的密钥配置为使用特定模式，如 **ctr** 或 **cbc**。如需更多信息，请参阅 *NIST SP 800-38A*。

2. 查看识别所生成密钥位置的顺序详情，如 **Secret href** 值所示：

```
$ 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                                                                                  |
| Secret href   | https://192.168.123.173:9311/v1/secrets/efcfec49-b9a3-4425-a9b6-5ba69cb18719 |
| Created       | 2018-01-24T04:24:33+00:00                                                             |
| Status        | ACTIVE                                                                                |
| Error code    | None                                                                                  |
| Error message | None                                                                                  |
+---------------+---------------------------------------------------------------------------------------+
```

3. 检索 secret 的详情：

```
$ openstack secret get https://192.168.123.173:9311/v1/secrets/efcfec49-b9a3-4425-a9b6-5ba69cb18719
+---------------+---------------------------------------------------------------------------------------+
| Field         | Value                                                                                 |
+---------------+---------------------------------------------------------------------------------------+
| Secret href   | https://192.168.123.173:9311/v1/secrets/efcfec49-b9a3-4425-a9b6-5ba69cb18719 |
| Name          | swift_key                                                                             |
| Created       | 2018-01-24T04:24:33+00:00                                                             |
| Status        | ACTIVE                                                                                |
| Content types | {u'default': u'application/octet-stream'}                                              |
| Algorithm     | aes                                                                                   |
| Bit length    | 256                                                                                   |
| Secret type   | symmetric                                                                             |
| Mode          | ctr                                                                                   |
| Expiration    | None                                                                                  |
+---------------+---------------------------------------------------------------------------------------+
```

2.6. 备份简单的加密加密密钥

要备份简单的加密加密密钥，请将包含主 KEK 的 **barbican.conf** 文件备份到安全强化的位置，然后备份 barbican 数据库。

重要

该流程包括生成测试 secret 和密钥的步骤。如果您已经为 secret 生成了一个密钥，请跳过测试密钥步骤并使用您生成的密钥。

前提条件

- OpenStack Key Manager 已安装并运行
- 您有一个安全强化的位置用于 KEK 备份

流程

1. 在 overcloud 上，生成新的 256 位密钥并将其存储在 barbican 中：

```
(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
+----------------+------------------------------------------------------------------+
| Field          | Value                                                            |
+----------------+------------------------------------------------------------------+
| Order href     | http://10.0.0.104:9311/v1/orders/2a11584d-851c-4bc2-83b7-35d04d3bae86 |
| Type           | Key                                                              |
| Container href | N/A                                                              |
| Secret href    | None                                                             |
| Created        | None                                                             |
| Status         | None                                                             |
| Error code     | None                                                             |
| Error message  | None                                                             |
+----------------+------------------------------------------------------------------+
```

2. 创建测试 secret：

```
(overcloud) [stack@undercloud-0 ~]$ openstack secret store --name testSecret --payload
'TestPayload'
+---------------+------------------------------------------------------------------+
| Field         | Value                                                            |
+---------------+------------------------------------------------------------------+
| Secret href   | http://10.0.0.104:9311/v1/secrets/93f62cfd-e008-401f-be74-bf057c88b04a |
| Name          | testSecret                                                       |
| Created       | None                                                             |
| Status        | None                                                             |
| Content types | None                                                             |
| Algorithm     | aes                                                              |
| Bit length    | 256                                                              |
| Secret type   | opaque                                                           |
| Mode          | cbc                                                              |
| Expiration    | None                                                             |
+---------------+------------------------------------------------------------------+
```

3. 确认创建了 test secret：

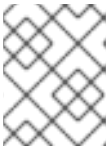
```
(overcloud) [stack@undercloud-0 ~]$ openstack secret list
+------------------------------------------------------------------+-----------+-----------------------
---+--------+-----------------------------------------+----------+-----------+------------+------+------
------+
```

```
| 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-5221-47e5-9887-608972a5fefb | swift_key  |
2018-06-19T18:24:40+00:00 | ACTIVE | {u'default': u'application/octet-stream'} | aes       |
256 | symmetric   | ctr  | None       |
+----------------------------------------------------------------------------+------------+-----------------------
---+--------+-----------------------------------------+-----------+------------+-------------+------+------
------+
```

4. 将包含主 KEK 的 **barbican.conf** 文件复制到安全强化的位置。

5. 登录到 **controller-0** 节点并检索 *barbican* 用户密码：

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



注意

只有用户 *barbican* 有权访问 *barbican* 数据库。因此，需要 barbican 用户密码来备份或恢复数据库。

6. 备份 *barbican* 数据库：

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

7. 检查数据库备份是否存储在 **/home/tripleo-admin** 中：

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

8. 在 overcloud 上，删除之前创建的 secret，并验证它们是否不再存在：

```
(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 ~]$
```

2.7. 从备份中恢复简单的加密加密密钥

要从备份中恢复 barbican 数据库，请使用 barbican 权限登录到 Controller 节点，并恢复 barbican 数据库。要从备份中恢复 KEK，请使用备份文件覆盖 **barbican.conf** 文件。

前提条件

- OpenStack Key Manager 已安装并运行
- 您有 **barbican.conf** 文件和 barbican 数据库的现有备份

流程

1. 登录到 **controller-0** 节点，检查控制器上是否有 **barbican** 数据库，其授予对 **barbican** 用户的访问权限，以恢复数据库：

```
[tripleo-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
[tripleo-admin@controller-0 ~]$
```

2. 将备份文件恢复到 **barbican** 数据库：

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

3. 使用之前备份的文件覆盖 **barbican.conf** 文件。

验证

- 在 overcloud 上，验证测试 secret 是否已成功恢复：

```
(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-5221-47e5-9887-608972a5fefb | swift_key  |
2018-06-19T18:24:40+00:00 | ACTIVE | {u'default': u'application/octet-stream'} | aes       |
256 | symmetric   | ctr  | None        |
+------------------------------------------------------------------------+-----------+-----------------------
---+--------+----------------------------------------+----------+-----------+-------------+------+------
------+
(overcloud) [stack@undercloud-0 ~]$
```

第 3 章 将 OPENSTACK KEY MANAGER (BARBICAN)与硬件安全模块(HSM)设备集成

将您的 Red Hat OpenStack Platform 部署与硬件安全模块(HSM)设备集成，以使用基于硬件的加密处理来提高安全状况。当您计划 OpenStack Key Manager 与 HSM 设备集成时，您必须选择一个受支持的加密类型和 HSM 设备，设置常规备份，并查看可能影响部署的任何其他信息或限制。

3.1. 将 OPENSTACK KEY MANAGER (BARBICAN)与 ATOS HSM 集成

要将 PKCS#11 后端与您的 Trustway Proteccio Net HSM 设备集成，请使用参数创建一个配置文件，以使用 HSM 连接 barbican。您可以通过在 **atos_hsms** 参数下列出两个或多个 HSM 来启用 HA。

规划

默认情况下，HSM 最多可以有 32 个并发连接。如果您超过这个数字，您可能会遇到 PKCS#11 客户端中的内存错误。您可以计算连接数量，如下所示：

- 每个控制器都有一个 **barbican-api** 和一个 **barbican-worker** 进程。
- 每个 Barbican API 进程通过 **N** Apache worker 执行 - （其中 **N** 默认为 CPU 的数量）。
- 每个 worker 都有一个到 HSM 的连接。

每个 **barbican-worker** 进程都有一个与数据库的连接。您可以使用 **BarbicanWorkers** heat 参数定义每个 API 进程的 Apache worker 数量。默认情况下，Apache worker 的数量与 CPU 数量匹配。

例如，如果您有三个控制器，每个控制器都有 32 个内核，则每个控制器上的 Barbican API 使用 32 个 Apache worker。因此，一个控制器会消耗所有 32 HSM 连接。为避免此争用，请限制为每个节点配置的 Barbican Apache worker 的数量。在本例中，将 **BarbicanWorkers** 设置为 **10**，以便所有三个控制器都能够为每个 HSM 进行 10 个并发连接。

前提条件

- 为 Atos HSM 提供厂商软件的受密码保护的 HTTPS 服务器

表 3.1. HTTPS 服务器提供的文件

File	Example	由提供
Proteccio 客户端软件 ISO 镜像文件	Proteccio1.09.05.iso	HSM Vendor
SSL 服务器证书	proteccio.CRT	HSM 管理员
SSL 客户端证书	client.CRT	HSM 管理员
SSL 客户端密钥	client.KEY	HSM 管理员

流程

1. 为 Barbican 创建 **configure-barbican.yaml** 环境文件并添加以下参数：

```
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: ip: 192.168.1.102
```

注意

atos_hsms 参数取代了 **os_hsm_ip_address** 和 **atos_server_cert_location** 的参数，这些参数将在以后的版本中被删除。

表 3.2. Heat 参数

参数	值
BarbicanSimpleCryptoGlobalDefault	这是一个布尔值，用于确定 **simplecrypto** 是否为全局默认值。
BarbicanPkcs11GlobalDefault	这是一个布尔值，用于确定 **PKCS#11** 是否为全局默认值。
BarbicanPkcs11CryptoSlotId	Barbican 使用的虚拟 HSM 的插槽 ID。
ATOSVars	
atos_client_iso_name	Atos 客户端软件 ISO 的文件名。这个值必须与 **atos_client_iso_location** 参数的 URL 中的文件名匹配。
atos_client_iso_location	URL，包括用户名和密码，用于指定 Proteccio 客户端软件 ISO 镜像的 HTTPS 服务器位置。
atos_client_cert_location	指定 SSL 客户端证书的 HTTPS 服务器位置的 URL，包括用户名和密码。
atos_client_key_location	指定 SSL 客户端密钥的 HTTPS 服务器位置的 URL，包括用户名和密码。这必须是上述客户端证书的匹配密钥。

参数	值
atos_hsms	指定 HSM 的名称、证书位置和 IP 地址的一个或多个 HSM 列表。当您在此列表中包含多个 HSM 时，Barbican 为负载平衡和高可用性配置 HSM。

2. 在部署命令中包含自定义 **configure-barbican.yaml**, **barbican.yaml** 和针对 ATOS 的 **barbican-backend-pkcs11-atos.yaml** 环境文件，以及与部署相关的任何其他环境文件：

```
$ 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/containers-prepare-parameter.yaml \
   -e /home/stack/templates/config_lvm.yaml \
   -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
   -e /home/stack/templates/network/network-environment.yaml \
   -e /home/stack/templates/hostnames.yml \
   -e /home/stack/templates/nodes_data.yaml \
   -e /home/stack/templates/extra_templates.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/templates/configure-barbican.yaml \
   --log-file overcloud_deployment_with_atos.log
```

验证

1. 创建测试 secret：

```
$ 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. 检索您刚才创建的 secret 的有效负载：

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

```
9e64-b664d574be53 --payload
+---------+-------------+
| Field   | Value       |
+---------+-------------+
| Payload | TestPayload |
+---------+-------------+
```

3.2. 将 OPENSTACK KEY MANAGER (BARBICAN)与 THALES LUNA NETWORK HSM 集成

要将 PKCS#11 后端与 Thales Luna Network HSM 设备集成用于基于硬件的加密处理，请使用 Ansible 角色在控制器上下载并安装 Thales Luna 客户端软件，并创建一个 Key Manager 配置文件使其包含预定义的 HSM IP 和凭证。

前提条件

- 为 Thales Luna Network HSM 提供厂商软件的受密码保护的 HTTPS 服务器。
- 供应商在压缩的 zip 归档中提供了 Luna Network HSM 客户端软件。

流程

1. 在 director 上安装 **ansible-role-lunasa-hsm** 角色：

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

2. 为 Key Manager (barbican)创建一个 **configure-barbican.yaml** 环境文件，并添加特定于您的环境的参数。

```
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
```

表 3.3. Heat 参数

参数	值
BarbicanSimpleCryptoGlobalDefault	这是一个布尔值，用于确定 simplecrypto 是否为全局默认值。

参数	值
BarbicanPkcs11GlobalDefault	这是一个布尔值，用于确定 PKCS#11 是否为全局默认值。
BarbicanPkcs11CryptoTokenLabel	如果您有一个 HSM，则参数的值是分区标签。如果您在两个或多个分区之间使用 HA，则这是您要提供给 HA 组的标签。
BarbicanPkcs11CryptoLogin	用于登录到 HSM 的 PKCS#11 密码，由 HSM 管理员提供。
LunasaVar	
lunasa_client_tarball_name	Luna 软件 tarball 的名称。
lunasa_client_tarball_location	指定 Luna 软件 tarball 的 HTTPS 服务器位置的 URL。
lunasa_client_installer_path	到 zipped tarball 中的 install.sh 脚本的路径。
lunasa_client_rotate_cert	（可选）当设为 true 时，将生成新客户端证书来替换任何现有证书。默认：false
lunasa_client_working_dir	（可选） Controller 节点中的工作目录。Default: /tmp/lunasa_client_install
lunasa_hsms	指定 name, hostname, admin_password, partition, 和 partition 序列号的一个或多个 HSM 列表。当在这个列表中包含多个 HSM 时，B Barbican 配置 HSM 以实现高可用性。

3. 在部署命令中包含自定义 **configure-barbican.yaml** 和 Thales 特定的 **barbican-backend-pkcs11-llunasa.yaml** 环境文件，以及与部署相关的任何其他模板：

```
$ 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/templates/configure-barbican.yaml \
  --log-file overcloud_deployment_with_luna.log
```

3.3. 将 OPENSTACK KEY MANAGER (BARBICAN)与 ENTRUST NSHIELD CONNECT XC HSM 集成

要将 PKCS#11 后端与您的 Entrust nShield Connect XC HSM 集成，请使用 Ansible 角色在 Controller 上下载并安装 Entrust 客户端软件，并创建一个 Barbican 配置文件使其包含预定义的 HSM IP 和凭证。

前提条件

- 为 Entrust nShield Connect XC 提供厂商软件的密码保护 HTTPS 服务器。

流程

1. 为 Barbican 创建 **configure-barbican.yaml** 环境文件，并添加特定于您的环境的参数。使用以下片断作为示例：

```
  parameter_defaults:
    VerifyGlanceSignatures: true
    SwiftEncryptionEnabled: true
    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_rfs_server_ip_address: 192.168.10.12
      thales_hsm_config_location: hsm-C90E-02E0-D947
      nShield_hsms:
        - name: hsm-name.example.com
          ip: 192.168.10.10
      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
```

表 3.4. Heat 参数

参数	值
BarbicanSimpleCryptoGlobalDefault	这是一个布尔值，用于确定 **simplecrypto** 是否为全局默认值。

参数	值
BarbicanPkcs11GlobalDefault	这是一个布尔值，用于确定 **PKCS#11** 是否为全局默认值。
BarbicanPkcs11CryptoSlotId	Barbican 使用的虚拟 HSM 的插槽 ID。
BarbicanPkcs11CryptoMKEKLabel	此参数定义 HSM 中生成的 mKEK 的名称。director 使用这个名称在 HSM 中创建此密钥。
BarbicanPkcs11CryptoHMACLabel	此参数定义 HSM 中生成的 HMAC 密钥的名称。director 使用这个名称在 HSM 中创建此密钥。
ThalesVars	
thales_client_working_dir	用户定义的临时工作目录。
thales_client_tarball_location	指定 Entrust 软件的 HTTPS 服务器位置的 URL。
thales_km_data_tarball_name	Entrust 软件 tarball 的名称。
thales_rfs_key	用于获取与 RFS 服务器的 SSH 连接的私钥。您必须将它添加为 RFS 服务器的授权密钥。

2. 包含自定义 **configure-barbican.yaml** 环境文件，以及 **barbican.yaml** 和 Thales 特定的 **barbican-backend-pkcs11-thales.yaml** 环境文件，以及运行 **openstack overcloud deploy** 命令时部署所需的任何其他模板：

```
$ 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/containers-prepare-parameter.yaml \
   -e /home/stack/templates/config_lvm.yaml \
   -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
   -e /home/stack/templates/network/network-environment.yaml \
   -e /home/stack/templates/hostnames.yml \
   -e /home/stack/templates/nodes_data.yaml \
   -e /home/stack/templates/extra_templates.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/templates/configure-barbican.yaml \
   --log-file overcloud_deployment_with_atos.log
```

验证

1. 创建测试 secret：

```
$ 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. 检索您刚才创建的 secret 的有效负载：

```
openstack secret get https://192.168.123.163/key-manager/v1/secrets/4cc5ffe0-eea2-449d-
9e64-b664d574be53 --payload
+---------+-------------+
| Field   | Value       |
+---------+-------------+
| Payload | TestPayload |
+---------+-------------+
```

3.3.1. 使用 Entrust nShield Connect 的负载均衡

现在，您可以通过指定有效 HSM 的数组在 Entrust nShield Connect HSMs 上启用负载共享。当列出多个 HSM 时，会启用负载共享。

这个功能在此发行版本中作为技术预览提供，因此不受红帽完全支持。它只应用于测试，不应部署在生产环境中。

有关技术预览功能的更多信息，请参阅覆盖范围详细信息。

流程

- 当您为 Entrust nShield Connect HSM 配置 **name** 和 **ip** 参数时，指定多个将启用负载共享：

```
parameter_defaults:
  ....
  ThalesVars:
    ....
    nshield_hsms:
      - name: hsm-name1.example.com
        ip: 192.168.10.10
      - name: hsm-nam2.example.com
        ip: 192.168.10.11
    ....
```

3.4. 轮转 MKEK 和 HMAC 密钥

您可以使用 director 更新来轮转 MKEK 和 HMAC 密钥。

注意

由于 Barbican 中的限制，MKEK 和 HMAC 具有相同的密钥类型。

流程

1. 在部署环境文件中添加以下参数：

```
BarbicanPkcs11CryptoRewrapKeys:          true
```

2. 更改 MKEK 和 HMAC 密钥上的标签，例如，如果您的标签与以下类似：

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

   您可以通过递增值来更改标签：

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

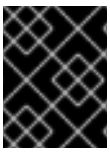
注意

不要更改 HMAC 密钥类型。

3. 使用 director 重新部署以更新。director 检查为 MKEK 和 HMAC 标记的密钥是否存在，然后创建它们。另外，如果 **BarbicanPkcs11CryptoRewrapKeys** 参数设置为 **True**，director 会调用 **barbican-manage hsm pkek_rewrap** 来重新打包所有现有 pKEK。

第 4 章 加密和验证 OPENSTACK 服务

您可以使用 barbican 来加密和验证多个 Red Hat OpenStack Platform 服务，如 Block Storage (cinder)加密密钥、块存储卷镜像、Object Storage (swift)对象和 Image Service (glance)镜像。



重要

Nova 在首次使用时格式化加密的卷（如果未加密）。然后，生成的块设备将提供给 Compute 节点。

容器化服务指南

- 不要更新您可能在物理节点的主机操作系统上找到的任何配置文件，例如 **/etc/cinder/cinder.conf**。容器化服务不引用此文件。
- 不要更新容器中运行的配置文件。重启容器后，更改将会丢失。
  相反，如果必须更改容器化服务，请更新 **/var/lib/config-data/puppet-generated/** 中的配置文件，用于生成容器。

  例如：

  - 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**

  重启容器后会应用更改。

4.1. 加密 OBJECT STORAGE (SWIFT) AT-REST 对象

默认情况下，上传到 Object Storage (swift)的对象存储是未加密的。因此，可以从文件系统中直接访问对象。如果在磁盘被丢弃前没有正确清除磁盘，这可能会带来安全风险。启用 barbican 后，对象存储服务(swift)可以透明地加密和解密存储(at-rest)对象。at-rest 加密与 in-transit 加密不同，它在存储在磁盘上时引用被加密的对象。

Swift 以透明的方式执行这些加密任务，对象在上传到 swift 时会自动加密，然后在提供给用户时自动解密。此加密和解密使用相同的(symmetric)密钥进行，该密钥存储在 barbican 中。

注意

在启用了加密并将数据添加到 swift 集群后，您无法禁用加密，因为数据现在以加密状态存储。因此，如果禁用了加密，数据将无法被读取，直到您使用相同的密钥重新启用加密为止。

前提条件

- OpenStack Key Manager 已安装并启用

流程

1. 在您的环境文件中包含 **SwiftEncryptionEnabled: True** 参数，然后使用 **/home/stack/overcloud_deploy.sh** 重新运行 **openstack overcloud deploy**。

2. 确认 swift 已配置为使用 at-rest 加密：

```
$ 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
```

   其结果应包括用于**加密**的条目。

4.2. 加密块存储(CINDER)卷

您可以使用 barbican 管理块存储(cinder)加密密钥。此配置使用 LUKS 对连接到您的实例的磁盘进行加密，包括引导磁盘。密钥管理对用户来说是透明的；当您使用 **luks** 作为加密类型创建新卷时，cinder 会为卷生成对称密钥 secret 并将其存储在 barbican 中。引导实例时（或附加加密的卷），nova 从 barbican 检索密钥，并将 secret 存储在本地 Compute 节点上的 Libvirt secret。

流程

1. 在运行 **cinder-volume** 和 **nova-compute** 服务的节点上，确认 nova 和 cinder 都配置为使用 barbican 进行密钥管理：

```
$ 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. 创建使用加密的卷模板。当您创建新卷时，可以在这里定义的设置中建模：

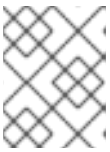
```
$ openstack volume type create --encryption-provider nova.volume.encryptors.luks.LuksEncryptor --encryption-cipher aes-xts-plain64 --encryption-key-size 256 --encryption-control-location front-end LuksEncryptor-Template-256
+-------------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Field       | Value                                                                                                                                                                            |
+-------------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| description | None                                                                                                                                                                             |
| encryption  | cipher='aes-xts-plain64', control_location='front-end', encryption_id='9df604d0-8584-4ce8-b450-e13e6316c4d3', key_size='256', provider='nova.volume.encryptors.luks.LuksEncryptor' |
| id          | 78898a82-8f4c-44b2-a460-40a5da9e4d59                                                                                                                                             |
| is_public   | True                                                                                                                                                                             |
| name        | LuksEncryptor-Template-256
```

```
|
+-------------+-----------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
```

3. 创建一个新卷，并指定它使用 **LuksEncryptor-Template-256** 设置：

```
$ openstack volume create --size 1 --type LuksEncryptor-Template-256 'Encrypted-Test-
Volume'
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| consistencygroup_id | None                                 |
| created_at          | 2018-01-22T00:19:06.000000           |
| description         | None                                 |
| encrypted           | True                                 |
| id                  | a361fd0b-882a-46cc-a669-c633630b5c93 |
| migration_status    | None                                 |
| multiattach         | False                                |
| name                | Encrypted-Test-Volume                |
| properties          |                                      |
| replication_status  | None                                 |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | LuksEncryptor-Template-256           |
| updated_at          | None                                 |
| user_id             | 0e73cb3111614365a144e7f8f1a972af     |
+---------------------+--------------------------------------+
```

生成的 secret 会自动上传到 barbican 后端。



注意

确保创建加密卷的用户具有项目的 **creator** barbican 角色。如需更多信息，请参阅 **授予用户对创建者角色的访问权限** 部分。

4. 获取 barbican secret UUID。这个值显示在 **encryption_key_id** 字段中。

```
$ cinder --os-volume-api-version 3.64 volume show Encrypted-Test-Volume
+------------------------------+-----------------------------------+
|Property                      |Value                              |
+------------------------------+-----------------------------------+
|attached_servers              |[]                                 |
|attachment_ids                |[]                                 |
|availability_zone             |nova                               |
|bootable                      |false                              |
|cluster_name                  |None                               |
|consistencygroup_id           |None                               |
|created_at                    |2022-07-28T17:35:26.000000         |
|description                   |None                               |
|encrypted                     |True                               |
```

```
|encryption_key_id            |0944b8a8-de09-4413-b2ed-38f6c4591dd4 |
|group_id                     |None                                 |
|id                           |a0b51b97-0392-460a-abfa-093022a120f3 |
|metadata                     |                                     |
|migration_status             |None                                 |
|multiattach                  |False                                |
|name                         |vol                                  |
|os-vol-host-attr:host        |hostgroup@tripleo_iscsi#tripleo_iscsi|
|os-vol-mig-status-attr:migstat|None                                |
|os-vol-mig-status-attr:name_id|None                                |
|os-vol-tenant-attr:tenant_id |a2071ece39b3440aa82395ff7707996f     |
|provider_id                  |None                                 |
|replication_status           |None                                 |
|service_uuid                 |471f0805-072e-4256-b447-c7dd10ceb807 |
|shared_targets               |False                                |
|size                         |1                                    |
|snapshot_id                  |None                                 |
|source_volid                 |None                                 |
|status                       |available                            |
|updated_at                   |2022-07-28T17:35:26.000000           |
|user_id                      |ba311b5c2b8e438c951d1137333669d4     |
|volume_type                  |LUKS                                 |
|volume_type_id               |cc188ace-f73d-4af5-bf5a-d70ccc5a401c |
+-----------------------------+-------------------------------------+
```

注意

您必须使用 **--os-volume-api-version 3.64** 参数和 Cinder CLI 来显示 **encryption_key_id** 值。没有对应的 OpenStack CLI 命令。

5. 使用 barbican 确认存在磁盘加密密钥。在本例中，时间戳与 LUKS 卷创建时间匹配：

```
$ openstack secret list
+------------------------------------------------------------------------------+------+----------
---------+--------+-----------------------------------------+----------+------------+-------------+------
+------------+
| Secret href                                                  | Name | Created                | Status
| Content types                    | Algorithm | Bit length | Secret type | Mode | Expiration |
+------------------------------------------------------------------------------+------+----------
---------+--------+-----------------------------------------+----------+------------+-------------+------
+------------+
| https://192.168.123.169:9311/v1/secrets/0944b8a8-de09-4413-b2ed-38f6c4591dd4 | None |
2018-01-22T02:23:15+00:00 | ACTIVE | {u'default': u'application/octet-stream'} | aes       |
256 | symmetric   | None | None       |
+------------------------------------------------------------------------------+------+----------
---------+--------+-----------------------------------------+----------+------------+-------------+------
+------------+
```

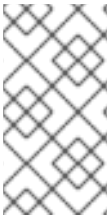
6. 将新卷附加到现有实例。例如：

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

然后，卷会呈现在客户端操作系统中，并可使用内置工具挂载。

4.2.1. 将块存储卷迁移到 OpenStack Key Manager

如果您之前使用 **ConfKeyManager** 管理磁盘加密密钥，您可以通过扫描数据库中的 **encryption_key_id** 条目来将卷迁移到 OpenStack Key Manager。每个条目都会获取一个新的 barbican 密钥 ID，并保留现有的 **ConfKeyManager** secret。



注意

- 在以前的版本中，您可以使用 **ConfKeyManager** 为加密的卷重新分配所有权。对于其密钥由 barbican 管理的卷，这不可能。
- 激活 barbican 不会破坏现有的 **keymgr** 卷。

前提条件

在迁移前，请查看 Barbican-managed 加密卷和使用 **ConfKeyManager** 的卷之间的以下区别：

- 您无法转让加密卷的所有权，因为目前无法传输 barbican secret 的所有权。
- barbican 对允许读取和删除 secret 更严格的限制，这可能会影响一些 Cinder 卷操作。例如，用户无法附加、分离或删除其他用户的卷。

流程

1. 部署 barbican 服务。
2. 将 **creator** 角色添加到 cinder 服务。例如：

```
#openstack role create creator
#openstack role add --user cinder creator  --project service
```

3. 重启 **cinder-volume** 和 **cinder-backup** 服务。**cinder-volume** 和 **cinder-backup** 服务会自动开始迁移过程。您可以检查日志文件以查看迁移的状态信息：
   - **cinder-volume** - 迁移存储在 cinder 的 Volumes 和 Snapshots 表中的密钥。
   - **cinder-backup** - 迁移 Backups 表中的密钥。
4. 监控指示迁移完成的消息的日志，并检查没有更多卷正在使用 **ConfKeyManager** all-zeros 加密密钥 ID。
5. 从 **cinder.conf** 和 **nova.conf** 中删除 **fixed_key** 选项。您必须确定哪些节点配置了此设置。
6. 从 cinder 服务中删除 **创建者** 角色。

验证

- 启动进程后，日志文件中会显示其中一个条目。这表明迁移是否已正确启动，或标识了它遇到的问题：
  - **不迁移加密密钥，因为 ConfKeyManager 仍在使用。**
  - **不迁移加密密钥，因为 ConfKeyManager 的 fixed_key 没有被使用。**

- **不迁移加密密钥，因为不支持迁移到 'XXX' key_manager 后端。** - 这个消息不太可能出现；在遇到 barbican 以外的另一个密钥管理器后端时，这个安全检查是处理代码的安全检查。这是因为代码只支持一个迁移场景： *从* ConfKeyManager *到* barbican。
- **不迁移加密密钥，因为没有与此主机关联的卷。** - 当 **cinder-volume** 在多个主机上运行，而特定主机没有与之关联的卷时会出现这种情况。这是因为每个主机都需要负责处理自己的卷。
- **开始迁移 ConfKeyManager 键。**
- **将卷 <UUID> 加密密钥迁移到 Barbican** - 迁移期间，将检查所有主机的卷；如果卷仍在使用 ConfKeyManager 的密钥 ID （它只包括零 (**00000000-0000-0000-0000-000000000000**)，则会出现这个消息。
  - 对于 **cinder-backup**，此消息使用稍有不同的大写： **迁移卷 [...]** 或 **迁移备份 [...]**

- 每个主机检查其所有卷后，主机会显示一个摘要状态消息：

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

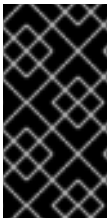
- 您还可以看到以下条目：
  - **使用 ConfKeyManager 的全零加密密钥 ID 仍有 %d 卷。**
  - **使用 ConfKeyManager 的全零加密密钥 ID 仍有 %d 备份。**
    这两个消息都可能出现在 **cinder-volume** 和 **cinder-backup** 日志中。每个服务仅处理其自身条目的迁移，但该服务知道其他的状态。因此，**cinder-volume** 知道 **cinder-backup** 是否仍然有要迁移的备份，**cinder-backup** 知道 **cinder-volume** 服务是否有要迁移的卷。

虽然每个主机只迁移自己的卷，但摘要消息是基于全局评估，以确定任何卷是否仍然需要迁移，确认所有卷的迁移已完成。

cleanup

将您的密钥 ID 迁移到 barbican 后，固定密钥会保留在配置文件中。这可能会造成对某些用户的安全问题，因为 **fixed_key** 值没有在 **.conf** 文件中加密。

要解决这个问题，您可以从 nova 和 cinder 配置中手动删除 **fixed_key** 值。但是，在进行前，首先完成测试并查看日志文件的输出，因为仍依赖于这个值的磁盘无法访问。



重要

encryption_key_id 最近添加到 **Backup** 表中，作为 Queens 发行版本的一部分。因此，已存在的加密卷的备份可能会存在。all-zeros **encryption_key_id** 存储在备份本身上，但它不会出现在 **Backup** 数据库中。因此，迁移过程无法了解某些加密卷的备份是否仍然存在，仍依赖于所有零 **ConfKeyMgr** 密钥 ID。

1. 查看现有的 **fixed_key** 值。这两个服务的值都必须匹配。

```
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
```

重要

备份现有的 **fixed_key** 值。如果出现问题，或者需要恢复使用旧加密密钥的备份，这允许您恢复该值。

2. 删除 **fixed_key** 值：

```
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
```

故障排除

只有在请求者具有 **creator** 角色时，才能创建 barbican secret。这意味着 cinder 服务本身需要创建者角色，否则会出现类似如下的日志序列：

1. **开始迁移 ConfKeyManager 键。**
2. **将卷 <UUID> 加密密钥迁移到 Barbican**
3. **迁移加密密钥时出错：禁止 Secret 创建尝试 - 请查看您的用户/项目权限**
4. **使用 ConfKeyManager 的全零加密密钥 ID 仍有 %d 卷。**

key 消息是第三个：**Secret 创建尝试。**要解决这个问题，更新 **cinder** 帐户的权限：

1. 运行 **openstack role add --project service --user cinder creator**
2. 重启 **cinder-volume** 和 **cinder-backup** 服务。

因此，在迁移时下一次尝试应该会成功。

4.3. 验证 BLOCK STORAGE (CINDER) 卷镜像

Block Storage Service (cinder)会在从镜像创建卷过程中自动验证任何已下载的已签名。签名在镜像写入卷前进行验证。要提高性能，您可以使用 Block Storage Image-Volume 缓存存储验证的镜像以创建新卷。

注意

Red Hat Ceph Storage 或 RBD 卷不支持 Cinder 镜像签名验证。

流程

1. 登录到 Controller 节点。
2. 选择以下选项之一：
   - 查看 **卷** 日志中的 cinder 镜像验证活动 **/var/log/containers/cinder/cinder-volume.log**。例如，您可以在实例引导时预期以下条目：

```
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
```

- 使用 **openstack volume list** 和 **cinder volume show** 命令：

  a. 使用 **openstack volume list** 命令找到卷 ID。

  b. 在计算节点上运行 **cinder volume show** 命令：

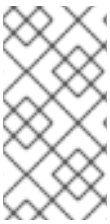
  ```
  cinder volume show <VOLUME_ID>
  ```

3. 找到 **volume_image_metadata** 部分，其中包含 验证了：**True** 的行签名。

```
$ 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-6fbb080bb483          |
| 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                                           |
+--------------------------------+-----------------------------------------------+
```

注意

快照保存为镜像服务(glance)镜像。如果您将 Compute 服务(nova)配置为检查已签名的镜像，则必须从 glance 手动下载镜像，为镜像签名，然后重新上传镜像。无论快照来自使用签名镜像创建的实例，还是从从签名镜像创建的卷引导的实例，都是如此。



注意

卷可以上传为镜像服务(glance)镜像。如果原始卷是可引导的，则镜像可用于在块存储服务(cinder)中创建可引导卷。如果您将块存储服务配置为检查已签名的镜像，则必须从 glance 中手动下载镜像，计算镜像签名并更新所有适当的镜像签名属性，然后才能使用镜像。更多信息请参阅 第 4.5 节 “验证快照”。

其他资源

- 配置块存储服务(cinder)

4.3.1. 自动删除卷镜像加密密钥

将加密卷上传到镜像服务(glance)时，块存储服务(cinder)在密钥管理服务(barbican)中创建一个加密密钥。这会在加密密钥和存储的镜像之间创建一个 1:1 关系。

加密密钥删除可防止密钥管理服务无限度地消耗资源。块存储、密钥管理和镜像服务自动管理加密卷的密钥，包括删除密钥。

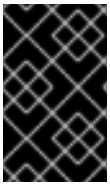
块存储服务会自动将两个属性添加到卷镜像中：

- **cinder_encryption_key_id** - 密钥管理服务为特定镜像存储的加密密钥的标识符。
- **cinder_encryption_key_deletion_policy** - 告知镜像服务是否删除与此镜像关联的密钥的策略。

重要

这些属性的值会被自动分配。为**避免意外的数据丢失，请不要调整这些值**。

当您创建卷镜像时，块存储服务会将 **cinder_encryption_key_deletion_policy** 属性设置为 **on_image_deletion**。当您删除卷镜像时，如果 **cinder_encryption_key_deletion_policy** 等于 **on_image_deletion** _policy，则镜像服务会删除对应的加密密钥。



重要

红帽不推荐手动操作 **cinder_encryption_key_id** 或 **cinder_encryption_key_deletion_policy** 属性。如果您使用由 **cinder_encryption_key_id** 值标识的加密密钥用于任何其他目的，则风险数据丢失。

4.4. 签名镜像服务(GLANCE)镜像

当您配置镜像服务(glance)以验证上传的镜像是否已被篡改时，您必须先为镜像签名，然后才能使用这些镜像启动实例。使用 **openssl** 命令使用存储在 barbican 中的密钥为镜像签名，然后使用附带的签名信息将镜像上传到 glance。因此，在每次使用前会验证镜像的签名，如果签名不匹配，实例构建过程会失败。

前提条件

- OpenStack Key Manager 已安装并启用

流程

1. 在环境文件中，使用 **VerifyGlanceSignatures: True** 设置启用镜像验证。您必须重新运行 **openstack overcloud deploy** 命令，才能使此设置生效。
2. 要验证 glance 镜像验证是否已启用，请在 overcloud Compute 节点上运行以下命令：

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

注意

如果您使用 Ceph 作为镜像和计算服务的后端，则会创建一个 CoW 克隆。因此，无法执行镜像签名验证。

3. 确认 glance 已配置为使用 barbican ：

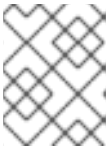
```
$ sudo crudini --get /var/lib/config-data/puppet-generated/glance_api/etc/glance/glance-
api.conf key_manager backend
castellan.key_manager.barbican_key_manager.BarbicanKeyManager
```

4. 生成证书：

```
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
```

5. 将证书添加到 barbican secret 存储中：

```
$ 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' -f value
https://192.168.123.170:9311/v1/secrets/5df14c2b-f221-4a02-948e-48a61edd3f5b
```



注意

记录生成的 UUID 以供稍后的步骤使用。在本例中，证书的 UUID 是 **5df14c2b-f221-4a02-948e-48a61edd3f5b**。

6. 使用 **private_key.pem** 为镜像签名并生成 **.signature** 文件。例如：

```
$ 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
```

7. 将生成的 **.signature** 文件转换为 *base64* 格式：

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

8. 将 *base64* 值加载到变量中，以便在后续命令中使用它：

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

9. 将已签名的镜像上传到 glance。对于 **img_signature_certificate_uuid**，您必须指定之前上传到 barbican 的签名密钥的 UUID：

```
 openstack image create \
--container-format bare --disk-format qcow2 \
--property img_signature="$cirros_signature_b64" \
--property img_signature_certificate_uuid="5df14c2b-f221-4a02-948e-48a61edd3f5b"\
--property img_signature_hash_method="SHA-256" \
--property img_signature_key_type="RSA-PSS" cirros_0_4_0_signed \
--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                |
lcI7nGgoKxnCyOcsJ4abbEZEpzXByFPIgiPeiT+Otjz0yvW00KNN3fI0AA6tn9EXrp7fb2xBDE4UaO3v |
|                              |
IFquV/s3mU4LcCiGdBAl3pGsMlmZZIQFVNcUPOaayS1kQYKY7kxYmU9iq/AZYyPw37KQI52smC/zoO54 |
|                              | zZ+JpnfwIsM=                                                                 |
| img_signature_certificate_uuid | ba3641c2-6a3d-445a-8543-851a68110eab
|
| 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                        | 9f812310df904e6ea01e1bacb84c9f1a
|
| protected                    | False                                                                        |
| size                         | None                                                                         |
| status                       | queued                                                                       |
| tags                         | []                                                                           |
| updated_at                   | 2018-01-23T05:37:31Z                                                         |
| virtual_size                 | None                                                                         |
| visibility                   | shared                                                                       |
+------------------------------+------------------------------------------------------------------------------
----+
```

10. 您可以在 Compute 日志中查看 glance 的镜像验证活动：**/var/log/containers/nova/nova-compute.log**。例如，您可以在实例引导时预期以下条目：

```
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
```

4.5. 验证快照

快照保存为镜像服务(glance)镜像。如果您将 Compute 服务(nova)配置为检查已签名的镜像，则快照必须经过签名，即使它们是从带有签名镜像的实例创建。

流程

1. 从 glance 下载快照

```
openstack image save --file <local-file-name> <image-name>
```

2. 生成签名以验证快照。这是生成签名以验证任何镜像时所使用的进程。如需更多信息，请参阅验证 镜像服务(glance)镜像。

3. 更新镜像属性：

```
openstack image set \
  --property img_signature="$cirros_signature_b64" \
  --property img_signature_certificate_uuid="5df14c2b-f221-4a02-948e-48a61edd3f5b" \
  --property img_signature_hash_method="SHA-256" \
  --property img_signature_key_type="RSA-PSS" \
  <image_id_of_the_snapshot>
```

4. 可选：从文件系统中删除下载的 glance 镜像：

```
rm <local-file-name>
```