



# Red Hat OpenStack Platform 16.2

## コンテナ化されたサービスへの移行

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



# Red Hat OpenStack Platform 16.2 コンテナ化されたサービスへの移行

---

コンテナ化された 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

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Transitioning\_to\_Containerized\_Services.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

本ガイドは、コンテナで実行される 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

前書き .....	3
多様性を受け入れるオープンソースの強化 .....	4
RED HAT ドキュメントへのフィードバック (英語のみ) .....	5
第1章 はじめに .....	6
1.1. コンテナ化されたサービスおよび KOLLA .....	6
第2章 コンテナイメージの取得および変更 .....	7
2.1. コンテナイメージの準備 .....	7
2.2. コンテナイメージ準備のパラメーター .....	7
2.3. コンテナイメージタグ付けのガイドライン .....	11
2.4. プライベートレジストリーからのコンテナイメージの取得 .....	12
2.5. イメージ準備エントリーの階層化 .....	14
2.6. 準備プロセスにおけるイメージの変更 .....	15
2.7. コンテナイメージの既存パッケージの更新 .....	16
2.8. コンテナイメージへの追加 RPM ファイルのインストール .....	16
2.9. カスタム DOCKERFILE を使用したコンテナイメージの変更 .....	17
2.10. コンテナイメージ管理用 SATELLITE サーバーの準備 .....	17
第3章 コンテナを使用したアンダークラウドのインストール .....	21
3.1. DIRECTOR の設定 .....	21
3.2. DIRECTOR の設定パラメーター .....	21
3.3. DIRECTOR のインストール .....	27
3.4. コンテナ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 .....	28
第4章 コンテナベースのオーバークラウドのデプロイおよび更新 .....	29
4.1. オーバークラウドのデプロイ .....	29
4.2. オーバークラウドの更新 .....	29
第5章 コンテナ化されたサービスの操作 .....	30
5.1. コンテナ化されたサービスの管理 .....	30
5.2. コンテナ化されたサービスに関するトラブルシューティング .....	33
第6章 SYSTEMD サービスとコンテナ化されたサービスの比較 .....	36
6.1. SYSTEMD サービスとコンテナ化されたサービスの比較 .....	36
6.2. SYSTEMD のログの場所とコンテナベースのログの場所の比較 .....	38
6.3. SYSTEMD の設定とコンテナベースの設定の比較 .....	39



# 前書き

## 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、[弊社の CTO、Chris Wright のメッセージ](#) を参照してください。



## RED HAT ドキュメントへのフィードバック (英語のみ)

弊社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。

### ドキュメントへのダイレクトフィードバック (DDF) 機能の使用 (英語版のみ)

特定の文章、段落、またはコードブロックに対して直接コメントを送付するには、DDF の **Add Feedback** 機能を使用してください。なお、この機能は英語版のドキュメントでのみご利用いただけます。

1. **Multi-page HTML** 形式でドキュメントを表示します。
2. ドキュメントの右上隅に **Feedback** ボタンが表示されていることを確認してください。
3. コメントするテキスト部分をハイライト表示します。
4. **Add Feedback** をクリックします。
5. **Add Feedback** フィールドにコメントを入力します。
6. (オプション) ドキュメントチームが連絡を取り問題についてお伺いできるように、ご自分のメールアドレスを追加します。
7. **Submit** をクリックします。

## 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。ここでは、以下の点について説明します。

- コンテナイメージを取得および変更する方法
- コンテナ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ベースの環境に移行するために、コンテナ化された 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. コンテナ化されたサービスおよび KOLLA

Red Hat OpenStack Platform (RHOSP) の各主要サービスは、コンテナ内で実行されます。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。これには次の効果があります。

- デプロイ中、RHOSP は Red Hat カスタマーポータルからコンテナイメージをプルして実行する。
- **podman** コマンドは、サービスの起動や停止などの管理機能を実行する。
- コンテナをアップグレードするには、新しいコンテナイメージをプルし、既存のコンテナを新しいバージョンのコンテナ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セットを使用します。

## 第2章 コンテナイメージの取得および変更

コンテナ化されたオーバークラウドには、必要なコンテナ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. コンテナイメージの準備

オーバークラウドのインストールには、コンテナ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環境ファイルが必要です。この環境ファイルを生成してカスタマイズし、コンテナイメージの準備に使用します。



#### 注記

オーバークラウド用に特定のコンテナイメージバージョンを設定する必要がある場合は、イメージを特定のバージョンに固定する必要があります。詳しくは、「[Pinning container images for the overcloud](#)」を参照してください。

#### 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イメージ準備ファイルを生成します。

```
$ sudo 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. 要件に合わせて **containers-prepare-parameter.yaml** を変更します。

### 2.2. コンテナイメージ準備のパラメーター

コンテナ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** ストラテジーの各設定で使用するこ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
<b>excludes</b>	設定からイメージ名を除外する正規表現の一覧
<b>includes</b>	設定に含める正規表現の一覧。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。 <b>includes</b> パラメーターを指定すると、 <b>excludes</b> の設定はすべて無視されます。
<b>modify_append_tag</b>	対象となるイメージのタグに追加する文字列。たとえば、16.2.3-5.161のタグが付けられたイメージをプルし、 <b>modify_append_tag</b> を <b>-hotfix</b> に設定すると、director は最終イメージを 16.2.3-5.161-hotfix とタグ付けします。
<b>modify_only_with_labels</b>	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。
<b>modify_role</b>	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
<b>modify_vars</b>	<b>modify_role</b> に渡す変数のディクショナリー

パラメーター	説明
<b>push_destination</b>	<p>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を定義します。</p> <ul style="list-style-type: none"> <li>● <b>true</b> に設定すると、<b>push_destination</b> はホスト名を使用して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。これが推奨される方法です。</li> <li>● <b>false</b> に設定すると、ローカルレジストリーへのプッシュは実行されず、ノードはソースから直接イメージをプルします。</li> <li>● カスタムの値に設定すると、director はイメージを外部のローカルレジストリーにプッシュします。</li> </ul> <p>実稼働環境でこのパラメーターを <b>false</b> に設定した場合、イメージを Red Hat Container Catalog から直接プルする際に、すべてのオーバークラウドノードが同時に外部接続を通じて Red Hat Container Catalog からイメージをプルするため、帯域幅の問題が発生する可能性があります。コンテナイメージをホストする Red Hat Satellite Server から直接プルする場合にのみ、<b>false</b> を使用します。</p> <p><b>push_destination</b> パラメーターが <b>false</b> に設定されているか、または定義されておらずリモートレジストリーで認証が必要な場合は、<b>ContainerImageRegistryLogin</b> パラメーターを <b>true</b> に設定し、<b>ContainerImageRegistryCredentials</b> パラメーターで認証情報を追加します。</p>
<b>pull_source</b>	元のコンテナイメージをプルするソースレジストリー
<b>set</b>	初期イメージの取得場所を定義する、 <b>キー:値</b> 定義のディクショナリー
<b>tag_from_label</b>	<p>指定したコンテナイメージのメタデータラベルの値を使用して、すべてのイメージのタグを作成し、そのタグが付けられたイメージをプルします。たとえば、<b>tag_from_label: {version}-{release}</b> を設定すると、director は <b>version</b> および <b>release</b> ラベルを使用して新しいタグを作成します。あるコンテナについてバージョン 16.2.3 に、リリースは <b>5.161</b> に設定され、タグ 16.2.3-5.1 が設定される可能性があります。<b>set</b> ディクショナリーで <b>tag</b> を定義していない場合に限り、director はこのパラメーターを使用します。</p>



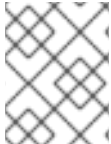
## 重要

イメージをアンダークラウドにプッシュする場合は、**push\_destination: UNDERCLOUD\_IP:PORT** の代わりに **push\_destination: true** を使用します。**push\_destination: true** 手法を使用することで、IPv4 アドレスおよび IPv6 アドレスの両方で一貫性が確保されます。

**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。

キー	説明
<b>ceph_image</b>	Ceph Storage コンテナイメージの名前
<b>ceph_namespace</b>	Ceph Storage コンテナイメージの名前空間
<b>ceph_tag</b>	Ceph Storage コンテナイメージのタグ
<b>ceph_alertmanager_image</b> <b>ceph_alertmanager_namespace</b> <b>ceph_alertmanager_tag</b>	Ceph Storage Alert Manager コンテナイメージの名前、namespace、およびタグ。
<b>ceph_grafana_image</b> <b>ceph_grafana_namespace</b> <b>ceph_grafana_tag</b>	Ceph Storage Grafana コンテナイメージの名前、namespace、およびタグ。
<b>ceph_node_exporter_image</b> <b>ceph_node_exporter_namespace</b> <b>ceph_node_exporter_tag</b>	Ceph Storage Node Exporter コンテナイメージの名前、namespace、およびタグ。
<b>ceph_prometheus_image</b> <b>ceph_prometheus_namespace</b> <b>ceph_prometheus_tag</b>	Ceph Storage Prometheus コンテナイメージの名前、namespace、およびタグ。
<b>name_prefix</b>	各 OpenStack サービスイメージの接頭辞
<b>name_suffix</b>	各 OpenStack サービスイメージの接尾辞
<b>namespace</b>	各 OpenStack サービスイメージの名前空間
<b>neutron_driver</b>	使用する OpenStack Networking (neutron) コンテナを定義するのに使用するドライバー。標準の <b>neutron-server</b> コンテナに設定するには、null 値を使用します。OVN ベースのコンテナを使用するには、 <b>ovn</b> に設定します。

キー	説明
<b>tag</b>	ソースからの全イメージに特定のタグを設定します。定義されていない場合は、director は Red Hat OpenStack Platform のバージョン番号をデフォルト値として使用します。このパラメーターは、 <b>tag_from_label</b> の値よりも優先されます。



### 注記

コンテナイメージでは、Red Hat OpenStack Platform のバージョンに基づいたマルチストリームタグが使用されます。したがって、今後 **latest** タグは使用されません。

## 2.3. コンテナイメージタグ付けのガイドライン

Red Hat コンテナレジストリーでは、すべての Red Hat OpenStack Platform コンテナイメージをタグ付けするのに、特定のバージョン形式が使用されます。この形式は、**version-release** のように各コンテナのラベルのメタデータに従います。

### バージョン

Red Hat OpenStack Platform のメジャーおよびマイナーバージョンに対応します。これらのバージョンは、1つまたは複数のリリースが含まれるストリームとして機能します。

### release

バージョンストリーム内の、特定コンテナイメージバージョンのリリースに対応します。

たとえば、Red Hat OpenStack Platform の最新バージョンが 16.2.3 で、コンテナイメージのリリースが **5.161** の場合、コンテナイメージのタグは 16.2.3-5.161 となります。

Red Hat コンテナレジストリーでは、コンテナイメージバージョンの最新リリースとリンクするメジャーおよびマイナー **version** タグのセットも使用されます。たとえば、16.2 と 16.2.3 の両方が、16.2.3 コンテナストリームの最新 **release** とリンクします。16.2 の新規マイナーリリースが公開されると、16.2 タグは新規マイナーリリースストリームの最新 **release** とリンクします。一方、16.2.3 タグは、引き続き 16.2.3 ストリーム内の最新 **release** とリンクします。

**ContainerImagePrepare** パラメーターには 2 つのサブパラメーターが含まれ、これを使用してダウンロードするコンテナイメージを定義することができます。これらのサブパラメーターは、**set** ディクショナリー内の **tag** パラメーターおよび **tag\_from\_label** パラメーターです。以下のガイドラインを使用して、**tag** または **tag\_from\_label** のどちらを使用するかを判断してください。

- **tag** のデフォルト値は、お使いの OpenStack Platform のメジャーバージョンです。本バージョンでは、16.2 です。これは常に最新のマイナーバージョンおよびリリースに対応します。

```
parameter_defaults:
  ContainerImagePrepare:
    - set:
      ...
      tag: 16.2
      ...
```

- OpenStack Platform コンテナイメージの特定マイナーバージョンに変更するには、タグをマイナーバージョンに設定します。たとえば、16.2.2 に変更するには、**tag** を 16.2.2 に設定します。

-

```
parameter_defaults:
  ContainerImagePrepare:
    - set:
      ...
      tag: 16.2.2
      ...
```

- **tag** を設定すると、インストールおよび更新時に、director は必ず **tag** で設定したバージョンの最新のコンテナイメージ **release** をダウンロードします。
- **tag** を設定しないと、director は最新のメジャーバージョンと共に **tag\_from\_label** の値を使用します。

```
parameter_defaults:
  ContainerImagePrepare:
    - set:
      ...
      # tag: 16.2
      ...
      tag_from_label: '{version}-{release}'
```

- **tag\_from\_label** パラメーターは、Red Hat コンテナレジストリーから検査する最新コンテナイメージリリースのラベルメタデータからタグを生成します。たとえば、特定のコンテナのラベルは、以下の **version** および **release** メタデータを使用します。

```
"Labels": {
  "release": "5.161",
  "version": "16.2.3",
  ...
}
```

- **tag\_from\_label** のデフォルト値は **{version}-{release}** です。これは、各コンテナイメージのバージョンおよびリリースのメタデータラベルに対応します。たとえば、コンテナイメージの **version** に 16.2.3 が、**release** に 5.161 が、それぞれ設定されている場合、コンテナイメージのタグは 16.2.3-5.161 となります。
- **tag** パラメーターは、常に **tag\_from\_label** パラメーターよりも優先されます。**tag\_from\_label** を使用するには、コンテナ準備の設定で **tag** パラメーターを省略します。
- **tag** および **tag\_from\_label** の主な相違点は、次のとおりです。director が **tag** を使用してイメージをプルする場合は、Red Hat コンテナレジストリーがバージョンストリーム内の最新イメージリリースとリンクするメジャーまたはマイナーバージョンのタグだけにに基づきます。一方、**tag\_from\_label** を使用する場合は、director がタグを生成して対応するイメージをプルできるように、各コンテナイメージのメタデータの検査を行います。

## 2.4. プライベートレジストリーからのコンテナイメージの取得

**registry.redhat.io** レジストリーにアクセスしてイメージをプルするには、認証が必要です。**registry.redhat.io** およびその他のプライベートレジストリーで認証するには、**containers-prepare-parameter.yaml** ファイルに **ContainerImageRegistryCredentials** および **ContainerImageRegistryLogin** パラメーターを含めます。

### ContainerImageRegistryCredentials

一部のコンテナイメージレジストリーでは、イメージにアクセスするのに認証が必要です。そのよう



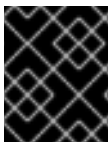
な場合には、**containers-prepare-parameter.yaml** 環境ファイルの **ContainerImageRegistryCredentials** パラメーターを使用します。**ContainerImageRegistryCredentials** パラメーターは、プライベートレジストリーの URL に基づくキーのセットを使用します。それぞれのプライベートレジストリーの URL は、独自のキーと値のペアを使用して、ユーザー名 (キー) およびパスワード (値) を定義します。これにより、複数のプライベートレジストリーに対して認証情報を指定することができます。

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
      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。

複数のレジストリーの認証情報を指定するには、**ContainerImageRegistryCredentials** でレジストリーごとに複数のキー/ペアの値を設定します。

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
      set:
        namespace: registry.redhat.io/...
      ...
    - push_destination: true
      set:
        namespace: registry.internalsite.com/...
      ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
    registry.internalsite.com:
      myuser2: '0th3rp@55w0rd!'
      '192.0.2.1:8787':
        myuser3: '@n0th3rp@55w0rd!'
```



### 重要

デフォルトの **ContainerImagePrepare** パラメーターは、認証が必要な **registry.redhat.io** からコンテナイメージをプルします。

詳細は、「[Red Hat コンテナレジストリーの認証](#)」を参照してください。

## ContainerImageRegistryLogin

**ContainerImageRegistryLogin** パラメーターを使用して、コンテナイメージを取得するために、オーバークラウドノードシステムがリモートレジストリーにログインする必要があるかどうかを制御します。このような状況は、アンダークラウドを使用してイメージをホストする代わりに、オーバークラ

ウドノードがイメージを直接プルする場合に発生します。

特定の設定について、**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、**ContainerImageRegistryLogin** を `true` に設定する必要があります。

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: false
  set:
    namespace: registry.redhat.io/...
    ...
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
  ContainerImageRegistryLogin: true
```

ただし、オーバークラウドノードに **ContainerImageRegistryCredentials** で定義されたレジストリーホストへのネットワーク接続がなく、**ContainerImageRegistryLogin** を `true` に設定すると、ログインを試みる際にデプロイメントが失敗する可能性があります。オーバークラウドノードに **ContainerImageRegistryCredentials** で定義されたレジストリーホストへのネットワーク接続がない場合、**push\_destination** を `true` に、**ContainerImageRegistryLogin** を `false` に設定して、オーバークラウドノードがアンダークラウドからイメージをプルできるようにします。

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
  set:
    namespace: registry.redhat.io/...
    ...
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
  ContainerImageRegistryLogin: false
```

## 2.5. 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、**16.0-44** 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/rhosp-rhel8
    name_prefix: openstack-
    name_suffix: ""
- push_destination: true
  includes:
```

```
- nova-api
set:
  namespace: registry.redhat.io/rhosp-rhel8
  tag: 16.2-44
```

**includes** および **excludes** のパラメーターでは、各エントリーのイメージの絞り込みをコントロールするのに正規表現が使用されます。**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。イメージが一致するとみなされるためには、名前に **includes** または **excludes** の正規表現の値が含まれている必要があります。

## 2.6. 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、変更したそのイメージで直ちにオーバークラウドをデプロイすることが可能です。



### 注記

Red Hat OpenStack Platform(RHOSP)director は、Ceph コンテナ用ではなく、RHOSP コンテナの準備中にイメージの変更をサポートしています。

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。

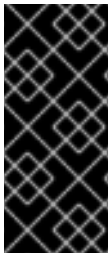
Ansible ロール **tripleo-modify-image** は要求されたロールインターフェースに従い、変更のユースケースに必要な処理を行います。**ContainerImagePrepare** パラメーターの変更固有のキーを使用して、変更をコントロールします。

- **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** 変数をそのロールに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します。

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



### 重要

**openstack tripleo container image prepare** コマンドを使用するには、アンダークラウドに実行中の **image-serve** レジストリーが含まれている必要があります。したがって、**image-serve** レジストリーがインストールされないため、新しいアンダークラウドのインストール前にこのコマンドを実行することはできません。アンダークラウドが正常にインストールされた後に、このコマンドを実行することができます。

## 2.7. コンテナイメージの既存パッケージの更新



### 注記

Red Hat OpenStack Platform(RHOSP)director は、Ceph コンテナ用ではなく、RHOSP コンテナのコンテナイメージ上の既存パッケージの更新をサポートします。

### 手順

- 以下の **ContainerImagePrepare** エントリーの例では、アンダークラウドホストの dnf リポジトリ設定を使用してコンテナイメージのパッケージをすべて更新します。

```
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
...
```

## 2.8. コンテナイメージへの追加 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、ホットフィックスやローカルパッケージビルドなど、パッケージリポジトリ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。



### 注記

Red Hat OpenStack Platform(RHOSP)director は、Ceph コンテナ用ではなく、RHOSP コンテナのコンテナイメージへの追加 RPM ファイルをインストールすることをサポートしています。

### 手順

- 以下の **ContainerImagePrepare** エントリーの例では、**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
...

```

## 2.9. カスタム DOCKERFILE を使用したコンテナイメージの変更

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できます。 **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。



### 注記

Red Hat OpenStack Platform(RHOSP)director は、Ceph コンテナ用ではなく、RHOSP コンテナのカスタム Dockerfile でコンテナイメージの変更をサポートしています。

### 手順

1. 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
...

```

2. **/home/stack/nova-custom/Dockerfile** ファイル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/rhosp-rhel8/openstack-nova-compute:latest

USER "root"

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

USER "nova"

```

## 2.10. コンテナイメージ管理用 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 サーバーにプルし、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。コンテナイメージ管理についての詳細は、『Red Hat Satellite 6 コンテンツ管理ガイド』の「[コンテナイメージの管理](#)」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. すべてのコンテナイメージの一覧を作成します。

```
$ sudo podman search --limit 1000 "registry.redhat.io/rhosp-rhel8/openstack" --format="{{.Name }}" | sort > 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** 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、Satellite 組織に新規製品(**OSP{vernum} Containers**)を作成します。

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

このカスタム製品に、イメージを保管します。

4. **satellite\_images** ファイルからオーバークラウドのコンテナイメージを追加します。

```
$ while read IMAGE; do \
  IMAGE_NAME=$(echo $IMAGE | cut -d"/" -f3 | sed "s/openstack-//g") ; \
  IMAGE_NOURL=$(echo $IMAGE | sed "s/registry.redhat.io///g") ; \
  hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name $IMAGE_NOURL \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name $IMAGE_NAME ; done < satellite_images
```

5. Ceph Storage 4 コンテナイメージを追加します。

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

6. コンテナイメージを同期します。

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

Satellite サーバー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 サーバー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。

7. 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イメージを利用可能にする場合には、コンテナ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「[コンテンツビューの管理](#)」を参照してください。
8. **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" \
  --lifecycle-environment "production" \
  --product "OSP 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 コンテナイメージのタグが表示されます。

9. 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、イメージを準備するデフォルトの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。

```
$ sudo 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。

10. **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。

- **push\_destination**: 選択したコンテナイメージの管理手段に応じて、これを **true** または **false** に設定します。このパラメーターを **false** に設定すると、オーバークラウドノードはイメージを直接 Satellite からプルします。このパラメーターを **true** に設定すると、director はイメージを Satellite からアンダークラウドレジストリーにプルし、オーバークラウドはそのイメージをアンダークラウドレジストリーからプルします。
- **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-myosp16-osp\_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-osp\_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 コンテナイメージの場所を定義するこれらの追加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: false
  set:
    ceph_image: acme-production-myosp16_1-osp_containers-rhceph-4
    ceph_namespace: satellite.example.com:5000
    ceph_tag: latest
    name_prefix: acme-production-myosp16_1-osp_containers-
    name_suffix: ""
    namespace: satellite.example.com:5000
    neutron_driver: null
    tag: '16.2'
  ...
```



### 注記

Red Hat Satellite Server に保存されている特定のコンテナイメージバージョンを使用するには、**タグ** キー/値のペアを、**セット** デクショナリーで特定のバージョンに設定します。たとえば、16.2.2 イメージストリームを使用するには、set デクショナリーに **tag: 16.2.2** を設定します。

**undercloud.conf** 設定ファイルで **containers-prepare-parameter.yaml** 環境ファイルを定義する必要があります。定義しないと、アンダークラウドはデフォルト値を使用します。

```
container_images_file = /home/stack/containers-prepare-parameter.yaml
```



## 第3章 コンテナ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。設定のベースとするためにデフォルトのテンプレートをコピーするには、以下の手順を実施します。

#### 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。

#### 重要

少なくとも、コンテナイメージの設定が含まれる環境ファイルに **container\_images\_file** パラメーターを設定する必要があります。このパラメーターに適切なファイルを正しく設定しないと、director は **ContainerImagePrepare** パラメーターからコンテナイメージのルールセットを取得することや、**ContainerImageRegistryCredentials** パラメーターからコンテナレジストリーの認証情報を取得することができなくなります。

#### デフォルト値

**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** に設定する必要があります。

#### 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アップします。デプロイメントコマンドの実行後もデプロイメント時に使用した一時ファイルをそのまま残すには、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

コンテナ管理用の CLI ツール。このパラメーターは、 **podman** に設定したままにしてください。Red Hat Enterprise Linux 8.4 がサポートするのは、 **podman** だけです。

#### container\_healthcheck\_disabled

コンテナ化されたサービスのヘルスチェックを無効にします。Red Hat は、ヘルスチェックを有効にし、このオプションを **false** に設定したままにすることを推奨します。

#### container\_images\_file

コンテナイメージ情報が含まれる heat 環境ファイル。このファイルには、以下のエントリーを含めることができます。

- 必要なすべてのコンテナ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** パラメーターを有効にし、 **enabled\_hardware\_types** の一覧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\_nova、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

イントロスペクションの 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** 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

アンダークラウドに **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、メトリックデータを消費する Red Hat CloudForms などの他の製品を使用する場合に必要です。

#### 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 コンテナイメージの URL。未設定のままにします。

#### heat\_native

**heat-all** を使用してホストベースのアンダークラウド設定を実行します。**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。

## ipv6\_address\_mode

アンダークラウドのプロビジョニングネットワーク用の IPv6 アドレス設定モード。このパラメーターに設定できる値の一覧を以下に示します。

- dhcpv6-stateless: ルーター広告 (RA) を使用するアドレス設定と DHCPv6 を使用するオプションの情報
- dhcpv6-stateful: DHCPv6 を使用するアドレス設定とオプションの情報。

## 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。標準の PXE を使用するには、このパラメーターを **false** に設定します。PowerPC デプロイメント、またはハイブリッド PowerPC および x86 デプロイメントの場合は、この値を **false** に設定しません。

## 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: em0: <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 em0
        valid_lft 3462sec preferred_lft 3462sec
    inet6 fe80::5054:ff:fe75:2409/64 scope link
        valid_lft forever preferred_lft forever
3: em1: <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 **em0** を使用し、プロビジョニング NIC は、現在設定されていない **em1** を使用します。この場合は、**local\_interface** を **em1** 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。この IP アドレスが環境内の既存の 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、この値をデフォルトの **192.168.24.1/24** のままにします。

IPv6 の場合、ステートフル接続とステートレス接続の両方をサポートするには、ローカル IP アドレス接頭辞の長さが **/64** である必要があります。

## local\_mtu

**local\_interface** に使用する最大伝送単位 (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 または YAML 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、**undercloud.conf** で設定されたネットワークパラメーターを無視します。ボンディン

グを設定する場合、またはインターフェースにオプションを追加する場合に、このパラメーターを使用します。アンダークラウドネットワークインターフェースのカスタマイズに関する詳しい情報は、『Director Installation and Usage』の「[Configuring undercloud network interfaces](#)」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。

#### overcloud\_domain\_name

オーバークラウドをデプロイする際に使用する DNS ドメイン名



#### 注記

オーバークラウドを設定する際に、**CloudDomain** にこのパラメーター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で、デフォルトロールファイルを上書きするのに使用するロールファイル。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、このパラメーターは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ペ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。デフォルト値に含まれるのは、**ctlplane-subnet** サブネットのみです。詳細は、「[サブネット](#)」を参照してください。

#### templates

上書きするヒート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。**DEBUG** ログレベルを有効にするには、この値を **true** に設定します。

#### 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、システムのホスト名設定をすべて適切に定義する必要があります。

#### undercloud\_log\_file

アンダークラウドのインストールログおよび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

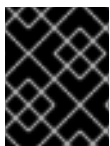
アンダークラウドのインストール時にパッケージを更新するかどうかを定義します。

## 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。



### 重要

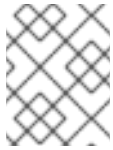
director がサブネットを作成した後、director はサブネットの IP アドレスを変更することはできません。

#### 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。



### 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。

## 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。

## dns\_nameservers

サブネットに固有の DNS ネームサーバー。サブネットにネームサーバーが定義されていない場合には、サブネットは **undercloud\_nameservers** パラメーターで定義されるネームサーバーを使用します。

## 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。

## 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

検査プロセス中に使用するこのネットワーク上のノードの一時的な IP 範囲。この範囲は、**dhcp\_start** と **dhcp\_end** で定義された範囲と重複することはできませんが、同じ IP サブネット内になければなりません。

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
```

このコマンドにより、director の設定スクリプトが起動します。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、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 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も自動的に起動します。以下のコマンドを使用して、有効になったコンテナを確認することができます。

```
[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 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主要な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。director を使用して、RHOSP 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します。

#### 手順

1. アンダークラウドノードで、**stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. **dnf update** コマンドで director の主なパッケージを更新します。

```
$ sudo dnf update -y python3-tripleoclient* tripleo-ansible ansible
```

4. **openstack undercloud upgrade** コマンドで、アンダークラウドの環境を更新します。

```
$ openstack undercloud upgrade
```

5. アンダークラウドの更新プロセスが完了するまで待ちます。
6. 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
```

7. ノードがブートするまで待ちます。



## 第4章 コンテナ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 コンテナ化されたサービスの操作

本章では、コンテナを管理するコマンドの例および OpenStack Platform コンテナ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. コンテナ化されたサービスの管理

Red Hat OpenStack Platform (RHOSP) 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。

#### コンテナとイメージの一覧表示

実行中のコンテナ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
```

#### コンテナのプロパティの確認

コンテナまたはコンテナイメージのプロパティを表示するには、**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の管理

以前のバージョンの OpenStack Platform では、コンテナは Docker およびそのデーモンで管理されていました。OpenStack Platform 16 では、Systemd サービスインターフェースでコンテナのライフサイクルが管理されます。それぞれのコンテナはサービスであり、Systemd コマンドを実行して各コンテナ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

コンテナ内のサービス設定ファイルに加えた変更は、コンテナの再起動後には元に戻ります。これは、コンテナ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** コンテナ内の `/etc/keystone/keystone.conf` を編集してコンテナを再起動すると、そのコンテナ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 コンテナの監視

Systemd タイマーインターフェースは、コンテナのヘルスチェックを管理します。各コンテナ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", "..."}]}}
```

コンテナタイマーを停止、起動、再起動、およびコンテナ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。チェックを手動で開始することもできます。



## 注記

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

## コンテナログの確認

OpenStack Platform 16 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、すべてのコンテナの標準出力 (stdout) と標準エラー (stderr) が、コンテナごとに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 コンテナを設定します。これにより、**container-puppet-\*** コンテナ等の削除されたコンテナを含め、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。

コンテナが置き換えられた場合には、新しいコンテナは同じログファイルにログを出力します。**podman** はコンテナ ID ではなくコンテナ名を使用するためです。

**podman logs** コマンドを使用して、コンテナ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
```

## コンテナへのアクセス

コンテナ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。

### コンテナログの確認

各コンテナは、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のエラーの原因が記載されています。

### コンテナの検査

状況によっては、コンテナ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。

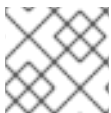
```
$ sudo podman inspect keystone
```

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、稼働中のコンテナ内でコマンドを実行します。たとえば、**keystone** コンテナ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
```

### コンテナのエクスポート

コンテナ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化されたサービスの比較

本章では、コンテナ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
OpenStack Image Storage (glance)	<b>tripleo_glance_api.service</b>	<b>glance_api</b>
HAProxy	<b>tripleo_haproxy.service</b>	<b>haproxy</b>
OpenStack Orchestration (heat)	<b>tripleo_heat_api.service</b> <b>tripleo_heat_api_cfn.service</b> <b>tripleo_heat_api_cron.service</b> <b>tripleo_heat_engine.service</b>	<b>heat_api</b> <b>heat_api_cfn</b> <b>heat_api_cron</b> <b>heat_engine</b>
OpenStack Bare Metal (ironic)	<b>tripleo_ironic_api.service</b> <b>tripleo_ironic_conductor.service</b> <b>tripleo_ironic_inspector.service</b> <b>tripleo_ironic_inspector_dnsmasq.service</b> <b>tripleo_ironic_neutron_agent.service</b> <b>tripleo_ironic_pxe_http.service</b> <b>tripleo_ironic_pxe_tftp.service</b> <b>tripleo_iscsid.service</b>	<b>ironic_api</b> <b>ironic_conductor</b> <b>ironic_inspector</b> <b>ironic_inspector_dnsmasq</b> <b>ironic_neutron_agent</b> <b>ironic_pxe_http</b> <b>ironic_pxe_tftp</b> <b>iscsid</b>
Keepalived	<b>tripleo_keepalived.service</b>	<b>keepalived</b>
OpenStack Identity (keystone)	<b>tripleo_keystone.service</b> <b>tripleo_keystone_cron.service</b>	<b>keystone</b> <b>keystone_cron</b>
Logrotate	<b>tripleo_logrotate_crond.service</b>	<b>logrotate_crond</b>
Memcached	<b>tripleo_memcached.service</b>	<b>memcached</b>



コンポーネント	Systemd サービス	コンテナ
OpenStack Workflow (mistral)	<b>tripleo_mistral_api.service</b> <b>tripleo_mistral_engine.service</b> <b>tripleo_mistral_event_engine.service</b> <b>tripleo_mistral_executor.service</b>	<b>mistral_api</b> <b>mistral_engine</b> <b>mistral_event_engine</b> <b>mistral_executor</b>
MySQL	<b>tripleo_mysql.service</b>	<b>mysql</b>
OpenStack Networking (neutron)	<b>tripleo_neutron_api.service</b> <b>tripleo_neutron_dhcp.service</b> <b>tripleo_neutron_l3_agent.service</b> <b>tripleo_neutron_ovs_agent.service</b>	<b>neutron_api</b> <b>neutron_dhcp</b> <b>neutron_l3_agent</b> <b>neutron_ovs_agent</b>
OpenStack Compute (nova)	<b>tripleo_nova_api.service</b> <b>tripleo_nova_api_cron.service</b> <b>tripleo_nova_compute.service</b> <b>tripleo_nova_conductor.service</b> <b>tripleo_nova_metadata.service</b> <b>tripleo_nova_placement.service</b> <b>tripleo_nova_scheduler.service</b>	<b>nova_api</b> <b>nova_api_cron</b> <b>nova_compute</b> <b>nova_conductor</b> <b>nova_metadata</b> <b>nova_placement</b> <b>nova_scheduler</b>
RabbitMQ	<b>tripleo_rabbitmq.service</b>	<b>rabbitmq</b>
OpenStack Object Storage (swift)	<b>tripleo_swift_account_reaper.service</b> <b>tripleo_swift_account_server.service</b> <b>tripleo_swift_container_server.service</b> <b>tripleo_swift_container_updater.service</b> <b>tripleo_swift_object_expirer.service</b> <b>tripleo_swift_object_server.service</b> <b>tripleo_swift_object_updater.service</b> <b>tripleo_swift_proxy.service</b> <b>tripleo_swift_rsync.service</b>	<b>swift_account_reaper</b> <b>swift_account_server</b> <b>swift_container_server</b> <b>swift_container_updater</b> <b>swift_object_expirer</b> <b>swift_object_server</b> <b>swift_object_updater</b> <b>swift_proxy</b> <b>swift_rsync</b>

コンポーネント	Systemd サービス	コンテナ
OpenStack Messaging (zaqar)	<b>tripleo_zaqar.service</b>	<b>zaqar</b>
	<b>tripleo_zaqar_websocket.service</b>	<b>zaqar_websocket</b>

## 6.2. SYSTEMD のログの場所とコンテナ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	コンテナログ
aodh	<b>/var/log/aodh/</b>	<b>/var/log/containers/aodh/</b> <b>/var/log/containers/httpd/aodh-api/</b>
ceilometer	<b>/var/log/ceilometer/</b>	<b>/var/log/containers/ceilometer/</b>
cinder	<b>/var/log/cinder/</b>	<b>/var/log/containers/cinder/</b> <b>/var/log/containers/httpd/cinder-api/</b>
glance	<b>/var/log/glance/</b>	<b>/var/log/containers/glance/</b>
gnocchi	<b>/var/log/gnocchi/</b>	<b>/var/log/containers/gnocchi/</b> <b>/var/log/containers/httpd/gnocchi-api/</b>
heat	<b>/var/log/heat/</b>	<b>/var/log/containers/heat/</b> <b>/var/log/containers/httpd/heat-api/</b> <b>/var/log/containers/httpd/heat-api-cfn/</b>
horizon	<b>/var/log/horizon/</b>	<b>/var/log/containers/horizon/</b> <b>/var/log/containers/httpd/horizon/</b>

OpenStack サービス	Systemd サービスのログ	コンテナログ
keystone	<code>/var/log/keystone/</code>	<code>/var/log/containers/keystone</code> <code>/var/log/containers/httpd/keystone/</code>
databases	<code>/var/log/mariadb/</code> <code>/var/log/mongodb/</code> <code>/var/log/mysql.log</code>	<code>/var/log/containers/mysql/</code>
neutron	<code>/var/log/neutron/</code>	<code>/var/log/containers/neutron/</code> <code>/var/log/containers/httpd/neutron-api/</code>
nova	<code>/var/log/nova/</code>	<code>/var/log/containers/nova/</code> <code>/var/log/containers/httpd/nova-api/</code> <code>/var/log/containers/httpd/placement/</code>
panko		<code>/var/log/containers/panko/</code> <code>/var/log/containers/httpd/panko-api/</code>
rabbitmq	<code>/var/log/rabbitmq/</code>	<code>/var/log/containers/rabbitmq/</code>
redis	<code>/var/log/redis/</code>	<code>/var/log/containers/redis/</code>
swift	<code>/var/log/swift/</code>	<code>/var/log/containers/swift/</code>

### 6.3. SYSTEMD の設定とコンテナ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、コンテナにマウントされます。また、それぞれの該当コンテナ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	コンテナの設定
aodh	<code>/etc/aodh/</code>	<code>/var/lib/config-data/puppet-generated/aodh/</code>

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

OpenStack サービス	Systemd サービスの設定	コンテナの設定
nova	<b>/etc/nova/</b>	<b>/var/lib/config-data/puppet-generated/nova/etc/nova/</b> <b>/var/lib/config-data/puppet-generated/etc/placement/</b>
panko		<b>/var/lib/config-data/puppet-generated/panko/etc/panko</b>
rabbitmq	<b>/etc/rabbitmq/</b>	<b>/var/lib/config-data/puppet-generated/rabbitmq/etc/rabbitmq/</b>
redis	<b>/etc/redis/</b> <b>/etc/redis.conf</b>	<b>/var/lib/config-data/puppet-generated/redis/etc/redis/</b> <b>/var/lib/config-data/puppet-generated/redis/etc/redis.conf</b>
swift	<b>/etc/swift/</b>	<b>/var/lib/config-data/puppet-generated/swift/etc/swift/</b> <b>/var/lib/config-data/puppet-generated/swift_ringbuilder/etc/swift/</b>