Red Hat OpenStack Platform 13

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: 2023-04-10

Red Hat OpenStack Platform 13 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

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。

第1章 概要

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。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。

本リリースでは、barbican は Block Storage (cinder) および Compute (nova) コンポーネントとの統合を提供します。

第2章 バックエンドの選択

シークレット (証明書、API キー、パスワードなど) は、暗号化された Blob として barbican データベースに保存することも、セキュアなストレージシステムに直接保存することもできます。

シークレットを暗号化された Blob として barbican データベースに保管するには、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シークレットのブロブを暗号化します。このキーは、プレーンテキストで **barbican.conf** ファイルに保存されます。

> **注記**
>
> simple crypto プラグインは、現在 Red Hat がサポートしている唯一のプラグインです。

- **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キー暗号化キー (KEK) 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 KEK は、ハードウェアセキュリティーモジュール (HSM) に格納されているマスター KEK によって暗号化されます。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 プロトコルを介して 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 KEK 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。

> **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。

2.1. バックエンド間の移行

Barbican を使用すると、プロジェクトに異なるバックエンドを定義することができます。プロジェクトにマッピングが存在しない場合は、シークレットはグローバルのデフォルトバックエンドに保存されます。つまり、複数のバックエンドを設定することは可能ですが、少なくとも 1 つのグローバルバックエンドが定義されている必要があります。異なるバックエンド用に提供された heat テンプレートには、各バックエンドをデフォルトとして設定するパラメーターが含まれます。

特定のバックエンドにシークレットを保存してから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間に、古い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引き続き利用できます。

第3章 BARBICAN のインストール

Red Hat OpenStack Platform では、barbican はデフォルトで有効になっていません。以下の手順では、既存の OpenStack デプロイメントに barbican をデプロイする方法について説明します。barbican はコンテナー化されたサービスとして実行されるため、この手順では新しいコンテナーイメージの準備およびアップロード方法についても説明します。

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。**PKCS11** や DogTag などの追加のバックエンドも提供されていますが、このリリースではサポートされていません。

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。これにより、barbican をインストールするように director に指示します (**openstack overcloud deploy [...]** に含まれている場合)。

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

- **BarbicanSimpleCryptoGlobalDefault** - 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。
- その他のオプションも設定可能です。
  - **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. このステップでは、barbican 用の新しいコンテナーイメージを準備します。**configure-barbican.yaml** と関連するすべてのテンプレートファイルを含める必要があります。デプロイメントに合わせて次の例を変更します。

```
$ openstack overcloud container image prepare \
 --namespace example.lab.local:5000/rhosp13 \
 --tag 2018-06-06.1 \
 --push-destination 192.168.100.1:8787 \
 --output-images-file ~/container-images-with-barbican.yaml \
 -e /home/stack/virt/config_lvm.yaml \
 -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
 -e /home/stack/virt/network/network-environment.yaml \
 -e /home/stack/virt/hostnames.yml \
 -e /home/stack/virt/nodes_data.yaml \
```

```
 -e /home/stack/virt/extra_templates.yaml \
 -e /home/stack/virt/docker-images.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/configure-barbican.yaml
```

3. 新しいコンテナーイメージをアンダークラウドレジストリーにアップロードします。

```
$ openstack overcloud container image upload --debug --config-file container-images-with-barbican.yaml
```

4. 新しい環境ファイルを準備します。

```
$ openstack overcloud container image prepare \
  --tag 2018-06-06.1 \
  --namespace 192.168.100.1:8787/rhosp13 \
  --output-env-file ~/container-parameters-with-barbican.yaml \
  -e /home/stack/virt/config_lvm.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /home/stack/virt/network/network-environment.yaml \
  -e /home/stack/virt/hostnames.yml \
  -e /home/stack/virt/nodes_data.yaml \
  -e /home/stack/virt/extra_templates.yaml \
  -e /home/stack/virt/docker-images.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/configure-barbican.yaml
```

5. これらの変更をデプロイメントに適用するには、オーバークラウドを更新し、先ほど **openstack overcloud deploy [...]** で使用したすべての heat テンプレートファイルを指定します。以下に例を示します。

```
$ 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/virt/config_lvm.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
    -e /home/stack/virt/network/network-environment.yaml \
    -e /home/stack/virt/hostnames.yml \
    -e /home/stack/virt/nodes_data.yaml \
    -e /home/stack/virt/extra_templates.yaml \
    -e /home/stack/container-parameters-with-barbican.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/services/barbican.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/barbican-backend-simple-crypto.yaml \
    -e /home/stack/configure-barbican.yaml \
    --log-file overcloud_deployment_38.log
```

3.1. オーバークラウドの作成者ロールへのユーザーの追加

ユーザーは、barbican シークレットの作成および編集、またはシークレットを barbican に保存する暗号化されたボリュームを作成するには、**creator** ロールのメンバーである必要があります。

1. 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。

2. 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
```

3.1.1. barbican 機能のテスト

本セクションでは、barbican が正常に動作していることをテストする方法を説明します。

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

3.2. ポリシーについて

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、keystone プロジェクトロール (以前に作成した **creator** など) 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。

3.2.1. デフォルトポリシーの表示

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。**barbican** ソースコードから生成することで、デフォルトポリシーを表示できます。

1. 追加のコンポーネントをダウンロードしてインストールする可能性があるため、非実稼働システムで次の手順を実行します。この例では、**queens** ブランチに切り替えるため、別のバージョンを使用する場合はこれを調整する必要があります。

   ```
   git clone https://github.com/openstack/barbican
   cd /home/stack/barbican
   git checkout origin/stable/queens
   tox -e genpolicy
   ```

   これにより、サブディレクトリー内にデフォルト設定を含むポリシーファイル **etc/barbican/policy.yaml.sample** が生成されます。このパスは、システムの **/etc** ディレクトリーではなく、リポジトリー内のサブディレクトリーを参照することに注意してください。このファイルのコンテンツは、以下の手順で説明します。

2. 生成した **policy.yaml.sample** ファイルには、barbican で使用されるポリシーが記述されています。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**: シークレットの削除、作成/編集、および読み取りを行うことができます。
   - **creator**: シークレットの作成/編集および読み取りが可能です。シークレットを削除できません。
   - **observer**: データの読み取りのみが可能です。
   - **audit**: メタデータのみを読み取ることができます。シークレットを読み取りできません。
     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 ロールを一覧表示します。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。

3. 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4章 BARBICAN でのシークレットの管理

4.1. シークレットの一覧表示

シークレットは URI によって識別されます。これは href の値として示されます。以下の例では、直前の手順で作成したシークレットを示しています。

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

4.2. 新規シークレットの追加

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://192.168.123.163:9311/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                                                                              |
+---------------+-----------------------------------------------------------------------------------+
```

4.3. シークレットの更新

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 **update** 関数を使用してペイロードを追加することができます。以下に例を示します。

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

4.4. シークレットの削除

URI を指定してシークレットを削除できます。以下に例を示します。

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

4.5. 対称鍵の生成

対称鍵は、nova のディスク暗号化や swift オブジェクトの暗号化など、特定のタスクに適しています。

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**: 生成された鍵は、**ctr** または **cbc** などの特定のモードを使用するように設定できます。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                                                                              |
+---------------+-----------------------------------------------------------------------------------+
```

4.6. バックアップおよびリストアキー

暗号化キーのバックアップおよびリストアのプロセスは、バックエンドのタイプによって異なります。

4.6.1. シンプルな暗号化バックエンドをバックアップおよび復元します。

シンプルな暗号化 バックエンドには、2 つの異なるコンポーネント (KEK とデータベース) のバックアップが必要です。バックアップおよび復元プロセスを定期的にテストすることが推奨されます。

4.6.1.1. KEK のバックアップおよび復元

シンプルな暗号化 バックエンドの場合、マスター KEK が書き込まれている **barbican.conf** ファイルをバックアップする必要があります。このファイルは、セキュリティーが強化された場所にバックアップされる必要があります。実際のデータは、次のセクションで説明されているように、Barbican データベースに暗号化された状態に保存されます。

- バックアップから鍵を復元するには、復元された **barbican.conf** を既存の **barbican.conf** にコピーする必要があります。

4.6.1.2. バックエンドデータベースのバックアップおよびリストア

この手順では、簡単な暗号化バックエンド向けに、barbican データベースをバックアップおよび復元する方法を説明します。これは、キーを生成し、シークレットを barbican にアップロードします。その後、barbican データベースのバックアップを作成し、作成したシークレットを削除します。次に、データベースを復元し、先に作成したシークレットが復元されていることを確認します。

注記

これは重要な要件であるため、KEK もバックアップするようにしてください。これは、前のセクションで説明します。

4.6.1.2.1. テストシークレットの作成

1. オーバークラウドで **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。

```
(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. シークレットが作成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       |
+------------------------------------------------------------------------+------------+---------------------------
---+--------+-----------------------------------------+-----------+------------+-------------+------+------
------+
```

4.6.1.2.2. barbican データベースのバックアップ

controller-0 ノードにログインした状態で以下の手順を実行します。

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。

1. **barbican** ユーザーのパスワードを取得します。以下に例を示します。

```
[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",
```

2. **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
```

3. データベースのバックアップは **/home/heat-admin**に保存されます。

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

4.6.1.2.3. テストシークレットの削除

1. 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。以下に例を示します。

```
(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 ~]$
```

4.6.1.2.4. データベースを復元します。

controller-0 ノードにログインした状態で以下の手順を実行します。

1. 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 ~]$
```

9) バックアップファイルを **barbican** データベースに戻します。

+

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

4.6.1.2.5. 復元プロセスの確認

1. 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5章 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。

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。

注記

設定ファイルを更新する場合には、特定の OpenStack サービスはコンテナー内で実行されるようになったことを認識する必要があります。これは、keystone、nova、cinder などのサービスが対象です。その結果、考慮すべき管理プラクティスがいくつかあります。

- 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。

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。

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。

```
$ 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。

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。

5.1. 既存のボリューム鍵の BARBICAN への移行

以前のバージョンでは、デプロイメントでディスク暗号化キーの管理に **ConfKeyManager** が使用される可能性がありました。そのため、固定キーが生成され、nova および cinder の設定ファイルに保存されていました。以下の手順で、キー ID を barbican に移行することができます。このユーティリティーは、barbican への移行についてスコープ内の **encryption_key_id** エントリーのデータベースをスキャンすることで機能します。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。

注記

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。

有効な場合、移行プロセスは自動的に実行されますが、次のセクションで説明されているように、一部の設定が必要になります。実際の移行は **cinder-volume** および **cinder-backup** プロセスで実行され、cinder ログファイル内の進捗を追跡できます。

- **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。

5.1.1. 移行手順の概要

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。
4. **cinder-volume** および **cinder-backup** は、移行プロセスを自動的に開始します。
5. 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。
6. **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。
7. cinder サービスから **creator** ロールを削除します。

5.1.2. 動作の違い

barbican が管理する暗号化ボリュームは、**ConfKeyManager** を使用するボリュームとは異なる動作をします。

- 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。

5.1.3. 移行プロセスの確認

本セクションでは、移行タスクのステータスを表示する方法を説明します。プロセスを起動すると、これらのエントリーの 1 つがログに表示されます。これは、移行が正しく開始するか、または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る点に注意してください。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。確認が終わったら、**fixed_key** の設定を **cinder.conf** および **nova.conf** から削除します。詳細は、Clean up the fixed keys のセクションを参照してください。

5.1.4. 移行プロセスのトラブルシューティング

5.1.4.1. ロール割り当て

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。

5.1.5. 固定キーの削除

> **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。

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。

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
```

2. 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。

3. **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
```

第6章 保存されている SWIFT オブジェクトの暗号化

デフォルトでは、オブジェクトストレージにアップロードされるオブジェクトは暗号化されずに保存されます。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。

6.1. SWIFT 用の AT-REST 暗号化の有効化

1. 環境ファイルに **SwiftEncryptionEnabled: True** を追加し、**/home/stack/overcloud_deploy.sh** を使用して **openstack overcloud deploy** を再実行することで、swift 暗号化機能を有効にすることができます。**Barbican のインストール**の章で説明されているように、barbican を有効にする必要があります。

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。

第7章 GLANCE イメージの検証

Barbican を有効にした後に、Image サービス (glance) を設定して、アップロードしたイメージが改ざんされていないことを確認できます。この実装では、イメージは最初に barbican に保管されるキーで署名されます。その後、イメージは、付随する署名情報と共に glance にアップロードされます。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。

barbican と glance の統合とは、**openssl** コマンドを秘密鍵と共に使用してアップロードする前に glance イメージを署名できることを意味します。

7.1. GLANCE イメージ検証の有効化

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。

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。

7.2. イメージの検証

検証用の glance イメージを設定するには、以下の手順を実施します。

1. 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
```

2. 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
```

3. 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。

4. **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
```

5. 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
```

6. 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)
```

7. 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                       | 25c96942084f61e008d593b6c2cfda00
|
| container_format               | bare                                                                 |
| created_at                     | 2018-01-23T05:37:31Z                                                 |
| disk_format                    | qcow2                                                                |
| id                             | d3396fa0-2ea2-4832-8a77-d36fa3f2ab27                                 |
| img_signature                  |
lcI7nGgoKxnCyOcsJ4abbEZEpzXByFPIgiPeiT+Otjz0yvW00KNN3fI0AA6tn9EXrp7fb2xBDE4Ua
O3v |
|                                |
IFquV/s3mU4LcCiGdBAl3pGsMlmZZIQFVNcUPOaayS1kQYKY7kxYmU9iq/AZYyPw37KQI52s
mC/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                         | active                                                                         |
| tags                           | []                                                                             |
| updated_at                     | 2018-01-23T05:37:31Z                                                           |
| virtual_size                   | None                                                                           |
| visibility                     | shared                                                                         |
+--------------------------------+--------------------------------------------------------------------------------
----+
```

8. glance のイメージ検証のアクティビティーを 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
```