Red Hat OpenStack Platform 15

コンテナー化されたサービスへの移行

コンテナー化された OpenStack Platform サービスの操作に関する基本ガイド

Last Updated: 2022-09-15

Red Hat OpenStack Platform 15 コンテナー化されたサービスへの移行

コンテナー化された OpenStack Platform サービスの操作に関する基本ガイド

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

本ガイドは、コンテナーで実行される OpenStack Platform サービスの操作に慣れるのに役立つ基本情報をユーザーに提供します。

目次

第1章 はじめに

以前のバージョンの Red Hat OpenStack Platform は、Systemd の管理するサービスを使用していました。しかし、より新しいバージョンの OpenStack Platform は、コンテナーを使用してサービスを実行するようになりました。コンテナー化された OpenStack Platform サービスがどのように動作するか、理解が十分ではない管理者もいます。本ガイドの目的は、OpenStack Platform のコンテナーイメージおよびコンテナー化されたサービスを理解するのに役立つ情報を提供することです。ここでは、以下の点について説明します。

- コンテナーイメージを取得および変更する方法
- コンテナー化されたサービスをオーバークラウドで管理する方法
- コンテナーと Systemd サービスの相違点の理解

本書の主目的は、Systemd ベースの環境からコンテナーベースの環境に移行するために、コンテナー化された OpenStack Platform サービスに関する十分な知識を習得するのに役立つ情報を提供することです。

1.1. コンテナー化されたサービスおよび KOLLA

Red Hat OpenStack Platform の各主要サービスは、コンテナー内で実行されます。このことにより、それぞれのサービスが、ホストから独立した専用の分離名前空間内に維持されます。この構成には、以下のような特徴があります。

- Red Hat カスタマーポータルからコンテナーイメージをプルして実行することで、サービスのデプロイメントが実施される。
- **podman** コマンドを実行し、サービスの起動/停止などの管理機能を操作する。
- コンテナーをアップグレードするには、新しいコンテナーイメージをプルし、既存のコンテナーを新しいバージョンのコンテナーに置き換える必要がある。

Red Hat OpenStack Platform は、**kolla** ツールセットによりビルド/管理されるコンテナーセットを使用します。

第2章 コンテナーイメージの取得および変更

コンテナー化されたオーバークラウドには、必要なコンテナーイメージを含むレジストリーへのアクセスが必要です。本章では、Red Hat OpenStack Platform 向けのコンテナーイメージを使用するためのレジストリーおよびアンダークラウドとオーバークラウドの設定の準備方法について説明します。

2.1. コンテナーイメージの準備

オーバークラウドの設定には、イメージの取得先およびその保存方法を定義するための初期レジストリーの設定が必要です。コンテナーイメージを準備するための環境ファイルを生成およびカスタマイズするには、以下の手順を実施します。

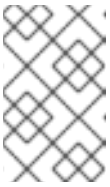
手順

1. アンダークラウドホストに stack ユーザーとしてログインします。
2. デフォルトのコンテナーイメージ準備ファイルを生成します。

```
$ openstack tripleo container image prepare default \
  --local-push-destination \
  --output-env-file containers-prepare-parameter.yaml
```

上記のコマンドでは、以下の追加オプションを使用しています。

- **--local-push-destination**: コンテナーイメージの保管場所として、アンダークラウド上のレジストリーを設定します。つまり、director は必要なイメージを Red Hat Container Catalog からプルし、それをアンダークラウド上のレジストリーにプッシュします。director はこのレジストリーをコンテナーイメージのソースとして使用します。Red Hat Container Catalog から直接プルする場合には、このオプションを省略します。
- **--output-env-file**: 環境ファイルの名前です。このファイルには、コンテナーイメージを準備するためのパラメーターが含まれます。ここでは、ファイル名は **containers-prepare-parameter.yaml** です。



注記

アンダークラウドとオーバークラウド両方のコンテナーイメージのソースを定義するのに、同じ **containers-prepare-parameter.yaml** ファイルを使用することができます。

3. **containers-prepare-parameter.yaml** を編集し、要求に合わせて変更を加えます。

2.2. コンテナーイメージ準備のパラメーター

コンテナー準備用のデフォルトファイル (**containers-prepare-parameter.yaml**) には、**ContainerImagePrepare** Heat パラメーターが含まれます。このパラメーターで、イメージのセットを準備するためのさまざまな設定を定義します。

```
parameter_defaults:
  ContainerImagePrepare:
  - (strategy one)
```

```
- (strategy two)
- (strategy three)
...
```

それぞれの設定では、サブパラメーターのセットにより使用するイメージやイメージの使用方法を定義することができます。以下の表には、**ContainerImagePrepare** の各設定で使用することのできるサブパラメーターの情報をまとめています。

パラメーター	説明
excludes	設定から除外するイメージの名前に含まれる文字列のリスト
includes	設定に含めるイメージの名前に含まれる文字列のリスト。少なくとも 1 つのイメージ名が既存のイメージと一致している必要があります。**includes** パラメーターを指定すると、**excludes** の設定はすべて無視されます。
modify_append_tag	対象となるイメージのタグに追加する文字列。たとえば、**14.0-89** のタグが付けられたイメージをプルし、**modify_append_tag** を **-hotfix** に設定すると、director は最終イメージを **14.0-89-hotfix** とタグ付けします。
modify_only_with_labels	変更するイメージを絞り込むイメージラベルのディクショナリー。イメージが定義したラベルと一致する場合には、director はそのイメージを変更プロセスに含めます。
modify_role	イメージのアップロード中 (ただし目的のレジストリーにプッシュする前) に実行する Ansible ロール名の文字列
modify_vars	**modify_role** に渡す変数のディクショナリー
push_destination	アップロードプロセス中にイメージをプッシュするレジストリーの名前空間。このパラメーターで名前空間を指定すると、すべてのイメージパラメーターでもこの名前空間が使用されます。**true** に設定すると、**push_destination** はアンダークラウドレジストリーの名前空間に設定されます。実稼働環境では、このパラメーターを **false** に設定することは推奨されません。これが **false** に設定されている (または何も設定されていない) 時にリモートレジストリーで認証が必要な場合は、**ContainerImageRegistryLogin** パラメーターを **true** に設定し、**ContainerImageRegistryCredentials** パラメーターで認証情報を提供します。

パラメーター	説明
pull_source	元のコンテナーイメージをプルするソースレジストリー
set	初期イメージの取得場所を定義する、**キー:値** 定義のディクショナリー
tag_from_label	得られたイメージをタグ付けするラベルパターンを定義します。通常は、**{version}-{release}** に設定します。

set パラメーターには、複数の **キー:値** 定義を設定することができます。以下の表には、キーの情報をまとめています。

キー	説明
ceph_image	Ceph Storage コンテナーイメージの名前
ceph_namespace	Ceph Storage コンテナーイメージの名前空間
ceph_tag	Ceph Storage コンテナーイメージのタグ
name_prefix	各 OpenStack サービスイメージの接頭辞
name_suffix	各 OpenStack サービスイメージの接尾辞
namespace	各 OpenStack サービスイメージの名前空間
neutron_driver	使用する OpenStack Networking (neutron) コンテナーを定義するのに使用するドライバー。標準の **neutron-server** コンテナーに設定するには、null 値を使用します。OVN ベースのコンテナーを使用するには、**ovn** に設定します。
tag	ソースレジストリーからプルするイメージを識別するために director が使用するタグ。通常、このキーは **latest** に設定したままにします。

ContainerImageRegistryCredentials パラメーターは、コンテナーレジストリーをそのレジストリーに対して認証を行うためのユーザー名とパスワードにマッピングします。

コンテナーレジストリーでユーザー名およびパスワードが必要な場合には、**ContainerImageRegistryCredentials** を使用して以下の構文でその値を設定することができます。

```
ContainerImagePrepare:
- push_destination: 192.168.24.1:8787
  set:
    namespace: registry.redhat.io/...
```

```
    ...
ContainerImageRegistryCredentials:
  registry.redhat.io:
    my_username: my_password
```

上記の例の **my_username** および **my_password** を、実際の認証情報に置き換えてください。Red Hat では、個人のユーザー認証情報を使用する代わりに、レジストリーサービスアカウントを作成し、それらの認証情報を使用して **registry.redhat.io** コンテンツにアクセスすることを推奨します。詳しくは、「Red Hat コンテナーレジストリーの認証」を参照してください。

ContainerImageRegistryLogin パラメーターは、デプロイ中のシステムのレジストリーへのログインを制御するために使用されます。**push_destination** が false に設定されている、または使用されていない場合は、これを **true** に設定する必要があります。

```
ContainerImagePrepare:
- set:
    namespace: registry.redhat.io/...
    ...
ContainerImageRegistryCredentials:
  registry.redhat.io:
    my_username: my_password
ContainerImageRegistryLogin: true
```

2.3. イメージ準備エントリーの階層化

ContainerImagePrepare パラメーターの値は YAML リストです。したがって、複数のエントリーを指定することができます。以下の例で、2 つのエントリーを指定するケースを説明します。この場合、director はすべてのイメージの最新バージョンを使用しますが、**nova-api** イメージについてのみ、**15.0-44** とタグ付けされたバージョンを使用します。

```
ContainerImagePrepare:
- tag_from_label: "{version}-{release}"
  push_destination: true
  excludes:
  - nova-api
  set:
    namespace: registry.redhat.io/rhosp15-rhel8
    name_prefix: openstack-
    name_suffix: ''
    tag: latest
- push_destination: true
  includes:
  - nova-api
  set:
    namespace: registry.redhat.io/rhosp15-rhel8
    tag: 15.0-44
```

includes および **excludes** のエントリーで、それぞれのエントリーでのイメージの絞り込みをコントロールします。**includes** 設定と一致するイメージが、**excludes** と一致するイメージに優先します。一致するとみなされるためには、イメージ名に **includes** または **excludes** の設定値が含まれていなければなりません。

2.4. 準備プロセスにおけるイメージの変更

イメージの準備中にイメージを変更し、変更したイメージで直ちにデプロイすることが可能です。イメージを変更するシナリオを以下に示します。

- デプロイメント前にテスト中の修正でイメージが変更される、継続的インテグレーションのパイプラインの一部として。
- ローカルの変更をテストおよび開発のためにデプロイしなければならない、開発ワークフローの一部として。
- 変更をデプロイしなければならないが、イメージビルドパイプラインでは利用することができない場合。たとえば、プロプライエタリーアドオンの追加または緊急の修正など。

準備プロセス中にイメージを変更するには、変更する各イメージで Ansible ロールを呼び出します。ロールはソースイメージを取得して必要な変更を行い、その結果をタグ付けします。prepare コマンドでイメージを目的のレジストリーにプッシュし、変更したイメージを参照するように Heat パラメーターを設定することができます。

Ansible ロール **tripleo-modify-image** は要求されたロールインターフェースに従い、変更のユースケースに必要な処理を行います。変更は、**ContainerImagePrepare** パラメーターの変更固有のキーを使用してコントロールします。

- **modify_role** では、変更する各イメージについて呼び出す Ansible ロールを指定します。
- **modify_append_tag** は、ソースイメージタグの最後に文字列を追加します。これにより、そのイメージが変更されていることが明確になります。すでに **push_destination** レジストリーに変更されたイメージが含まれている場合には、このパラメーターを使用して変更を省略します。イメージを変更する場合には、必ず **modify_append_tag** を変更することを推奨します。
- **modify_vars** は、ロールに渡す Ansible 変数のディクショナリーです。

tripleo-modify-image ロールが処理するユースケースを選択するには、**tasks_from** 変数をそのロールで必要なファイルに設定します。

イメージを変更する **ContainerImagePrepare** エントリーを開発およびテストする場合には、イメージが想定どおりに変更されることを確認するために、追加のオプションを指定せずにイメージの準備コマンドを実行することを推奨します。

```
sudo openstack tripleo container image prepare \
  -e ~/containers-prepare-parameter.yaml
```

2.5. コンテナーイメージの既存パッケージの更新

以下の **ContainerImagePrepare** エントリーは、アンダークラウドホストの dnf リポジトリー設定を使用してイメージのパッケージをすべて更新する例です。

```
ContainerImagePrepare:
- push_destination: true
  ...
  modify_role: tripleo-modify-image
  modify_append_tag: "-updated"
  modify_vars:
    tasks_from: yum_update.yml
    compare_host_packages: true
    yum_repos_dir_path: /etc/yum.repos.d
  ...
```

2.6. コンテナーイメージへの追加 RPM ファイルのインストール

RPM ファイルのディレクトリーをコンテナーイメージにインストールすることができます。この機能は、ホットフィックス、ローカルパッケージビルド、またはパッケージリポジトリーからは入手できないパッケージのインストールに役立ちます。たとえば、以下の **ContainerImagePrepare** エントリーにより、**nova-compute** イメージだけにホットフィックスパッケージがインストールされます。

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
  modify_role: tripleo-modify-image
  modify_append_tag: "-hotfix"
  modify_vars:
    tasks_from: rpm_install.yml
    rpms_path: /home/stack/nova-hotfix-pkgs
  ...
```

2.7. カスタム DOCKERFILE を使用したコンテナーイメージの変更

柔軟性を高めるために、Dockerfile を含むディレクトリーを指定して必要な変更を加えることが可能です。**tripleo-modify-image** ロールを呼び出すと、ロールは **Dockerfile.modified** ファイルを生成し、これにより **FROM** ディレクティブが変更され新たな **LABEL** ディレクティブが追加されます。以下の例では、**nova-compute** イメージでカスタム Dockerfile が実行されます。

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
  modify_role: tripleo-modify-image
  modify_append_tag: "-hotfix"
  modify_vars:
    tasks_from: modify_image.yml
    modify_dir_path: /home/stack/nova-custom
  ...
```

/home/stack/nova-custom/Dockerfile の例を以下に示します。**USER** root ディレクティブを実行した後は、元のイメージのデフォルトユーザーに戻す必要があります。

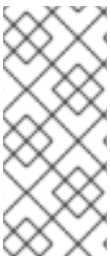
```
FROM registry.redhat.io/rhosp15-rhel8/openstack-nova-compute:latest

USER "root"

COPY customize.sh /tmp/
RUN /tmp/customize.sh

USER "nova"
```

2.8. コンテナーイメージ管理用 SATELLITE サーバーの準備

Red Hat Satellite 6 には、レジストリーの同期機能が備わっています。これにより、複数のイメージを

Satellite Server にプルし、アプリケーションライフサイクルの一環として管理することができます。また、他のコンテナー対応システムも Satellite をレジストリーとして使うことができます。コンテナーイメージ管理の詳細は、**『Red Hat Satellite 6 コンテンツ管理ガイド』**の「コンテナーイメージの管理」を参照してください。

以下の手順は、Red Hat Satellite 6 の **hammer** コマンドラインツールを使用した例を示しています。組織には、例として **ACME** という名称を使用しています。この組織は、実際に使用する Satellite 6 の組織に置き換えてください。



注記

この手順では、**registry.redhat.io** のコンテナーイメージにアクセスするために認証情報が必要です。Red Hat では、個人のユーザー認証情報を使用する代わりに、レジストリーサービスアカウントを作成し、それらの認証情報を使用して **registry.redhat.io** コンテンツにアクセスすることを推奨します。詳しくは、「Red Hat コンテナーレジストリーの認証」を参照してください。

手順

1. すべてのコンテナーイメージの一覧を作成します。

```
$ sudo podman search --limit 1000 "registry.redhat.io/rhosp15-rhel8" | awk '{ print $2 }' | grep -v beta | sed "s/registry.redhat.io\///g" | tail -n+2 > satellite_images
```

2. Satellite 6 の **hammer** ツールがインストールされているシステムに **satellite_images_names** ファイルをコピーします。あるいは、**『Hammer CLI ガイド』**に記載の手順に従って、アンダークラウドに **hammer** ツールをインストールします。

3. 以下の **hammer** コマンドを実行して、実際の Satellite 組織に新規製品(**OSP15 Containers**)を作成します。

```
$ hammer product create \
  --organization "ACME" \
  --name "OSP15 Containers"
```

このカスタム製品に、イメージを保管します。

4. 製品にベースコンテナーイメージを追加します。

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP15 Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhosp15-rhel8/openstack-base \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name base
```

5. **satellite_images** ファイルからオーバークラウドのコンテナーイメージを追加します。

```
$ while read IMAGE; do \
  IMAGENAME=$(echo $IMAGE | cut -d"/" -f2 | sed "s/openstack-//g" | sed "s/:.*//g") ; \
  hammer repository create \
```

```
  --organization "ACME" \
  --product "OSP15 Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name $IMAGE \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name $IMAGENAME ; done < satellite_images_names
```

6. Ceph Storage 4 コンテナーイメージを追加します。

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP15 Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph-beta/rhceph-4-rhel8 \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name rhceph-4-rhel8
```

7. コンテナーイメージを同期します。

```
$ hammer product synchronize \
  --organization "ACME" \
  --name "OSP15 Containers"
```

Satellite Server が同期を完了するまで待ちます。

注記

設定によっては、**hammer** から Satellite Server のユーザー名およびパスワードが要求される場合があります。設定ファイルを使って自動的にログインするように **hammer** を設定することができます。詳しくは、**『Red Hat Satellite Hammer CLI ガイド』**の「認証」セクションを参照してください。

8. お使いの Satellite 6 サーバーでコンテンツビューが使われている場合には、新たなバージョンのコンテンツビューを作成してイメージを反映し、アプリケーションライフサイクルの環境に従ってプロモートします。この作業は、アプリケーションライフサイクルをどのように構成するかに大きく依存します。たとえば、ライフサイクルに **production** という名称の環境があり、その環境でコンテナーイメージを利用可能にする場合には、コンテナーイメージを含むコンテンツビューを作成し、そのコンテンツビューを **production** 環境にプロモートします。詳しくは、『Red Hat Satellite コンテンツ管理ガイド』の「コンテンツビューによるコンテナーイメージの管理」を参照してください。

9. **base** イメージに使用可能なタグを確認します。

```
$ hammer docker tag list --repository "base" \
  --organization "ACME" \
  --environment "production" \
  --content-view "myosp15" \
  --product "OSP15 Containers"
```

このコマンドにより、特定環境のコンテンツビューでの OpenStack Platform コンテナーイメージのタグが表示されます。

10. アンダークラウドに戻り、Satellite サーバーをソースとして使用して、イメージ準備用のデフォルト環境ファイルを生成します。以下のサンプルコマンドを実行して環境ファイルを生成します。

```
(undercloud) $ openstack tripleo container image prepare default \
  --output-env-file containers-prepare-parameter.yaml
```

- **--output-env-file**: 環境ファイルの名前です。このファイルには、アンダークラウド用コンテナーイメージを準備するためのパラメーターが含まれます。ここでは、ファイル名は **containers-prepare-parameter.yaml** です。

11. **containers-prepare-parameter.yaml** ファイルを編集して以下のパラメーターを変更します。

- **namespace**: Satellite サーバー上のレジストリーの URL およびポート。Red Hat Satellite のデフォルトのレジストリーポートは 5000 です。

- **name_prefix**: プレフィックスは Satellite 6 の命名規則に基づきます。これは、コンテンツビューを使用するかどうかによって異なります。

  - コンテンツビューを使用する場合、構成は **[org]-[environment]-[content view]-[product]-** です。たとえば、**acme-production-myosp15-osp15_containers-** のようになります。

  - コンテンツビューを使用しない場合、構成は **[org]-[product]-** です。たとえば、**acme-osp15_containers-** のようになります。

- **ceph_namespace**、**ceph_image**、**ceph_tag**: Ceph Storage を使用する場合には、Ceph Storage のコンテナーイメージの場所を定義する追加のパラメーターを指定します。**ceph_image** に Satellite 固有のプレフィックスが追加された点に注意してください。このプレフィックスは、**name_prefix** オプションと同じ値です。

Satellite 固有のパラメーターが含まれる環境ファイルの例を、以下に示します。

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
      ceph_image: acme-production-myosp15-osp15_containers-rhceph-4
      ceph_namespace: satellite.example.com:5000
      ceph_tag: latest
      name_prefix: acme-production-myosp15-osp15_containers-
      name_suffix: ''
      namespace: satellite.example.com:5000
      neutron_driver: null
      tag: latest
      ...
    tag_from_label: '{version}-{release}'
```

アンダークラウドおよびオーバークラウドの両方を作成する際に、この環境ファイルを使用します。

第3章 コンテナーを使用したアンダークラウドのインストール

本章では、コンテナーベースのアンダークラウドを作成し、最新の状態に維持する方法について説明します。

3.1. DIRECTOR の設定

director のインストールプロセスでは、director が **stack** ユーザーのホームディレクトリーから読み取る **undercloud.conf** 設定ファイルに、特定の設定が必要になります。以下の手順では、デフォルトのテンプレートをベースに使用して設定を行う方法についてを説明します。

手順

1. デフォルトのテンプレートを **stack** ユーザーのホームディレクトリーにコピーします。

```
[stack@director ~]$ cp \
  /usr/share/python-tripleoclient/undercloud.conf.sample \
  ~/undercloud.conf
```

2. **undercloud.conf** ファイルを編集します。このファイルには、アンダークラウドを設定するための設定値が含まれています。パラメーターを省略したり、コメントアウトした場合には、アンダークラウドのインストールでデフォルト値が使用されます。

3.2. DIRECTOR の設定パラメーター

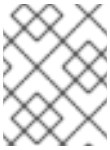
以下の一覧で、**undercloud.conf** ファイルを設定するパラメーターについて説明します。エラーを避けるために、パラメーターは決して該当するセクションから削除しないでください。

デフォルト

undercloud.conf ファイルの **[DEFAULT]** セクションで定義されているパラメーターを以下に示します。

additional_architectures

オーバークラウドがサポートする追加の (カーネル) アーキテクチャーの一覧。現在、オーバークラウドは **ppc64le** アーキテクチャーをサポートしています。



注記

ppc64le のサポートを有効にする場合には、**ipxe_enabled** を **False** に設定する必要もあります。

certificate_generation_ca

要求した証明書に署名する CA の **certmonger** のニックネーム。**generate_service_certificate** パラメーターを設定した場合に限り、このオプションを使用します。**local** CA を選択する場合は、certmonger はローカルの CA 証明書を **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** に抽出し、証明書をトラストチェーンに追加します。

clean_nodes

デプロイメントを再実行する前とイントロスペクションの後にハードドライブを消去するかどうかを定義します。

cleanup

一時ファイルをクリーンナップします。デプロイメント時に使用した一時ファイルをコマンド実行後もそのまま残すには、このパラメーターを **False** に設定します。ファイルを残すと、生成されたファイルのデバッグを行う場合やエラーが発生した場合に役に立ちます。

container_cli

コンテナー管理用の CLI ツール。Red Hat Enterprise Linux 8 は **podman** のみをサポートするため、このパラメーターは **podman** に設定したままにしてください。

container_healthcheck_disabled

コンテナー化されたサービスのヘルスチェックを無効にします。このオプションは **false** に設定したままにして、ヘルスチェックを有効な状態に維持することを推奨します。

container_images_file

コンテナーイメージ情報が含まれる Heat 環境ファイル。このパラメーターは、以下のいずれかに設定します。

- 必要なすべてのコンテナーイメージのパラメーター。
- 必要なイメージの準備を実施する **ContainerImagePrepare** パラメーター。このパラメーターが含まれるファイルの名前は、通常 **containers-prepare-parameter.yaml** です。

container_insecure_registries

podman が使用するセキュアではないレジストリーの一覧。プライベートコンテナーレジストリー等の別のソースからイメージをプルする場合には、このパラメーターを使用します。多くの場合、**podman** は Red Hat Container Catalog または Satellite サーバー (アンダークラウドが Satellite に登録されている場合) のいずれかからコンテナーイメージをプルするための証明書を持ちます。

container_registry_mirror

設定により **podman** が使用するオプションの **registry-mirror**

custom_env_files

アンダークラウドのインストールに追加する新たな環境ファイル。

deployment_user

アンダークラウドをインストールするユーザー。現在のデフォルトユーザー (**stack**) を使用する場合には、このパラメーターを未設定のままにします。

discovery_default_driver

自動的に登録されたノードのデフォルトドライバーを設定します。**enable_node_discovery** を有効にし、**enabled_hardware_types** ファイルにドライバーを含める必要があります。

enable_ironic、enable_ironic_inspector、enable_mistral、enable_tempest、enable_validations、enable_zaqar

director で有効にするコアサービスを定義します。これらのパラメーターは **true** に設定されたままにします。

enable_node_discovery

introspection ramdisk を PXE ブートする不明なノードを自動的に登録します。新規ノードは、**fake_pxe** ドライバーをデフォルトとして使用しますが、**discovery_default_driver** を設定して上書きすることもできます。また、イントロスペクションルールを使用して、新しく登録したノードにドライバーの情報を指定することもできます。

enable_novajoin

アンダークラウドの **novajoin** メタデータサービスをインストールするかどうかを定義します。

enable_routed_networks

ルーティングされたコントロールプレーンネットワークのサポートを有効にするかどうかを定義します。

enable_swift_encryption

保存データの Swift 暗号化を有効にするかどうかを定義します。

enable_telemetry

アンダークラウドに OpenStack Telemetry サービス (gnocchi、aodh、panko) をインストールするかどうかを定義します。Telemetry サービスを自動的にインストール/設定するには、**enable_telemetry** パラメーターを **true** に設定します。デフォルト値は **false** で、アンダークラウド上の telemetry が無効になります。このパラメーターは、Red Hat CloudForms などのメトリックデータを消費する他の製品を使用している場合に必要です。

enabled_hardware_types

アンダークラウドで有効にするハードウェアタイプの一覧。

generate_service_certificate

アンダークラウドのインストール時に SSL/TLS 証明書を生成するかどうかを定義します。これは **undercloud_service_certificate** パラメーターに使用します。アンダークラウドのインストールで、作成された証明書 **/etc/pki/tls/certs/undercloud-[undercloud_public_vip].pem** を保存します。**certificate_generation_ca** パラメーターで定義される CA はこの証明書に署名します。

heat_container_image

使用する heat コンテナーイメージの URL。未設定のままにします。

heat_native

ネイティブの heat テンプレートを使用します。**true** のままにします。

hieradata_override

director に Puppet hieradata を設定するための **hieradata** オーバーライドファイルへのパス。これにより、サービスに対して **undercloud.conf** パラメーター以外のカスタム設定を行うことができます。設定すると、アンダークラウドのインストールでこのファイルが **/etc/puppet/hieradata** ディレクトリーにコピーされ、階層の最初のファイルに設定されます。この機能の使用 方法についての詳細は、「アンダークラウドへの hieradata の設定 」を参照してください。

inspection_extras

イントロスペクション時に追加のハードウェアコレクションを有効化するかどうかを定義します。このパラメーターを使用するには、イントロスペクションイメージに **python-hardware** または **python-hardware-detect** パッケージが必要です。

inspection_interface

ノードのイントロスペクションに director が使用するブリッジ。これは、director の設定により作成されるカスタムのブリッジです。**LOCAL_INTERFACE** でこのブリッジをアタッチします。これは、デフォルトの **br-ctlplane** のままにします。

inspection_runbench

ノードイントロスペクション時に一連のベンチマークを実行します。ベンチマークを有効にするには、このパラメーターを **true** に設定します。このオプションは、登録ノードのハードウェアを検査する際にベンチマーク分析を実行する場合に必要です。

ipa_otp

IPA サーバーにアンダークラウドノードを登録するためのワンタイムパスワードを定義します。これは、**enable_novajoin** が有効な場合に必要です。

ipxe_enabled

iPXE か標準の PXE のいずれを使用するか定義します。デフォルトは **true** で iPXE を有効化します。**false** に指定すると、標準の PXE に設定されます。

local_interface

director のプロビジョニング NIC 用に選択するインターフェース。director は、DHCP および PXE ブートサービスにもこのデバイスを使用します。この値を選択したデバイスに変更します。接続されているデバイスを確認するには、**ip addr** コマンドを使用します。**ip addr** コマンドの出力結果の例を、以下に示します。

■

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen
1000
    link/ether 52:54:00:75:24:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.178/24 brd 192.168.122.255 scope global dynamic eth0
       valid_lft 3462sec preferred_lft 3462sec
    inet6 fe80::5054:ff:fe75:2409/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state DOWN
    link/ether 42:0b:c2:a5:c1:26 brd ff:ff:ff:ff:ff:ff
```

この例では、外部 NIC は **eth0** を、プロビジョニング NIC は未設定の **eth1** を使用します。今回は、**local_interface** を **eth1** に設定します。この設定スクリプトにより、このインターフェースが **inspection_interface** パラメーターで定義したカスタムのブリッジにアタッチされます。

local_ip

director のプロビジョニング NIC 用に定義する IP アドレス。director は、DHCP および PXE ブートサービスにもこの IP アドレスを使用します。環境内の既存の IP アドレスまたはサブネットと競合するなどの理由により、プロビジョニングネットワークに別のサブネットを使用する場合以外は、この値はデフォルトの **192.168.24.1/24** のままにします。

local_mtu

local_interface に使用する MTU。アンダークラウドでは 1500 以下にします。

local_subnet

PXE ブートと DHCP インターフェースに使用するローカルサブネット。**local_ip** アドレスがこのサブネットに含まれている必要があります。デフォルトは **ctlplane-subnet** です。

net_config_override

ネットワーク設定のオーバーライドテンプレートへのパス。このパラメーターを設定すると、アンダークラウドは JSON 形式のテンプレートを使用して **os-net-config** でネットワークを設定します。アンダークラウドは、**undercloud.conf** に設定されているネットワークパラメーターを無視します。**/usr/share/python-tripleoclient/undercloud.conf.sample** の例を参照してください。

networks_file

heat をオーバーライドするネットワークファイル

output_dir

状態、処理された heat テンプレート、および Ansible デプロイメントファイルを出力するディレクトリー

overcloud_domain_name

オーバークラウドのデプロイ時に使用する DNS ドメイン名。

注記

オーバークラウドを設定する際に、**CloudDomain** パラメーターに overcloud_domain_name と同じ値を設定する必要があります。オーバークラウドを設定する際に、環境ファイルでこのパラメーターを設定します。

roles_file

アンダークラウドのインストール用に上書きを行うロールファイル。director のインストールにデフォルトのロールファイルが使用されるように、未設定のままにすることを強く推奨します。

scheduler_max_attempts

スケジューラーがインスタンスのデプロイを試行する最大回数。スケジューリング時に競合状態にならないように、この値を 1 度にデプロイする予定のベアメタルノードの数以上に指定する必要があります。

service_principal

この証明書を使用するサービスの Kerberos プリンシパル。FreeIPA など CA で Kerberos プリンシパルが必要な場合に限り、このパラメーターを使用します。

subnets

プロビジョニングおよびイントロスペクション用のルーティングネットワークのサブネットの一覧。詳しくは、「サブネット」を参照してください。デフォルト値に含まれるのは、**ctlplane-subnet** サブネットのみです。

templates

上書きする heat テンプレートファイル

undercloud_admin_host

SSL/TLS 経由の director の管理 API エンドポイントに定義する IP アドレスまたはホスト名。director の設定により、IP アドレスは **/32** ネットマスクを使用するルーティングされた IP アドレスとして director のソフトウェアブリッジに接続されます。

undercloud_debug

アンダークラウドサービスのログレベルを **DEBUG** に設定します。この値は **true** に設定して有効化します。

undercloud_enable_selinux

デプロイメント時に、SELinux を有効または無効にします。問題をデバッグする場合以外は、この値を **true** に設定したままにすることを強く推奨します。

undercloud_hostname

アンダークラウドの完全修飾ホスト名を定義します。設定すると、アンダークラウドのインストールで全システムのホスト名が設定されます。未設定のままにすると、アンダークラウドは現在のホスト名を使用しますが、ユーザーは適切に全システムのホスト名の設定を行う必要があります。

undercloud_log_file

アンダークラウドのインストール/アップグレードログを保管するログファイルへのパス。デフォルトでは、ログファイルはホームディレクトリー内の **install-undercloud.log** です。たとえば、**/home/stack/install-undercloud.log** のようになります。

undercloud_nameservers

アンダークラウドのホスト名解決に使用する DNS ネームサーバーの一覧

undercloud_ntp_servers

アンダークラウドの日付と時刻を同期できるようにする Network Time Protocol サーバーの一覧

undercloud_public_host

SSL/TLS 経由の director のパブリック API エンドポイントに定義する IP アドレスまたはホスト名。director の設定により、IP アドレスは **/32** ネットマスクを使用するルーティングされた IP アドレスとして director のソフトウェアブリッジに接続されます。

undercloud_service_certificate

OpenStack SSL/TLS 通信の証明書の場所とファイル名。理想的には、信頼できる認証局から、この証明書を取得します。それ以外の場合は、独自の自己署名の証明書を生成します。

undercloud_timezone

アンダークラウド用ホストのタイムゾーン。タイムゾーンを指定しなければ、director は既存のタイムゾーン設定を使用します。

undercloud_update_packages

アンダークラウドのインストール時にパッケージを更新するかどうかを定義します。

サブネット

undercloud.conf ファイルには、各プロビジョニングサブネットの名前が付いたセクションがあります。たとえば、**ctlplane-subnet** という名前のサブネットを作成するには、**undercloud.conf** ファイルで以下のような設定を使用します。

```
[ctlplane-subnet]
cidr = 192.168.24.0/24
dhcp_start = 192.168.24.5
dhcp_end = 192.168.24.24
inspection_iprange = 192.168.24.100,192.168.24.120
gateway = 192.168.24.1
masquerade = true
```

プロビジョニングネットワークは、環境に応じて、必要なだけ指定することができます。

gateway

オーバークラウドインスタンスのゲートウェイ。外部ネットワークにトラフィックを転送するアンダークラウドのホストです。director に別の IP アドレスを使用する場合または直接外部ゲートウェイを使用する場合以外は、この値はデフォルト (**192.168.24.1**) のままにします。

注記

director 設定は、適切な **sysctl** カーネルパラメーターを使用して IP フォワーディングも自動的に有効化します。

cidr

オーバークラウドインスタンスの管理に director が使用するネットワーク。これは、アンダークラウドの **neutron** サービスが管理するプロビジョニングネットワークです。プロビジョニングネットワークに別のサブネットを使用しない限り、この値はデフォルト (**192.168.24.0/24**) のままにします。

masquerade

外部ネットワークへのアクセスのために、**cidr** で定義したネットワークをマスカレードするかどうかを定義します。このパラメーターにより、director 経由で外部ネットワークにアクセスすることができるように、プロビジョニングネットワークにネットワークアドレス変換 (NAT) の一部メカニズムが提供されます。

dhcp_start、dhcp_end

オーバークラウドノードの DHCP 割り当て範囲 (開始アドレスと終了アドレス)。ノードを割り当てるのに十分な IP アドレスがこの範囲に含まれるようにします。

dhcp_exclude

DHCP 割り当て範囲で除外する IP アドレス

host_routes

このネットワーク上のオーバークラウドインスタンス用の Neutron が管理するサブネットのホストルート。このパラメーターにより、アンダークラウド上の **local_subnet** のホストルートも設定されます。

inspection_iprange

director のイントロスペクションサービスが PXE ブートとプロビジョニングプロセスの際に使用する IP アドレス範囲。値をコンマ区切り、この IP アドレス範囲の開始アドレスおよび終了アドレスを定義します。たとえば、**192.168.24.100,192.168.24.120** のように定義します。この範囲には、使用するノードに十分な数の IP アドレスが含まれるようにし、**dhcp_start** と **dhcp_end** の範囲とは競合しないように設定してください。

これらのパラメーターの値は、構成に応じて変更してください。完了したら、ファイルを保存します。

3.3. DIRECTOR のインストール

director をインストールして基本的なインストール後タスクを行うには、以下の手順を実施します。

手順

1. 以下のコマンドを実行して、アンダークラウドに director をインストールします。

```
[stack@director ~]$ openstack undercloud install
```

このコマンドで、director の設定スクリプトを起動します。director により追加のパッケージがインストールされ、**undercloud.conf** の設定に応じてサービスが設定されます。このスクリプトは、完了までに数分かかります。

スクリプトにより、完了時には 2 つのファイルが生成されます。

- **undercloud-passwords.conf**: director サービスの全パスワード一覧
- **stackrc**: director のコマンドラインツールへアクセスできるようにする初期化変数セット

2. このスクリプトは、全 OpenStack Platform サービスのコンテナーも自動的に起動します。以下のコマンドを使用して、有効化されたコンテナーを確認してください。

```
[stack@director ~]$ sudo podman ps
```

3. **stack** ユーザーを初期化してコマンドラインツールを使用するには、以下のコマンドを実行します。

```
[stack@director ~]$ source ~/stackrc
```

プロンプトには、OpenStack コマンドがアンダークラウドに対して認証および実行されることが表示されるようになります。

```
(undercloud) [stack@director ~]$
```

director のインストールが完了しました。これで、director のコマンドラインツールが使用できるようになりました。

3.4. コンテナー化されたアンダークラウドのマイナーアップデートの実施

director では、アンダークラウドノード上のパッケージを更新するためのコマンドが提供されています。これにより、OpenStack Platform 環境の現行バージョン内のマイナーアップデートを実行することができます。

手順

1. director に **stack** ユーザーとしてログインします。
2. **dnf** コマンドを実行して、director の主要なパッケージをアップグレードします。

```
$ sudo dnf update -y python3-tripleoclient* openstack-tripleo-common openstack-tripleo-heat-templates
```

3. director は **openstack undercloud upgrade** コマンドを使用して、アンダークラウドの環境を更新します。以下のコマンドを実行します。

```
$ openstack undercloud upgrade
```

4. アンダークラウドのアップグレードプロセスが完了するまで待ちます。

5. アンダークラウドをリブートして、オペレーティングシステムのカーネルとその他のシステムパッケージを更新します。

```
$ sudo reboot
```

6. ノードがブートするまで待ちます。

第4章 コンテナーベースのオーバークラウドのデプロイおよび更新

本章では、コンテナーベースのオーバークラウドを作成し、最新の状態に維持する方法について説明します。

4.1. オーバークラウドのデプロイ

最小構成のオーバークラウドをデプロイする方法を、以下の手順で説明します。デプロイの結果、2 つのノードを持つ基本的なオーバークラウド (1 つのコントローラーノードおよび 1 つのコンピュートノード) が得られます。

手順

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. **deploy** コマンドを実行し、オーバークラウドイメージの場所を定義したファイル (通常は **overcloud_images.yaml**) を含めます。

```
(undercloud) $ openstack overcloud deploy --templates \
  -e /home/stack/templates/overcloud_images.yaml \
  --ntp-server pool.ntp.org
```

3. オーバークラウドがデプロイメントを完了するまで待ちます。

4.2. オーバークラウドの更新

コンテナー化され たオーバークラウドの更新に関する情報は、『Red Hat OpenStack Platform の最新状態の維持 』を参照してください。

第5章 コンテナー化されたサービスの操作

本章では、コンテナーを管理するコマンドの例および OpenStack Platform コンテナーに関するトラブルシューティング方法について説明します。

5.1. コンテナー化されたサービスの管理

OpenStack Platform では、アンダークラウドおよびオーバークラウドノード上のコンテナー内でサービスが実行されます。特定の状況では、1 つのホスト上で個別のサービスを制御する必要がある場合があります。本項では、コンテナー化されたサービスを管理するためにノード上で実行することのできる、一般的なコマンドについて説明します。

コンテナーとイメージの一覧表示

実行中のコンテナーを一覧表示するには、以下のコマンドを実行します。

```
$ sudo podman ps
```

コマンド出力に停止中またはエラーの発生したコンテナーを含めるには、コマンドに **--all** オプションを追加します。

```
$ sudo podman ps --all
```

コンテナーイメージを一覧表示するには、以下のコマンドを実行します。

```
$ sudo podman images
```

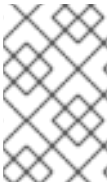
コンテナーの属性の確認

コンテナーまたはコンテナーイメージのプロパティーを表示するには、**podman inspect** コマンドを使用します。たとえば、**keystone** コンテナーを検査するには、以下のコマンドを実行します。

```
$ sudo podman inspect keystone
```

Systemd サービスを使用したコンテナーの管理

以前のバージョンの OpenStack Platform では、コンテナーは Docker およびそのデーモンで管理されていました。OpenStack Platform 15 では、Systemd サービスインターフェースでコンテナーのライフサイクルが管理されます。それぞれのコンテナーはサービスであり、以下のコマンドを実行して各コンテナーに関する特定の操作を実施します。



注記

Systemd は再起動ポリシーを適用するため、Podman CLI を使用してコンテナーを停止、起動、および再起動することは推奨されません。その代わりに、Systemd サービスコマンドを使用してください。

コンテナーのステータスを確認するには、**systemctl status** コマンドを実行します。

```
$ sudo systemctl status tripleo_keystone
● tripleo_keystone.service - keystone container
  Loaded: loaded (/etc/systemd/system/tripleo_keystone.service; enabled; vendor preset: disabled)
  Active: active (running) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
```

```
  Main PID: 29012 (podman)
    CGroup: /system.slice/tripleo_keystone.service
            └─29012 /usr/bin/podman start -a keystone
```

コンテナーを停止するには、**systemctl stop** コマンドを実行します。

```
$ sudo systemctl stop tripleo_keystone
```

コンテナーを起動するには、**systemctl start** コマンドを実行します。

```
$ sudo systemctl start tripleo_keystone
```

コンテナーを再起動するには、**systemctl restart** コマンドを実行します。

```
$ sudo systemctl restart tripleo_keystone
```

コンテナーステータスを監視するデーモンはないので、以下の状況では Systemd はほとんどのコンテナーを自動的に再起動します。

- **podman stop** コマンドの実行など、明瞭な終了コードまたはシグナル。
- 起動後に podman コンテナーがクラッシュするなど、不明瞭な終了コード
- 不明瞭なシグナル
- コンテナーの起動に 1 分 30 秒以上かかった場合のタイムアウト

Systemd サービスに関する詳しい情報は、**systemd.service** のドキュメント を参照してください。

注記

コンテナー内のサービス設定ファイルに加えた変更は、コンテナーの再起動後には元に戻ります。これは、コンテナーがノードのローカルファイルシステム上の **/var/lib/config-data/puppet-generated/** にあるファイルに基づいてサービス設定を再生成するためです。たとえば、**keystone** コンテナー内の **/etc/keystone/keystone.conf** を編集してコンテナーを再起動すると、そのコンテナーはノードのローカルシステム上にある **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf** を使用して設定を再生成します。再起動前にコンテナー内で加えられた変更は、この設定によって上書きされます。

Systemd タイマーを使用した podman コンテナーの監視

Systemd タイマーインターフェースは、コンテナーのヘルスチェックを管理します。各コンテナーのタイマーがサービスユニットを実行し、そのユニットがヘルスチェックスクリプトを実行します。

すべての OpenStack Platform コンテナーのタイマーを一覧表示するには、**systemctl list-timers** コマンドを実行し、出力を **tripleo** が含まれる行に限定します。

```
$ sudo systemctl list-timers | grep tripleo
Mon 2019-02-18 20:18:30 UTC  1s left      Mon 2019-02-18 20:17:26 UTC  1min 2s ago
tripleo_nova_metadata_healthcheck.timer           tripleo_nova_metadata_healthcheck.service
Mon 2019-02-18 20:18:33 UTC  4s left      Mon 2019-02-18 20:17:03 UTC  1min 25s ago
tripleo_mistral_engine_healthcheck.timer        tripleo_mistral_engine_healthcheck.service
Mon 2019-02-18 20:18:34 UTC  5s left      Mon 2019-02-18 20:17:23 UTC  1min 5s ago
```

```
tripleo_keystone_healthcheck.timer                tripleo_keystone_healthcheck.service
Mon 2019-02-18 20:18:35 UTC  6s left      Mon 2019-02-18 20:17:13 UTC  1min 15s ago
tripleo_memcached_healthcheck.timer               tripleo_memcached_healthcheck.service
(...)
```

特定のコンテナータイマーのステータスを確認するには、healthcheck サービスに対して **systemctl status** コマンドを実行します。

```
$ sudo systemctl status tripleo_keystone_healthcheck.service
● tripleo_keystone_healthcheck.service - keystone healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.service; disabled; vendor
preset: disabled)
   Active: inactive (dead) since Mon 2019-02-18 20:22:46 UTC; 22s ago
  Process: 115581 ExecStart=/usr/bin/podman exec keystone /openstack/healthcheck (code=exited,
status=0/SUCCESS)
 Main PID: 115581 (code=exited, status=0/SUCCESS)

Feb 18 20:22:46 undercloud.localdomain systemd[1]: Starting keystone healthcheck...
Feb 18 20:22:46 undercloud.localdomain podman[115581]: {"versions": {"values": [{"status": "stable",
"updated": "2019-01-22T00:00:00Z", "..."}]}]}}
Feb 18 20:22:46 undercloud.localdomain podman[115581]: 300 192.168.24.1:35357 0.012 seconds
Feb 18 20:22:46 undercloud.localdomain systemd[1]: Started keystone healthcheck.
```

コンテナータイマーを停止、起動、再起動、およびコンテナータイマーのステータスを表示するには、**.timer** Systemd リソースに対して該当する **systemctl** コマンドを実行します。たとえば、**tripleo_keystone_healthcheck.timer** リソースのステータスを確認するには、以下のコマンドを実行します。

```
$ sudo systemctl status tripleo_keystone_healthcheck.timer
● tripleo_keystone_healthcheck.timer - keystone container healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.timer; enabled; vendor preset:
disabled)
   Active: active (waiting) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
```

healthcheck サービスは無効だが、そのサービスのタイマーが存在し有効になっている場合には、チェックは現在タイムアウトしているが、タイマーに従って実行されることを意味します。いつでもチェックを手動で開始することができます。

注記

podman ps コマンドは、コンテナーのヘルスステータスを表示しません。

コンテナーログの確認

OpenStack Platform 15 では、新たなロギングディレクトリー **/var/log/containers/stdout** が導入されています。ここには、すべてのコンテナーの標準出力 (stdout) と標準エラー (stderr) が、コンテナーごとに 1 つのファイルに統合されて保存されます。

paunch および **container-puppet.py** スクリプトは、出力を **/var/log/containers/stdout** ディレクトリーにプッシュするように podman コンテナーを設定します。これにより、**container-puppet-*** コンテナー等の削除されたコンテナーを含め、すべてのログのコレクションが作成されます。

また、ホストはこのディレクトリーにログローテーションを適用し、大きな容量のファイルがディスク容量を消費する問題を防ぎます。

コンテナーが置き換えられた場合には、新しいコンテナーは同じログファイルにログを出力します。**podman** はコンテナー ID ではなくコンテナー名を使用するよう設定されているためです。

podman logs コマンドを使用して、コンテナー化されたサービスのログを確認することもできます。たとえば、**keystone** コンテナーのログを確認するには、以下のコマンドを実行します。

```
$ sudo podman logs keystone
```

コンテナーへのアクセス

コンテナー化されたサービスのシェルに入るには、**podman exec** コマンドを使用して **/bin/bash** を起動します。たとえば、**keystone** コンテナーのシェルに入るには、以下のコマンドを実行します。

```
$ sudo podman exec -it keystone /bin/bash
```

root ユーザーとして **keystone** コンテナーのシェルに入るには、以下のコマンドを実行します。

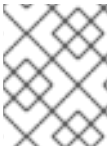
```
$ sudo podman exec --user 0 -it <NAME OR ID> /bin/bash
```

コンテナーから出るには、以下のコマンドを実行します。

```
# exit
```

5.2. コンテナー化されたサービスに関するトラブルシューティング

オーバークラウドのデプロイメント中またはデプロイメント後にコンテナー化されたサービスでエラーが発生した場合には、以下の推奨事項に従って、エラーの根本的な原因を特定してください。



注記

これらのコマンドを実行する前には、オーバークラウドノードにログイン済みであることを確認し、これらのコマンドをアンダークラウドで実行しないようにしてください。

コンテナーログの確認

各コンテナーは、主要プロセスからの標準出力を保持します。この出力はログとして機能し、コンテナー実行時に実際に何が発生したのかを特定するのに役立ちます。たとえば、**keystone** コンテナーのログを確認するには、以下のコマンドを使用します。

```
$ sudo podman logs keystone
```

大半の場合は、このログにコンテナーのエラーの原因が記載されています。

コンテナーの検査

状況によっては、コンテナーに関する情報を検証する必要がある場合があります。たとえば、以下のコマンドを使用して **keystone** コンテナーのデータを確認します。

```
$ sudo podman inspect keystone
```

これにより、ローレベルの設定データが含まれた JSON オブジェクトが提供されます。その出力を **jq** コマンドにパイプで渡して、特定のデータを解析することが可能です。たとえば、**keystone** コンテナーのマウントを確認するには、以下のコマンドを実行します。

```
$ sudo podman inspect keystone | jq .[0].Mounts
```

--format オプションを使用して、データを単一行に解析することもできます。これは、コンテナーデータのセットに対してコマンドを実行する場合に役立ちます。たとえば、**keystone** コンテナーを実行するのに使用するオプションを再生成するには、以下のように **inspect** コマンドに **--format** オプションを指定して実行します。

```
$ sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range .Mounts}} -v
{{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}' keystone
```

注記

--format オプションは、Go 構文を使用してクエリーを作成します。

これらのオプションを **podman run** コマンドと共に使用して、トラブルシューティング目的のコンテナーを再度作成します。

```
$ OPTIONS=$( sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range
.Mounts}} -v {{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}'
keystone )
$ sudo podman run --rm $OPTIONS /bin/bash
```

コンテナー内でのコマンドの実行

状況によっては、特定の Bash コマンドでコンテナー内の情報を取得する必要がある場合があります。このような場合には、以下の **podman** コマンドを使用して、稼働中のコンテナー内でコマンドを実行します。たとえば、**keystone** コンテナーで次のコマンドを実行します。

```
$ sudo podman exec -ti keystone <COMMAND>
```

注記

-ti オプションを指定すると、コマンドは対話式の擬似ターミナルで実行されます。

<COMMAND> は必要なコマンドに置き換えます。たとえば、各コンテナーには、サービスの接続を確認するためのヘルスチェックスクリプトがあります。**keystone** にヘルスチェックスクリプトを実行するには、以下のコマンドを実行します。

```
$ sudo podman exec -ti keystone /openstack/healthcheck
```

コンテナーのシェルにアクセスするには、コマンドとして **/bin/bash** を使用して **podman exec** を実行します。

```
$ sudo podman exec -ti keystone /bin/bash
```

コンテナーのエクスポート

コンテナーに障害が発生した場合には、ファイルの内容を詳細に調べる必要があります。この場合は、コンテナーの全ファイルシステムを **tar** アーカイブとしてエクスポートすることができます。たとえば、**keystone** コンテナーのファイルシステムをエクスポートするには、以下のコマンドを実行します。

```
$ sudo podman export keystone -o keystone.tar
```

このコマンドにより **keystone.tar** アーカイブが作成されます。これを抽出して、調べることができます。

第6章 SYSTEMD サービスとコンテナー化されたサービスの比較

本章では、コンテナー化されたサービスと Systemd サービスの相違点を示す参考資料を提供します。

6.1. SYSTEMD サービスとコンテナー化されたサービスの比較

Systemd ベースのサービスと、Systemd サービスで制御する **podman** コンテナー間の相関を以下の表に示します。

コンポーネント	Systemd サービス	コンテナー
OpenStack Image Storage (glance)	**tripleo_glance_api.service**	**glance_api**
HAProxy	**tripleo_haproxy.service**	**haproxy**
OpenStack Orchestration (heat)	**tripleo_heat_api.service** **tripleo_heat_api_cfn.service** **tripleo_heat_api_cron.service** **tripleo_heat_engine.service**	**heat_api** **heat_api_cfn** **heat_api_cron** **heat_engine**
OpenStack Bare Metal (ironic)	**tripleo_ironic_api.service** **tripleo_ironic_conductor.service** **tripleo_ironic_inspector.service** **tripleo_ironic_inspector_dnsmasq.service** **tripleo_ironic_neutron_agent.service** **tripleo_ironic_pxe_http.service** **tripleo_ironic_pxe_tftp.service** **tripleo_iscsid.service**	**ironic_api** **ironic_conductor** **ironic_inspector** **ironic_inspector_dnsmasq** **ironic_neutron_agent** **ironic_pxe_http** **ironic_pxe_tftp** **iscsid**
Keepalived	**tripleo_keepalived.service**	**keepalived**
OpenStack Identity (keystone)	**tripleo_keystone.service** **tripleo_keystone_cron.service**	**keystone** **keystone_cron**
Logrotate	**tripleo_logrotate_crond.service**	**logrotate_crond**
Memcached	**tripleo_memcached.service**	**memcached**

コンポーネント	Systemd サービス	コンテナー
OpenStack Workflow (mistral)	**tripleo_mistral_api.service** **tripleo_mistral_engine.service** **tripleo_mistral_event_engine.service** **tripleo_mistral_executor.service**	**mistral_api** **mistral_engine** **mistral_event_engine** **mistral_executor**
MySQL	**tripleo_mysql.service**	**mysql**
OpenStack Networking (neutron)	**tripleo_neutron_api.service** **tripleo_neutron_dhcp.service** **tripleo_neutron_l3_agent.service** **tripleo_neutron_ovs_agent.service**	**neutron-dnsmasq-qdhcp-[UUID]** **neutron_api** **neutron_dhcp** **neutron_l3_agent** **neutron_ovs_agent**
OpenStack Compute (nova)	**tripleo_nova_api.service** **tripleo_nova_api_cron.service** **tripleo_nova_compute.service** **tripleo_nova_conductor.service** **tripleo_nova_metadata.service** **tripleo_nova_placement.service** **tripleo_nova_scheduler.service**	**nova_api** **nova_api_cron** **nova_compute** **nova_conductor** **nova_metadata** **nova_placement** **nova_scheduler**
RabbitMQ	**tripleo_rabbitmq.service**	**rabbitmq**

コンポーネント	Systemd サービス	コンテナー
OpenStack Object Storage (swift)	**tripleo_swift_account_reaper.service** **tripleo_swift_account_server.service** **tripleo_swift_container_server.service** **tripleo_swift_container_updater.service** **tripleo_swift_object_expirer.service** **tripleo_swift_object_server.service** **tripleo_swift_object_updater.service** **tripleo_swift_proxy.service** **tripleo_swift_rsync.service**	**swift_account_reaper** **swift_account_server** **swift_container_server** **swift_container_updater** **swift_object_expirer** **swift_object_server** **swift_object_updater** **swift_proxy** **swift_rsync**
OpenStack Messaging (zaqar)	**tripleo_zaqar.service** **tripleo_zaqar_websocket.service**	**zaqar** **zaqar_websocket**

6.2. SYSTEMD のログの場所とコンテナーベースのログの場所の比較

Systemd ベースの OpenStack ログと対応するコンテナーベースの等価ログを、以下の表に示します。すべてのコンテナーベースのログは物理ホストにマウントされたディレクトリーに保管されるので、物理ホストからログにアクセスすることができます。

OpenStack サービス	Systemd サービスのログ	コンテナーログ
aodh	**/var/log/aodh/**	**/var/log/containers/aodh/** **/var/log/containers/httpd/aodh-api/**
ceilometer	**/var/log/ceilometer/**	**/var/log/containers/ceilometer/**
cinder	**/var/log/cinder/**	**/var/log/containers/cinder/** **/var/log/containers/httpd/cinder-api/**
glance	**/var/log/glance/**	**/var/log/containers/glance/**
gnocchi	**/var/log/gnocchi/**	**/var/log/containers/gnocchi/** **/var/log/containers/httpd/gnocchi-api/**

OpenStack サービス	Systemd サービスのログ	コンテナーログ
heat	**/var/log/heat/**	**/var/log/containers/heat/** **/var/log/containers/httpd/heat-api/** **/var/log/containers/httpd/heat-api-cfn/**
horizon	**/var/log/horizon/**	**/var/log/containers/horizon/** **/var/log/containers/httpd/horizon/**
keystone	**/var/log/keystone/**	**/var/log/containers/keystone** **/var/log/containers/httpd/keystone/**
databases	**/var/log/mariadb/** **/var/log/mongodb/** **/var/log/mysqld.log**	**/var/log/containers/mysql/**
neutron	**/var/log/neutron/**	**/var/log/containers/neutron/** **/var/log/containers/httpd/neutron-api/**
nova	**/var/log/nova/**	**/var/log/containers/nova/** **/var/log/containers/httpd/nova-api/** **/var/log/containers/httpd/nova-placement/**
panko		**/var/log/containers/panko/** **/var/log/containers/httpd/panko-api/**
rabbitmq	**/var/log/rabbitmq/**	**/var/log/containers/rabbitmq/**
redis	**/var/log/redis/**	**/var/log/containers/redis/**
swift	**/var/log/swift/**	**/var/log/containers/swift/**

6.3. SYSTEMD の設定とコンテナーベースの設定の比較

Systemd ベースの OpenStack 設定と対応するコンテナーベースの等価設定を、以下の表に示します。すべてのコンテナーベースの設定の場所は物理ホストで利用可能で、コンテナーにマウントされます。また、それぞれの該当コンテナー内の設定にマージされます (**kolla** により)。

OpenStack サービス	Systemd サービスの設定	コンテナーの設定
aodh	**/etc/aodh/**	**/var/lib/config-data/puppet-generated/aodh/**
ceilometer	**/etc/ceilometer/**	**/var/lib/config-data/puppet-generated/ceilometer/etc/ceilometer/**
cinder	**/etc/cinder/**	**/var/lib/config-data/puppet-generated/cinder/etc/cinder/**
glance	**/etc/glance/**	**/var/lib/config-data/puppet-generated/glance_api/etc/glance/**
gnocchi	**/etc/gnocchi/**	**/var/lib/config-data/puppet-generated/gnocchi/etc/gnocchi/**
haproxy	**/etc/haproxy/**	**/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/**
heat	**/etc/heat/**	**/var/lib/config-data/puppet-generated/heat/etc/heat/** **/var/lib/config-data/puppet-generated/heat_api/etc/heat/** **/var/lib/config-data/puppet-generated/heat_api_cfn/etc/heat/**
horizon	**/etc/openstack-dashboard/**	**/var/lib/config-data/puppet-generated/horizon/etc/openstack-dashboard/**
keystone	**/etc/keystone/**	**/var/lib/config-data/puppet-generated/keystone/etc/keystone/**
databases	**/etc/my.cnf.d/** **/etc/my.cnf**	**/var/lib/config-data/puppet-generated/mysql/etc/my.cnf.d/**

OpenStack サービス	Systemd サービスの設定	コンテナーの設定
neutron	**/etc/neutron/**	**/var/lib/config-data/puppet-generated/neutron/etc/neutron/**
nova	**/etc/nova/**	**/var/lib/config-data/puppet-generated/nova/etc/nova/** **/var/lib/config-data/puppet-generated/nova_placement/etc/nova/**
panko		**/var/lib/config-data/puppet-generated/panko/etc/panko**
rabbitmq	**/etc/rabbitmq/**	**/var/lib/config-data/puppet-generated/rabbitmq/etc/rabbitmq/**
redis	**/etc/redis/** **/etc/redis.conf**	**/var/lib/config-data/puppet-generated/redis/etc/redis/** **/var/lib/config-data/puppet-generated/redis/etc/redis.conf**
swift	**/etc/swift/**	**/var/lib/config-data/puppet-generated/swift/etc/swift/** **/var/lib/config-data/puppet-generated/swift_ringbuilder/etc/swift/**