Red Hat OpenStack Platform 16.2

Manage Secrets with OpenStack Key Manager

OpenStack Key Manager (barbican) を OpenStack デプロイメントと統合する方法。

Last Updated: 2024-02-07

Red Hat OpenStack Platform 16.2 Manage Secrets with OpenStack Key Manager

OpenStack Key Manager (barbican) を OpenStack デプロイメントと統合する方法。

OpenStack Team
rhos-docs@redhat.com

法律上の通知

概要

OpenStack Key Manager (barbican) を OpenStack デプロイメントと統合する方法。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに対するご意見をお聞かせください。ドキュメントの改善点があればお知らせください。

Jira でドキュメントのフィードバックを提供する

ドキュメントに関するフィードバックを提供するには、Create Issue フォームを使用します。Red Hat OpenStack Platform Jira プロジェクトで Jira Issue が作成され、フィードバックの進行状況を追跡できます。

1. Jira にログインしていることを確認してください。Jira アカウントをお持ちでない場合は、アカウントを作成してフィードバックを送信してください。
2. Create Issue をクリックして、**Create Issue** ページを開きます。
3. **Summary** フィールドと **Description** フィールドに入力します。**Description** フィールドに、ドキュメントの URL、章またはセクション番号、および問題の詳しい説明を入力します。フォーム内の他のフィールドは変更しないでください。
4. **Create** をクリックします。

第1章 OPENSTACK KEY MANAGER(BARBICAN) のデプロイと設定

OpenStack Key Manager (barbican) は Red Hat OpenStack Platform のシークレットマネージャーです。barbican API とコマンドラインを使用して、OpenStack サービスの使用する証明書、キー、パスワードを一元管理することができます。Red Hat OpenStack Platform では、barbican はデフォルトで有効になっていません。既存の OpenStack デプロイメントに barbican をデプロイすることができます。

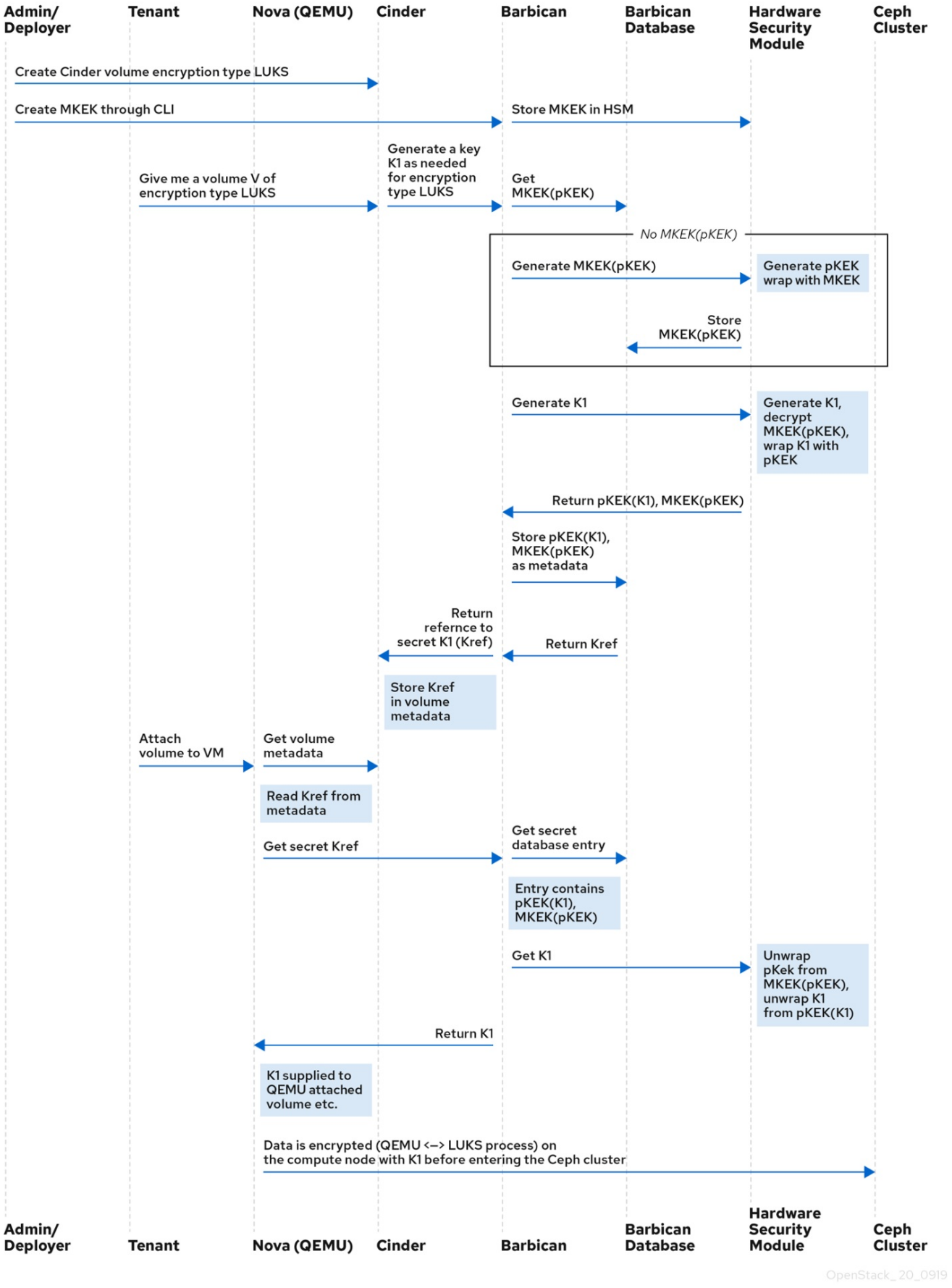
barbican は現在、本書で説明されている以下のユースケースをサポートします。

- **対称暗号鍵**: 中でも Block Storage (cinder) ボリュームの暗号化、一時ディスク暗号化、および Object Storage (swift) の暗号化に使用されます。
- **非対称鍵と証明書**: glance イメージの署名や検証などに使用されます。

OpenStack Key Manager は、Block Storage (cinder)、Networking (neutron)、および Compute (nova) コンポーネントと統合します。

1.1. OPENSTACK KEY MANAGER ワークフロー

以下の図は、OpenStack Key Manager が環境のシークレットを管理するために使用するワークフローを示しています。



1.2. OPENSTACK KEY MANAGER 暗号化タイプ

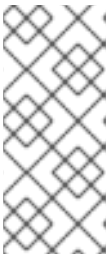
証明書、API キー、パスワードなどのシークレットは、barbican データベースまたはセキュアなストレージシステムに直接暗号化された blob に保存できます。簡単な暗号プラグインまたは PKCS#11 暗号プラグインを使用して、シークレットを暗号化できます。

シークレットを barbican データベースに暗号化された Blob として保存するには、以下のオプションを使用することができます。

- **Simple crypto プラグイン**: シンプルな暗号化プラグインはデフォルトで有効になっており、単一の対称キーを使用してすべてのシークレットペイロードを暗号化します。このキーは **barbican.conf** ファイルのプレーンテキストに保存されるため、このファイルへの不正アクセスを防ぐことが重要です。

- **PKCS#11 暗号化プラグイン**: PKCS#11 暗号化プラグインは、barbican データベースに保存されるプロジェクト固有の鍵暗号化キー (pKEK) を使用してシークレットを暗号化します。これらのプロジェクト固有の pKEK は、ハードウェアセキュリティーモジュール (HSM) に保存される主な鍵暗号化キー (MKEK) により暗号化されます。すべての暗号化および復号化操作は、in-process メモリーではなく、HSM に置かれます。PKCS#11 プラグインは、PKCS#11 API 経由で HSM と通信します。暗号化はセキュアなハードウェアで行われ、プロジェクトごとに異なる pKEK が使用されるため、このオプションは単純な暗号化プラグインよりも安全です。
  Red Hat は、以下の HSM のいずれかで PKCS#11 バックエンドをサポートします。

デバイス	リリースでサポートされます。	高可用性 (HA) のサポート
ATOS Trustway Proteccio NetHSM	16.0+	16.1+
Entrust nShield Connect HSM	16.0+	サポート対象外
Thales Luna Network HSM	16.1 以降 (テクノロジープレビュー)	16.1 以降 (テクノロジープレビュー)



注記

高可用性 (HA) オプション: barbican サービスは Apache 内で実行され、高可用性に HAProxy を使用するように director により設定されます。バックエンド層の HA オプションは、使用されているバックエンドによって異なります。たとえば、簡単な暗号化の場合、すべての barbican インスタンスには設定ファイル内に同じ暗号化キーがあり、これにより単純な HA 設定が作成されます。

1.2.1. 複数の暗号化メカニズムの設定

Barbican の 1 つのインスタンスを設定して、複数のバックエンドを使用することができます。これが完了したら、バックエンドを **global default** として指定する必要があります。プロジェクトごとにデフォルトのバックエンドを指定することもできます。プロジェクトのマッピングが存在しない場合は、そのプロジェクトのシークレットは、グローバルのデフォルトバックエンドを使用して保存されます。

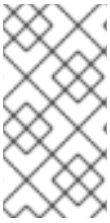
たとえば、Simple crypto および PKCS#11 プラグインの両方を使用するように Barbican を設定できます。Simple crypto をグローバルデフォルトとして設定すると、すべてのプロジェクトがそのバックエンドを使用します。これにより、PKCS#11 バックエンドを使用するプロジェクトを、そのプロジェクトの推奨バックエンドとして PKCS#11 を設定すると、そのプロジェクトを指定できます。

新規バックエンドに移行する場合には、グローバルのデフォルトまたはプロジェクト固有のバックエンドとして新しいバックエンドを有効にする時に、元のバックエンドを利用可能な状態にすることができ

ます。その結果、古いシークレットは古いバックエンドで使用でき、新規シークレットは新しいグローバルデフォルトバックエンドに保存されます。

1.3. キーマネージャーのデプロイ

OpenStack Key Manager をデプロイするには、まず barbican サービスの環境ファイルを作成し、追加の環境ファイルでオーバークラウドを再デプロイします。次に、ユーザーを **creator** に追加して barbican シークレットを作成および編集するか、シークレットを保存する暗号化されたボリュームを barbican に作成します。



注記

この手順では、barbican が **simple_crypto** バックエンドを使用するように設定します。別の設定を必要とする **PKCS#11** や、使用する HSM に依存するさまざまな heat テンプレートファイルなど、追加のバックエンドが利用できます。KMIP、Hashicorp Vault および DogTag などの他のバックエンドはサポートされません。

前提条件

- オーバークラウドがデプロイされ、実行されています。

手順

1. アンダークラウドノードで、barbican の環境ファイルを作成します。

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

BarbicanSimpleCryptoGlobalDefault は、このプラグインをグローバルのデフォルトプラグインとして設定します。

環境ファイルに以下のオプションを追加することもできます。

- **BarbicanPassword**: barbican サービスアカウントのパスワードを設定します。
- **BarbicanWorkers**: **barbican::wsgi::apache** のワーカー数を設定します。デフォルトで **'%{::processorcount}'** を使用します。
- **BarbicanDebug**: デバッグを有効にします。
- **BarbicanPolicies**: barbican 向けに設定するポリシーを定義します。ハッシュ値を使用します (例: **{ barbican-context_is_admin: { key: context_is_admin, value: 'role:admin' } }**)。このエントリーは **/etc/barbican/policy.json** に追加されます。ポリシーの詳細は、後のセクションで説明します。
- **BarbicanSimpleCryptoKek**: キー暗号化キー (KEK) は、指定がない場合は director によって生成されます。

2. スクリプトから以前追加したロール、テンプレート、または環境ファイルを削除せずに、**openstack overcloud deploy** コマンドに以下のファイルを追加します。
   - /usr/share/openstack-tripleo-heat-templates/environments/services/barbican.yaml

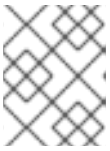
- /usr/share/openstack-tripleo-heat-templates/environments/barbican-backend-simple-crypto.yaml
- /home/stack/templates/configure-barbican.yaml

3. デプロイメントスクリプトを再度実行して、変更をデプロイメントに適用します。

```
$ 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. **creator** ロールの **id** を取得します。

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



注記

OpenStack Key Manager(barbican) がインストールされていないと、**creator** ロールは表示されません。

5. ユーザーを **creator** ロールに割り当て、関連するプロジェクトを指定します。この例では、**project_a** プロジェクトの **user1** という名前のユーザーが **creator** ロールに追加されます。

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

検証

1. テストシークレットを作成します。以下に例を示します。

```
$ 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. 作成したシークレットのペイロードを取得します。

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

1.4. キーマネージャーポリシーの表示

barbican はポリシーを使用して、キーの追加または削除など、シークレットに対してアクションを実行できるユーザーを決定します。これらのコントロールを実装するには、以前に作成した **creator** などの keystone プロジェクトロールは barbican 内部パーミッションにマッピングされます。その結果、これらのプロジェクトロールに割り当てられるユーザーは、対応する barbican パーミッションを受信します。

デフォルトのポリシーはコードで定義されるため、通常は修正は必要ありません。ポリシーが変更されていない場合は、環境内の既存のコンテナーを使用してデフォルトポリシーを表示できます。デフォルトポリシーに変更が加えられた場合には、デフォルトを確認する場合は、別のシステムを使用して **openstack-barbican-api** コンテナーを最初にプルします。

前提条件

- OpenStack Key Manager がデプロイされ、実行している。

手順

1. Red Hat 認証情報を使用して podman にログインします。

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

2. **openstack-barbican-api** コンテナーをプルします。

```
podman pull \
registry.redhat.io/rhosp-rhel8/openstack-barbican-api:16.2
```

3. 現在の作業ディレクトリーにポリシーファイルを生成します。

```
podman run -it \
registry.redhat.io/rhosp-rhel8/openstack-barbican-api:16.2 \
oslopolicy-policy-generator \
--namespace barbican > barbican-policy.yaml
```

検証

barbican-policy.yaml ファイルをチェックして、barbican が使用するポリシーを確認します。ポリシーは、ユーザーがシークレットおよびシークレットメタデータと対話する方法を定義する 4 つの異なるロールによって実装されます。ユーザーは、特定のロールに割り当てられているパーミッションを受け取ります。

admin

admin ロールは、すべてのプロジェクトでシークレットの読み取り、作成、編集、削除を実行できます。

creator

creator ロールは、作成者がスコープ指定されているプロジェクトのシークレットの読み取り、作成、編集、削除を実行できます。

observer

observer ロールは、シークレットの読み取りのみ実行できます。

audit

audit ロールは、メタデータの読み取りのみ実行できます。audit ロールは、シークレットの読み取りは実行できません。

たとえば、以下のエントリーは、各プロジェクトの **admin**、**observer**、および **creator** の keystone ロールをリスト表示します。右側には、**role:admin**、**role:observer**、および **role:creator** パーミッションが割り当てられていることを確認します。

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

#
#"observer": "role:observer"

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

これらのロールは barbican でグループ化することもできます。たとえば、**admin_or_creator** を指定するルールは、**rule:admin** または **rule:creator** のメンバーに適用できます。

ファイル内では、**secret:put** および **secret:delete** のアクションがあります。右側には、これらのアクションを実行するパーミッションがあるロールについて確認してください。以下の例では、**secret:delete** は、**admin** および **creator** ロールメンバーのみがシークレットエントリーを削除できることを意味します。さらに、ルールは、そのプロジェクトの **admin** または **creator** ロールのユーザーがそのプロジェクトのシークレットを削除できることを示しています。プロジェクトのマッチは、ポリシーにも定義される **secret_project_match** ルールで定義されます。

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

第2章 OPENSTACK KEY MANAGER (BARBICAN) を使用したシークレットおよび鍵の管理

OpenStack Key Manager を使用して、シークレットおよび暗号化鍵の作成、更新、および削除を行います。暗号化鍵および barbican データベースのバックアップやリストアを行うこともできます。暗号化鍵および barbican データベースを定期的にバックアップすることを推奨します。

2.1. シークレットの表示

シークレットのリストを表示するには、**openstack secret list** コマンドを実行します。リストには、URI、名前、種別、シークレットに関するその他の情報が含まれます。

手順

- シークレットのリストを表示します。

```
$ 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. シークレットの作成

シークレットを作成するには、**openstack secret store** コマンドを実行し、シークレットの名前とオプションでシークレットのペイロードを指定します。

手順

- シークレットを作成します。以下に例を示します。

```
$ 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. シークレットへのペイロードの追加

シークレットのペイロード (シークレットの削除以外) を変更することはできませんが、ペイロードを指定せずにシークレットを作成した場合は、後で **openstack secret update** コマンドを使用してペイロードを追加することができます。

手順

- シークレットにペイロードを追加します。

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

2.4. シークレットの削除

シークレットを削除するには、**openstack secret delete** コマンドを実行し、シークレットの URI を指定します。

手順

- 指定された URI のシークレットを削除します。

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

2.5. 対称鍵の生成

対称鍵を生成するには、**order create** コマンドを使用し、鍵を barbican に保存します。続いて、nova のディスク暗号化や swift オブジェクトの暗号化など、特定のタスクに対称鍵を使用します。

前提条件

- OpenStack Key Manager がインストールされ、動作している。

手順

1. **order create** を使用して新しい 256 ビットのキーを生成し、barbican に保存します。以下に例を示します。

```
$ 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 オプションを使用して、**ctr** や **cbc** などの特定のモードを使用するように生成される鍵を設定することもできます。詳細は、**NIST SP 800-38A** を参照してください。

2. オーダーの詳細を表示して、生成されたキーの場所を **Secret href** の値として特定します。

```
$ 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. シークレットの詳細を取得します。

```
$ 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. 簡単な CRYPTO 暗号化鍵のバックアップ

簡単な crypto 暗号化鍵をバックアップするには、メインの KEK が含まれる **barbican.conf** ファイルをセキュリティーが強化された場所にコピーしてから、barbican データベースをバックアップします。

重要

この手順には、テストのシークレットおよび鍵を生成するステップが含まれます。シークレットの鍵をすでに生成している場合は、テスト用鍵の手順を省略し、生成したキーを使用します。

前提条件

- OpenStack Key Manager がインストールされ、動作している。
- KEK のバックアップを保存するセキュリティーが強化された場所がある。

手順

1. オーバークラウドで新しい 256 ビットの鍵を生成し、barbican に保存します。

```
(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. テストシークレットを作成します。

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

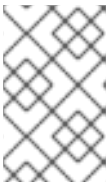
3. テストシークレットが作成されていることを確認します。

```
(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. メインの KEK が含まれる **barbican.conf** ファイルを、セキュリティーが強化された場所にコピーします。

5. **controller-0** ノードにログインし、**barbican** ユーザーパスワードを取得します。

```
[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",
```



注記

barbican データベースにアクセスできるのは、ユーザー **barbican** のみです。そのため、barbican ユーザーのパスワードはデータベースをバックアップまたは復元するために必要です。

6. **barbican** データベースをバックアップします。

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

7. データベースのバックアップが **/home/heat-admin** に保存されていることを確認します。

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

8. オーバークラウドで、以前に作成したシークレットを削除し、それらのシークレットが存在しないことを確認します。

```
(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. バックアップからの簡単な CRYPTO 暗号化鍵のリストア

barbican データベースをバックアップからリストアするには、barbican パーミッションでコントローラーノードにログインして、barbican データベースをリストアします。バックアップから KEK をリストアするには、バックアップファイルで **barbican.conf** ファイルを上書きします。

前提条件

- OpenStack Key Manager がインストールされ、動作している。
- **barbican.conf** ファイルと barbican データベースの既存のバックアップがある。

手順

1. **controller-0** ノードにログインし、コントローラー上に **barbican** ユーザーにデータベースを復元するためのアクセスを付与する **barbican** データベースがあることを確認します。

```
[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 ~]$
```

2. **barbican** データベースにバックアップファイルをリストアします。

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

3. 以前にバックアップしたファイルで **barbican.conf** ファイルを上書きします。

検証

- オーバークラウドで、テストシークレットが正常に復元されたことを確認します。

```
(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 OPENSTACK KEY MANAGER (BARBICAN) と HARDWARE SECURITY MODULE (HSM) アプライアンスのインテグレーション

ハードウェアセキュリティーモジュール (HSM) アプライアンスと Red Hat OpenStack Platform デプロイメントを統合し、ハードウェアベースの暗号化処理を使用してセキュリティーオポストシャーを向上させます。OpenStack Key Manager と HSM アプライアンスとのインテグレーションを計画する場合には、サポートされている暗号化種別および HSM アプライアンスを選択し、通常のバックアップを設定し、デプロイメントに影響する可能性のある他の情報や制限を確認する必要があります。

3.1. OPENSTACK KEY MANAGER (BARBICAN) と ATOS HSM のインテグレーション

PKCS#11 バックエンドを Trustway Proteccio Net HSM アプライアンスと統合するには、barbican を HSM に接続するパラメーターで設定ファイルを作成します。**atos_hsms** パラメーターの下に 2 つ以上の HSM をリスト表示して、HA を有効にすることができます。

プランニング

デフォルトでは、HSM の同時接続の最大数は 32 個あります。この数値を超えると、PKCS#11 クライアントからのメモリーエラーが発生することがあります。以下に示すように、接続数を計算することができます。

- 各コントローラーには 1 つの **barbican-api** と 1 つの **barbican-worker** プロセスがあります。
- 各 Barbican API プロセスは **N** 個の Apache ワーカーで実行されます (**N** はデフォルトで CPU の数に設定されます)。
- 各ワーカーには HSM への 1 つの接続があります。

各 **barbican-worker** プロセスには、データベースに 1 つのコネクションがあります。**BarbicanWorkers** heat パラメーターを使用して、各 API プロセスの Apache ワーカー数を定義することができます。デフォルトでは、Apache ワーカーの数は CPU 数と一致します。

たとえば、32 コアを持つ 3 つのコントローラーがある場合は、各コントローラー上の Barbican API は 32 Apache ワーカーを使用します。これにより、1 つのコントローラーは、利用可能な 32 HSM 接続をすべて消費します。この競合を回避するには、各ノードに設定された Barbican Apache ワーカーの数を制限します。この例では、**BarbicanWorkers** を **10** に設定します。これにより、これら 3 つのコントローラーすべてが HSM に対して同時接続 10 個となるようにします。

前提条件

- Atos HSM にベンダーソフトウェアを提供するパスワードで保護された HTTPS サーバー

表3.1 HTTPS サーバーが提供するファイル

ファイル	例	提供元
Proteccio Client Software ISO イメージファイル	Proteccio1.09.05.iso	HSM ベンダー
SSL サーバー証明書	proteccio.CRT	HSM 管理者

ファイル	例	提供元
SSL クライアント証明書	client.CRT	HSM 管理者
SSL クライアントキー	client.KEY	HSM 管理者

手順

1. Barbican 用の **configure-barbican.yaml** 環境ファイルを作成し、以下のパラメーターを追加します。

```
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
```

注記

atos_hsms パラメーターは、非推奨となったパラメーター **atos_hsm_ip_address** および **atos_server_cert_location** に置き換わるもので、今後のリリースで削除される予定です。

表3.2 heat パラメーター

パラメーター	値
BarbicanSimpleCryptoGlobalDefault	これは、**simplecrypto** がグローバルデフォルトであるかどうかを決定するブール値です。
BarbicanPkcs11GlobalDefault	これは、**PKCS#11** がグローバルデフォルトであるかどうかを決定するブール値です。
BarbicanPkcs11CryptoSlotId	Barbican によって使用される仮想 HSM のスロット ID。
ATOSVars	

パラメーター	値
atos_client_iso_name	Atos クライアントソフトウェア ISO のファイル名。この値は、**atos_client_iso_location** パラメーターの URL のファイル名と一致する必要があります。
atos_client_iso_location	Proteccio Client Software ISO イメージの HTTPS サーバーの場所を指定するユーザー名とパスワードを含む URL。
atos_client_cert_location	SSL クライアント証明書の HTTPS サーバーの場所を指定するユーザー名とパスワードを含む URL。
atos_client_key_location	SSL クライアントキーの HTTPS サーバーの場所を指定するユーザー名とパスワードを含む URL。これは、上記のクライアント証明書に一致するキーである必要があります。
atos_hsms	HSM の名前、証明書の場所、および IP アドレスを指定する 1 つ以上の HSM のリスト。このリストに複数の HSM を含めると、Barbican は負荷分散と高可用性向けに HSM を設定します。

2. デプロイメントコマンドおよびご自分のデプロイメントに該当するその他の環境ファイルと共に、カスタムの **configure-barbican.yaml**、**barbican.yaml**、および ATOS 固有の **barbican-backend-pkcs11-atos.yaml** 環境ファイルを追加します。

```
$ 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
```

検証

1. テストシークレットを作成します。

```
$ 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. 作成したシークレットのペイロードを取得します。

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

3.2. OPENSTACK KEY MANAGER (BARBICAN) と THALES LUNA NETWORK HSM の統合

ハードウェアベースの暗号化処理用に、PKCS#11 バックエンドを Thales Luna Network HSM アプライアンスと統合するには、Ansible ロールを使用して、コントローラーに Thales クライアントソフトウェアをダウンロードしてインストールします。また、Key Manager 設定ファイルを作成し、事前に定義された HSM IP および認証情報を追加します。

前提条件

- Thales Luna Network HSM にベンダーソフトウェアを提供する、パスワードで保護された HTTPS サーバー。
- ベンダーが提供する Luna Network HSM クライアントソフトウェアは、圧縮された zip アーカイブに用意されています。

手順

1. director に **ansible-role-lunasa-hsm** ロールをインストールします。

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

2. Key Manager (barbican) 用の **configure-barbican.yaml** 環境ファイルを作成し、ご自分の環境に固有のパラメーターを追加します。

```
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 パラメーター

パラメーター	値
BarbicanSimpleCryptoGlobalDefault	これは、simplecrypto がグローバルデフォルトであるかどうかを決定するブール値です。
BarbicanPkcs11GlobalDefault	これは、PKCS#11 がグローバルデフォルトであるかどうかを決定するブール値です。
BarbicanPkcs11CryptoTokenLabel	HSM が 1 つある場合、パラメーターの値はパーティションラベルです。2 つ以上のパーティション間で HA を使用している場合、これは HA グループに指定するラベルになります。
BarbicanPkcs11CryptoLogin	HSM 管理者が提供する HSM にログインするために使用される PKCS#11 パスワード。
LunasaVar	
lunasa_client_tarball_name	Luna ソフトウェア tarball の名前。
lunasa_client_tarball_location	Luna ソフトウェア tarball の HTTPS サーバーの場所を指定する URL。
lunasa_client_installer_path	zip 形式の tarball の install.sh スクリプトへのパス。
lunasa_client_rotate_cert	(オプション) true に設定すると、既存の証明書を置き換える新しいクライアント証明書が生成されます。デフォルト: false
lunasa_client_working_dir	(オプション) コントローラーノードの作業ディレクトリー。Default: /tmp/lunasa_client_install

パラメーター	値
lunasa_hsms	名前、hostname、admin_password、partition、およびパーティションシリアル番号を指定する 1 つ以上の HSM のリスト。このリストに複数の HSM を含めると、Barbican は高可用性向けに HSM を設定します。

3. デプロイメントコマンドにカスタムの **configure-barbican.yaml** および Thales 固有の **barbican-backend-pkcs11-llunasa.yaml** 環境ファイルと、デプロイメントに関連するその他のテンプレートを追加します。

```
$ 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. OPENSTACK KEY MANAGER (BARBICAN) と ENTRUST NSHIELD CONNECT XC のインテグレーション

PKCS#11 バックエンドを Entrust nShield Connect XC HSM と統合するには、Ansible ロールを使用して、コントローラーに Entrust クライアントソフトウェアをダウンロードしてインストールします。また、Barbican 設定ファイルを作成し、事前に定義された HSM IP および認証情報を追加します。

前提条件

- Entrust nShield Connect XC にベンダーソフトウェアを提供する、パスワードで保護される HTTPS サーバー。

手順

1. Barbican 用の **configure-barbican.yaml** 環境ファイルを作成し、お使いの環境に固有のパラメーターを追加します。以下のスニペットを例として使用します。

```
    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 パラメーター

パラメーター	値
BarbicanSimpleCryptoGlobalDefault	これは、**simplecrypto** がグローバルデフォルトであるかどうかを決定するブール値です。
BarbicanPkcs11GlobalDefault	これは、**PKCS#11** がグローバルデフォルトであるかどうかを決定するブール値です。
BarbicanPkcs11CryptoSlotId	Barbican によって使用される仮想 HSM のスロット ID。
BarbicanPkcs11CryptoMKEKLabel	このパラメーターは、HSM で生成された mKEK の名前を定義します。director はこのキーを使用して HSM にこのキーを作成します。
BarbicanPkcs11CryptoHMACLabel	このパラメーターは、HSM で生成された HMAC の名前を定義します。director はこのキーを使用して HSM にこのキーを作成します。
ThalesVars	
thales_client_working_dir	ユーザー定義の一時作業ディレクトリー。
thales_client_tarball_location	Entrust ソフトウェアの HTTPS サーバーの場所を指定する URL。

パラメーター	値
thales_km_data_tarball_name	Entrust ソフトウェア tarball の名前。
thales_rfs_key	RFS サーバーへの SSH 接続の取得に使用されるプライベートキー。これを承認されたキーとして RFS サーバーに追加する必要があります。

2. **barbican.yaml** および Thales 固有の **barbican-backend-pkcs11-thales.yaml** 環境ファイル、ならびに **openstack overcloud deploy** コマンドを実行する際にデプロイメントに必要なその他すべてのテンプレートに加えて、カスタムの **configure-barbican.yaml** 環境ファイルを追加します。

```
$ 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
```

検証

1. テストシークレットを作成します。

```
$ 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. 作成したシークレットのペイロードを取得します。

```
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. Entrust nShield Connect を使用した負荷分散

有効な HSM のアレイを指定することにより、Entrust nShield Connect HSM で負荷分散を有効にできるようになりました。複数の HSM がリストされている場合は、負荷分散が有効になります。

この機能は、本リリースではテクノロジープレビューとして提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。

テクノロジープレビュー機能の詳しい情報は、対象範囲の詳細 を参照してください。

手順

- Entrust nShield Connect HSM の **name** と **ip** パラメーターを設定するときに、複数を指定すると、負荷分散が有効になります。

```
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. MKEK 鍵および HMAC 鍵のローテーション

director の更新を使用して、MKEK キーおよび HMAC キーをローテーションできます。

注記

Barbican の制限により、MKEK と HMAC は同じキータイプです。

手順

1. デプロイメントの環境ファイルに以下のパラメーターを追加します。

```
BarbicanPkcs11CryptoRewrapKeys:         true
```

2. たとえば、ラベルが以下のようであれば、MKEK キーおよび HMAC 鍵のラベルを変更します。

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

値をインクリメントしてラベルを変更できます。

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

注記

HMAC 鍵タイプは変更しないでください。

3. director を使用して再デプロイし、更新を適用します。director は、MKEK および HMAC のラベルが付けられたキーが存在するかどうかを確認してから、作成します。また、**BarbicanPkcs11CryptoRewrapKeys** パラメーターが **True** に設定されていると、director は **barbican-manage hsm pkek_rewrap** を呼び出して、既存のすべての pKEK を再度ラップします。

第4章 OPENSTACK サービスの暗号化および検証

barbican を使用して、Block Storage (cinder) 暗号化鍵、Block Storage ボリュームイメージ、Object Storage (swift) オブジェクト、および Image サービス (glance) イメージなどのさまざまな Red Hat OpenStack Platform サービスを暗号化および検証することができます。

重要

Nova は、暗号化されていない場合に初回使用時に暗号化されたボリュームをフォーマットします。作成されるブロックデバイスは、コンピュートノードに提示されます。

コンテナー化されたサービスに対するガイドライン

- 物理ノードのホストオペレーティングシステム上の設定ファイル (例: **/etc/cinder/cinder.conf**) は更新しないでください。コンテナー化されたサービスはこのようなファイルを参照しません。
- コンテナー内で実行されている設定ファイルは更新しないでください。コンテナーを再起動すると、変更が失われます。
  代わりに、コンテナー化されたサービスを変更する必要がある場合は、コンテナーの生成に使用される **/var/lib/config-data/puppet-generated/** の設定ファイルを更新します。

  以下に例を示します。

  - 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**

  変更は、コンテナーを再起動した後に適用されます。

4.1. 保存中の OBJECT STORAGE (SWIFT) オブジェクトの暗号化

デフォルトでは、Object Storage (swift) にアップロードしたオブジェクトは暗号化されず保管されます。したがって、ファイルシステムからオブジェクトに直接アクセスすることが可能です。このため、ディスクを破棄する前に適切に消去しなかった場合には、セキュリティーリスクとなってしまいます。barbican を有効にすると、Object Storage サービス (swift) は、保管されている (at-rest) オブジェクトを透過的に暗号化および復号化できます。at-rest 暗号化は、in-transit 暗号化とは異なり、ディスクに保管されている間にオブジェクトが暗号化されることを指します。

Swift はこれらの暗号化タスクを透過的に実行し、オブジェクトは swift にアップロードされる際には自動的に暗号化され、ユーザーに提供される際には自動的に復号化されます。この暗号化と復号化は、Barbican に保管されている同じ (対称) キーを使用して処理されます。

注記

データが暗号化された状態で保存されているので、暗号化を有効にし、swift クラスターにデータを追加した後には暗号化を無効にすることはできません。その結果、同じキーで暗号化を再度有効にするまで、暗号化が無効になっている場合は、データは読み取りできなくなります。

前提条件

- OpenStack Key Manager がインストールされ、有効である。

手順

1. 環境ファイルに **SwiftEncryptionEnabled: True** パラメーターを追加してから、**/home/stack/overcloud_deploy.sh** を使用して **openstack overcloud deploy** を再実行します。

2. swift が at-rest 暗号化を使用するように設定されていることを確認します。

```
$ 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
```

結果には、**encryption** のエントリーが含まれている必要があります。

4.2. BLOCK STORAGE (CINDER) ボリュームの暗号化

barbican を使用して Block Storage (cinder) の暗号化キーを管理することができます。この設定は、LUKS を使用して、インスタンスに接続されているディスク (ブートディスクを含む) を暗号化します。キー管理はユーザーに透過的です。暗号化種別として **luks** を使用して新規ボリュームを作成すると、cinder はボリュームの対称キーシークレットを生成し、それを barbican に保存します。インスタンスをブート (または暗号化されたボリュームをアタッチ) する場合は、nova はキーを barbican から取得して、そのシークレットをコンピュートノード上の Libvirt シークレットとしてローカルに保存します。

手順

1. **cinder-volume** および **nova-compute** サービスを実行しているノードで、nova と cinder の両方がキー管理に barbican を使用するように設定されていることを確認します。

```
$ 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. 暗号化を使用するボリュームテンプレートを作成します。新しいボリュームを作成すると、設定からモデル化できます。

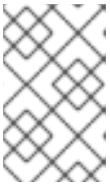
```
$ openstack volume type create --encryption-provider nova.volume.encryptors.luks.LuksEncryptor --encryption-cipher aes-xts-plain64 --encryption-key-size 256 --encryption-control-location front-end LuksEncryptor-Template-256
+-------------+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| 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. 新しいボリュームを作成して、**LuksEncryptor-Template-256** 設定を使用するように指定します。

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

生成されるシークレットは barbican バックエンドに自動的にアップロードされます。



注記

暗号化されたボリュームを作成するユーザーに、プロジェクトで **creator** barbican ロールがあることを確認します。詳細は、**creator ロールへのユーザーアクセスの付与** セクションを参照してください。

4. barbican を使用して、ディスクの暗号化キーが存在することを確認します。この例では、タイムスタンプが LUKS ボリュームの作成時間と一致します。

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

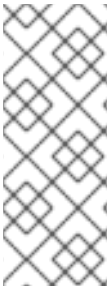
5. 新規ボリュームを既存のインスタンスにアタッチします。以下に例を示します。

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

その後、ボリュームはゲストオペレーティングシステムに表示され、組み込みツールを使用してマウントできます。

4.2.1. Block Storage ボリュームの OpenStack Key Manager への移行

以前に **ConfKeyManager** を使用してディスク暗号化鍵を管理していた場合は、データベースをスキャンして、barbican への移行のスコープ内に **encryption_key_id** エントリーの有無を確認し、ボリュームを OpenStack Key Manager に移行できます。各エントリーは新しい barbican キー ID を取得し、既存の **ConfKeyManager** シークレットが保持されます。



注記

- 以前は、**ConfKeyManager** を使用して暗号化されたボリュームの所有権を再割り当てすることができました。これは、barbican が管理するキーを持つボリュームにはできません。
- barbican をアクティベートすると、既存の **keymgr** ボリュームが破損しません。

前提条件

移行前に、Barbican が管理する暗号化ボリュームと、**ConfKeyManager** を使用するボリューム間の以下の違いを確認してください。

- 現在、barbican シークレットの所有権を転送することができないため、暗号化されたボリュームの所有権は移動できません。
- barbican は、シークレットの読み取りと削除ができるかより厳しいので、一部の cinder ボリューム操作に影響を及ぼす可能性があります。たとえば、別のユーザーのボリュームの割り当て、割り当て解除、または削除はできません。

手順

1. barbican サービスをデプロイします。
2. **creator** ロールを cinder サービスに追加します。以下に例を示します。

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

3. **cinder-volume** および **cinder-backup** サービスを再起動します。**cinder-volume** および **cinder-backup** サービスは、移行プロセスを自動的に開始します。ログファイルを確認して、移行についてのステータス情報を表示できます。

   - **cinder-volume**: cinder の Volumes および Snapshots テーブルに保存される鍵を移行します。
   - **cinder-backup**: Backups テーブルの鍵を移行します。

4. 移行が完了したことを示すメッセージのログを監視し、ボリュームが **ConfKeyManager** オールゼロ暗号鍵 ID を使用していないことを確認します。

5. **cinder.conf** および **nova.conf** から **fixed_key** オプションを削除します。この設定が設定されているノードを判別する必要があります。

6. cinder サービスから **creator** ロールを削除します。

検証

- プロセスを起動すると、これらのエントリーの 1 つがログファイルに表示されます。これは、移行が正しく開始するか、発生した問題を特定するかどうかを示します。
  - **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.** - このメッセージが表示されることはほとんどありません。barbican 以外の別の Key Manager バックエンドに遭遇していたコードを処理するための安全チェックです。これは、コードが 1 つの移行シナリオ (ConfKeyManager **から** barbican **へ**) のみをサポートする ためです。
  - **Not migrating encryption keys because there are no volumes associated with this host.** - これは、 **cinder-volume** が複数のホストで実行され、特定のホストにボリュームが関連付けられていない場合に生じる可能性があります。これは、すべてのホストが独自のボリュームを処理するため発生します。
  - **Starting migration of ConfKeyManager keys.**
  - **Migrating volume <UUID> encryption key to Barbican** - 移行時に、ホストのボリュームがすべて検証され、ボリュームが ConfKeyManager のキー ID を使用している場合に (すべてがゼロ (**00000000-0000-0000-0000-000000000000**) であることで確認)、このメッセージが表示されます。
    - **cinder-backup** では、このメッセージは若干異なる大文字を使用します。**Migrating Volume [...]** または **Migrating Backup [...]**
- 各ホストがすべてのボリュームを検査すると、ホストはサマリーステータスメッセージを表示します。

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

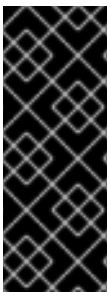
- 以下のエントリーも表示される場合があります。
  - **There are still %d volume(s) using the ConfKeyManager's all-zeros encryption key ID.**
  - **There are still %d backup(s) using the ConfKeyManager's all-zeros encryption key ID.**
    これらのメッセージはいずれも **cinder-volume** および **cinder-backup** ログに表示される場合があります。各サービスは独自のエントリーの移行のみを処理しますが、サービスは他のステータスを認識します。これにより、**cinder-volume** は **cinder-backup** に移行するバックアップがまだあるかどうかを認識し、**cinder-backup** は **cinder-volume** サービスに移行するボリュームがあるかどうかを認識します。

各ホストは自身のボリュームのみを移行しますが、概要メッセージは、ボリュームの移行が依然として必要なかどうかについて、グローバルアセスメントに基づいています。これにより、すべてのボリュームの移行が完了しているかどうかを確認できます。

クリーンアップ

キー ID を barbican に移行すると、固定キーが設定ファイル内に残ります。**fixed_key** の値が **.conf** ファイルで暗号化されないため、一部のユーザーにセキュリティー上の問題が発生することがあります。

これに対処するには、nova および cinder の設定から **fixed_key** の値を手動で削除します。ただし、最初にログファイルの出力が完了してから、続行する前にログファイルの出力を確認します。この値がまだ依存するディスクにはアクセスできないためです。



重要

最近 **encryption_key_id** は、Queens リリースの一環として、**Backup** テーブルにのみ追加されました。その結果、暗号化されたボリュームの既存のバックアップが存在する可能性があります。すべてがゼロの **encryption_key_id** はバックアップ自体に保存されますが、**Backup** データベースには表示されません。そのため、移行プロセスでは、暗号化されたボリュームのバックアップが存在し、all-zero の **ConfKeyMgr** キー ID に依存するかを知ることはできません。

1. 既存の **fixed_key** の値を確認します。値は、両方のサービスと一致している必要があります。

```
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

既存の **fixed_key** の値のバックアップを作成します。これにより、何らかの問題が発生した場合や、古い暗号鍵を使用するバックアップを復元する必要がある場合に、値を復元できます。

2. **fixed_key** の値を削除します。

```
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
```

トラブルシューティング

barbican シークレットは、要求に **creator** ロールがある場合にのみ作成できます。これは、cinder サービス自体が作成者ロールが必要なことを意味します。それ以外の場合は、以下のようなログシーケンスが発生します。

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.**

鍵に関するメッセージは、3 番目の **Secret creation attempt not allowed** です。問題を修正するには、**cinder** アカウントの権限を更新します。

1. **openstack role add --project service --user cinder creator** を実行します。
2. **cinder-volume** および **cinder-backup** サービスを再起動します。

その結果、次の移行試行に成功します。

4.3. BLOCK STORAGE (CINDER) ボリュームイメージの検証

Block Storage サービス (cinder) は、イメージの作成時に、ダウンロードされたイメージの署名を自動的に検証します。署名は、イメージがボリュームに書き込まれる前に検証されます。パフォーマンスを向上させるために、Block Storage Image-Volume キャッシュを使用して、新規ボリュームを作成するための検証済みイメージを保存できます。

注記

cinder イメージの署名の検証は、Red Hat Ceph Storage または RBD ボリュームではサポートされません。

手順

1. コントローラーノードにログインします。
2. 以下のいずれかのオプションを選択します。
   - **Volume** ログ **/var/log/containers/cinder/cinder-volume.log** で cinder のイメージ検証アクティビティーを表示します。
     たとえば、インスタンスの起動時に以下のエントリーが表示されるはずです。

     ```
     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
     ```

   - **openstack volume list** および **cinder volume show** コマンドを使用します。
     a. **openstack volume list** コマンドを使用して、ボリューム ID を見つけます。
     b. コンピュートノードで **cinder volume show** コマンドを実行します。

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

3. **volume_image_metadata** セクションで **signature verified : True** の行を探します。

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

注記

スナップショットは、Image サービス (glance) イメージとして保存されます。Compute サービス (nova) が署名済みのイメージを確認するように設定する場合には、glance からイメージを手動でダウンロードし、そのイメージに署名してからイメージを再アップロードする必要があります。これは、スナップショットが署名済みイメージで作成されたインスタンスから、または署名済みイメージから作成されたボリュームから起動したインスタンスの場合に該当します。

注記

ボリュームは、Image サービス (glance) イメージとしてアップロードすることができます。元のボリュームがブート可能な場合には、このイメージを使用して Block Storage サービス (cinder) でブート可能なボリュームを作成できます。署名済みイメージを確認するように Block Storage サービスを設定している場合は、イメージを使用する前に、glance からイメージを手動でダウンロードし、イメージ署名の算出を行い、適切なイメージ署名属性をすべて更新する必要があります。詳細は、「スナップショットの検証」 を参照してください。

関連情報

- Block Storage サービス (cinder) の設定

4.3.1. ボリュームイメージ暗号化キーの自動削除

Block Storage サービス (cinder) が暗号化されたボリュームを Image サービス (glance) にアップロードする際に、Key Management サービス (barbican) に暗号鍵を作成します。これにより、暗号鍵と保存されるイメージに 1 対 1 の関係が形成されます。

暗号鍵を削除することで、Key Management サービスがリソースを無制限に消費するのを防ぐことができます。Block Storage サービス、Key Management サービス、および Image サービスは、暗号化されたボリュームの鍵を自動的に管理します。これには、鍵の削除が含まれます。

Block Storage サービスは、自動的に 2 つの属性をボリュームイメージに追加します。

- **cinder_encryption_key_id**: Key Management サービスが特定のイメージ用に保存する暗号鍵の識別子
- **cinder_encryption_key_deletion_policy**: Image サービスはこのポリシーにしたがって、このイメージに関連付けられた鍵を削除するかどうかを Key Management サービスに指示します。

重要

これらの属性の値は、自動的に割り当てられます。**意図しないデータ損失を避けるため、これらの値を調整しないでください。**

ボリュームイメージを作成すると、Block Storage サービスは **cinder_encryption_key_deletion_policy** 属性を **on_image_deletion** に設定します。**cinder_encryption_key_deletion_policy** が **on_image_deletion** に設定されている場合、ボリュームイメージを削除すると、Image サービスは対応する暗号鍵を削除します。

重要

Red Hat では、**cinder_encryption_key_id** または **cinder_encryption_key_deletion_policy** 属性を手動で操作することを推奨しません。**cinder_encryption_key_id** の値で識別される暗号鍵を他の目的で使用すると、データが失われる危険性があります。

4.4. IMAGE SERVICE (GLANCE) イメージの署名

アップロードされたイメージが改ざんされていないことを確認するように Image Service (glance) を設定する場合、それらのイメージを使用してインスタンスを開始する前に、イメージに署名する必要があります。**openssl** コマンドを使用して、barbican に保存されているキーを使用してイメージに署名し、

イメージをアップロードして、付随する署名情報を一目で確認します。その結果、各使用前にイメージの署名が検証され、署名が一致しない場合、インスタンスのビルドプロセスに失敗しています。

前提条件

- OpenStack Key Manager がインストールされ、有効である。

手順

1. 環境ファイルで、**VerifyGlanceSignatures: True** の設定でイメージの検証を有効にします。この設定を有効にするには、**openstack overcloud deploy** コマンドを再実行する必要があります。
2. glance イメージの検証が有効化されていることを確認するには、オーバークラウドのコンピュートノードで以下のコマンドを実行します。

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

注記

イメージおよび Compute サービスのバックエンドに Ceph を使用する場合、CoW クローンが作成されます。したがって、イメージ署名の検証は実行できません。

3. glance が barbican を使用するように設定されていることを確認します。

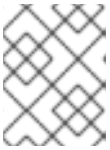
```
$ sudo crudini --get /var/lib/config-data/puppet-generated/glance_api/etc/glance/glance-
api.conf key_manager backend
castellan.key_manager.barbican_key_manager.BarbicanKeyManager
```

4. 証明書を生成します。

```
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. barbican のシークレットストアに証明書を追加します。

```
$ 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
```



注記

後のステップで使用するために、生成された UUID を記録します。以下の例では、証明書の UUID は **5df14c2b-f221-4a02-948e-48a61edd3f5b** です。

6. **private_key.pem** を使用してイメージに署名し、**.signature** ファイルを生成します。以下に例を示します。

```
$ 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. 作成される **.signature** ファイルを **base64** 形式に変換します。

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

8. 後続のコマンドで **base64** 値を変数に読み込みます。

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

9. 署名付きイメージを glance にアップロードします。**img_signature_certificate_uuid** の場合、前のステップで barbican にアップロードした署名キーの UUID を指定する必要があります。

```
 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 |
|                                 |
lFquV/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. glance のイメージ検証のアクティビティーを次の Compute ログで表示することができます。**/var/log/containers/nova/nova-compute.log**たとえば、インスタンスの起動時に以下のエントリーが表示されるはずです。

```
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. スナップショットの検証

スナップショットは、Image サービス (glance) イメージとして保存されます。Compute サービス (nova) が署名済みイメージをチェックするように設定した場合には、署名されたイメージを持つインスタンスからスナップショットが作成された場合でも、スナップショットは署名する必要があります。

手順

1. glance からスナップショットをダウンロードする

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

2. 署名を生成してスナップショットを検証します。これは、イメージの検証に署名を生成する時に使用するプロセスと同じです。詳細は、Validating Image Service (glance) images を参照してください。

3. イメージの属性を更新します。

```
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. (オプション) ダウンロードした glance イメージをファイルシステムから削除します。

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