# Red Hat OpenStack Platform 16.1

## オーバークラウドの高度なカスタマイズ

Red Hat OpenStack Platform director を使用して高度な機能を設定する方法

Last Updated: 2021-05-09

# Red Hat OpenStack Platform 16.1 オーバークラウドの高度なカスタマイズ

Red Hat OpenStack Platform director を使用して高度な機能を設定する方法

OpenStack Team
rhos-docs@redhat.com

## 法律上の通知



## 概要

Red Hat OpenStack Platform director を使用して、Red Hat OpenStack Platform (RHOSP) のエンタープライズ環境向けに特定の高度な機能を設定します。これには、ネットワークの分離、ストレージの設定、SSL 通信、一般的な設定の方法が含まれます。

# 目次

# 多様性を受け入れるオープンソースの強化

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

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 オーバークラウド設定の概要

Red Hat OpenStack Platform (RHOSP) director は、完全な機能を実装した OpenStack 環境 (オーバークラウドとしても知られる) をプロビジョニング/作成するためのツールセットを提供します。基本的なオーバークラウドの準備と設定については、**『director のインストールと使用方法』**に記載しています。ただし、実稼働環境レベルのオーバークラウドには、以下のような追加設定が必要となる場合があります。

- 既存のネットワークインフラストラクチャーにオーバークラウドを統合するための基本的なネットワーク設定
- 特定の OpenStack ネットワークトラフィック種別を対象とする個別の VLAN 上でのネットワークトラフィックの分離
- パブリックエンドポイント上の通信をセキュリティー保護するための SSL 設定
- NFS、iSCSI、Red Hat Ceph Storage、および複数のサードパーティー製ストレージデバイスなどのストレージオプション
- Red Hat コンテンツ配信ネットワークまたは内部の Red Hat Satellite 5/6 サーバーへのノードの登録
- さまざまなシステムレベルのオプション
- OpenStack サービスの多様なオプション

**注記**

本ガイドに記載する例は、オーバークラウドを設定するためのオプションのステップです。これらのステップは、オーバークラウドに追加の機能を提供する場合にのみ必要です。環境の要件に該当するステップを使用してください。

# 第2章 HEAT テンプレートの概要

本章のカスタム設定では、heat テンプレートおよび環境ファイルを使用して、オーバークラウドの特定の機能を定義します。本項には、Red Hat OpenStack Platform director に関連した heat テンプレートの構造や形式を理解するための基本的な説明を記載します。

## 2.1. HEAT テンプレート

director は、Heat Orchestration Template (HOT) をオーバークラウドデプロイメントプランのテンプレート形式として使用します。HOT 形式のテンプレートは、通常 YAML 形式で表現されます。テンプレートの目的は、OpenStack Orchestration (heat) が作成するリソースのコレクションであるスタックを定義および作成し、リソースを設定することです。リソースとは、コンピュートリソース、ネットワーク設定、セキュリティーグループ、スケーリングルール、カスタムリソースなどの Red Hat OpenStack Platform (RHOSP) のオブジェクトを指します。

heat テンプレートは、3 つの主要なセクションで構成されます。

**parameters**

これらは、heat に渡される設定 (スタックのカスタマイズが可能) およびパラメーターのデフォルト値 (値を渡さない場合) です。これらの設定がテンプレートの **parameters** セクションで定義されます。

**resources**

**resources** セクションを使用して、このテンプレートを使用してスタックをデプロイする際に作成することができるリソース (コンピュートインスタンス、ネットワーク、ストレージボリューム等) を定義します。Red Hat OpenStack Platform (RHOSP) には、全コンポーネントに対応するコアリソースのセットが含まれています。これらは、スタックの一部として作成/設定する固有のオブジェクトです。RHOSP には、全コンポーネントに対応するコアリソースのセットが含まれています。これらがテンプレートの **resources** セクションで定義されます。

**outputs**

**outputs** セクションを使用して、スタックの作成後にクラウドユーザーがアクセスできるアウトプットパラメーターを宣言します。クラウドユーザーはこれらのパラメーターを使用して、デプロイしたインスタンスの IP アドレスやスタックの一部としてデプロイされた Web アプリケーションの URL 等のスタックの詳細を要求することができます。

基本的な heat テンプレートの例:

```
heat_template_version: 2013-05-23

description: > A very basic Heat template.

parameters:
  key_name:
    type: string
    default: lars
    description: Name of an existing key pair to use for the instance
  flavor:
    type: string
    description: Instance type for the instance to be created
    default: m1.small
  image:
    type: string
    default: cirros
    description: ID or name of the image to use for the instance
```

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      name: My Cirros Instance
      image: { get_param: image }
      flavor: { get_param: flavor }
      key_name: { get_param: key_name }

output:
  instance_name:
    description: Get the instance's name
    value: { get_attr: [ my_instance, name ] }
```

このテンプレートは、リソース種別 **type: OS::Nova::Server** を使用して、クラウドユーザーが指定する特定のフレーバー、イメージ、およびキーで **my_instance** というインスタンスを作成します。このスタックは、**My Cirros Instance** という **instance_name** の値を返すことができます。

heat がテンプレートを処理する際には、テンプレートのスタックとリソーステンプレートの子スタックセットを作成します。これにより、テンプレートで定義するメインのスタックを最上位とするスタックの階層が作成されます。以下のコマンドを使用して、スタックの階層を表示することができます。

```
$ openstack stack list --nested
```

## 2.2. 環境ファイル

環境ファイルとは特別な種類のテンプレートで、これを使用して heat テンプレートをカスタマイズすることができます。コアの heat テンプレートに加えて、環境ファイルをデプロイメントコマンドに追加することができます。環境ファイルには、3 つの主要なセクションが含まれます。

**resource_registry**

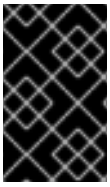
このセクションでは、他の heat テンプレートにリンクしたカスタムのリソース名を定義します。これにより、コアリソースコレクションに存在しないカスタムのリソースを作成することができます。

**parameters**

これらは、最上位のテンプレートのパラメーターに適用する共通設定です。たとえば、入れ子状のスタックをデプロイするテンプレートの場合には (リソースレジストリーマッピング等)、パラメーターは最上位のテンプレートにのみ適用され、入れ子状のリソースのテンプレートには適用されません。

**parameter_defaults**

これらのパラメーターは、全テンプレートのパラメーターのデフォルト値を変更します。たとえば、入れ子状のスタックをデプロイする heat テンプレートの場合には (リソースレジストリーマッピングなど)、パラメーターのデフォルト値がすべてのテンプレートに適用されます。



**重要**

パラメーターがオーバークラウドのすべてのスタックテンプレートに適用されるように、オーバークラウド用にカスタムの環境ファイルを作成する場合には、**parameters** ではなく **parameter_defaults** を使用します。

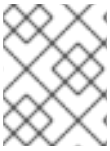
基本的な環境ファイルの例:

```
resource_registry:
  OS::Nova::Server::MyServer: myserver.yaml

parameter_defaults:
  NetworkName: my_network

parameters:
  MyIP: 192.168.0.1
```

特定の heat テンプレート (**my_template.yaml**) からスタックを作成する際に、この環境ファイル (**my_env.yaml**) を追加します。**my_env.yaml** ファイルにより、**OS::Nova::Server::MyServer** という新しいリソース種別が作成されます。**myserver.yaml** ファイルは、このリソース種別を実装する heat テンプレートファイルで、このファイルでの設定が元の設定よりも優先されます。**my_template.yaml** ファイルに **OS::Nova::Server::MyServer** リソースを含めることができます。

**MyIP** は、この環境ファイルと共にデプロイを行うメインの heat テンプレートにしかパラメーターを適用しません。この例では、**MyIP** は **my_template.yaml** のパラメーターにのみ適用されます。

**NetworkName** は、メインの heat テンプレート **my_template.yaml** とメインのテンプレートに含まれるリソースに関連付けられたテンプレート (上記の例では **OS::Nova::Server::MyServer** リソースおよびその **myserver.yaml** テンプレート) の両方に適用されます。



**注記**

RHOSP が heat テンプレートファイルをカスタムテンプレートリソースとして使用するには、ファイルの拡張子を .yaml または .template のいずれかにする必要があります。

## 2.3. オーバークラウドのコア HEAT テンプレート

director には、オーバークラウド用のコア heat テンプレートコレクションおよび環境ファイルコレクションが含まれます。このコレクションは、**/usr/share/openstack-tripleo-heat-templates** に保存されています。

このテンプレートコレクションの主なファイルおよびディレクトリーは、以下のとおりです。

**overcloud.j2.yaml**

オーバークラウド環境を作成するのに director が使用するメインのテンプレートファイル。このファイルでは Jinja2 構文を使用してテンプレートの特定セクションを繰り返し、カスタムロールを作成します。Jinja2 フォーマットは、オーバークラウドのデプロイメントプロセス中に YAML にレンダリングされます。

**overcloud-resource-registry-puppet.j2.yaml**

オーバークラウド環境を作成するのに director が使用するメインの環境ファイル。このファイルは、オーバークラウドイメージ上に保存される Puppet モジュールの設定セットを提供します。director により各ノードにオーバークラウドのイメージが書き込まれると、heat はこの環境ファイルに登録されているリソースを使用して各ノードの Puppet 設定を開始します。このファイルでは Jinja2 構文を使用してテンプレートの特定セクションを繰り返し、カスタムロールを作成します。Jinja2 フォーマットは、オーバークラウドのデプロイメントプロセス中に YAML にレンダリングされます。

**roles_data.yaml**

このファイルにはオーバークラウド内のロールの定義が含まれ、サービスを各ロールにマッピングします。

**network_data.yaml**

このファイルには、オーバークラウド内のネットワーク定義、およびそれらのサブネット、割り当てプール、仮想 IP のステータス等の属性が含まれます。デフォルトの **network_data.yaml** ファイルにはデフォルトのネットワーク (External、Internal Api、Storage、Storage Management、Tenant、Management) が含まれます。カスタムの **network_data.yaml** ファイルを作成して、**openstack overcloud deploy** コマンドに **-n** オプションで追加することができます。

**plan-environment.yaml**

このファイルには、オーバークラウドプランのメタデータの定義が含まれます。これには、プラン名、使用するメインのテンプレート、およびオーバークラウドに適用する環境ファイルが含まれます。

**capabilities-map.yaml**

このファイルには、オーバークラウドプランの環境ファイルのマッピングが含まれます。

**deployment**

このディレクトリーには、heat テンプレートが含まれます。**overcloud-resource-registry-puppet.j2.yaml** 環境ファイルは、このディレクトリーのファイルを使用して、各ノードに Puppet の設定が適用されるようにします。

**environments**

このディレクトリーには、オーバークラウドの作成に使用可能なその他の heat 環境ファイルが含まれます。これらの環境ファイルは、作成された Red Hat OpenStack Platform (RHOSP) 環境の追加の機能を有効にします。たとえば、ディレクトリーには Cinder NetApp のバックエンドストレージを有効にする環境ファイル (**cinder-netapp-config.yaml**) が含まれています。

**network**

このディレクトリーには、分離ネットワークおよびポートを作成するのに使用できる heat テンプレートのセットが含まれます。

**puppet**

このディレクトリーには、Puppet 設定を制御するテンプレートが含まれます。**overcloud-resource-registry-puppet.j2.yaml** 環境ファイルは、このディレクトリーのファイルを使用して、各ノードに Puppet の設定が適用されるようにします。

**puppet/services**

このディレクトリーには、全サービス設定用のレガシー heat テンプレートが含まれます。**puppet/services** ディレクトリー内のほとんどのテンプレートが、**deployment** ディレクトリーのテンプレートに置き換えられています。

**extraconfig**

このディレクトリーには、追加機能を有効にするのに使用できるテンプレートが含まれます。

**firstboot**

このディレクトリーには、ノードの初回作成時に director が使用する **first_boot** スクリプトの例が含まれています。

## 2.4. プランの環境メタデータ

プランの環境メタデータファイルで、オーバークラウドプランのメタデータを定義することができます。director は、オーバークラウドの作成時およびオーバークラウドプランのインポート/エクスポート時にメタデータを適用します。

プランの環境ファイルを使用して、OpenStack Workflow (Mistral) サービスを介して director が実行可能なワークフローを定義します。プランの環境メタデータファイルには、以下のパラメーターが含まれます。

**version**

テンプレートのバージョン

name
: オーバークラウドプランおよびプランのファイルを保管するのに使用する OpenStack Object Storage (swift) 内のコンテナーの名前

template
: オーバークラウドのデプロイメントに使用するコアの親テンプレート。これは、大半の場合は **overcloud.yaml.j2** テンプレートをレンダリングしたバージョンの **overcloud.yaml** です。

environments
: 使用する環境ファイルの一覧を定義します。各環境ファイルの名前および相対位置を **path** サブパラメーターで指定します。

parameter_defaults
: オーバークラウドで使用するパラメーターのセット。これは、標準の環境ファイルの **parameter_defaults** セクションと同じように機能します。

passwords
: オーバークラウドのパスワードに使用するパラメーターのセット。これは、標準の環境ファイルの **parameter_defaults** セクションと同じように機能します。通常、このセクションは director が無作為に生成したパスワードにより自動的に設定されます。

workflow_parameters
: このパラメータを使用して、OpenStack Workflow (mistral) の名前空間にパラメーターのセットを指定します。このパラメーターを使用して、特定のオーバークラウドパラメーターを自動生成することができます。

プランの環境ファイルの構文の例を、以下のスニペットに示します。

```
version: 1.0
name: myovercloud
description: 'My Overcloud Plan'
template: overcloud.yaml
environments:
- path: overcloud-resource-registry-puppet.yaml
- path: environments/containers-default-parameters.yaml
- path: user-environment.yaml
parameter_defaults:
  ControllerCount: 1
  ComputeCount: 1
  OvercloudComputeFlavor: compute
  OvercloudControllerFlavor: control
workflow_parameters:
  tripleo.derive_params.v1.derive_parameters:
    num_phy_cores_per_numa_node_for_pmd: 2
```

**-p** オプションを使用して、**openstack overcloud deploy** コマンドにプランの環境メタデータファイルを追加することができます。

```
(undercloud) $ openstack overcloud deploy --templates \
  -p /my-plan-environment.yaml \
  [OTHER OPTIONS]
```

以下のコマンドを使用して、既存のオーバークラウドプラン用のプランメタデータを確認することもできます。

```
(undercloud) $ openstack object save overcloud plan-environment.yaml --file -
```

## 2.5. オーバークラウド作成時の環境ファイルの追加

**-e** オプションを使用して、デプロイメントコマンドに環境ファイルを追加します。必要に応じていくつでも環境ファイルを追加することができます。ただし、後で指定する環境ファイルで定義されるパラメーターとリソースが優先されることになるため、環境ファイルの順番は重要です。以下の例では、2つの環境ファイルに共通のリソース種別 **OS::TripleO::NodeExtraConfigPost** と共通のパラメーター **TimeZone** が含まれます。

**environment-file-1.yaml**

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/template-1.yaml

parameter_defaults:
  RabbitFDLimit: 65536
  TimeZone: 'Japan'
```

**environment-file-2.yaml**

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/template-2.yaml

parameter_defaults:
  TimeZone: 'Hongkong'
```

デプロイメントコマンドに両方の環境ファイルを含めます。

```
$ openstack overcloud deploy --templates -e environment-file-1.yaml -e environment-file-2.yaml
```

**openstack overcloud deploy** コマンドは、以下のプロセスを順に実行します。

1. コア heat テンプレートコレクションからデフォルト設定を読み込みます。
2. **environment-file-1.yaml** の設定を適用します。この設定により、デフォルト設定と共通している設定は上書きされます。
3. **environment-file-2.yaml** の設定を適用します。この設定により、デフォルト設定および **environment-file-1.yaml** と共通している設定は上書きされます。

これにより、オーバークラウドのデフォルト設定が以下のように変更されます。

- **OS::TripleO::NodeExtraConfigPost** リソースは、**environment-file-2.yaml** で定義されるとおりに **/home/stack/templates/template-2.yaml** に設定されます。
- **TimeZone** パラメーターは、**environment-file-2.yaml** で定義されるとおりに **Hongkong** に設定されます。
- **RabbitFDLimit** パラメーターは、**environment-file-1.yaml** で定義されるとおりに **65536** に設定されます。この値は、**environment-file-2.yaml** によっては変更されません。

この手法を使用して、複数の環境ファイルの値が競合することなく、オーバークラウドのカスタム設定を定義することができます。

## 2.6. カスタムのコア HEAT テンプレートの使用

オーバークラウドの作成時に、director は **/usr/share/openstack-tripleo-heat-templates** にある heat テンプレートのコアセットを使用します。このコアテンプレートコレクションをカスタマイズする場合は、以下の git ワークフローを使用してカスタムテンプレートコレクションを管理します。

### カスタムテンプレートコレクションの初期化

heat テンプレートコレクションが含まれる初期 git リポジトリーを作成します。

1. テンプレートコレクションを **/home/stack/templates** ディレクトリーにコピーします。

```
$ cd ~/templates
$ cp -r /usr/share/openstack-tripleo-heat-templates .
```

2. カスタムテンプレートのディレクトリーに移動して git リポジトリーを初期化します。

```
$ cd ~/templates/openstack-tripleo-heat-templates
$ git init .
```

3. Git のユーザー名およびメールアドレスを設定します。

```
$ git config --global user.name "<USER_NAME>"
$ git config --global user.email "<EMAIL_ADDRESS>"
```

**<USER_NAME>** を使用するユーザー名に置き換えます。**<EMAIL_ADDRESS>** をご自分のメールアドレスに置き換えます。

4. 初期コミットに向けて全テンプレートをステージします。

```
$ git add *
```

5. 初期コミットを作成します。

```
$ git commit -m "Initial creation of custom core heat templates"
```

これで、最新のコアテンプレートコレクションを格納する初期 **master** ブランチが作成されます。このブランチは、カスタムブランチのベースとして使用し、新規テンプレートバージョンをこのブランチにマージします。

### カスタムブランチの作成と変更のコミット

カスタムブランチを使用して、コアテンプレートコレクションの変更を保管します。以下の手順に従って、**my-customizations** ブランチを作成してカスタマイズを追加します。

1. **my-customizations** ブランチを作成して、そのブランチに切り替えます。

```
$ git checkout -b my-customizations
```

2. カスタムブランチ内のファイルを編集します。

3. 変更を git にステージします。

```
$ git add [edited files]
```

4. カスタムブランチに変更をコミットします。

```
-
```

```
$ git commit -m "[Commit message for custom changes]"
```

このコマンドにより、変更がコミットとして **my-customizations** ブランチに追加されます。**master** ブランチを更新するには、**master** から **my-customizations** にリベースすると、git はこれらのコミットを更新されたテンプレートに追加します。これは、カスタマイズをトラッキングして、今後テンプレートが更新された際にそれらを再生するのに役立ちます。

### カスタムテンプレートコレクションの更新

アンダークラウドを更新する際に、**openstack-tripleo-heat-templates** パッケージも更新を受け取る可能性があります。このような場合には、カスタムテンプレートコレクションも更新する必要があります。

1. **openstack-tripleo-heat-templates** パッケージのバージョンを環境変数として保存します。

```
$ export PACKAGE=$(rpm -qv openstack-tripleo-heat-templates)
```

2. テンプレートコレクションのディレクトリーに移動して、更新されたテンプレート用に新規ブランチを作成します。

```
$ cd ~/templates/openstack-tripleo-heat-templates
$ git checkout -b $PACKAGE
```

3. そのブランチの全ファイルを削除して、新しいバージョンに置き換えます。

```
$ git rm -rf *
$ cp -r /usr/share/openstack-tripleo-heat-templates/* .
```

4. 初期コミット用にすべてのテンプレートを追加します。

```
$ git add *
```

5. パッケージ更新のコミットを作成します。

```
$ git commit -m "Updates for $PACKAGE"
```

6. このブランチを master にマージします。git 管理システム (例: GitLab) を使用している場合には、管理ワークフローを使用してください。git をローカルで使用している場合には、**master** ブランチに切り替えてから **git merge** コマンドを実行してマージします。

```
$ git checkout master
$ git merge $PACKAGE
```

**master** ブランチに最新のコアテンプレートコレクションが含まれるようになりました。これで、**my-customization** ブランチを更新されたコレクションからリベースできます。

### カスタムブランチのリベース

以下の手順に従って **my-customization** ブランチを更新します。

1. **my-customizations** ブランチに切り替えます。

```
$ git checkout my-customizations
```

2. このブランチを **master** からリベースします。

```
$ git rebase master
```

これにより、**my-customizations** ブランチが更新され、このブランチに追加されたカスタムコミットが再生されます。

また、リベース中に発生した競合を解決する必要もあります。

1. どのファイルで競合が発生しているかを確認します。

```
$ git status
```

2. 特定したテンプレートファイルで競合を解決します。
3. 解決したファイルを追加します。

```
$ git add [resolved files]
```

4. リベースを続行します。

```
$ git rebase --continue
```

### カスタムテンプレートのデプロイメント

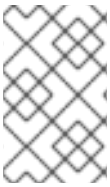
以下の手順に従って、カスタムテンプレートコレクションをデプロイします。

1. 必ず **my-customization** ブランチに切り替えた状態にします。

```
git checkout my-customizations
```

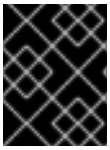
2. **openstack overcloud deploy** コマンドに **--templates** オプションを付けて、ローカルのテンプレートディレクトリーを指定して実行します。

```
$ openstack overcloud deploy --templates /home/stack/templates/openstack-tripleo-heat-templates [OTHER OPTIONS]
```



**注記**

ディレクトリーの指定をせずに **--templates** オプションを使用すると、director はデフォルトのテンプレートディレクトリー (**/usr/share/openstack-tripleo-heat-templates**) を使用します。



**重要**

Red Hat は、heat テンプレートコレクションを変更する代わりに「4章 *設定フック*」に記載の方法を使用することを推奨します。

## 2.7. JINJA2 構文のレンダリング

**/usr/share/openstack-tripleo-heat-templates** のコア heat テンプレートには、**j2.yaml** のファイル拡張子を持つファイルが多数含まれています。これらのファイルには Jinja2 テンプレート構文が含まれ、director はこれらのファイルを **.yaml** 拡張子を持つ等価な静的 heat テンプレートにレンダリングしま

す。たとえば、メインの **overcloud.j2.yaml** ファイルは **overcloud.yaml** にレンダリングされます。director はレンダリングされた **overcloud.yaml** ファイルを使用します。

Jinja2 タイプの heat テンプレートでは、Jinja2 構文を使用して反復値のパラメーターおよびリソースを作成します。たとえば、**overcloud.j2.yaml** ファイルには以下のスニペットが含まれます。

```
parameters:
...
{% for role in roles %}
  ...
  {{role.name}}Count:
    description: Number of {{role.name}} nodes to deploy
    type: number
    default: {{role.CountDefault|default(0)}}
  ...
{% endfor %}
```

director が Jinja2 構文をレンダリングする場合、director は **roles_data.yaml** ファイルで定義されるロールを繰り返し処理し、**{{role.name}}Count** パラメーターにロール名を代入します。デフォルトの **roles_data.yaml** ファイルには 5 つのロールが含まれ、ここでの例からは以下のパラメーターが作成されます。

- **ControllerCount**
- **ComputeCount**
- **BlockStorageCount**
- **ObjectStorageCount**
- **CephStorageCount**

レンダリング済みバージョンのパラメーターの例を以下に示します。

```
parameters:
  ...
  ControllerCount:
    description: Number of Controller nodes to deploy
    type: number
    default: 1
  ...
```

director がレンダリングするのは、コア heat テンプレートディレクトリー内からの Jinja2 タイプのテンプレートおよび環境ファイルだけです。Jinja2 テンプレートをレンダリングする際の正しい設定方法を、以下のユースケースで説明します。

### ユースケース 1: デフォルトのコアテンプレート

**テンプレートのディレクトリー: /usr/share/openstack-tripleo-heat-templates/**

**環境ファイル: /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.j2.yaml**

director はデフォルトのコアテンプレートの場所を使用し (**--templates**)、**network-isolation.j2.yaml** ファイルを **network-isolation.yaml** にレンダリングします。**openstack overcloud deploy** コマンドの実行時には、**-e** オプションを使用してレンダリングした **network-isolation.yaml** ファイルの名前を指定します。

```
$ openstack overcloud deploy --templates \
    -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml
    ...
```

### ユースケース 2: カスタムコアテンプレート

**テンプレートのディレクトリー: /home/stack/tripleo-heat-templates**

**環境ファイル: /home/stack/tripleo-heat-templates/environments/network-isolation.j2.yaml**

director はカスタムコアテンプレートの場所を使用し (**--templates /home/stack/tripleo-heat-templates**)、カスタムコアテンプレート内の **network-isolation.j2.yaml** ファイルを **network-isolation.yaml** にレンダリングします。**openstack overcloud deploy** コマンドの実行時には、**-e** オプションを使用してレンダリングした **network-isolation.yaml** ファイルの名前を指定します。

```
$ openstack overcloud deploy --templates /home/stack/tripleo-heat-templates \
    -e /home/stack/tripleo-heat-templates/environments/network-isolation.yaml
    ...
```

### ユースケース 3: 誤った設定

**テンプレートのディレクトリー: /usr/share/openstack-tripleo-heat-templates/**

**環境ファイル: /home/stack/tripleo-heat-templates/environments/network-isolation.j2.yaml**

director はカスタムコアテンプレートの場所を使用します (**--templates /home/stack/tripleo-heat-templates**)。しかし、選択した **network-isolation.j2.yaml** はカスタムコアテンプレート内に存在しないので、**network-isolation.yaml** にはレンダリングされません。この設定ではデプロイメントに失敗します。

### Jinja2 構文の静的テンプレートへの処理

**process-templates.py** スクリプトを使用して、**openstack-tripleo-heat-templates** の Jinja2 構文を静的テンプレートセットにレンダリングします。**process-templates.py** スクリプトで **openstack-tripleo-heat-templates** コレクションのコピーをレンダリングするには、**openstack-tripleo-heat-templates** ディレクトリーに移動します。

```
$ cd /usr/share/openstack-tripleo-heat-templates
```

静的コピーを保存するカスタムディレクトリーを定義する **-o** オプションを指定して、**tools** ディレクトリーにある **process-templates.py** スクリプトを実行します。

```
$ ./tools/process-templates.py -o ~/openstack-tripleo-heat-templates-rendered
```

これにより、すべての Jinja2 テンプレートがレンタリング済みの YAML バージョンに変換され、結果が **~/openstack-tripleo-heat-templates-rendered** に保存されます。

# 第3章 HEAT パラメーター

director テンプレートコレクション内の各 heat テンプレートには、**parameters** セクションがあります。このセクションには、特定のオーバークラウドサービス固有の全パラメーターの定義が含まれます。これには、以下のパラメーターが含まれます。

- **overcloud.j2.yaml**: デフォルトのベースパラメーター
- **roles_data.yaml**: コンポーザブルロールのデフォルトパラメーター
- **deployment/*.yaml**: 特定のサービスのデフォルトパラメーター

これらのパラメーターの値は、以下の方法で変更することができます。

1. カスタムパラメーター用の環境ファイルを作成します。
2. その環境ファイルの **parameter_defaults** セクションにカスタムのパラメーターを追加します。
3. **openstack overcloud deploy** コマンドでその環境ファイルを指定します。

## 3.1. 例 1: タイムゾーンの設定

タイムゾーンを設定するための heat テンプレート (**puppet/services/time/timezone.yaml**) には **TimeZone** パラメーターが含まれています。**TimeZone** パラメーターの値を空白のままにすると、オーバークラウドはデフォルトで時刻を **UTC** に設定します。

タイムゾーンの一覧を取得するには、**timedatectl list-timezones** コマンドを実行します。アジアのタイムゾーンを取得するコマンド例を以下に示します。

```
$ sudo timedatectl list-timezones|grep "Asia"
```

タイムゾーンを特定したら、環境ファイルの **TimeZone** パラメーターを設定します。以下に示す環境ファイルの例では、**TimeZone** の値を **Asia/Tokyo** に設定しています。

```
parameter_defaults:
  TimeZone: 'Asia/Tokyo'
```

## 3.2. 例 2: ネットワーク分散仮想ルーティング (DVR) の有効化

OpenStack Networking (neutron) API 用の heat テンプレート (**deployment/neutron/neutron-api-container-puppet.yaml**) には、分散仮想ルーティング (DVR) を有効化/無効化するためのパラメーターが含まれています。このパラメーターのデフォルト値は **false** です。有効にするには、環境ファイルの以下のエントリーを使用します。

```
parameter_defaults:
  NeutronEnableDVR: false
```

## 3.3. 例 3: RABBITMQ ファイル記述子の上限の設定

特定の設定では、RabbitMQ サーバーのファイル記述子の上限を高くする必要がある場合があります。**deployment/rabbitmq/rabbitmq-container-puppet.yaml** heat テンプレートを使用して、**RabbitFDLimit** パラメーターに新しい上限を設定します。環境ファイルに以下のエントリーを追加

します。

```
parameter_defaults:
  RabbitFDLimit: 65536
```

## 3.4. 例 4: パラメーターの有効化および無効化

デプロイメント時にパラメーターを初期設定し、それ以降のデプロイメント操作 (更新またはスケーリング操作など) ではそのパラメーターを無効にしなければならない場合があります。たとえば、オーバークラウドの作成時にカスタム RPM を含めるには、環境ファイルに以下のエントリーを追加します。

```
parameter_defaults:
  DeployArtifactURLs: ["http://www.example.com/myfile.rpm"]
```

それ以降のデプロイメントでこのパラメーターを無効にするには、パラメーターを削除するだけでは不十分です。削除するのではなく、パラメーターに空の値を設定する必要があります。

```
parameter_defaults:
  DeployArtifactURLs: []
```

これにより、それ以降のデプロイメント操作ではパラメーターは設定されなくなります。

## 3.5. 例 5: ロールベースのパラメーター

**[ROLE]Parameters** パラメーターを使用して特定のロールのパラメーターを設定します。ここで、**[ROLE]** はコンポーザブルロールに置き換えてください。

たとえば、director はコントローラーノードとコンピュートノードの両方に **sshd** を設定します。コントローラーノードとコンピュートノードに異なる **sshd** パラメーターを設定するには、**ControllerParameters** パラメーターと **ComputeParameters** パラメーターの両方が含まれる環境ファイルを作成し、特定のロールごとに **sshd** パラメーターを設定します。

```
parameter_defaults:
  ControllerParameters:
    BannerText: "This is a Controller node"
  ComputeParameters:
    BannerText: "This is a Compute node"
```

## 3.6. 変更するパラメーターの特定

Red Hat OpenStack Platform director は、設定用のパラメーターを多数提供しています。場合によっては、設定する特定のオプションとそれに対応する director のパラメーターを特定するのが困難なことがあります。director を使用してオプションを設定するには、以下のワークフローに従ってオプションを確認し、特定のオーバークラウドパラメーターにマッピングしてください。

1. 設定するオプションを特定します。そのオプションを使用するサービスを書き留めておきます。

2. このオプションに対応する Puppet モジュールを確認します。Red Hat OpenStack Platform 用の Puppet モジュールは director ノードの **/etc/puppet/modules** にあります。各モジュールは、特定のサービスに対応しています。たとえば、**keystone** モジュールは OpenStack Identity

(keystone) に対応しています。

- 選択したオプションを制御する変数が Puppet モジュールに含まれている場合には、次のステップに進んでください。
- 選択したオプションを制御する変数が Puppet モジュールに含まれていない場合には、そのオプションには hieradata は存在しません。可能な場合には、オーバークラウドがデプロイメントを完了した後でオプションを手動で設定することができます。

3. コア heat テンプレートコレクションに hieradata 形式の Puppet 変数が含まれているかどうかを確認します。**deployment/*** は通常、同じサービスの Puppet モジュールに対応します。たとえば、**deployment/keystone/keystone-container-puppet.yaml** テンプレートは、**keystone** モジュールの hieradata を提供します。
   - heat テンプレートが Puppet 変数用の hieradata を設定している場合には、そのテンプレートは変更することのできる director ベースのパラメーターも開示する必要があります。
   - heat テンプレートが Puppet 変数用の hieradata を設定していない場合には、環境ファイルを使用して、設定フックにより hieradata を渡します。hieradata のカスタマイズに関する詳しい情報は、「Puppet: ロール用 hieradata のカスタマイズ」 を参照してください。

### ワークフローの例

たとえば、OpenStack Identity (keystone) の通知の形式を変更するには、ワークフローを使用して、以下の手順を実施します。

1. 設定する OpenStack パラメーターを特定します (**notification_format**)。
2. **keystone** Puppet モジュールで **notification_format** の設定を検索します。

```
$ grep notification_format /etc/puppet/modules/keystone/manifests/*
```

   この場合は、**keystone** モジュールは **keystone::notification_format** の変数を使用してこのオプションを管理します。

3. **keystone** サービステンプレートでこの変数を検索します。

```
$ grep "keystone::notification_format" /usr/share/openstack-tripleo-heat-
templates/deployment/keystone/keystone-container-puppet.yaml
```

   このコマンドの出力には、director が **KeystoneNotificationFormat** パラメーターを使用して **keystone::notification_format** hieradata を設定していることが表示されます。

最終的なマッピングは、以下の表のとおりです。

| director のパラメーター | Puppet hieradata | OpenStack Identity (keystone) のオプション |
|---|---|---|
| **KeystoneNotificationFormat** | **keystone::notification_format** | **notification_format** |

オーバークラウドの環境ファイルで **KeystoneNotificationFormat** を設定すると、オーバークラウドの設定中に **keystone.conf** ファイルの **notification_format** オプションが設定されます。

# 第4章 設定フック

設定フックを使用して、オーバークラウドのデプロイメントプロセスに独自のカスタム設定関数を挿入します。メインのオーバークラウドサービスの設定前後にカスタム設定を挿入するためのフックや、Puppet ベースの設定を変更/追加するためのフックを作成することができます。

## 4.1. 初回ブート: 初回ブート設定のカスタマイズ

オーバークラウドの初回作成後に、director は **cloud-init** を使用して全ノードで設定を行います。**NodeUserData** リソース種別を使用して、**cloud-init** を呼び出すことができます。

**OS::TripleO::NodeUserData**

すべてのノードに適用する **cloud-init** 設定

**OS::TripleO::Controller::NodeUserData**

コントローラーノードに適用する **cloud-init** 設定

**OS::TripleO::Compute::NodeUserData**

コンピュートノードに適用する **cloud-init** 設定

**OS::TripleO::CephStorage::NodeUserData**

Ceph Storage ノードに適用する **cloud-init** 設定

**OS::TripleO::ObjectStorage::NodeUserData**

オブジェクトストレージノードに適用する **cloud-init** 設定

**OS::TripleO::BlockStorage::NodeUserData**

ブロックストレージノードに適用する **cloud-init** 設定

**OS::TripleO::[ROLE]::NodeUserData**

カスタムノードに適用する **cloud-init** 設定。**[ROLE]** をコンポーザブルロール名に置き換えてください。

以下の例では、全ノード上でカスタム IP アドレスを使用してネームサーバーを更新します。

1. 各ノードの **resolv.conf** ファイルに特定のネームサーバーを追加するためのスクリプトを実行する、基本的な heat テンプレート **~/templates/nameserver.yaml** を作成します。**OS::TripleO::MultipartMime** リソース種別を使用して、この設定スクリプトを送信することができます。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

resources:
  userdata:
    type: OS::Heat::MultipartMime
    properties:
      parts:
      - config: {get_resource: nameserver_config}

  nameserver_config:
    type: OS::Heat::SoftwareConfig
    properties:
      config: |
        #!/bin/bash
```

```
        echo "nameserver 192.168.1.1" >> /etc/resolv.conf

outputs:
  OS::stack_id:
    value: {get_resource: userdata}
```

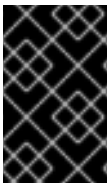
2. **OS::TripleO::NodeUserData** リソース種別として heat テンプレートを登録する環境ファイル **~/templates/firstboot.yaml** を作成します。

```
resource_registry:
  OS::TripleO::NodeUserData: /home/stack/templates/nameserver.yaml
```

3. オーバークラウドに初回ブートの設定を追加するには、その他の環境ファイルと共にこの環境ファイルをスタックに追加します。

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/firstboot.yaml \
    ...
```

これにより、ノード作成後の初回起動時に設定がすべてのノードに追加されます。これ以降は (たとえば、オーバークラウドスタックの更新時)、これらのテンプレートを追加してもこれらのスクリプトは実行されません。



**重要**

**NodeUserData** リソースを登録することができるのは、1 つのリソースにつき 1 つの heat テンプレートだけです。別の heat テンプレートに登録すると、使用する heat テンプレートがそのテンプレートに変わります。

## 4.2. 事前設定: 特定のオーバークラウドロールのカスタマイズ

**重要**

本書の以前のバージョンでは、**OS::TripleO::Tasks::*PreConfig** リソースを使用してロールごとに事前設定フックを提供していました。heat テンプレートコレクションでは、これらのフックを特定の用途に使用する必要があるので、これらを個別の用途に使用すべきではありません。その代わりに、以下に概要を示す **OS::TripleO::*ExtraConfigPre** フックを使用してください。

オーバークラウドは、OpenStack コンポーネントのコア設定に Puppet を使用します。director の提供するフックのセットを使用して、初回のブートが完了してコア設定が開始される前に、特定ノードロールのカスタム設定を行うことができます。これには、以下のフックが含まれます。

**OS::TripleO::ControllerExtraConfigPre**

Puppet のコア設定前にコントローラーノードに適用される追加の設定

**OS::TripleO::ComputeExtraConfigPre**

Puppet のコア設定前にコンピュートノードに適用される追加の設定

**OS::TripleO::CephStorageExtraConfigPre**

Puppet のコア設定前に Ceph Storage ノードに適用される追加の設定

**OS::TripleO::ObjectStorageExtraConfigPre**

Puppet のコア設定前にオブジェクトストレージノードに適用される追加の設定

**OS::TripleO::BlockStorageExtraConfigPre**

Puppet のコア設定前にブロックストレージノードに適用される追加の設定

**OS::TripleO::[ROLE]ExtraConfigPre**

Puppet のコア設定前にカスタムノードに適用される追加の設定。**[ROLE]** をコンポーザブルロール名に置き換えてください。

以下の例では、特定ロールの全ノードの **resolv.conf** ファイルに変数のネームサーバーを追加します。

1. ノードの **resolv.conf** ファイルに変数のネームサーバーを書き込むスクリプトを実行する、基本的な heat テンプレート **~/templates/nameserver.yaml** を作成します。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  server:
    type: string
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string

resources:
  CustomExtraConfigPre:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" > /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeploymentPre:
    type: OS::Heat::SoftwareDeployment
    properties:
      server: {get_param: server}
      config: {get_resource: CustomExtraConfigPre}
      actions: ['CREATE','UPDATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

outputs:
  deploy_stdout:
    description: Deployment reference, used to trigger pre-deploy on changes
    value: {get_attr: [CustomExtraDeploymentPre, deploy_stdout]}
```

上記の例では、**resources** セクションには以下のパラメーターが含まれます。

**CustomExtraConfigPre**

ここでは、ソフトウェア設定を定義します。上記の例では、Bash **スクリプト** を定義し、heat が **_NAMESERVER_IP_** を **nameserver_ip** パラメーターに保管された値に置き換えます。

CustomExtraDeploymentPre

この設定により、**CustomExtraConfigPre** リソースで定義したソフトウェア設定を実行します。以下の点に注意してください。

- 適用する設定を heat が認識するように、**config** パラメーターは **CustomExtraConfigPre** リソースを参照します。
- **server** パラメーターは、オーバークラウドノードのマッピングを取得します。これは親テンプレートにより提供されるパラメーターで、このフックのテンプレートには必須です。
- **actions** パラメーターは、設定を適用するタイミングを定義します。上記の例では、オーバークラウドが作成または更新された時に設定を適用します。設定可能なアクションは **CREATE**、**UPDATE**、**DELETE**、**SUSPEND**、および **RESUME** です。
- **input_values** では **deploy_identifier** というパラメーターを定義し、親テンプレートからの **DeployIdentifier** を格納します。このパラメーターにより、各デプロイメント更新のリソースにタイムスタンプが提供されます。これにより、これ以降のオーバークラウド更新時にリソースが再度適用されるようになります。

2. heat テンプレートをロールベースのリソース種別に登録する環境ファイル **~/templates/pre_config.yaml** を作成します。たとえば、コントローラーノードだけに設定を適用するには、**ControllerExtraConfigPre** フックを使用します。

```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

3. その他の環境ファイルと共に環境ファイルをスタックに追加します。

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/pre_config.yaml \
    ...
```

これにより、オーバークラウドの初回作成またはそれ以降の更新において、コア設定前にすべてのコントローラーノードに設定が適用されます。

**重要**

各リソースを登録することができるのは、1 つのフックにつき 1 つの heat テンプレートだけです。別の heat テンプレートに登録すると、使用する heat テンプレートがそのテンプレートに変わります。

## 4.3. 事前設定: 全オーバークラウドロールのカスタマイズ

オーバークラウドは、OpenStack コンポーネントのコア設定に Puppet を使用します。director の提供するフックを使用して、初回のブートが完了してコア設定が開始される前に、すべてのノード種別を設定することができます。

**OS::TripleO::NodeExtraConfig**

Puppet のコア設定前に全ノードロールに適用される追加の設定

以下の例では、各ノードの **resolv.conf** ファイルに変数のネームサーバーを追加します。

1. 各ノードの **resolv.conf** ファイルに変数のネームサーバーを追加するためのスクリプトを実行する、基本的な heat テンプレート **~/templates/nameserver.yaml** を作成します。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  server:
    type: string
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string

resources:
  CustomExtraConfigPre:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeploymentPre:
    type: OS::Heat::SoftwareDeployment
    properties:
      server: {get_param: server}
      config: {get_resource: CustomExtraConfigPre}
      actions: ['CREATE','UPDATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

outputs:
  deploy_stdout:
    description: Deployment reference, used to trigger pre-deploy on changes
    value: {get_attr: [CustomExtraDeploymentPre, deploy_stdout]}
```

上記の例では、**resources** セクションには以下のパラメーターが含まれます。

**CustomExtraConfigPre**

このパラメーターは、ソフトウェア設定を定義します。上記の例では、Bash **スクリプト** を定義し、heat が **_NAMESERVER_IP_** を **nameserver_ip** パラメーターに保管された値に置き換えます。

CustomExtraDeploymentPre

このパラメーターにより、**CustomExtraConfigPre** リソースで定義したソフトウェア設定を実行します。以下の点に注意してください。

- 適用する設定を heat が認識するように、**config** パラメーターは **CustomExtraConfigPre** リソースを参照します。
- **server** パラメーターは、オーバークラウドノードのマッピングを取得します。これは親テンプレートにより提供されるパラメーターで、このフックのテンプレートには必須です。
- **actions** パラメーターは、設定を適用するタイミングを定義します。上記の例では、オーバークラウドが作成または更新された時にのみ設定を適用します。設定可能なアクションは **CREATE**、**UPDATE**、**DELETE**、**SUSPEND**、および **RESUME** です。
- **input_values** パラメーターでは **deploy_identifier** というサブパラメーターを定義し、親テンプレートからの **DeployIdentifier** を格納します。このパラメーターにより、各デプロイメント更新のリソースにタイムスタンプが提供されます。これにより、これ以降のオーバークラウド更新時にリソースが再度適用されるようになります。

2. **OS::TripleO::NodeExtraConfig** リソース種別として heat テンプレートを登録する環境ファイル **~/templates/pre_config.yaml** を作成します。

```
resource_registry:
  OS::TripleO::NodeExtraConfig: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

3. その他の環境ファイルと共に環境ファイルをスタックに追加します。

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/pre_config.yaml \
    ...
```

これにより、オーバークラウドの初回作成またはそれ以降の更新において、コア設定前にすべてのノードに設定が適用されます。

**重要**

**OS::TripleO::NodeExtraConfig** を登録することができるのは 1 つの heat テンプレートだけです。別の heat テンプレートに登録すると、使用する heat テンプレートがそのテンプレートに変わります。

## 4.4. 設定後: 全オーバークラウドロールのカスタマイズ

**重要**

本書の以前のバージョンでは、**OS::TripleO::Tasks::*PostConfig** リソースを使用してロールごとに設定後フックを提供していました。heat テンプレートコレクションでは、これらのフックを特定の用途に使用する必要があるので、これらを個別の用途に使用すべきではありません。その代わりに、以下に概要を示す **OS::TripleO::NodeExtraConfigPost** フックを使用してください。

オーバークラウドの初回作成時または更新時において、オーバークラウドの作成が完了してからすべてのロールに設定の追加が必要となる可能性があります。このような場合には、以下の設定後フックを使用します。

**OS::TripleO::NodeExtraConfigPost**

Puppet のコア設定後に全ノードロールに適用される追加の設定

以下の例では、各ノードの **resolv.conf** ファイルに変数のネームサーバーを追加します。

1. 各ノードの **resolv.conf** ファイルに変数のネームサーバーを追加するためのスクリプトを実行する、基本的な heat テンプレート **~/templates/nameserver.yaml** を作成します。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  servers:
    type: json
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string
  EndpointMap:
    default: {}
    type: json

resources:
  CustomExtraConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeployments:
    type: OS::Heat::SoftwareDeploymentGroup
    properties:
      servers:  {get_param: servers}
      config: {get_resource: CustomExtraConfig}
```

```
    actions: ['CREATE','UPDATE']
    input_values:
      deploy_identifier: {get_param: DeployIdentifier}
```

上記の例では、**resources** セクションには以下のパラメーターが含まれます。

CustomExtraConfig

ここでは、ソフトウェア設定を定義します。上記の例では、Bash **スクリプト** を定義し、heat が **_NAMESERVER_IP_** を **nameserver_ip** パラメーターに保管された値に置き換えます。

CustomExtraDeployments

この設定により、**CustomExtraConfig** リソースで定義したソフトウェア設定を実行します。以下の点に注意してください。

- 適用する設定を heat が認識するように、**config** パラメーターは **CustomExtraConfig** リソースを参照します。
- **servers** パラメーターは、オーバークラウドノードのマッピングを取得します。これは親テンプレートにより提供されるパラメーターで、このフックのテンプレートには必須です。
- **actions** パラメーターは、設定を適用するタイミングを定義します。上記の例では、オーバークラウドが作成または更新された時に設定を適用します。設定可能なアクションは **CREATE**、**UPDATE**、**DELETE**、**SUSPEND**、および **RESUME** です。
- **input_values** では **deploy_identifier** というパラメーターを定義し、親テンプレートからの **DeployIdentifier** を格納します。このパラメーターにより、各デプロイメント更新のリソースにタイムスタンプが提供されます。これにより、これ以降のオーバークラウド更新時にリソースが再度適用されるようになります。

2. **OS::TripleO::NodeExtraConfigPost:** リソース種別として heat テンプレートを登録する環境ファイル **~/templates/post_config.yaml** を作成します。

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

3. その他の環境ファイルと共に環境ファイルをスタックに追加します。

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/post_config.yaml \
    ...
```

これにより、オーバークラウドの初回作成またはそれ以降の更新において、コア設定の完了後にすべてのノードに設定が適用されます。

**重要**

**OS::TripleO::NodeExtraConfigPost** を登録することができるのは 1 つの heat テンプレートだけです。別の heat テンプレートに登録すると、使用する heat テンプレートがそのテンプレートに変わります。

## 4.5. PUPPET: ロール用 HIERADATA のカスタマイズ

heat テンプレートコレクションにはパラメーターのセットが含まれ、これを使用して特定のノード種別に追加の設定を渡すことができます。これらのパラメーターでは、ノードの Puppet 設定用 hieradata として設定を保存します。

**ControllerExtraConfig**

すべてのコントローラーノードに追加する設定

**ComputeExtraConfig**

すべてのコンピュートノードに追加する設定

**BlockStorageExtraConfig**

すべてのブロックストレージノードに追加する設定

**ObjectStorageExtraConfig**

すべてのオブジェクトストレージノードに追加する設定

**CephStorageExtraConfig**

すべての Ceph Storage ノードに追加する設定

**[ROLE]ExtraConfig**

コンポーザブルロールに追加する設定。**[ROLE]** をコンポーザブルロール名に置き換えてください。

**ExtraConfig**

すべてのノードに追加する設定

1. デプロイ後の設定プロセスに設定を追加するには、**parameter_defaults** セクションにこれらのパラメーターが記載された環境ファイルを作成します。たとえば、コンピュートホストに確保するメモリーを 1024 MB に増やし VNC キーマップを日本語に指定するには、**ComputeExtraConfig** パラメーターの以下のエントリーを使用します。

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::reserved_host_memory: 1024
    nova::compute::vnc_keymap: ja
```

2. ご自分のデプロイメントに該当するその他の環境ファイルと共に、この環境ファイルを **openstack overcloud deploy** コマンドに追加します。

**重要**

それぞれのパラメーターを定義することができるのは 1 度だけです。さらに定義すると、以前の値が上書きされます。

## 4.6. PUPPET: 個別のノードの HIERADATA のカスタマイズ

heat テンプレートコレクションを使用して、個別のノードの Puppet hieradata を設定することができます。

1. ノードのイントロスペクションデータからシステム UUID を特定します。

```
$ openstack baremetal introspection data save 9dcc87ae-4c6d-4ede-81a5-9b20d7dc4a14 |
jq .extra.system.product.uuid
```

このコマンドは、システム UUID を返します。以下に例を示します。

```
"f5055c6c-477f-47fb-afe5-95c6928c407f"
```

2. ノード固有の hieradata を定義して **per_node.yaml** テンプレートを事前設定フックに登録する環境ファイルを作成します。**NodeDataLookup** パラメーターに、設定するノードのシステム UUID を指定します。

```
resource_registry:
  OS::TripleO::ComputeExtraConfigPre: /usr/share/openstack-tripleo-heat-
templates/puppet/extraconfig/pre_deploy/per_node.yaml
parameter_defaults:
  NodeDataLookup: '{"f5055c6c-477f-47fb-afe5-95c6928c407f":
{"nova::compute::vcpu_pin_set": [ "2", "3" ]}}'
```

3. ご自分のデプロイメントに該当するその他の環境ファイルと共に、この環境ファイルを **openstack overcloud deploy** コマンドに追加します。

**per_node.yaml** テンプレートは、各システム UUID に対応するノード上に、定義する hieradata が含まれる hieradata ファイルのセットを生成します。UUID が定義されていない場合には、生成される hieradata ファイルは空になります。上記の例では、**per_node.yaml** テンプレートは **OS::TripleO::ComputeExtraConfigPre** フックで定義されたすべてのコンピュートノード上で実行されますが、システム UUID が **f5055c6c-477f-47fb-afe5-95c6928c407f** のコンピュートノードのみが hieradata を受け取ります。

このメカニズムを使用して、特定の要件に応じて各ノードを個別に調整することができます。

**NodeDataLookup** についての詳しい情報は、**『コンテナー化された Red Hat Ceph を持つオーバークラウドのデプロイ』**の「異なる構成の Ceph Storage ノードへのディスクレイアウトのマッピング」を参照してください。

## 4.7. PUPPET: カスタムのマニフェストの適用

特定の状況では、追加のコンポーネントをオーバークラウドノードにインストールして設定する場合があります。そのためには、カスタムの Puppet マニフェストを使用して、主要な設定が完了してからノードに適用します。基本的な例として、各ノードに **motd** をインストールするケースを考えます。

1. Puppet 設定を起動する heat テンプレート **~/templates/custom_puppet_config.yaml** を作成します。

```
heat_template_version: 2014-10-16

description: >
  Run Puppet extra configuration to set new MOTD

parameters:
  servers:
    type: json

resources:
  ExtraPuppetConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      config: {get_file: motd.pp}
      group: puppet
```

```
    options:
      enable_hiera: True
      enable_facter: False

  ExtraPuppetDeployments:
    type: OS::Heat::SoftwareDeploymentGroup
    properties:
      config: {get_resource: ExtraPuppetConfig}
      servers: {get_param: servers}
```

この例では、テンプレート内に **/home/stack/templates/motd.pp** を追加し、設定するノードに渡します。**motd.pp** ファイルには、**motd** のインストールと設定に必要な Puppet クラスが含まれています。

2. **OS::TripleO::NodeExtraConfigPost:** リソース種別として heat テンプレートを登録する環境ファイル **~templates/puppet_post_config.yaml** を作成します。

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/custom_puppet_config.yaml
```

3. ご自分のデプロイメントに該当するその他の環境ファイルと共に、この環境ファイルを **openstack overcloud deploy** コマンドに追加します。

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/puppet_post_config.yaml \
    ...
```

これにより、**motd.pp** からの設定がオーバークラウド内の全ノードに適用されます。

# 第5章 ANSIBLE ベースのオーバークラウド登録

director は、Ansible ベースのメソッドを使用して、オーバークラウドノードを Red Hat カスタマーポータルまたは Red Hat Satellite Server に登録します。

以前のバージョンの Red Hat OpenStack Platform の **rhel-registration** メソッドを使用していた場合には、それを無効にして Ansible ベースのメソッドに切り替える必要があります。詳しい情報は、「rhsm コンポーザブルサービスへの切り替え」および「rhel-registration から rhsm へのマッピング」を参照してください。

director ベースの登録メソッドに加えて、デプロイメント後に手動で登録することもできます。詳細は、「手動による Ansible ベースの登録の実行」を参照してください。

## 5.1. RED HAT SUBSCRIPTION MANAGER (RHSM) コンポーザブルサービス

**rhsm** コンポーザブルサービスを使用して、Ansible を介してオーバークラウドノードを登録することができます。デフォルトの **roles_data** ファイルの各ロールには、**OS::TripleO::Services::Rhsm** リソースが含まれており、これはデフォルトで無効になっています。サービスを有効にするには、このリソースを **rhsm** コンポーザブルサービスのファイルに登録します。

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
```

**rhsm** コンポーザブルサービスは **RhsmVars** パラメーターを受け入れます。これを使用して、登録に必要な複数のサブパラメーターを定義することができます。

```
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_release: 8.2
```

**RhsmVars** パラメーターをロール固有のパラメーター (例: **ControllerParameters**) と共に使用することにより、異なるノードタイプ用の特定のリポジトリーを有効化する場合に柔軟性を提供することもできます。

## 5.2. RHSMVARS サブパラメーター

**rhsm** コンポーザブルサービスを設定する際に、以下のサブパラメーターを **RhsmVars** パラメーターの一部として使用します。利用可能な Ansible パラメーターについての詳細は、ロールに関するドキュメント を参照してください。

| rhsm | 説明 |
|---|---|
| **rhsm_method** | 登録の方法を選択します。**portal**、**satellite**、または **disable** のいずれかです。 |
| **rhsm_org_id** | 登録に使用する組織。この ID を特定するには、アンダークラウドノードから **sudo subscription-manager orgs** を実行します。プロンプトが表示されたら Red Hat の認証情報を入力して、出力される **Key** の値を使用します。組織の ID についての詳細は、「Red Hat Subscription Management における組織 ID について理解する」を参照してください。 |
| **rhsm_pool_ids** | 使用するサブスクリプションプール ID。サブスクリプションを自動でアタッチしない場合は、このパラメーターを使用します。この ID を特定するには、アンダークラウドノードから **sudo subscription-manager list --available --all --matches="*Red Hat OpenStack*"** を実行し、出力される **Pool ID** の値を使用します。 |
| **rhsm_activation_key** | 登録に使用するアクティベーションキー |
| **rhsm_autosubscribe** | このパラメーターを使用して、互換性のあるサブスクリプションを自動的にこのシステムにアタッチします。この機能を有効にするには、値を **true** に設定します。 |
| **rhsm_baseurl** | コンテンツを取得するためのベース URL。デフォルトの URL は Red Hat コンテンツ配信ネットワークです。Satellite サーバーを使用している場合は、この値を Satellite サーバーコンテンツリポジトリーのベース URL に変更します。 |
| **rhsm_server_hostname** | 登録用のサブスクリプション管理サービスのホスト名。デフォルトは Red Hat Subscription Management のホスト名です。Satellite サーバーを使用している場合は、この値を Satellite サーバーのホスト名に変更します。 |
| **rhsm_repos** | 有効にするリポジトリーの一覧 |
| **rhsm_username** | 登録用のユーザー名。可能な場合には、登録にアクティベーションキーを使用します。 |
| **rhsm_password** | 登録用のパスワード。可能な場合には、登録にアクティベーションキーを使用します。 |
| **rhsm_release** | リポジトリー固定用の Red Hat Enterprise Linux リリース。Red Hat OpenStack Platform の場合、このパラメーターは 8.2 に設定されます。 |
| **rhsm_rhsm_proxy_hostname** | HTTP プロキシーのホスト名 (例: **proxy.example.com**) |
| **rhsm_rhsm_proxy_port** | HTTP プロキシー通信用のポート (例: **8080**) |

| rhsm | 説明 |
|---|---|
| **rhsm_rhsm_proxy_user** | HTTP プロキシーにアクセスするためのユーザー名 |
| **rhsm_rhsm_proxy_password** | HTTP プロキシーにアクセスするためのパスワード |

**重要**

**rhsm_method** が **portal** に設定されている場合に限り、**rhsm_activation_key** と **rhsm_repos** を同時に使用することができます。**rhsm_method** が **satellite** に設定されている場合は、**rhsm_activation_key** または **rhsm_repos** のどちらか一方しか使用することができません。

## 5.3. RHSM コンポーザブルサービスを使用したオーバークラウドの登録

**rhsm** コンポーザブルサービスを有効にして設定する環境ファイルを作成します。director はこの環境ファイルを使用して、ノードを登録し、サブスクライブします。

**手順**

1. 設定を保存するための環境ファイルを **templates/rhsm.yml** という名前で作成します。
2. 環境ファイルに設定を追加します。以下に例を示します。

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
    rhsm_method: "portal"
    rhsm_release: 8.2
```

   - **resource_registry** セクションは、各ロールで利用可能な **OS::TripleO::Services::Rhsm** リソースに **rhsm** コンポーザブルサービスを関連付けます。
   - **RhsmVars** の変数は、Red Hat の登録を設定するためにパラメーターを Ansible に渡します。

3. 環境ファイルを保存します。

## 5.4. 異なるロールに対する RHSM コンポーザブルサービスの適用

**rhsm** コンポーザブルサービスをロールごとに適用することができます。たとえば、コントローラーノード、コンピュートノード、および Ceph Storage ノードに、異なる設定セットを適用することができます。

**手順**

1. 設定を保存するための環境ファイルを **templates/rhsm.yml** という名前で作成します。
2. 環境ファイルに設定を追加します。以下に例を示します。

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  ControllerParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-8-for-x86_64-baseos-eus-rpms
        - rhel-8-for-x86_64-appstream-eus-rpms
        - rhel-8-for-x86_64-highavailability-eus-rpms
        - ansible-2.9-for-rhel-8-x86_64-rpms
        - advanced-virt-for-rhel-8-x86_64-rpms
        - openstack-16.1-for-rhel-8-x86_64-rpms
        - rhceph-4-mon-for-rhel-8-x86_64-rpms
        - rhceph-4-tools-for-rhel-8-x86_64-rpms
        - fast-datapath-for-rhel-8-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 8.2
  ComputeParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-8-for-x86_64-baseos-eus-rpms
        - rhel-8-for-x86_64-appstream-eus-rpms
        - rhel-8-for-x86_64-highavailability-eus-rpms
        - ansible-2.9-for-rhel-8-x86_64-rpms
        - advanced-virt-for-rhel-8-x86_64-rpms
        - openstack-16.1-for-rhel-8-x86_64-rpms
        - rhceph-4-tools-for-rhel-8-x86_64-rpms
        - fast-datapath-for-rhel-8-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 8.2
  CephStorageParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-8-for-x86_64-baseos-rpms
        - rhel-8-for-x86_64-appstream-rpms
        - rhel-8-for-x86_64-highavailability-rpms
        - ansible-2.9-for-rhel-8-x86_64-rpms
```

```
    - openstack-16.1-deployment-tools-for-rhel-8-x86_64-rpms
    - rhceph-4-osd-for-rhel-8-x86_64-rpms
    - rhceph-4-tools-for-rhel-8-x86_64-rpms
  rhsm_username: "myusername"
  rhsm_password: "p@55w0rd!"
  rhsm_org_id: "1234567"
  rhsm_pool_ids: "68790a7aa2dc9dc50a9bc39fabc55e0d"
  rhsm_method: "portal"
  rhsm_release: 8.2
```

**resource_registry** は、各ロールで利用可能な **OS::TripleO::Services::Rhsm** リソースに **rhsm** コンポーザブルサービスを関連付けます。

**ControllerParameters**、**ComputeParameters**、および **CephStorageParameters** パラメーターは、それぞれ個別の **RhsmVars** パラメーターを使用して対応するロールにサブスクリプション情報を渡します。

**注記**

Red Hat Ceph Storage のサブスクリプションおよび Ceph Storage 固有のリポジトリーを使用するように、**CephStorageParameters** パラメーター内の **RhsmVars** パラメーターを設定します。**rhsm_repos** パラメーターに、コントローラーノードおよびコンピュートノードに必要な Extended Update Support (EUS) リポジトリーではなく、標準の Red Hat Enterprise Linux リポジトリーが含まれるようにします。

3. 環境ファイルを保存します。

## 5.5. RED HAT SATELLITE SERVER へのオーバークラウドの登録

ノードを Red Hat カスタマーポータルではなく Red Hat Satellite に登録するには、**rhsm** コンポーザブルサービスを有効にして設定する環境ファイルを作成します。

**手順**

1. 設定を保存するための環境ファイルを **templates/rhsm.yml** という名前で作成します。
2. 環境ファイルに設定を追加します。以下に例を示します。

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  RhsmVars:
    rhsm_activation_key: "myactivationkey"
    rhsm_method: "satellite"
    rhsm_org_id: "ACME"
    rhsm_server_hostname: "satellite.example.com"
    rhsm_baseurl: "https://satellite.example.com/pulp/repos"
    rhsm_release: 8.2
```

**resource_registry** は、各ロールで利用可能な **OS::TripleO::Services::Rhsm** リソースに **rhsm** コンポーザブルサービスを関連付けます。

**RhsmVars** の変数は、Red Hat の登録を設定するためにパラメーターを Ansible に渡します。

3. 環境ファイルを保存します。

## 5.6. RHSM コンポーザブルサービスへの切り替え

従来の **rhel-registration** メソッドは、bash スクリプトを実行してオーバークラウドの登録を処理します。この方法のスクリプトと環境ファイルは、**/usr/share/openstack-tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/** のコア heat テンプレートコレクションにあります。

**rhel-registration** メソッドを **rhsm** コンポーザブルサービスに切り替えるには、以下の手順を実施します。

**手順**

1. **rhel-registration** 環境ファイルは、今後のデプロイメント操作から除外します。通常は、以下のファイルを除外します。
   - **rhel-registration/environment-rhel-registration.yaml**
   - **rhel-registration/rhel-registration-resource-registry.yaml**
2. カスタムの **roles_data** ファイルを使用する場合には、**roles_data** ファイルの各ロールに必ず **OS::TripleO::Services::Rhsm** コンポーザブルサービスを含めてください。以下に例を示します。

```
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  ...
  ServicesDefault:
    ...
    - OS::TripleO::Services::Rhsm
    ...
```

3. **rhsm** コンポーザブルサービスのパラメーター用の環境ファイルを今後のデプロイメント操作に追加します。

このメソッドは、**rhel-registration** パラメーターを **rhsm** サービスのパラメーターに置き換えて、サービスを有効にする heat リソースを変更します。具体的には、

```
resource_registry:
  OS::TripleO::NodeExtraConfig: rhel-registration.yaml
```

上記の行を以下のように変更します。

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
```

デプロイメントに **/usr/share/openstack-tripleo-heat-templates/environments/rhsm.yaml** 環境ファイルを追加して、サービスを有効にすることもできます。

## 5.7. RHEL-REGISTRATION から RHSM へのマッピング

**rhel-registration** の方法から **rhsm** の方法への情報の移行を容易に行うには、以下の表を使用してパラメーターと値をマッピングします。

| rhel-registration | rhsm / RhsmVars |
|---|---|
| **rhel_reg_method** | **rhsm_method** |
| **rhel_reg_org** | **rhsm_org_id** |
| **rhel_reg_pool_id** | **rhsm_pool_ids** |
| **rhel_reg_activation_key** | **rhsm_activation_key** |
| **rhel_reg_auto_attach** | **rhsm_autosubscribe** |
| **rhel_reg_sat_url** | **rhsm_satellite_url** |
| **rhel_reg_repos** | **rhsm_repos** |
| **rhel_reg_user** | **rhsm_username** |
| **rhel_reg_password** | **rhsm_password** |
| **rhel_reg_release** | **rhsm_release** |
| **rhel_reg_http_proxy_host** | **rhsm_rhsm_proxy_hostname** |
| **rhel_reg_http_proxy_port** | **rhsm_rhsm_proxy_port** |
| **rhel_reg_http_proxy_username** | **rhsm_rhsm_proxy_user** |
| **rhel_reg_http_proxy_password** | **rhsm_rhsm_proxy_password** |

## 5.8. RHSM コンポーザブルサービスを使用したオーバークラウドのデプロイ

Ansible がオーバークラウドノードの登録プロセスを制御するように、**rhsm** コンポーザブルサービスを使用してオーバークラウドをデプロイします。

**手順**

1. **openstack overcloud deploy** コマンドで **rhsm.yml** 環境ファイルを指定します。

```
openstack overcloud deploy \
  <other cli args> \
  -e ~/templates/rhsm.yaml
```

これにより、Ansible のオーバークラウドの設定と、Ansible ベースの登録が有効化されます。

2. オーバークラウドのデプロイメントが完了するまで待ちます。

3. オーバークラウドノードのサブスクリプション情報を確認します。たとえば、コントローラーノードにログインして、以下のコマンドを実行します。

```
$ sudo subscription-manager status
$ sudo subscription-manager list --consumed
```

## 5.9. 手動による ANSIBLE ベースの登録の実行

director ノードで動的インベントリースクリプトを使用して、デプロイしたオーバークラウドで、手動による Ansible ベースの登録を行うことができます。このスクリプトを使用して、ホストグループとしてノードロールを定義します。続いて **ansible-playbook** を使用して定義したノードロールに対して Playbook を実行します。コントローラーノードを手動で登録するには、以下の例の Playbook を使用します。

**手順**

1. ノードの登録に **redhat_subscription** モジュールを使用する Playbook を作成します。たとえば、以下の Playbook はコントローラーノードに適用されます。

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      - ansible-2.9-for-rhel-8-x86_64-rpms
      - advanced-virt-for-rhel-8-x86_64-rpms
      - openstack-16.1-for-rhel-8-x86_64-rpms
      - rhceph-4-mon-for-rhel-8-x86_64-rpms
      - fast-datapath-for-rhel-8-x86_64-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
        org_id: 1234567
        release: 8.2
        pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
    - name: Disable all repos
      command: "subscription-manager repos --disable *"
    - name: Enable Controller node repos
      command: "subscription-manager repos --enable {{ item }}"
      with_items: "{{ repos }}"
```

- このプレイには 3 つのタスクが含まれます。
  - ノードを登録する。

- 自動的に有効化されるリポジトリーをすべて無効にする。
- コントローラーノードに関連するリポジトリーだけを有効にする。リポジトリーは **repos** 変数でリストされます。

2. オーバークラウドのデプロイ後に、Ansible がオーバークラウドに対して Playbook (**ansible-osp-registration.yml**) を実行するように以下のコマンドを実行することができます。

```
$ ansible-playbook -i /usr/bin/tripleo-ansible-inventory ansible-osp-registration.yml
```

このコマンドにより、以下のアクションが行われます。

- 動的インベントリースクリプトを実行し、ホストとそのグループの一覧を取得する。
- Playbook の **hosts** パラメーターで定義されているグループ (この場合はコントローラーグループ) 内のノードに、その Playbook のタスクを適用する。

# 第6章 コンポーザブルサービスとカスタムロール

オーバークラウドは、通常コントローラーノードやコンピュートノードなどの事前定義済みロールのノードと、異なる種別のストレージノードで構成されます。これらのデフォルトの各ロールには、director ノード上にあるコアの heat テンプレートコレクションで定義されているサービスセットが含まれます。ただし、特定のサービスのセットが含まれるカスタムロールを作成することもできます。

この柔軟性により、異なるロール上に異なるサービスの組み合わせを作成することができます。本章では、カスタムロールのアーキテクチャー、コンポーザブルサービス、およびそれらを使用する方法について説明します。

## 6.1. サポートされるロールアーキテクチャー

カスタムロールとコンポーザブルサービスを使用する場合には、以下のアーキテクチャーを使用することができます。

**デフォルトアーキテクチャー**

デフォルトの **roles_data** ファイルを使用します。すべての Controller サービスが単一の Controller ロールに含まれます。

**サポートされるスタンドアロンのロール**

**/usr/share/openstack-tripleo-heat-templates/roles** 内の事前定義済みファイルを使用して、カスタムの **roles_data** ファイルを生成します。詳細な情報は、「サポートされるカスタムロール」を参照してください。

**カスタムコンポーザブルサービス**

専用のロールを作成し、それらを使用してカスタムの **roles_data** ファイルを生成します。限られたコンポーザブルサービスの組み合わせしかテスト/検証されていない点に注意してください。Red Hat では、すべてのコンポーザブルサービスの組み合わせに対してサポートを提供することはできません。

## 6.2. ロール

### 6.2.1. roles_data ファイルの検証

**roles_data** ファイルには、director がノードにデプロイする YAML 形式のロール一覧が含まれます。それぞれのロールには、そのロールを構成するすべてのサービスの定義が含まれます。以下のスニペット例を使用して、**roles_data** の構文を説明します。

```
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
- name: Compute
  description: |
    Basic Compute Node role
  ServicesDefault:
    - OS::TripleO::Services::AuditD
```

```
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CephClient
  ...
```

コア heat テンプレートコレクションには、デフォルトの **roles_data** ファイル (**/usr/share/openstack-tripleo-heat-templates/roles_data.yaml**) が含まれています。デフォルトのファイルには、以下のロール種別の定義が含まれます。

- **Controller**
- **Compute**
- **BlockStorage**
- **ObjectStorage**
- **CephStorage**

デプロイメント時、**openstack overcloud deploy** コマンドには、デフォルトの **roles_data.yaml** ファイルが含まれます。ただし、**-r** 引数を使用して、このファイルをカスタムの **roles_data** ファイルでオーバーライドすることができます。

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-custom.yaml
```

### 6.2.2. roles_data ファイルの作成

カスタムの **roles_data** ファイルは、手動で作成することができますが、個別のロールテンプレートを使用して自動生成することも可能です。director は、ロールテンプレートの管理とカスタムの **roles_data** ファイルの自動生成を行うためのコマンドをいくつか提供しています。

1. デフォルトロールのテンプレートを一覧表示します。

```
$ openstack overcloud roles list
BlockStorage
CephStorage
Compute
ComputeHCI
ComputeOvsDpdk
Controller
...
```

2. **openstack overcloud roles show** コマンドを使用して、YAML 形式のロール定義を表示します。

```
$ openstack overcloud roles show Compute
```

3. カスタムの **roles_data** ファイルを生成します。**openstack overcloud roles generate** コマンドを使用して、複数の事前定義済みロールを単一のファイルに統合します。たとえば、**Controller**、**Compute**、および **Networker** ロールが含まれる **roles_data.yaml** ファイルを生成するには、以下のコマンドを実行します。

```
$ openstack overcloud roles generate -o ~/roles_data.yaml Controller Compute Networker
```

**-o** オプションを使用して、出力ファイルの名前を定義します。

このコマンドにより、カスタムの **roles_data** ファイルが作成されます。ただし、上記の例では、**Controller** と **Networker** ロールを使用しており、その両方に同じネットワークエージェントが含まれています。つまり、ネットワークサービスが **Controller** ロールから **Networker** ロールにスケーリングされ、オーバークラウドは **コントローラー** ノードと **ネットワーカー** ノード間でネットワークサービスの負荷のバランスを取ります。

必要なその他のロールと共に独自のカスタム **Controller** ロールを作成して、この **Networker** ロールをスタンドアロンにすることができます。これにより、独自のカスタムロールから **roles_data** ファイルを生成できるようになります。

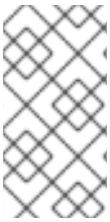
4. コア heat テンプレートコレクションから **stack** ユーザーのホームディレクトリーにディレクトリーをコピーします。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles ~/.
```

5. このディレクトリー内でカスタムロールファイルを追加または変更します。このディレクトリーをカスタムロールのソースとして使用するには、ロールのサブコマンドに **--roles-path** オプションを指定します。

```
$ openstack overcloud roles generate -o my_roles_data.yaml \
  --roles-path ~/roles \
  Controller Compute Networker
```

このコマンドにより、**~/roles** ディレクトリー内の個々のロールから、単一の **my_roles_data.yaml** ファイルが生成されます。



**注記**

デフォルトのロールコレクションには、**ControllerOpenStack** ロールも含まれます。このロールには、**Networker**、**Messaging**、および **Database** ロールのサービスは含まれません。**ControllerOpenStack** は、スタンドアロンの **Networker**、**Messaging**、および **Database** ロールと組み合わせて使用することができます。

### 6.2.3. サポートされるカスタムロール

以下の表で、利用可能なカスタムロールについて説明します。カスタムロールテンプレートは、**/usr/share/openstack-tripleo-heat-templates/roles** ディレクトリーにあります。

| ロール | 説明 | ファイル |
|---|---|---|
| **BlockStorage** | OpenStack Block Storage (cinder) ノード | **BlockStorage.yaml** |
| **CephAll** | 完全なスタンドアロンの Ceph Storage ノード。OSD、MON、Object Gateway (RGW)、Object Operations (MDS)、Manager (MGR)、および RBD Mirroring が含まれます。 | **CephAll.yaml** |
| **CephFile** | スタンドアロンのスケールアウト Ceph Storage ファイルロール。OSD および Object Operations (MDS) が含まれます。 | **CephFile.yaml** |

| ロール | 説明 | ファイル |
|---|---|---|
| **CephObject** | スタンドアロンのスケールアウト Ceph Storage オブジェクトロール。OSD および Object Gateway (RGW) が含まれます。 | **CephObject.yaml** |
| **CephStorage** | Ceph Storage OSD ノードロール | **CephStorage.yaml** |
| **ComputeAlt** | 代替のコンピュートノードロール | **ComputeAlt.yaml** |
| **ComputeDVR** | DVR 対応のコンピュートノードロール | **ComputeDVR.yaml** |
| **ComputeHCI** | ハイパーコンバージドインフラストラクチャーを持つコンピュートノード。Compute および Ceph OSD サービスが含まれます。 | **ComputeHCI.yaml** |
| **ComputeInstanceHA** | コンピュートインスタンス HA ノードロール。**environments/compute-instanceha.yaml** 環境ファイルと共に使用します。 | **ComputeInstanceHA.yaml** |
| **ComputeLiquidio** | Cavium Liquidio Smart NIC を持つコンピュートノード | **ComputeLiquidio.yaml** |
| **ComputeOvsDpdkRT** | コンピュート OVS DPDK RealTime ロール | **ComputeOvsDpdkRT.yaml** |
| **ComputeOvsDpdk** | コンピュート OVS DPDK ロール | **ComputeOvsDpdk.yaml** |
| **ComputePPC64LE** | ppc64le サーバー用 Compute ロール | **ComputePPC64LE.yaml** |
| **ComputeRealTime** | リアルタイムのパフォーマンスに最適化された Compute ロール。このロールを使用する場合には、**overcloud-realtime-compute** イメージが利用可能で、ロール固有のパラメーター **IsolCpusList**、**NovaComputeCpuDedicatedSet**、および **NovaComputeCpuSharedSet** がリアルタイムコンピュートノードのハードウェアに応じて設定されている必要があります。 | **ComputeRealTime.yaml** |
| **ComputeSriovRT** | コンピュート SR-IOV RealTime ロール | **ComputeSriovRT.yaml** |
| **ComputeSriov** | コンピュート SR-IOV ロール | **ComputeSriov.yaml** |
| **Compute** | 標準のコンピュートノードロール | **Compute.yaml** |

| ロール | 説明 | ファイル |
| --- | --- | --- |
| **ControllerAllNovaStandalone** | データベース、メッセージング、ネットワーク設定、および OpenStack Compute (nova) コントロールコンポーネントを持たない Controller ロール。**Database**、**Messaging**、**Networker**、および **Novacontrol** ロールと組み合わせて使用します。 | **ControllerAllNovaStandalone.yaml** |
| **ControllerNoCeph** | コア Controller サービスは組み込まれているが Ceph Storage (MON) コンポーネントを持たない Controller ロール。このロールはデータベース、メッセージング、およびネットワーク機能を処理しますが、Ceph Storage 機能は処理しません。 | **ControllerNoCeph.yaml** |
| **ControllerNovaStandalone** | OpenStack Compute (nova) コントロールコンポーネントが含まれない Controller ロール。**Novacontrol** ロールと組み合わせて使用します。 | **ControllerNovaStandalone.yaml** |
| **ControllerOpenstack** | データベース、メッセージング、およびネットワーク設定コンポーネントが含まれない Controller ロール。**Database**、**Messaging**、および **Networker** ロールと組み合わせて使用します。 | **ControllerOpenstack.yaml** |
| **ControllerStorageNfs** | すべてのコアサービスが組み込まれ、Ceph NFS を使用する Controller ロール。このロールはデータベース、メッセージング、およびネットワーク機能を処理します。 | **ControllerStorageNfs.yaml** |
| **Controller** | すべてのコアサービスが組み込まれた Controller ロール。このロールはデータベース、メッセージング、およびネットワーク機能を処理します。 | **Controller.yaml** |
| **ControllerSriov** (ML2/OVN) | 通常の Controller ロールと同じですが、OVN Metadata エージェントがデプロイされます。 | **ControllerSriov.yaml** |
| **Database** | スタンドアロンのデータベースロール。Pacemaker を使用して Galera クラスターとして管理されるデータベース。 | **Database.yaml** |
| **HciCephAll** | ハイパーコンバージドインフラストラクチャーおよびすべての Ceph Storage サービスを持つコンピュートノード。OSD、MON、Object Gateway (RGW)、Object Operations (MDS)、Manager (MGR)、および RBD Mirroring が含まれます。 | **HciCephAll.yaml** |
| **HciCephFile** | ハイパーコンバージドインフラストラクチャーおよび Ceph Storage ファイルサービスを持つコンピュートノード。OSD および Object Operations (MDS) が含まれます。 | **HciCephFile.yaml** |

| ロール | 説明 | ファイル |
|---|---|---|
| **HciCephMon** | ハイパーコンバージドインフラストラクチャーおよび Ceph Storage ブロックサービスを持つコンピュートノード。OSD、MON、および Manager が含まれます。 | **HciCephMon.yaml** |
| **HciCephObject** | ハイパーコンバージドインフラストラクチャーおよび Ceph Storage オブジェクトサービスを持つコンピュートノード。OSD および Object Gateway (RGW) が含まれます。 | **HciCephObject.yaml** |
| **IronicConductor** | Ironic Conductor ノードロール | **IronicConductor.yaml** |
| **Messaging** | スタンドアロンのメッセージングロール。Pacemaker を使用して管理される RabbitMQ。 | **Messaging.yaml** |
| **Networker** (ML2/OVS) | ML2/OVS でのスタンドアロンのネットワーク設定ロール。単独で OpenStack Networking (neutron) エージェントを実行します。デプロイメントで ML2/OVN メカニズムドライバーが使用される場合は、「ML2/OVN でのカスタム Networker ロールの作成」を参照してください。 | **Networker.yaml** |
| **NetworkerSriov** (ML2/OVN) | 通常の Networker ロールと同じですが、OVN Metadata エージェントがデプロイされます。「ML2/OVN でのカスタム Networker ロールの作成」を参照してください。 | **NetworkerSriov.yaml** |
| **Novacontrol** | スタンドアロンの **nova-control** ロール。単独で OpenStack Compute (nova) コントロールエージェントを実行します。 | **Novacontrol.yaml** |
| **ObjectStorage** | Swift オブジェクトストレージノードロール | **ObjectStorage.yaml** |
| **Telemetry** | すべてのメトリックおよびアラームサービスを持つ Telemetry ロール | **Telemetry.yaml** |

### 6.2.4. ML2/OVN でのカスタム Networker ロールの作成

デプロイメントで ML2/OVN メカニズムドライバーが使用される場合にカスタムの Networker ロールをデプロイするには、環境ファイルを使用してネットワーカーノードのロールのパラメーターを設定し、コントローラーノードから Networker ロールを消去する必要があります。

**neutron-ovn-dvr-ha.yaml** 等の環境ファイルを使用します。

**手順**

1. コントローラーノードで **OVNCMSOptions** を消去します。
   - 

```
ControllerParameters:
  OVNCMSOptions: ""
```

2. ネットワーカーノードで、**OVNCMSOptions** を **'enable-chassis-as-gw'** に設定します。

```
NetworkerParameters:
  OVNCMSOptions: "enable-chassis-as-gw"
```

3. SR-IOV を設定してカスタム Networker ロールをデプロイするには、ネットワーカーノードに以下の設定を追加します。

```
NetworkerSriovParameters:
  OVNCMSOptions: "enable-chassis-as-gw"
```

**注記**

本リリースでは、ML2/OVN デプロイメントで SR-IOV とネイティブ OVN DHCP の組み合わせを使用する場合、以下の制限が適用されます。

- すべてのポートに対して HA シャーシグループが 1 つしかないため、すべての外部ポートは単一のゲートウェイノード上でスケジュールされる。
- 外部ポートは論理ルーターのゲートウェイポートと共存しないため、VLAN テナントネットワークでは、VF (直接) ポートでの North-South ルーティングは SR-IOV では機能しない。Bug #1875852 を参照してください。

### 6.2.5. ロールパラメーターの考察

それぞれのロールには以下のパラメーターが含まれます。

**name**

**(必須)** 空白または特殊文字を含まないプレーンテキスト形式のロール名。選択した名前により、他のリソースとの競合が発生しないことを確認します。たとえば、**Network** の代わりに **Networker** を名前に使用します。

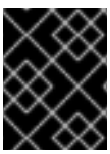
**description**

**(オプション)** プレーンテキスト形式のロールの説明

**tags**

**(オプション)** ロールのプロパティーを定義するタグの YAML リスト。このパラメーターを使用して **controller** と **primary** タグの両方で、プライマリーロールを定義します。

```
- name: Controller
  ...
  tags:
    - primary
    - controller
  ...
```



**重要**

プライマリーロールをタグ付けしない場合には、最初に定義するロールがプライマリーロールになります。このロールが Controller ロールとなるようにしてください。

networks

ロール上で設定するネットワークの YAML リストまたはディクショナリー。YAML リストを使用する場合には、各コンポーザブルネットワークの一覧を含めます。

```
networks:
  - External
  - InternalApi
  - Storage
  - StorageMgmt
  - Tenant
```

ディクショナリーを使用する場合には、各ネットワークをコンポーザブルネットワークの特定の **subnet** にマッピングします。

```
networks:
  External:
    subnet: external_subnet
  InternalApi:
    subnet: internal_api_subnet
  Storage:
    subnet: storage_subnet
  StorageMgmt:
    subnet: storage_mgmt_subnet
  Tenant:
    subnet: tenant_subnet
```

デフォルトのネットワークに
は、**External**、**InternalApi**、**Storage**、**StorageMgmt**、**Tenant**、**Management** が含まれます。

CountDefault

**(任意)** このロールにデプロイするデフォルトのノード数を定義します。

HostnameFormatDefault

**(任意)** このロールに対するホスト名のデフォルトの形式を定義します。デフォルトの命名規則では、以下の形式が使用されます。

```
[STACK NAME]-[ROLE NAME]-[NODE ID]
```

たとえば、コントローラーノード名はデフォルトで以下のように命名されます。

```
overcloud-controller-0
overcloud-controller-1
overcloud-controller-2
...
```

disable_constraints

**(任意)** director によるデプロイ時に OpenStack Compute (nova) および OpenStack Image Storage (glance) の制約を無効にするかどうかを定義します。事前にプロビジョニングされたノードでオーバークラウドをデプロイする場合に、このパラメーターを使用します。詳しくは、**『director のインストールと使用方法』**の「事前にプロビジョニングされたノードを使用した基本的なオーバークラウドの設定」を参照してください。

update_serial

**(任意)** OpenStack の更新オプション時に同時に更新するノードの数を定義します。**roles_data.yaml** ファイルのデフォルト設定は以下のとおりです。

- コントローラー、オブジェクトストレージ、および Ceph Storage ノードのデフォルトは **1** です。
- コンピュートおよびブロックストレージノードのデフォルトは **25** です。

このパラメーターをカスタムロールから省いた場合のデフォルトは **1** です。

ServicesDefault

**(任意)** ノード上で追加するデフォルトのサービス一覧を定義します。詳細な情報は、「コンポーザブルサービスアーキテクチャーの考察」を参照してください。

これらのパラメーターを使用して、新規ロールを作成すると共にロールに追加するサービスを定義することができます。

**openstack overcloud deploy** コマンドは、**roles_data** ファイルのパラメーターをいくつかのJinja2 ベースのテンプレートに統合します。たとえば、特定の時点で **overcloud.j2.yaml** heat テンプレートは **roles_data.yaml** のロールの一覧を繰り返し適用し、対応する各ロール固有のパラメーターとリソースを作成します。

たとえば、**overcloud.j2.yaml** heat テンプレートの各ロールのリソース定義は、以下のスニペットのようになります。

```
  {{role.name}}:
    type: OS::Heat::ResourceGroup
    depends_on: Networks
    properties:
      count: {get_param: {{role.name}}Count}
      removal_policies: {get_param: {{role.name}}RemovalPolicies}
      resource_def:
        type: OS::TripleO::{{role.name}}
        properties:
          CloudDomain: {get_param: CloudDomain}
          ServiceNetMap: {get_attr: [ServiceNetMap, service_net_map]}
          EndpointMap: {get_attr: [EndpointMap, endpoint_map]}
...
```

このスニペットには、Jinja2 ベースのテンプレートが **{{role.name}}** の変数を組み込み、各ロール名を **OS::Heat::ResourceGroup** リソースとして定義しているのが示されています。これは、次に **roles_data** ファイルのそれぞれの **name** パラメーターを使用して、対応する各 **OS::Heat::ResourceGroup** リソースを命名します。

### 6.2.6. 新規ロールの作成

コンポーザブルサービスのアーキテクチャーを使用して、デプロイメントの要件に応じて新規ロールを作成することができます。たとえば、OpenStack Dashboard (**horizon**) だけをホストする新しい **Horizon** ロールを作成するケースを考えます。

**手順**

1. デフォルトの **roles** ディレクトリーのカスタムコピーを作成します。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles ~/.
```

2. ~/**roles**/**Horizon.yaml** という名前の新規ファイルを作成して、ベースおよびコアの OpenStack Dashboard サービスが含まれる **Horizon** ロールを新規作成します。

```
- name: Horizon
  CountDefault: 1
  HostnameFormatDefault: '%stackname%-horizon-%index%'
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::Horizon
```

デフォルトのオーバークラウドに常に **Horizon** ノードが含まれるように、**CountDefault** を **1** に設定します。

3. (オプション) 既存のオーバークラウド内でサービスをスケーリングする場合は、既存のサービスを **Controller** ロール上に保持します。新規オーバークラウドを作成して、OpenStack Dashboard がスタンドアロンのロールに残るようにするには、**Controller** ロールの定義から OpenStack Dashboard コンポーネントを削除します。

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    ...
    - OS::TripleO::Services::GnocchiMetricd
    - OS::TripleO::Services::GnocchiStatsd
    - OS::TripleO::Services::HAproxy
    - OS::TripleO::Services::HeatApi
    - OS::TripleO::Services::HeatApiCfn
    - OS::TripleO::Services::HeatApiCloudwatch
    - OS::TripleO::Services::HeatEngine
    # - OS::TripleO::Services::Horizon               # Remove this service
    - OS::TripleO::Services::IronicApi
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Keepalived
    ...
```

4. ~/**roles** ディレクトリーをソースに使用して、新しい **roles_data-horizon.yaml** ファイルを生成します。

```
$ openstack overcloud roles generate -o roles_data-horizon.yaml \
  --roles-path ~/roles \
  Controller Compute Horizon
```

5. このロールに新しいフレーバーを定義して、特定のノードをタグ付けできるようにする必要がある場合があります。この例では、以下のコマンドを使用して **horizon** フレーバーを作成します。

```
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 horizon
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property "capabilities:profile"="horizon" horizon
$ openstack flavor set --property resources:VCPU=0 --property resources:MEMORY_MB=0 -
-property resources:DISK_GB=0 --property resources:CUSTOM_BAREMETAL=1 horizon
```

6. ノードを新規フレーバーにタグ付けします。

```
$ openstack baremetal node set --property capabilities='profile:horizon,boot_option:local'
58c3d07e-24f2-48a7-bbb6-6843f0e8ee13
```

7. 以下の環境ファイルのスニペットを使用して、Horizon ノードの数とフレーバーを定義します。

```
parameter_defaults:
  OvercloudHorizonFlavor: horizon
  HorizonCount: 1
```

8. ご自分のデプロイメントに該当するその他の環境ファイルと共に、新しい **roles_data-horizon.yaml** ファイルおよび環境ファイルを **openstack overcloud deploy** コマンドに追加します。

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-horizon.yaml -e
~/templates/node-count-flavor.yaml
```

この設定により、コントローラーノードが 1 台、コンピュートノードが 1 台、ネットワーカーノードが 1 台の 3 ノード構成のオーバークラウドが作成されます。オーバークラウドのノード一覧を表示するには、以下のコマンドを実行します。

```
$ openstack server list
```

## 6.3. コンポーザブルサービス

### 6.3.1. ガイドラインおよび制限事項

コンポーザブルロールのアーキテクチャーには、以下のガイドラインおよび制限事項があることに注意してください。

Pacemaker により管理されないサービスの場合:

- スタンドアロンのカスタムロールにサービスを割り当てることができます。
- 初回のデプロイメント後に追加のカスタムロールを作成してそれらをデプロイし、既存のサービスをスケーリングすることができます。

Pacemaker により管理されるサービスの場合:

- スタンドアロンのカスタムロールに Pacemaker の管理対象サービスを割り当てることができます。

- Pacemaker のノード数の上限は 16 です。Pacemaker サービス (**OS::TripleO::Services::Pacemaker**) を 16 のノードに割り当てた場合には、それ以降のノードは、代わりに Pacemaker Remote サービス (**OS::TripleO::Services::PacemakerRemote**) を使用する必要があります。同じロールで Pacemaker サービスと Pacemaker Remote サービスを割り当てることはできません。

- Pacemaker の管理対象サービスが割り当てられていないロールには、Pacemaker サービス (**OS::TripleO::Services::Pacemaker**) を追加しないでください。

- **OS::TripleO::Services::Pacemaker** または **OS::TripleO::Services::PacemakerRemote** のサービスが含まれているカスタムロールはスケールアップまたはスケールダウンできません。

一般的な制限事項

- メジャーバージョン間のアップグレードプロセス中にカスタムロールとコンポーザブルサービスを変更することはできません。

- オーバクラウドのデプロイ後には、ロールのサービスリストを変更することはできません。オーバークラウドのデプロイ後にサービスリストを変更すると、デプロイでエラーが発生して、ノード上に孤立したサービスが残ってしまう可能性があります。

### 6.3.2. コンポーザブルサービスアーキテクチャーの考察

コア heat テンプレートコレクションには、コンポーザブルサービスのテンプレートセットが 2 つ含まれています。

- **deployment** には、主要な OpenStack サービスのテンプレートが含まれます。

- **puppet/services** には、コンポーザブルサービスを設定するためのレガシーテンプレートが含まれます。互換性を維持するために、一部のコンポーザブルサービスは、このディレクトリーからのテンプレートを使用する場合があります。多くの場合、コンポーザブルサービスは **deployment** ディレクトリーのテンプレートを使用します。

各テンプレートには目的を特定する記述が含まれています。たとえば、**deployment/time/ntp-baremetal-puppet.yaml** サービステンプレートには以下のような記述が含まれます。

```
description: >
  NTP service deployment using puppet, this YAML file
  creates the interface between the HOT template
  and the puppet manifest that actually installs
  and configure NTP.
```

これらのサービステンプレートは、Red Hat OpenStack Platform デプロイメント固有のリソースとして登録されます。これは、**overcloud-resource-registry-puppet.j2.yaml** ファイルで定義されている一意な heat リソース名前空間を使用して、各リソースを呼び出すことができることを意味します。サービスはすべて、リソース種別に **OS::TripleO::Services** 名前空間を使用します。

一部のリソースは、直接コンポーザブルサービスのベーステンプレートを使用します。

```
resource_registry:
  ...
  OS::TripleO::Services::Ntp: deployment/time/ntp-baremetal-puppet.yaml
  ...
```

ただし、コアサービスにはコンテナーが必要なので、コンテナー化されたサービステンプレートを使用します。たとえば、コンテナー化された **keystone** サービスでは、以下のリソースを使用します。

```
resource_registry:
  ...
  OS::TripleO::Services::Keystone: deployment/keystone/keystone-container-puppet.yaml
  ...
```

通常、これらのコンテナー化されたテンプレートは、依存関係を追加するために他のテンプレートを参照します。たとえば、**deployment/keystone/keystone-container-puppet.yaml** テンプレートは、**ContainersCommon** リソースにベーステンプレートの出力を保管します。

```
resources:
  ContainersCommon:
    type: ../containers-common.yaml
```

これにより、コンテナー化されたテンプレートは、**containers-common.yaml** テンプレートからの機能やデータを取り込むことができます。

**overcloud.j2.yaml** heat テンプレートには、**roles_data.yaml** ファイル内の各カスタムロールのサービス一覧を定義するための Jinja2-based コードのセクションが含まれています。

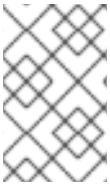
```
{{role.name}}Services:
  description: A list of service resources (configured in the heat
               resource_registry) which represent nested stacks
               for each service that should get installed on the {{role.name}} role.
  type: comma_delimited_list
  default: {{role.ServicesDefault|default([])}}
```

デフォルトのロールの場合は、これにより次のサービス一覧パラメーターが作成されます: **ControllerServices**、**ComputeServices**、**BlockStorageServices**、**ObjectStorageServices**、**CephStorageServices**

**roles_data.yaml** ファイル内の各カスタムロールのデフォルトのサービスを定義します。たとえば、デフォルトの Controller ロールには、以下の内容が含まれます。

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Core
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
...
```

これらのサービスは、次に **ControllerServices** パラメーターのデフォルト一覧として定義されます。



**注記**

環境ファイルを使用してサービスパラメーターのデフォルト一覧を上書きすることもできます。たとえば、環境ファイルで **ControllerServices** を **parameter_default** として定義して、**roles_data.yaml** ファイルからのサービス一覧を上書きすることができます。

### 6.3.3. ロールへのサービスの追加と削除

サービスの追加と削除の基本的な方法では、ノードロールのデフォルトサービス一覧のコピーを作成してからサービスを追加/削除します。たとえば、OpenStack Orchestration (heat) をコントローラーノードから削除するケースを考えます。

1. デフォルトの **roles** ディレクトリーのカスタムコピーを作成します。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles ~/.
```

2. **~/roles/Controller.yaml** ファイルを編集して、**ServicesDefault** パラメーターのサービス一覧を変更します。OpenStack Orchestration のサービスまでスクロールしてそれらを削除します。

```
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
    - OS::TripleO::Services::HeatApi            # Remove this service
    - OS::TripleO::Services::HeatApiCfn         # Remove this service
    - OS::TripleO::Services::HeatApiCloudwatch  # Remove this service
    - OS::TripleO::Services::HeatEngine         # Remove this service
    - OS::TripleO::Services::MySQL
    - OS::TripleO::Services::NeutronDhcpAgent
```

3. 新しい **roles_data** ファイルを生成します。

```
$ openstack overcloud roles generate -o roles_data-no_heat.yaml \
  --roles-path ~/roles \
  Controller Compute Networker
```

4. **openstack overcloud deploy** コマンドを実行する際に、この新しい **roles_data** ファイルを指定します。

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-no_heat.yaml
```

このコマンドにより、コントローラノードには OpenStack Orchestration のサービスがインストールされない状態でオーバークラウドがデプロイされます。

**注記**

また、カスタムの環境ファイルを使用して、**roles_data** ファイル内のサービスを無効にすることもできます。無効にするサービスを **OS::Heat::None** リソースにリダイレクトします。以下に例を示します。

```
resource_registry:
  OS::TripleO::Services::HeatApi: OS::Heat::None
  OS::TripleO::Services::HeatApiCfn: OS::Heat::None
  OS::TripleO::Services::HeatApiCloudwatch: OS::Heat::None
  OS::TripleO::Services::HeatEngine: OS::Heat::None
```

### 6.3.4. 無効化されたサービスの有効化

一部のサービスはデフォルトで無効化されています。これらのサービスは、**overcloud-resource-registry-puppet.j2.yaml** ファイルで null 操作 (**OS::Heat::None**) として登録されます。たとえば、Block Storage のバックアップサービス (**cinder-backup**) は無効化されています。

```
  OS::TripleO::Services::CinderBackup: OS::Heat::None
```

このサービスを有効化するには、**puppet/services** ディレクトリー内の対応する heat テンプレートにリソースをリンクする環境ファイルを追加します。一部のサービスには、**environments** ディレクトリー内に事前定義済みの環境ファイルがあります。たとえば、Block Storage のバックアップサービスは、以下のエントリーが含まれる **environments/cinder-backup.yaml** ファイルを使用します。

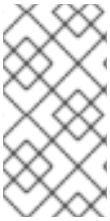
```
resource_registry:
  OS::TripleO::Services::CinderBackup: ../podman/services/pacemaker/cinder-backup.yaml
...
```

このエントリーにより、デフォルトの null 操作のリソースが上書きされ、これらのサービスが有効になります。**openstack overcloud deploy** コマンドの実行時に、この環境ファイルを指定します。

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-
templates/environments/cinder-backup.yaml
```

### 6.3.5. サービスなしの汎用ノードの作成

OpenStack サービスを一切設定しない汎用の Red Hat Enterprise Linux 8.2 ノードを作成することができます。これは、コアの Red Hat OpenStack Platform (RHOSP) 環境外でソフトウェアをホストする必要がある場合に役立ちます。たとえば、RHOSP は、Kibana や Sensu 等のモニタリングツールとの統合を提供します。詳細は、**『監視ツール設定ガイド』**を参照してください。Red Hat は、それらのモニタリングツールに対するサポートは提供しませんが、director では、それらのツールをホストする汎用の Red Hat Enterprise Linux 8.2 ノードの作成が可能です。



**注記**

汎用ノードは、ベースの Red Hat Enterprise Linux 8 イメージではなく、ベースの **overcloud-full** イメージを引き続き使用します。これは、ノードには何らかの Red Hat OpenStack Platform ソフトウェアがインストールされていますが、有効化または設定されていないことを意味します。

1. カスタムの **roles_data.yaml** ファイルに **ServicesDefault** の一覧が含まれない汎用ロールを作成します。

```
- name: Generic
- name: Controller
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
- name: Compute
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
```

既存の **Controller** ロールおよび **Compute** ロールを維持するようにしてください。

2. 必要な汎用 Red Hat Enterprise Linux 8 ノードの数およびプロビジョニングするノードを選択する際のフレーバーを指定する環境ファイル **generic-node-params.yaml** を作成します。

```
parameter_defaults:
  OvercloudGenericFlavor: baremetal
  GenericCount: 1
```

3. **openstack overcloud deploy** コマンドを実行する際に、ロールのファイルと環境ファイルの両方を指定します。

```
$ openstack overcloud deploy --templates \
-r ~/templates/roles_data_with_generic.yaml \
-e ~/templates/generic-node-params.yaml
```

この設定により、コントローラーノードが 1 台、コンピュートノードが 1 台、汎用 Red Hat Enterprise Linux 8 ノードが 1 台の 3 ノード構成の環境がデプロイされます。

# 第7章 コンテナー化されたサービス

director は、OpenStack Platform のコアサービスをオーバークラウド上にコンテナーとしてインストールします。本項では、コンテナー化されたサービスがどのように機能するかについての背景情報を記載します。

## 7.1. コンテナー化されたサービスのアーキテクチャー

director は、OpenStack Platform のコアサービスをオーバークラウド上にコンテナーとしてインストールします。コンテナー化されたサービス用のテンプレートは、**/usr/share/openstack-tripleo-heat-templates/deployment/** にあります。

コンテナー化されたサービスを使用するすべてのノードで、ロールの **OS::TripleO::Services::Podman** サービスを有効にする必要があります。カスタムロール設定用の **roles_data.yaml** ファイルを作成する際には、ベースのコンポーザブルサービスと共に **OS::TripleO::Services::Podman** サービスを追加します。たとえば、**IronicConductor** ロールには、以下の定義を使用します。

```
- name: IronicConductor
  description: |
    Ironic Conductor node role
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Storage:
      subnet: storage_subnet
  HostnameFormatDefault: '%stackname%-ironic-%index%'
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::BootParams
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::IpaClient
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::IronicPxe
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MetricsQdr
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::Podman
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timesync
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
```

## 7.2. コンテナー化されたサービスのパラメーター

コンテナー化されたサービスのテンプレートにはそれぞれ、**outputs** セクションがあります。このセクションでは、OpenStack Orchestration (heat) サービスに渡すデータセットを定義します。テンプレートには、標準のコンポーザブルサービスパラメーター (「ロールパラメーターの考察」を参照) に加えて、コンテナーの設定固有のパラメーターセットが含まれます。

**puppet_config**

サービスの設定時に Puppet に渡すデータ。初期のオーバークラウドデプロイメントステップでは、director は、コンテナー化されたサービスが実際に実行される前に、サービスの設定に使用するコンテナーのセットを作成します。このパラメーターには以下のサブパラメーターが含まれます。

- **config_volume**: 設定を格納するマウント済みのボリューム
- **puppet_tags**: 設定中に Puppet に渡すタグ。OpenStack は、これらのタグを使用して Puppet の実行を特定サービスの設定リソースに制限します。たとえば、OpenStack Identity (keystone) のコンテナー化されたサービスは、**keystone_config** タグを使用して、設定コンテナーで **keystone_config** Puppet リソースを実行します。
- **step_config**: Puppet に渡される設定データ。これは通常、参照されたコンポーザブルサービスから継承されます。
- **config_image**: サービスを設定するためのコンテナーイメージ

**kolla_config**

設定ファイルの場所、ディレクトリーのパーミッション、およびサービスを起動するためにコンテナー上で実行するコマンドを定義するコンテナー固有のデータセット

**docker_config**

サービスの設定コンテナーで実行するタスク。すべてのタスクは以下に示すステップにグループ化され、director が段階的にデプロイメントを行うのに役立ちます。

- **ステップ 1**: ロードバランサーの設定
- **ステップ 2**: コアサービス (データベース、Redis)
- **ステップ 3**: OpenStack Platform サービスの初期設定
- **ステップ 4**: OpenStack Platform サービスの全般設定
- **ステップ 5**: サービスのアクティブ化

**host_prep_tasks**

ベアメタルノードがコンテナー化されたサービスに対応するための準備タスク

## 7.3. コンテナーイメージの準備

オーバークラウドのインストールには、コンテナー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、コンテナーイメージの準備に使用します。

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

## 7.4. 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。

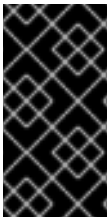
```
parameter_defaults:
  ContainerImagePrepare:
  - (strategy one)
  - (strategy two)
  - (strategy three)
  ...
```

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** | 設定からイメージ名を除外する正規表現の一覧 |
| **includes** | 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。**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。 |

| パラメーター | 説明 |
| --- | --- |
| **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、16.1.3-5.161 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 16.1.3-5.161-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を定義します。<br>● **true** に設定すると、**push_destination** 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。<br>● **false** 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。<br>● 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。<br>実稼働環境でこのパラメーターを **false** 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。コンテナー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、**false** を使用します。<br>**push_destination** パラメーターが **false** に設定されている (または定義されていない) 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含めます。 |

| 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、**key: value** 定義のディクショナリー |
| **tag_from_label** | 指定したコンテナー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、**tag_from_label: {version}-{release}** を設定した場合、director は **version** および **release** ラベルを使用して新しいタグを構築します。あるコンテナーについて **version** を 16.1.3 に、**release** を **5.161** に設定した場合、タグは 16.1.3-5.161 となります。**set** ディクショナリーで **tag** 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。 |



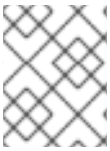
**重要**

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** ではなく **push_destination: true** を使用します。**push_destination: true** 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 **key: value** 定義を設定することができます。

| 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 |

| キー | 説明 |
| --- | --- |
| **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 **tag_from_label** の値よりも優先されます。 |



**注記**

コンテナーイメージでは、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。

# 第8章 コンテナーイメージタグ付けのガイドライン

Red Hat コンテナーレジストリーでは、すべての Red Hat OpenStack Platform コンテナー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ーのラベルのメタデータに従います。

**version**

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、特定コンテナー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 16.1.3 で、コンテナーイメージのリリースが **5.161** の場合、コンテナーイメージのタグは 16.1.3-5.161 となります。

Red Hat コンテナーレジストリーでは、コンテナー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、16.1 と 16.1.3 の両方が、16.1.3 コンテナーストリームの最新 **release** とリンクします。16.1 の新規マイナーリリースが公開されると、16.1 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、16.1.3 タグは、引き続き 16.1.3 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ー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、16.1 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: 16.1
      ...
```

- OpenStack Platform コンテナー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、16.1.2 に変更するには、**tag** を 16.1.2 に設定します。

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 16.1.2
      ...
```

- **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ーイメージ **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: 16.1
    ...
  tag_from_label: '{version}-{release}'
```

- **tag_from_label** パラメーターは、Red Hat コンテナーレジストリーから検査する最新コンテナー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、特定のコンテナーのラベルは、以下の **version** および **release** メタデータを使用します。

```
"Labels": {
  "release": "5.161",
  "version": "16.1.3",
  ...
}
```

- **tag_from_label** のデフォルト値は **{version}-{release}** です。これは、各コンテナー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、コンテナーイメージの **version** に 16.1.3 が、 **release** に 5.161 が、それぞれ設定されている場合、コンテナーイメージのタグは 16.1.3-5.161 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、コンテナー準備の設定で **tag** パラメーターを省略します。

- **tag** と **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 コンテナー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、各コンテナーイメージのメタデータの検査を行います。

# 第9章 プライベートレジストリーからのコンテナーイメージの取得

**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との間で認証を行うには、**containers-prepare-parameter.yaml** ファイルに **ContainerImageRegistryCredentials** および **ContainerImageRegistryLogin** パラメーターを追加します。

### ContainerImageRegistryCredentials

一部のコンテナー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** からコンテナーイメージをプルします。

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

**ContainerImageRegistryLogin** パラメーターを使用して、コンテナーイメージを取得するの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** で定義し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** に設定している場合は、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** で定義し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は、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、**push_destination** を **true** に、**ContainerImageRegistryLogin** を **false** に、それぞれ設定します。

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

## 9.1. 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: 16.1-44
```

**includes** および **excludes** 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。

## 9.2. 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。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稼働中の **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。

## 9.3. 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コンテナー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
  ...
```

## 9.4. 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
  ...
```

## 9.5. 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 が含まれる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** 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"
```

## 9.6. ベンダープラグインのデプロイ

一部のサードパーティーハードウェアをブロックストレージのバックエンドとして使用するには、ベンダープラグインをデプロイする必要があります。以下の例で、Dell EMC ハードウェアをブロックストレージのバックエンドとして使用するために、ベンダープラグインをデプロイする方法について説明します。

サポート対象のバックエンドアプライアンスおよびドライバーに関する詳細は、**『ストレージガイド』**の「サードパーティーのストレージプロバイダー」を参照してください。

**手順**

1. オーバークラウド用に新たなコンテナーイメージファイルを作成します。

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

2. containers-prepare-parameter-dellemc.yaml ファイルを編集します。

3. メインの Red Hat OpenStack Platform コンテナーイメージの設定に **exclude** パラメーターを追加します。このパラメーターを使用して、ベンダーコンテナーイメージで置き換えるコンテナーイメージを除外します。以下の例では、コンテナーイメージは **cinder-volume** イメージです。

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
      excludes:
      - cinder-volume
      set:
        namespace: registry.redhat.io/rhosp-rhel8
        name_prefix: openstack-
        name_suffix: ''
        tag: 16.1
        ...
      tag_from_label: "{version}-{release}"
```

4. **ContainerImagePrepare** パラメーターに、ベンダープラグインの代替コンテナーイメージが含まれる新しい設定を追加します。

```
parameter_defaults:
  ContainerImagePrepare:
    ...
    - push_destination: true
      includes:
        - cinder-volume
      set:
        namespace: registry.connect.redhat.com/dellemc
        name_prefix: openstack-
        name_suffix: -dellemc-rhosp16
        tag: 16.1-2
        ...
```

5. **ContainerImageRegistryCredentials** パラメーターに registry.connect.redhat.com レジストリーの認証情報を追加します。

```
parameter_defaults:
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      [service account username]: [service account password]
    registry.connect.redhat.com:
      [service account username]: [service account password]
```

6. **containers-prepare-parameter-dellemc.yaml** ファイルを保存します。

7. **openstack overcloud deploy** 等のデプロイメントコマンドに **containers-prepare-parameter-dellemc.yaml** ファイルを追加します。

```
$ openstack overcloud deploy --templates
    ...
    -e containers-prepare-parameter-dellemc.yaml
    ...
```

director がオーバークラウドをデプロイする際に、オーバークラウドは標準のコンテナーイメージの代わりにベンダーのコンテナーイメージを使用します。

**重要**

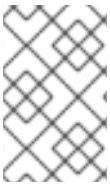
オーバークラウドデプロイメントの標準の **containers-prepare-parameter.yaml** ファイル

は、**containers-prepare-parameter-dellemc.yaml** ファイルにより置き換えられます。オーバークラウドのデプロイメントには、標準の **containers-prepare-parameter.yaml** ファイルを含めないでください。アンダークラウドのインストールおよび更新には、標準の **containers-prepare-parameter.yaml** ファイルを維持します。

# 第10章 基本的なネットワーク分離

特定の種別のネットワークトラフィックを分離してホストできるように、分離されたネットワークを使用するようにオーバークラウドを設定します。Red Hat OpenStack Platform には、このネットワーク分離を設定するための環境ファイルのセットが含まれています。ネットワーク設定のパラメーターをさらにカスタマイズするために、追加の環境ファイルが必要になる場合もあります。

- ネットワーク分離を有効にするための環境ファイル (**/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml**)
- ネットワークのデフォルト値を設定するための環境ファイル (**/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml**)
- IP 範囲、サブネット、仮想 IP 等のネットワーク設定を定義するための **network_data** ファイル。以下の例では、デフォルトのファイルをコピーし、それをご自分のネットワークに合わせて編集する方法について説明します。
- 各ノードの NIC レイアウトを定義するためのテンプレート。オーバークラウドのコアテンプレートコレクションには、さまざまなユースケースに対応する複数のデフォルトが含まれます。
- NIC を有効にするための環境ファイル。以下の例では、**environments** ディレクトリーにあるデフォルトファイルを使用しています。



**注記**

前述の一覧のファイルは一部 Jinja2 形式のファイルで、**.j2.yaml** の拡張子を持つものがあります。director は、デプロイメント中にこれらのファイルを **.yaml** バージョンにレンダリングします。

## 10.1. ネットワーク分離

デフォルトでは、オーバークラウドはサービスをプロビジョニングネットワークに割り当てます。ただし、director はオーバークラウドのネットワークトラフィックを分離したネットワークに分割することができます。分離ネットワークを使用するために、オーバークラウドにはこの機能を有効にする環境ファイルが含まれています。コア heat テンプレートの **environments/network-isolation.j2.yaml** ファイルは Jinja2 形式のファイルで、コンポーザブルネットワークファイル内の各ネットワークのポートおよび仮想 IP をすべて定義します。レンダリングすると、すべてのリソースレジストリーと共に **network-isolation.yaml** ファイルが同じ場所に生成されます。

```
resource_registry:
  # networks as defined in network_data.yaml
  OS::TripleO::Network::Storage: ../network/storage.yaml
  OS::TripleO::Network::StorageMgmt: ../network/storage_mgmt.yaml
  OS::TripleO::Network::InternalApi: ../network/internal_api.yaml
  OS::TripleO::Network::Tenant: ../network/tenant.yaml
  OS::TripleO::Network::External: ../network/external.yaml

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::StorageVipPort: ../network/ports/storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: ../network/ports/storage_mgmt.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: ../network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::ExternalVipPort: ../network/ports/external.yaml
  OS::TripleO::Network::Ports::RedisVipPort: ../network/ports/vip.yaml
```

```
# Port assignments by role, edit role definition to assign networks to roles.
# Port assignments for the Controller
OS::TripleO::Controller::Ports::StoragePort: ../network/ports/storage.yaml
OS::TripleO::Controller::Ports::StorageMgmtPort: ../network/ports/storage_mgmt.yaml
OS::TripleO::Controller::Ports::InternalApiPort: ../network/ports/internal_api.yaml
OS::TripleO::Controller::Ports::TenantPort: ../network/ports/tenant.yaml
OS::TripleO::Controller::Ports::ExternalPort: ../network/ports/external.yaml

# Port assignments for the Compute
OS::TripleO::Compute::Ports::StoragePort: ../network/ports/storage.yaml
OS::TripleO::Compute::Ports::InternalApiPort: ../network/ports/internal_api.yaml
OS::TripleO::Compute::Ports::TenantPort: ../network/ports/tenant.yaml

# Port assignments for the CephStorage
OS::TripleO::CephStorage::Ports::StoragePort: ../network/ports/storage.yaml
OS::TripleO::CephStorage::Ports::StorageMgmtPort: ../network/ports/storage_mgmt.yaml
```

このファイルの最初のセクションには、**OS::TripleO::Network::*** リソースのリソースレジストリーの宣言が含まれます。デフォルトでは、これらのリソースは、ネットワークを作成しない **OS::Heat::None** リソースタイプを使用します。これらのリソースを各ネットワークの YAML ファイルにリダイレクトすると、それらのネットワークの作成が可能となります。

次の数セクションで、各ロールのノードに IP アドレスを指定します。コントローラーノードでは、ネットワークごとに IP が指定されます。コンピュートノードとストレージノードは、ネットワークのサブネットでの IP が指定されます。

オーバークラウドネットワークのその他の機能 (「カスタムコンポーザブルネットワーク」および「カスタムネットワークインターフェーステンプレート」を参照) は、**network-isolation.yaml** 環境ファイルに依存します。したがって、デプロイメントコマンドにレンダリングした環境ファイルを追加する必要があります。

```
$ openstack overcloud deploy --templates \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  ...
```

## 10.2. 分離ネットワーク設定の変更

デフォルトの **network_data.yaml** ファイルをコピーし、コピーを修正してデフォルトの分離ネットワークを設定します。

**手順**

1. デフォルトの **network_data.yaml** ファイルをコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/network_data.yaml /home/stack/.
```

2. **network_data.yaml** ファイルのローカルコピーを編集し、ご自分のネットワーク要件に応じてパラメーターを変更します。たとえば、内部 API ネットワークには以下のデフォルトネットワーク情報が含まれます。

```
- name: InternalApi
  name_lower: internal_api
  vip: true
```

```
vlan: 201
ip_subnet: '172.16.2.0/24'
allocation_pools: [{'start': '172.16.2.4', 'end': '172.16.2.250'}]
```

各ネットワークについて、以下の値を編集します。

- **vlan**: このネットワークに使用する VLAN ID を定義します。
- **ip_subnet** および **ip_allocation_pools**: ネットワークのデフォルトサブネットおよび IP 範囲を設定します。
- **gateway**: ネットワークのゲートウェイを設定します。この値を使用して、外部ネットワークまたは必要に応じて他のネットワークのデフォルトルートを定義します。

**-n** オプションを使用して、カスタム **network_data.yaml** ファイルをデプロイメントに含めます。**-n** オプションを設定しないと、デプロイメントコマンドはデフォルトのネットワーク情報を使用します。

## 10.3. ネットワークインターフェーステンプレート

オーバークラウドのネットワーク設定には、ネットワークインターフェースのテンプレートセットが必要です。これらのテンプレートは YAML 形式の標準の heat テンプレートです。director がロール内の各ノードを正しく設定できるように、それぞれのロールには NIC のテンプレートが必要です。

すべての NIC のテンプレートには、標準の heat テンプレートと同じセクションが含まれています。

**heat_template_version**
使用する構文のバージョン

**description**
テンプレートを説明する文字列

**parameters**
テンプレートに追加するネットワークパラメーター

**resources**
**parameters** で定義したパラメーターを取得し、それらをネットワークの設定スクリプトに適用します。

**outputs**
設定に使用する最終スクリプトをレンダリングします。

**/usr/share/openstack-tripleo-heat-templates/network/config** のデフォルト NIC テンプレートは、Jinja2 構文を使用してテンプレートをレンダリングします。たとえば、**single-nic-vlans** 設定からの以下のスニペットにより、各ネットワークの VLAN セットがレンダリングされます。

```
{%- for network in networks if network.enabled|default(true) and network.name in role.networks %}
- type: vlan
  vlan_id:
    get_param: {{network.name}}NetworkVlanID
  addresses:
  - ip_netmask:
      get_param: {{network.name}}IpSubnet
{%- if network.name in role.default_route_networks %}
```

デフォルトのコンピュートノードでは、Storage、Internal API、および Tenant ネットワークのネットワーク情報だけがレンダリングされます。

```
- type: vlan
  vlan_id:
    get_param: StorageNetworkVlanID
  device: bridge_name
  addresses:
  - ip_netmask:
      get_param: StorageIpSubnet
- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  device: bridge_name
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet
- type: vlan
  vlan_id:
    get_param: TenantNetworkVlanID
  device: bridge_name
  addresses:
  - ip_netmask:
      get_param: TenantIpSubnet
```

デフォルトの Jinja2 ベースのテンプレートを標準の YAML バージョンにレンダリングする方法は、「「カスタムネットワークインターフェーステンプレート」」で説明します。この YAML バージョンを、カスタマイズのベースとして使用することができます。

## 10.4. デフォルトのネットワークインターフェーステンプレート

director の **/usr/share/openstack-tripleo-heat-templates/network/config/** には、ほとんどの標準的なネットワークシナリオに適するテンプレートが含まれています。以下の表は、各 NIC テンプレートセットおよびテンプレートを有効にするのに使用する必要がある環境ファイルの概要をまとめたものです。

**注記**

NIC テンプレートを有効にするそれぞれの環境ファイルには、接尾辞 **.j2.yaml** が使われます。これはレンダリング前の Jinja2 バージョンです。デプロイメントには、接尾辞に **.yaml** が使われるレンダリング済みのファイル名を指定するようにしてください。

| NIC ディレクトリー | 説明 | 環境ファイル |
|---|---|---|
| **single-nic-vlans** | 単一の NIC (**nic1**) がコントロールプレーンネットワークにアタッチされ、VLAN 経由でデフォルトの Open vSwitch ブリッジにアタッチされる。 | **environments/net-single-nic-with-vlans.j2.yaml** |
| **single-nic-linux-bridge-vlans** | 単一の NIC (**nic1**) がコントロールプレーンネットワークにアタッチされ、VLAN 経由でデフォルトの Linux ブリッジにアタッチされる。 | **environments/net-single-nic-linux-bridge-with-vlans** |

| NIC ディレクトリー | 説明 | 環境ファイル |
|---|---|---|
| **bond-with-vlans** | コントロールプレーンネットワークが **nic1** にアタッチされる。ボンディング構成の NIC (**nic2** および **nic3**) が VLAN 経由でデフォルトの Open vSwitch ブリッジにアタッチされる。 | **environments/net-bond-with-vlans.yaml** |
| **multiple-nics** | コントロールプレーンネットワークが **nic1** にアタッチされる。それ以降の NIC は **network_data.yaml** ファイルで定義されるネットワークに割り当てられる。デフォルトでは、Storage が **nic2** に、Storage Management が **nic3** に、Internal API が **nic4** に、Tenant が **br-tenant** ブリッジ上の **nic5** に、External がデフォルトの Open vSwitch ブリッジ上の **nic6** に割り当てられる。 | **environments/net-multiple-nics.yaml** |

**注記**

外部ネットワークを使用しないオーバークラウドデプロイ用の環境ファイル (例: **net-bond-with-vlans-no-external.yaml**) や IPv6 デプロイメント用の環境ファイル (例: **net-bond-with-vlans-v6.yaml**) も存在します。これらは後方互換のために提供されるもので、コンポーザブルネットワークでは機能しません。

それぞれのデフォルト NIC テンプレートセットには、**role.role.j2.yaml** テンプレートが含まれます。このファイルは、Jinja2 を使用して各コンポーザブルロールの追加ファイルをレンダリングします。たとえば、オーバークラウドで Compute、Controller、および Ceph Storage ロールが使用される場合には、デプロイメントにより、**role.role.j2.yaml** をベースに以下のようなテンプレートが新たにレンダリングされます。

- **compute.yaml**
- **controller.yaml**
- **ceph-storage.yaml**

## 10.5. 基本的なネットワーク分離の有効化

director には、基本的なネットワーク分離を有効にするためのテンプレートが含まれています。これらのファイルは、**/usr/share/openstack-tripleo-heat-templates/environments** ディレクトリーにあります。たとえば、テンプレートを使用して、基本的なネットワーク分離の VLAN が設定された単一の NIC にオーバークラウドをデプロイすることができます。以下のシナリオでは、**net-single-nic-with-vlans** テンプレートを使用します。

**手順**

1. **openstack overcloud deploy** コマンドを実行する際に、以下のレンダリング済み環境ファイルを追加するようにしてください。
    - カスタム **network_data.yaml** ファイル
    - デフォルトネットワーク分離ファイルのレンダリング済みファイル名
    - デフォルトネットワーク環境ファイルのレンダリング済みファイル名
    - デフォルトネットワークインターフェース設定ファイルのレンダリング済みファイル名
    - 設定に必要なその他の環境ファイル

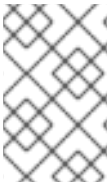
以下に例を示します。

```
$ openstack overcloud deploy --templates \
  ...
  -n /home/stack/network_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml \
  ...
```

### 10.5.1. カスタムコンポーザブルネットワーク

特定のネットワークトラフィックを異なるネットワークでホストする場合は、カスタムコンポーザブルネットワークを作成することができます。オーバークラウドに追加のコンポーザブルネットワークを設定するには、以下のファイルおよびテンプレートを設定する必要があります。

- ネットワーク分離を有効にするための環境ファイル (**/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml**)
- ネットワークのデフォルト値を設定するための環境ファイル (**/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml**)
- デフォルト以外の追加ネットワークを作成するためのカスタム **network_data** ファイル
- カスタムネットワークをロールに割り当てるためのカスタム **roles_data** ファイル
- 各ノードの NIC レイアウトを定義するためのテンプレート。オーバークラウドのコアテンプレートコレクションには、さまざまなユースケースに対応する複数のデフォルトが含まれます。
- NIC を有効にするための環境ファイル。以下の例では、**environments** ディレクトリーにあるデフォルトファイルを使用しています。
- ネットワーク設定パラメーターをカスタマイズするその他の環境ファイル。以下の例では、OpenStack サービスとコンポーザブルネットワークのマッピングをカスタマイズするための環境ファイルを用いています。



**注記**

前述の一覧のファイルは一部 Jinja2 形式のファイルで、**.j2.yaml** の拡張子を持つものがあります。director は、デプロイメント中にこれらのファイルを **.yaml** バージョンにレンダリングします。

#### 10.5.1.1. コンポーザブルネットワーク

デフォルトのオーバークラウドでは、以下に示す事前定義済みのネットワークセグメントのセットが使用されます。

- Control Plane
- Internal API
- Storage
- Storage Management
- Tenant
- External
- Management (オプション)

コンポーザブルネットワークを使用して、さまざまなサービス用のネットワークを追加することができます。たとえば、NFS トラフィック専用のネットワークがある場合には、複数のロールに提供できます。

director は、デプロイメントおよび更新フェーズでのカスタムネットワークの作成をサポートしています。このような追加のネットワークを、Ironic ベアメタルノード、システム管理に使用したり、異なるロール用に別のネットワークを作成するのに使用したりすることができます。また、これは、トラフィックが複数のネットワーク間でルーティングされる、分離型のデプロイメント用に複数のネットワークセットを作成するのに使用することもできます。

1 つのデータファイル (**network_data.yaml**) で、デプロイするネットワークの一覧を管理します。**-n** オプションを使用して、このファイルをデプロイメントコマンドに含めます。このオプションを指定しないと、デプロイメントにはデフォルトの **/usr/share/openstack-tripleo-heat-templates/network_data.yaml** ファイルが使用されます。

#### 10.5.1.2. コンポーザブルネットワークの追加

コンポーザブルネットワークを使用して、さまざまなサービス用のネットワークを追加します。たとえば、ストレージバックアップトラフィック専用のネットワークがある場合には、ネットワークを複数のロールに提供できます。

**手順**

1. デフォルトの **network_data.yaml** ファイルをコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/network_data.yaml /home/stack/.
```

2. **network_data.yaml** ファイルのローカルコピーを編集し、新規ネットワーク用のセクションを追加します。

```
- name: StorageBackup
  name_lower: storage_backup
  vlan: 21
  vip: true
  ip_subnet: '172.21.1.0/24'
  allocation_pools: [{'start': '171.21.1.4', 'end': '172.21.1.250'}]
  gateway_ip: '172.21.1.1'
```

**network_data.yaml** ファイルでは、以下のパラメーターを使用することができます。

**name**

人間が判読可能なネットワークの名前を設定します。必須のパラメーターは、このパラメーターだけです。判読性を向上させるために、**name_lower** を使用して名前を正規化することもできます。たとえば、**InternalApi** を **internal_api** に変更します。

**name_lower**

ネットワーク名の小文字バージョンを設定します。director は、この名前を **roles_data.yaml** ファイルのロールに割り当てられる該当ネットワークにマッピングします。

**vlan**

このネットワークに使用する VLAN を設定します。

**vip: true**

新規ネットワーク上に仮想 IP アドレス (VIP) を作成します。この IP は、サービス/ネットワーク間のマッピングパラメーター (**ServiceNetMap**) に一覧表示されるサービスのターゲット IP として使用されます。仮想 IP は Pacemaker を使用するロールにしか使用されない点に注意してください。オーバークラウドの負荷分散サービスにより、トラフィックがこれらの IP から対応するサービスのエンドポイントにリダイレクトされます。

**ip_subnet**

デフォルトの IPv4 サブネットを CIDR 形式で設定します。

**allocation_pools**

IPv4 サブネットの IP 範囲を設定します。

**gateway_ip**

ネットワークのゲートウェイを設定します。

**routes**

ネットワークに新たなルートを追加します。それぞれの追加ルートが含まれる JSON リストを使用します。それぞれのリスト項目には、ディクショナリーの値のマッピングが含まれます。構文例を以下に示します。

```
routes: [{'destination':'10.0.0.0/16', 'nexthop':'10.0.2.254'}]
```

**subnets**

このネットワーク内にある追加のルーティングされたサブネットを作成します。このパラメーターでは、ルーティングされたサブネット名の小文字バージョンが含まれる **dict** 値をキーとして指定し、**vlan**、**ip_subnet**、**allocation_pools**、および **gateway_ip** パラメーターをサブネットにマッピングする値として指定します。このレイアウトの例を以下に示します。

```
- name: StorageBackup
  name_lower: storage_backup
  vlan: 200
  vip: true
  ip_subnet: '172.21.0.0/24'
  allocation_pools: [{'start': '171.21.0.4', 'end': '172.21.0.250'}]
  gateway_ip: '172.21.0.1'
  subnets:
    storage_backup_leaf1:
      vlan: 201
      ip_subnet: '172.21.1.0/24'
      allocation_pools: [{'start': '171.21.1.4', 'end': '172.21.1.250'}]
      gateway_ip: '172.19.1.254'
```

このマッピングは、スパイン/リーフ型デプロイメントで一般的に使用されます。詳しくは、『スパイン/リーフ型ネットワーク』を参照してください。

**-n** オプションを使用して、カスタム **network_data.yaml** ファイルをデプロイメントコマンドに含めます。**-n** オプションを指定しないと、デプロイメントコマンドはデフォルトのネットワークセットを使用します。

#### 10.5.1.3. ロールへのコンポーザブルネットワークの追加

コンポーザブルネットワークをご自分の環境で定義したオーバークラウドロールに割り当てることができます。たとえば、カスタム **StorageBackup** ネットワークを Ceph Storage ノードに追加することができます。

**手順**

1. カスタム **roles_data.yaml** ファイルがまだない場合には、デフォルトをご自分のホームディレクトリーにコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/roles_data.yaml /home/stack/.
```

2. カスタム **roles_data.yaml** ファイルを編集します。
3. ネットワークを追加するロールの **networks** 一覧にネットワーク名を追加します。たとえば、**StorageBackup** ネットワークを Ceph Storage ロールに追加するには、以下のスニペット例を使用します。

```
- name: CephStorage
  description: |
    Ceph OSD Storage node role
  networks:
    - Storage
    - StorageMgmt
    - StorageBackup
```

4. カスタムネットワークを対応するロールに追加したら、ファイルを保存します。

**openstack overcloud deploy** コマンドを実行する際に、**-r** オプションを使用してカスタム **roles_data.yaml** ファイルを指定します。**-r** オプションを設定しないと、デプロイメントコマンドはデフォルトのロールセットとそれに対応する割り当て済みのネットワークを使用します。

#### 10.5.1.4. コンポーザブルネットワークへの OpenStack サービスの割り当て

各 OpenStack サービスは、リソースレジストリーでデフォルトのネットワーク種別に割り当てられます。これらのサービスは、そのネットワーク種別に割り当てられたネットワーク内の IP アドレスにバインドされます。OpenStack サービスはこれらのネットワークに分割されますが、実際の物理ネットワーク数はネットワーク環境ファイルに定義されている数と異なる可能性があります。環境ファイル (たとえば **/home/stack/templates/service-reassignments.yaml**) で新たにネットワークマッピングを定義することで、OpenStack サービスを異なるネットワーク種別に再割り当てすることができます。**ServiceNetMap** パラメーターにより、各サービスに使用するネットワーク種別が決定されます。

たとえば、ハイライトしたセクションを変更して、Storage Management ネットワークサービスを Storage Backup ネットワークに再割り当てすることができます。

```
parameter_defaults:
```

```
ServiceNetMap:
  SwiftMgmtNetwork: storage_backup
  CephClusterNetwork: storage_backup
```

これらのパラメーターを **storage_backup** に変更すると、対象のサービスは Storage Management ネットワークではなく、Storage Backup ネットワークに割り当てられます。つまり、**parameter_defaults** セットを設定するのは Storage Backup ネットワーク用だけで、Storage Management ネットワーク用に設定する必要はありません。

director はカスタムの **ServiceNetMap** パラメーター定義を **ServiceNetMapDefaults** から取得するデフォルト値の事前定義済みリストにマージして、デフォルト値を上書きします。director はカスタマイズされた設定を含む完全な一覧を **ServiceNetMap** に返し、その一覧は多様なサービスのネットワーク割り当ての設定に使用されます。

サービスマッピングは、Pacemaker を使用するノードの **network_data.yaml** ファイルで **vip: true** と設定されているネットワークに適用されます。オーバークラウドの負荷分散サービスにより、トラフィックが仮想 IP から特定のサービスのエンドポイントにリダイレクトされます。

**注記**

デフォルトサービスの全一覧は、**/usr/share/openstack-tripleo-heat-templates/network/service_net_map.j2.yaml** ファイルの **ServiceNetMapDefaults** パラメーターに記載されています。

#### 10.5.1.5. カスタムコンポーザブルネットワークの有効化

デフォルト NIC テンプレートの 1 つを使用してカスタムコンポーザブルネットワークを有効にします。以下の例では、VLAN が設定された単一 NIC のテンプレート (**net-single-nic-with-vlans**) を使用します。

**手順**

1. **openstack overcloud deploy** コマンドを実行する際に、以下のファイルを追加するようにしてください。
   - カスタム **network_data.yaml** ファイル
   - ネットワーク/ロール間の割り当てを定義するカスタム **roles_data.yaml** ファイル
   - デフォルトネットワーク分離のレンダリング済みファイル名
   - デフォルトネットワーク環境ファイルのレンダリング済みファイル名
   - デフォルトネットワークインターフェース設定のレンダリング済みファイル名
   - サービスの再割り当て等、ネットワークに関連するその他の環境ファイル

以下に例を示します。

```
$ openstack overcloud deploy --templates \
    ...
    -n /home/stack/network_data.yaml \
    -r /home/stack/roles_data.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
```

```
-e /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml \
-e /home/stack/templates/service-reassignments.yaml \
...
```

上記の例に示すコマンドにより、追加のカスタムネットワークを含め、コンポーザブルネットワークがオーバークラウドのノード全体にデプロイされます。

**重要**

管理ネットワーク等の新しいカスタムネットワークを導入する場合は、テンプレートを再度レンダリングする必要がある点に注意してください。ネットワーク名を **roles_data.yaml** ファイルに追加するだけでは不十分です。

#### 10.5.1.6. デフォルトネットワークの名前変更

**network_data.yaml** ファイルを使用して、デフォルトネットワークのユーザー表示名を変更することができます。

- InternalApi
- External
- Storage
- StorageMgmt
- Tenant

これらの名前を変更するのに、**name** フィールドを変更しないでください。代わりに、**name_lower** フィールドをネットワークの新しい名前に変更し、新しい名前で ServiceNetMap を更新します。

**手順**

1. **network_data.yaml** ファイルで、名前を変更する各ネットワークの **name_lower** パラメーターに新しい名前を入力します。

```
- name: InternalApi
  name_lower: MyCustomInternalApi
```

2. **service_net_map_replace** パラメーターに、**name_lower** パラメーターのデフォルト値を追加します。

```
- name: InternalApi
  name_lower: MyCustomInternalApi
  service_net_map_replace: internal_api
```

### 10.5.2. カスタムネットワークインターフェーステンプレート

「10章 *基本的なネットワーク分離*」を設定したら、実際の環境内のノードに適したカスタムネットワークインターフェーステンプレートのセットを作成することができます。たとえば、以下のファイルを含めることができます。

- ネットワーク分離を有効にするための環境ファイル (**/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml**)

- ネットワークのデフォルト値を設定するための環境ファイル (**/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml**)
- 各ノードの NIC レイアウトを定義するためのテンプレート。オーバークラウドのコアテンプレートコレクションには、さまざまなユースケースに対応する複数のデフォルトが含まれます。カスタム NIC テンプレートを作成するには、デフォルトの Jinja2 テンプレートをレンダリングしてカスタムテンプレートのベースにします。
- NIC を有効にするためのカスタム環境ファイル。以下の例では、カスタムインターフェーステンプレートを参照するカスタム環境ファイル (**/home/stack/templates/custom-network-configuration.yaml**) を用いています。
- ネットワーク設定パラメーターをカスタマイズするその他の環境ファイル
- ネットワークをカスタマイズする場合には、カスタム **network_data.yaml** ファイル
- 追加のネットワークまたはカスタムコンポーザブルネットワークを作成する場合には、カスタム **network_data.yaml** ファイルおよびカスタム **roles_data.yaml** ファイル

**注記**

前述の一覧のファイルは一部 Jinja2 形式のファイルで、**.j2.yaml** の拡張子を持つものがあります。director は、デプロイメント中にこれらのファイルを **.yaml** バージョンにレンダリングします。

#### 10.5.2.1. カスタムネットワークアーキテクチャー

デフォルトの NIC テンプレートは、特定のネットワーク構成には適しない場合があります。たとえば、特定のネットワークレイアウトに適した、専用のカスタム NIC テンプレートを作成しなければならない場合があります。また、コントロールサービスとデータサービスを異なる NIC に分離しなければならない場合があります。このような場合には、サービスから NIC への割り当てを以下のようにマッピングすることができます。

- NIC1 (プロビジョニング)
  - Provisioning / Control Plane
- NIC2 (コントロールグループ)
  - Internal API
  - Storage Management
  - External (パブリック API)
- NIC3 (データグループ)
  - Tenant Network (VXLAN トンネリング)
  - Tenant VLAN / Provider VLAN
  - Storage
  - External VLAN (Floating IP/SNAT)
- NIC4 (管理)
  - Management

#### 10.5.2.2. カスタマイズのためのデフォルトネットワークインターフェーステンプレートのレンダリング

カスタムインターフェーステンプレートの設定を簡素化するには、デフォルト NIC テンプレートの Jinja2 構文をレンダリングし、レンダリング済みのテンプレートをカスタム設定のベースとして使用します。

**手順**

1. **process-templates.py** スクリプトを使用して、**openstack-tripleo-heat-templates** コレクションのコピーをレンダリングします。

```
$ cd /usr/share/openstack-tripleo-heat-templates
$ ./tools/process-templates.py -o ~/openstack-tripleo-heat-templates-rendered
```

これにより、すべての Jinja2 テンプレートがレンタリング済みの YAML バージョンに変換され、結果が ~/**openstack-tripleo-heat-templates-rendered** に保存されます。

カスタムネットワークファイルまたはカスタムロールファイルを使用する場合には、それぞれ **-n** および **-r** オプションを使用して、それらのファイルを含めることができます。

```
$ ./tools/process-templates.py -o ~/openstack-tripleo-heat-templates-rendered -n
/home/stack/network_data.yaml -r /home/stack/roles_data.yaml
```

2. 複数 NIC の例をコピーします。

```
$ cp -r ~/openstack-tripleo-heat-templates-rendered/network/config/multiple-nics/
~/templates/custom-nics/
```

3. ご自分のネットワーク構成に適するように、**custom-nics** のテンプレートセットを編集します。

#### 10.5.2.3. ネットワークインターフェースのアーキテクチャー

「カスタマイズのためのデフォルトネットワークインターフェーステンプレートのレンダリング」でレンダリングするカスタム NIC テンプレートには、**parameters** および **resources** セクションが含まれます。

**パラメーター**

**parameters** セクションには、ネットワークインターフェース用の全ネットワーク設定パラメーターが記載されます。これには、サブネットの範囲や VLAN ID などが含まれます。heat テンプレートは、その親テンプレートから値を継承するので、このセクションは、変更せずにそのまま維持するべきです。ただし、ネットワーク環境ファイルを使用して一部のパラメーターの値を変更することが可能です。

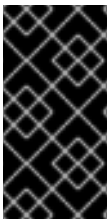
**リソース**

**resources** セクションには、ネットワークインターフェースの主要な設定を指定します。大半の場合、変更する必要があるのは **resources** セクションのみです。各 **resources** セクションは以下のヘッダーで始まります。

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
```

```
properties:
  group: script
  config:
    str_replace:
      template:
        get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
      params:
        $network_config:
          network_config:
```

このスニペットが実行するスクリプト (**run-os-net-config.sh**) により、**os-net-config** がノードのネットワーク属性を設定するのに使用する設定ファイルが作成されます。**network_config** セクションには、**run-os-net-config.sh** スクリプトに送信されるカスタムのネットワークインターフェースのデータが記載されます。このカスタムインターフェースデータは、デバイスの種別に基づいた順序で並べます。



**重要**

カスタム NIC テンプレートを作成する場合には、各 NIC テンプレートについて **run-os-net-config.sh** スクリプトの場所を絶対パスに設定する必要があります。スクリプトは、アンダークラウドの **/usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh** に保存されています。

#### 10.5.2.4. ネットワークインターフェースの参照

ネットワークインターフェースの設定には、以下のパラメーターが含まれます。

##### interface

単一のネットワークインターフェースを定義します。この設定では、実際のインターフェース名 (eth0、eth1、enp0s25) または番号によるインターフェース (nic1、nic2、nic3) を使用して各インターフェースを定義します。

```
- type: interface
  name: nic2
```

**表10.1 interface のオプション**

| オプション | デフォルト | 説明 |
|---|---|---|
| name | | インターフェース名 |
| use_dhcp | False | DHCP を使用して IP アドレスを取得します。 |
| use_dhcpv6 | False | DHCP を使用して v6 の IP アドレスを取得します。 |
| addresses | | インターフェースに割り当てられる IP アドレスの一覧 |

| オプション | デフォルト | 説明 |
|---|---|---|
| routes | | インターフェースに割り当てられるルートの一覧。詳細な情報は、「routes」を参照してください。 |
| mtu | 1500 | 接続の最大伝送単位 (MTU: Maximum Transmission Unit) |
| primary | False | プライマリーインターフェースとしてインターフェースを定義します。 |
| defroute | True | DHCP サービスにより提供されるデフォルトのルートを使用します。**use_dhcp** または **use_dhcpv6** を選択した場合に限り有効です。 |
| persist_mapping | False | システム名の代わりにデバイスのエイリアス設定を記述します。 |
| dhclient_args | なし | DHCP クライアントに渡す引数 |
| dns_servers | なし | インターフェースに使用する DNS サーバーの一覧 |
| ethtool_opts | | 特定の NIC で VXLAN を使用する際にスループットを向上させるには、このオプションを **"rx-flow-hash udp4 sdfn"** に設定します。 |

### vlan

VLAN を定義します。**parameters** セクションから渡された VLAN ID およびサブネットを使用します。

以下に例を示します。

```
- type: vlan
  vlan_id:{get_param: ExternalNetworkVlanID}
  addresses:
    - ip_netmask: {get_param: ExternalIpSubnet}
```

**表10.2 vlan のオプション**

| オプション | デフォルト | 説明 |
| --- | --- | --- |
| vlan_id | | VLAN ID |
| device | | VLAN の接続先となる親デバイス。VLAN が OVS ブリッジのメンバーではない場合に、このパラメーターを使用します。たとえば、このパラメーターを使用して、ボンディングされたインターフェースデバイスに VLAN を接続します。 |
| use_dhcp | False | DHCP を使用して IP アドレスを取得します。 |
| use_dhcpv6 | False | DHCP を使用して v6 の IP アドレスを取得します。 |
| addresses | | VLAN に割り当てられる IP アドレスの一覧 |
| routes | | VLAN に割り当てられるルートの一覧。詳細な情報は、「routes」を参照してください。 |
| mtu | 1500 | 接続の最大伝送単位 (MTU: Maximum Transmission Unit) |
| primary | False | プライマリーインターフェースとして VLAN を定義します。 |
| defroute | True | DHCP サービスにより提供されるデフォルトのルートを使用します。**use_dhcp** または **use_dhcpv6** を選択した場合に限り有効です。 |
| persist_mapping | False | システム名の代わりにデバイスのエイリアス設定を記述します。 |
| dhclient_args | なし | DHCP クライアントに渡す引数 |
| dns_servers | なし | VLAN に使用する DNS サーバーの一覧 |

### ovs_bond

Open vSwitch で、複数の **インターフェース** を結合するボンディングを定義します。これにより、冗長性や帯域幅が向上します。

以下に例を示します。

```
- type: ovs_bond
  name: bond1
  members:
  - type: interface
    name: nic2
  - type: interface
    name: nic3
```

**表10.3 ovs_bond のオプション**

| オプション | デフォルト | 説明 |
|---|---|---|
| name | | ボンディング名 |
| use_dhcp | False | DHCP を使用して IP アドレスを取得します。 |
| use_dhcpv6 | False | DHCP を使用して v6 の IP アドレスを取得します。 |
| addresses | | ボンディングに割り当てられる IP アドレスの一覧 |
| routes | | ボンディングに割り当てられるルートの一覧。詳細な情報は、「routes」を参照してください。 |
| mtu | 1500 | 接続の最大伝送単位 (MTU: Maximum Transmission Unit) |
| primary | False | プライマリーインターフェースとしてインターフェースを定義します。 |
| members | | ボンディングで使用するインターフェースオブジェクトの一覧 |
| ovs_options | | ボンディング作成時に OVS に渡すオプションのセット |
| ovs_extra | | ボンディングのネットワーク設定ファイルで OVS_EXTRA パラメーターとして設定するオプションのセット |

| オプション | デフォルト | 説明 |
| --- | --- | --- |
| defroute | True | DHCP サービスにより提供されるデフォルトのルートを使用します。**use_dhcp** または **use_dhcpv6** を選択した場合に限り有効です。 |
| persist_mapping | False | システム名の代わりにデバイスのエイリアス設定を記述します。 |
| dhclient_args | なし | DHCP クライアントに渡す引数 |
| dns_servers | なし | ボンディングに使用する DNS サーバーの一覧 |

### ovs_bridge

Open vSwitch で、複数の **interface**、**ovs_bond**、**vlan** オブジェクトを接続するブリッジを定義します。

ネットワークインターフェース種別 **ovs_bridge** には、パラメーター **name** を使用します。

**注記**

複数のブリッジがある場合は、デフォルト名の **bridge_name** を受け入れるのではなく、個別のブリッジ名を使用する必要があります。個別の名前を使用しないと、コンバージフェーズ時に 2 つのネットワークボンディングが同じブリッジに配置されます。

外部の tripleo ネットワークに OVS ブリッジを定義している場合は、**bridge_name** および **interface_name** の値を維持します。デプロイメントフレームワークが、これらの値を自動的にそれぞれ外部ブリッジ名および外部インターフェース名に置き換えるためです。

以下に例を示します。

```
- type: ovs_bridge
  name: bridge_name
  addresses:
  - ip_netmask:
      list_join:
      - /
      - - {get_param: ControlPlaneIp}
        - {get_param: ControlPlaneSubnetCidr}
  members:
    - type: interface
      name: interface_name
- type: vlan
  device: bridge_name
  vlan_id:
    {get_param: ExternalNetworkVlanID}
  addresses:
    - ip_netmask:
        {get_param: ExternalIpSubnet}
```

■

**注記**

OVS ブリッジは、Networking サービス (neutron) サーバーに接続して設定データを取得します。OpenStack の制御トラフィック (通常 Control Plane および Internal API ネットワーク) が OVS ブリッジに配置されていると、OVS がアップグレードされたり、管理ユーザーやプロセスによって OVS ブリッジが再起動されたりする度に、neutron サーバーへの接続が失われます。これにより、ダウンタイムが発生します。このような状況でダウンタイムが許容されない場合は、コントロールグループのネットワークを OVS ブリッジではなく別のインターフェースまたはボンディングに配置する必要があります。

- Internal API ネットワークをプロビジョニングインターフェース上の VLAN および 2 番目のインターフェース上の OVS ブリッジに配置すると、最小の設定を行うことができます。
- ボンディングを実装する場合は、少なくとも 2 つのボンディング (4 つのネットワークインターフェース) が必要です。コントロールグループを Linux ボンディング (Linux ブリッジ) に配置します。PXE ブート用のシングルインターフェースへの LACP フォールバックをスイッチがサポートしていない場合には、このソリューションには少なくとも 5 つの NIC が必要となります。

**表10.4 ovs_bridge のオプション**

| オプション | デフォルト | 説明 |
|---|---|---|
| name | | ブリッジ名 |
| use_dhcp | False | DHCP を使用して IP アドレスを取得します。 |
| use_dhcpv6 | False | DHCP を使用して v6 の IP アドレスを取得します。 |
| addresses | | ブリッジに割り当てられる IP アドレスの一覧 |
| routes | | ブリッジに割り当てられるルートの一覧。詳細な情報は、「routes」を参照してください。 |
| mtu | 1500 | 接続の最大伝送単位 (MTU: Maximum Transmission Unit) |
| members | | ブリッジで使用するインターフェース、VLAN、およびボンディングオブジェクトの一覧 |
| ovs_options | | ブリッジ作成時に OVS に渡すオプションのセット |

| オプション | デフォルト | 説明 |
| --- | --- | --- |
| ovs_extra | | ブリッジのネットワーク設定ファイルで OVS_EXTRA パラメーターとして設定するオプションのセット |
| defroute | True | DHCP サービスにより提供されるデフォルトのルートを使用します。**use_dhcp** または **use_dhcpv6** を選択した場合に限り有効です。 |
| persist_mapping | False | システム名の代わりにデバイスのエイリアス設定を記述します。 |
| dhclient_args | なし | DHCP クライアントに渡す引数 |
| dns_servers | なし | ブリッジに使用する DNS サーバーの一覧 |

### linux_bond

複数の **インターフェース** を結合する Linux ボンディングを定義します。これにより、冗長性や帯域幅が向上します。**bonding_options** パラメーターには、カーネルベースのボンディングオプションを指定するようにしてください。

以下に例を示します。

```
- type: linux_bond
  name: bond1
  members:
  - type: interface
    name: nic2
    primary: true
  - type: interface
    name: nic3
  bonding_options: "mode=802.3ad"
```

ボンディングが **nic2** の MAC アドレスを使用するように、 **nic2** には **primary: true** が設定される点に注意してください。

**表10.5 linux_bond のオプション**

| オプション | デフォルト | 説明 |
| --- | --- | --- |
| name | | ボンディング名 |

| オプション | デフォルト | 説明 |
|---|---|---|
| use_dhcp | False | DHCP を使用して IP アドレスを取得します。 |
| use_dhcpv6 | False | DHCP を使用して v6 の IP アドレスを取得します。 |
| addresses | | ボンディングに割り当てられる IP アドレスの一覧 |
| routes | | ボンディングに割り当てられるルートの一覧。「routes」を参照してください。 |
| mtu | 1500 | 接続の最大伝送単位 (MTU: Maximum Transmission Unit) |
| primary | False | プライマリーインターフェースとしてインターフェースを定義します。 |
| members | | ボンディングで使用するインターフェースオブジェクトの一覧 |
| bonding_options | | ボンディングを作成する際のオプションのセット |
| defroute | True | DHCP サービスにより提供されるデフォルトのルートを使用します。**use_dhcp** または **use_dhcpv6** を選択した場合に限り有効です。 |
| persist_mapping | False | システム名の代わりにデバイスのエイリアス設定を記述します。 |
| dhclient_args | なし | DHCP クライアントに渡す引数 |
| dns_servers | なし | ボンディングに使用する DNS サーバーの一覧 |

### linux_bridge

複数の **interface**、**linux_bond**、**vlan** オブジェクトを接続する Linux ブリッジを定義します。外部のブリッジは、パラメーターに 2 つの特殊な値も使用します。

- **bridge_name**: 外部ブリッジ名に置き換えます。
- **interface_name**: 外部インターフェースに置き換えます。

以下に例を示します。

```
- type: linux_bridge
  name: bridge_name
  addresses:
    - ip_netmask:
        list_join:
          - /
          - - {get_param: ControlPlaneIp}
            - {get_param: ControlPlaneSubnetCidr}
  members:
    - type: interface
      name: interface_name
- type: vlan
  device: bridge_name
  vlan_id:
    {get_param: ExternalNetworkVlanID}
  addresses:
    - ip_netmask:
        {get_param: ExternalIpSubnet}
```

**表10.6 linux_bridge のオプション**

| オプション | デフォルト | 説明 |
|---|---|---|
| name | | ブリッジ名 |
| use_dhcp | False | DHCP を使用して IP アドレスを取得します。 |
| use_dhcpv6 | False | DHCP を使用して v6 の IP アドレスを取得します。 |
| addresses | | ブリッジに割り当てられる IP アドレスの一覧 |
| routes | | ブリッジに割り当てられるルートの一覧。詳細な情報は、「routes」を参照してください。 |
| mtu | 1500 | 接続の最大伝送単位 (MTU: Maximum Transmission Unit) |
| members | | ブリッジで使用するインターフェース、VLAN、およびボンディングオブジェクトの一覧 |
| defroute | True | DHCP サービスにより提供されるデフォルトのルートを使用します。**use_dhcp** または **use_dhcpv6** を選択した場合に限り有効です。 |

| オプション | デフォルト | 説明 |
| --- | --- | --- |
| persist_mapping | False | システム名の代わりにデバイスのエイリアス設定を記述します。 |
| dhclient_args | なし | DHCP クライアントに渡す引数 |
| dns_servers | なし | ブリッジに使用する DNS サーバーの一覧 |

**routes**

ネットワークインターフェース、VLAN、ブリッジ、またはボンディングに適用するルートの一覧を定義します。

以下に例を示します。

```
- type: interface
  name: nic2
  ...
  routes:
    - ip_netmask: 10.1.2.0/24
      default: true
      next_hop:
        get_param: EC2MetadataIp
```

| オプション | デフォルト | 説明 |
| --- | --- | --- |
| ip_netmask | なし | 接続先ネットワークの IP およびネットマスク |
| default | False | このルートをデフォルトルートに設定します。**ip_netmask: 0.0.0.0/0** の設定と等価です。 |
| next_hop | なし | 接続先ネットワークに到達するのに使用するルーターの IP アドレス |

### 10.5.2.5. ネットワークインターフェースレイアウトの例

以下のスニペットはコントローラーノードの NIC テンプレートの例で、コントロールグループを OVS ブリッジから分離するカスタムネットワークシナリオの設定方法を示しています。

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
```

```
str_replace:
  template:
    get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
  params:
    $network_config:
      network_config:

      # NIC 1 - Provisioning
      - type: interface
        name: nic1
        use_dhcp: false
        addresses:
        - ip_netmask:
            list_join:
            - /
            - - get_param: ControlPlaneIp
              - get_param: ControlPlaneSubnetCidr
        routes:
        - ip_netmask: 169.254.169.254/32
          next_hop:
            get_param: EC2MetadataIp

      # NIC 2 - Control Group
      - type: interface
        name: nic2
        use_dhcp: false
      - type: vlan
        device: nic2
        vlan_id:
          get_param: InternalApiNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: InternalApiIpSubnet
      - type: vlan
        device: nic2
        vlan_id:
          get_param: StorageMgmtNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: StorageMgmtIpSubnet
      - type: vlan
        device: nic2
        vlan_id:
          get_param: ExternalNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: ExternalIpSubnet
        routes:
        - default: true
          next_hop:
            get_param: ExternalInterfaceDefaultRoute

      # NIC 3 - Data Group
      - type: ovs_bridge
        name: bridge_name
        dns_servers:
```

```
          get_param: DnsServers
        members:
        - type: interface
          name: nic3
          primary: true
        - type: vlan
          vlan_id:
            get_param: StorageNetworkVlanID
          addresses:
          - ip_netmask:
              get_param: StorageIpSubnet
        - type: vlan
          vlan_id:
            get_param: TenantNetworkVlanID
          addresses:
          - ip_netmask:
              get_param: TenantIpSubnet

        # NIC 4 - Management
        - type: interface
          name: nic4
          use_dhcp: false
          addresses:
          - ip_netmask: {get_param: ManagementIpSubnet}
          routes:
          - default: true
            next_hop: {get_param: ManagementInterfaceDefaultRoute}
```

このテンプレートは、4 つのネットワークインターフェースを使用し、タグ付けされた複数の VLAN デバイスを、番号によるインターフェース (**nic1** から **nic4**) に割り当てます。**nic3** では、このテンプレートは Storage ネットワークおよび Tenant ネットワークをホストする OVS ブリッジを作成します。その結果、以下のレイアウトが作成されます。

- NIC1 (プロビジョニング)
  - Provisioning / Control Plane
- NIC2 (コントロールグループ)
  - Internal API
  - Storage Management
  - External (パブリック API)
- NIC3 (データグループ)
  - Tenant Network (VXLAN トンネリング)
  - Tenant VLAN / Provider VLAN
  - Storage
  - External VLAN (Floating IP/SNAT)
- NIC4 (管理)
  - Management

#### 10.5.2.6. カスタムネットワークにおけるネットワークインターフェーステンプレートの考慮事項

コンポーザブルネットワークを使用する場合には、**process-templates.py** スクリプトによりレンダリングされる静的テンプレートに、**network_data.yaml** および **roles_data.yaml** ファイルで定義するネットワークおよびロールが含まれます。レンダリングされた NIC テンプレートに以下の項目が含まれるようにします。

- カスタムコンポーザブルネットワークを含む、各ロールの静的ファイル
- 各ロールの静的ファイルの正しいネットワーク定義

カスタムネットワークがロールで使用されなくても、各静的ファイルにはすべてのカスタムネットワークの全パラメーター定義が必要です。レンダリングされたテンプレートにこれらのパラメーターが含まれるようにします。たとえば、**StorageBackup** ネットワークを Ceph ノードだけに追加する場合、全ロールの NIC 設定テンプレートの **parameters** セクションにもこの定義を含める必要があります。

```
parameters:
  ...
  StorageBackupIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  ...
```

必要な場合には、VLAN ID とゲートウェイ IP の **parameters** 定義を含めることもできます。

```
parameters:
  ...
  StorageBackupNetworkVlanID:
    default: 60
    description: Vlan ID for the management network traffic.
    type: number
  StorageBackupDefaultRoute:
   description: The default route of the storage backup network.
   type: string
  ...
```

カスタムネットワーク用の **IpSubnet** パラメーターは、各ロールのパラメーター定義に含まれています。ただし、Ceph ロールは **StorageBackup** ネットワークを使用する唯一のロールなので、Ceph ロールの NIC 設定テンプレートのみがそのテンプレートの **network_config** セクションの **StorageBackup** パラメーターを使用することになります。

```
    $network_config:
      network_config:
      - type: interface
        name: nic1
        use_dhcp: false
        addresses:
        - ip_netmask:
            get_param: StorageBackupIpSubnet
```

#### 10.5.2.7. カスタムネットワーク環境ファイル

カスタムネットワーク環境ファイル (ここでは、**/home/stack/templates/custom-network-**

**configuration.yaml**) は heat の環境ファイルで、オーバークラウドのネットワーク環境を記述し、カスタムネットワークインターフェース設定テンプレートを参照します。IP アドレス範囲と合わせてネットワークのサブネットおよび VLAN を定義します。また、これらの値をローカルの環境用にカスタマイズします。

**resource_registry** のセクションには、各ノードロールのカスタムネットワークインターフェーステンプレートへの参照が含まれます。登録された各リソースには、以下の形式を使用します。

- **OS::TripleO::[ROLE]::Net::SoftwareConfig: [FILE]**

**[ROLE]** はロール名で、**[FILE]** はその特定のロールに対応するネットワークインターフェーステンプレートです。以下に例を示します。

```
resource_registry:
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/custom-nics/controller.yaml
```

**parameter_defaults** セクションには、各ネットワーク種別のネットワークオプションを定義するパラメーター一覧が含まれます。

#### 10.5.2.8. ネットワーク環境パラメーター

以下の表は、ネットワーク環境ファイルの **parameter_defaults** セクションで使用することのできるパラメーターをまとめたものです。これらのパラメーターで、ご自分の NIC テンプレートのデフォルトパラメーターの値を上書きします。

| パラメーター | 説明 | タイプ |
|---|---|---|
| **ControlPlaneDefaultRoute** | コントロールプレーン上のルーターの IP アドレスで、コントローラーノード以外のロールのデフォルトルートとして使用されます。ルーターの代わりに IP マスカレードを使用する場合には、この値をアンダークラウドの IP に設定します。 | 文字列 |
| **ControlPlaneSubnetCidr** | コントロールプレーン上で使用される IP ネットワークの CIDR ネットマスク。コントロールプレーンのネットワークが 192.168.24.0/24 を使用する場合には、CIDR は **24** になります。 | 文字列 (ただし、実際には数値) |
| ***NetCidr** | 特定のネットワークの完全なネットワークおよび CIDR ネットマスク。このパラメーターのデフォルト値は、**network_data.yaml** ファイルで規定するネットワークの **ip_subnet** に自動的に設定されます。例: **InternalApiNetCidr: 172.16.0.0/24** | 文字列 |

| パラメーター | 説明 | タイプ |
| --- | --- | --- |
| ***AllocationPools** | 特定のネットワークに対する IP 割り当て範囲。このパラメーターのデフォルト値は、**network_data.yaml** ファイルで規定するネットワークの **allocation_pools** に自動的に設定されます。例: **InternalApiAllocationPools: [{'start': '172.16.0.10', 'end': '172.16.0.200'}]** | ハッシュ |
| ***NetworkVlanID** | 特定ネットワーク上のノードの VLAN ID。このパラメーターのデフォルト値は、**network_data.yaml** ファイルで規定するネットワークの **vlan** に自動的に設定されます。例: **InternalApiNetworkVlanID: 201** | 数値 |
| ***InterfaceDefaultRoute** | 特定のネットワークのルーターアドレスで、ロールのデフォルトルートとして、あるいは他のネットワークへのルートに使用することができます。このパラメーターのデフォルト値は、**network_data.yaml** ファイルで規定するネットワークの **gateway_ip** に自動的に設定されます。例: **InternalApiInterfaceDefaultRoute: 172.16.0.1** | 文字列 |
| **DnsServers** | resolv.conf に追加する DNS サーバーの一覧。通常は、最大で 2 つのサーバーが許可されます。 | コンマ区切りリスト |
| **EC2MetadataIp** | オーバークラウドノードのプロビジョニングに使用されるメタデータサーバーの IP アドレス。この値をコントロールプレーン上のアンダークラウドの IP アドレスに設定します。 | 文字列 |
| **BondInterfaceOvsOptions** | ボンディングインターフェースのオプション (例: **BondInterfaceOvsOptions: "bond_mode=balance-slb"**) | 文字列 |

| パラメーター | 説明 | タイプ |
| --- | --- | --- |
| **NeutronExternalNetworkBridge** | OpenStack Networking (neutron) に使用する外部ブリッジ名のレガシー値。この値はデフォルトでは空欄になっています。したがって、**NeutronBridgeMappings** で複数の物理ブリッジを定義することができます。通常は、この値をオーバーライドしないでください。 | 文字列 |
| **NeutronFlatNetworks** | neutron プラグインで設定するフラットネットワークを定義します。外部ネットワークを作成できるように、デフォルト値は **datacentre** に設定されています (例: **NeutronFlatNetworks: "datacentre"**)。 | 文字列 |
| **NeutronBridgeMappings** | 使用する論理ブリッジから物理ブリッジへのマッピング。デフォルト値は、ホストの外部ブリッジ (**br-ex**) を物理名 (**datacentre**) にマッピングします。OpenStack Networking (neutron) プロバイダーネットワークまたは Floating IP ネットワークを作成する際に、論理名を参照します。例: **NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"** | 文字列 |
| **NeutronPublicInterface** | ネットワーク分離を使用しない場合に、ネットワークノード向けに **br-ex** にブリッジするインターフェースを定義します。通常、ネットワークを 2 つしか持たない小規模なデプロイメント以外では使用しません。例: **NeutronPublicInterface: "eth0"** | 文字列 |

| パラメーター | 説明 | タイプ |
| --- | --- | --- |
| **NeutronNetworkType** | OpenStack Networking (neutron) のテナントネットワーク種別。複数の値を指定するには、コンマ区切りリストを使用します。利用可能なネットワークがすべてなくなるまで、最初に指定した種別が使用されます。その後、次の種別が使用されます。例: **NeutronNetworkType: "vxlan"**。デフォルトの ML2 メカニズムドライバーである ML2/OVN メカニズムドライバーでは、vxlan はサポートされない点に注意してください。 | 文字列 |
| **NeutronTunnelTypes** | neutron テナントネットワークのトンネリング種別。複数の値を指定するには、コンマ区切りの文字列を使用します (例: **NeutronTunnelTypes: 'gre,vxlan'**)。デフォルトの ML2 メカニズムドライバーである ML2/OVN メカニズムドライバーでは、vxlan はサポートされない点に注意してください。 | 文字列 / コンマ区切りリスト |
| **NeutronTunnelIdRanges** | テナントネットワークの割り当てに使用可能にする GRE トンネリングの ID 範囲 (例: **NeutronTunnelIdRanges "1:1000"**) | 文字列 |
| **NeutronVniRanges** | テナントネットワークの割り当てに使用可能にする VXLAN VNI の ID 範囲 (例: **NeutronVniRanges: "1:1000"**) | 文字列 |
| **NeutronEnableTunnelling** | すべてのトンネル化ネットワークを有効にするか完全に無効にするかを定義します。トンネル化ネットワークを作成する予定がないことが明確な場合を除き、このパラメーターは有効のままにしてください。デフォルト値は **true** です。 | ブール値 |

| パラメーター | 説明 | タイプ |
| --- | --- | --- |
| **NeutronNetworkVLANRanges** | サポートする ML2 および Open vSwitch VLAN マッピングの範囲。デフォルトでは、物理ネットワーク **datacentre** 上の任意の VLAN を許可するように設定されています。複数の値を指定するには、コンマ区切りリストを使用します (例: **NeutronNetworkVLANRanges: "datacentre:1:1000,tenant:100:299,tenant:310:399"**)。 | 文字列 |
| **NeutronMechanismDrivers** | neutron テナントネットワークのメカニズムドライバー。デフォルト値は **ovn** です。複数の値を指定するには、コンマ区切りの文字列を使用します (例: **NeutronMechanismDrivers: 'openvswitch,l2population'**)。 | 文字列 / コンマ区切りリスト |

#### 10.5.2.9. カスタムネットワーク環境ファイルの例

NIC テンプレートを有効にしカスタムパラメーターを設定するための環境ファイルの例を、以下のスニペットに示します。

```
resource_registry:
  OS::TripleO::BlockStorage::Net::SoftwareConfig:
    /home/stack/templates/nic-configs/cinder-storage.yaml
  OS::TripleO::Compute::Net::SoftwareConfig:
    /home/stack/templates/nic-configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig:
    /home/stack/templates/nic-configs/controller.yaml
  OS::TripleO::ObjectStorage::Net::SoftwareConfig:
    /home/stack/templates/nic-configs/swift-storage.yaml
  OS::TripleO::CephStorage::Net::SoftwareConfig:
    /home/stack/templates/nic-configs/ceph-storage.yaml

parameter_defaults:
  # Gateway router for the provisioning network (or Undercloud IP)
  ControlPlaneDefaultRoute: 192.0.2.254
  # The IP address of the EC2 metadata server. Generally the IP of the Undercloud
  EC2MetadataIp: 192.0.2.1
  # Define the DNS servers (maximum 2) for the overcloud nodes
  DnsServers: ["8.8.8.8","8.8.4.4"]
  NeutronExternalNetworkBridge: """
```

#### 10.5.2.10. カスタム NIC を使用したネットワーク分離の有効化

ネットワーク分離およびカスタム NIC テンプレートを使用してオーバークラウドをデプロイするには、オーバークラウドのデプロイメントコマンドに該当するすべてのネットワーク環境ファイルを追加します。

**手順**

1. **openstack overcloud deploy** コマンドを実行する際に、以下のファイルを追加します。
   - カスタム **network_data.yaml** ファイル
   - デフォルトネットワーク分離のレンダリング済みファイル名
   - デフォルトネットワーク環境ファイルのレンダリング済みファイル名
   - カスタム NIC テンプレートへのリソースの参照を含むカスタム環境ネットワーク設定
   - 設定に必要なその他の環境ファイル

以下に例を示します。

```
$ openstack overcloud deploy --templates \
  ...
  -n /home/stack/network_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
  -e /home/stack/templates/custom-network-configuration.yaml \
  ...
```

- まず **network-isolation.yaml** ファイルを指定し、次に **network-environment.yaml** ファイルを指定します。それに続く **custom-network-configuration.yaml** は、前の 2 つのファイルからの **OS::TripleO::[ROLE]::Net::SoftwareConfig** リソースを上書きします。
- コンポーザブルネットワークを使用する場合には、このコマンドに **network_data.yaml** および **roles_data.yaml** ファイルを含めます。

## 10.5.3. その他のネットワーク設定

本章では、「「カスタムネットワークインターフェーステンプレート」」で説明した概念および手順に続いて、オーバークラウドネットワークの要素を設定する際に役立つその他の情報を提供します。

### 10.5.3.1. カスタムインターフェースの設定

インターフェースは個別に変更を加える必要がある場合があります。以下の例では、DHCP アドレスを使用してインフラストラクチャーネットワークに接続するための 2 つ目の NIC およびボンディング用の 3 つ目/4 つ目の NIC を使用するのに必要となる変更を紹介します。

```
network_config:
  # Add a DHCP infrastructure network to nic2
  - type: interface
    name: nic2
    use_dhcp: true
  - type: ovs_bridge
    name: br-bond
    members:
      - type: ovs_bond
        name: bond1
```

```
      ovs_options:
        get_param: BondInterfaceOvsOptions
      members:
        # Modify bond NICs to use nic3 and nic4
        - type: interface
          name: nic3
          primary: true
        - type: interface
          name: nic4
```

ネットワークインターフェースのテンプレートは、実際のインターフェース名 (**eth0**、**eth1**、**enp0s25**) または番号によるインターフェース (**nic1**、**nic2**、**nic3**) のいずれかを使用します。名前によるインターフェース (**eth0**、**eno2** など) ではなく、番号によるインターフェース (**nic1**、**nic2** など) を使用した場合には、ロール内のホストのネットワークインターフェースは、全く同じである必要はありません。たとえば、あるホストに **em1** と **em2** のインターフェースが指定されており、別のホストには **eno1** と **eno2** が指定されていても、両ホストの NIC は **nic1** および **nic2** として参照することができます。

番号によるインターフェースの順序は、名前によるネットワークインターフェース種別の順序に対応します。

- **eth0**、**eth1** などの **ethX**。これらは、通常オンボードのインターフェースです。
- **eno0**、**eno1** などの **enoX**。これらは、通常オンボードのインターフェースです。
- **enp3s0**、**enp3s1**、**ens3** などの英数字順の **enX** インターフェース。これらは、通常アドオンのインターフェースです。

番号による NIC スキームには、アクティブなインターフェースだけが含まれます (たとえば、インターフェースにスイッチに接続されたケーブルがあるかどうかが考慮されます)。4 つのインターフェースを持つホストと、6 つのインターフェースを持つホストがある場合は、**nic1** から **nic4** を使用して各ホストで 4 本のケーブルだけを結線します。

**nic1**、**nic2** ・・・としてマッピングする物理 NIC を事前に定義できるように、物理インターフェースを特定のエイリアスにハードコーディングすることができます。また、MAC アドレスを指定したエイリアスにマッピングすることもできます。

**注記**

通常、**os-net-config** はすでに接続済みの **UP** 状態のインターフェースしか登録しません。ただし、カスタムマッピングファイルを使用してインターフェースをハードコーディングすると、**DOWN** 状態のインターフェースであっても登録されます。

インターフェースは、環境ファイルを使用してエイリアスにマッピングされます。以下の例では、各ノードの **nic1** および **nic2** にエントリーが事前定義されます。

```
parameter_defaults:
  NetConfigDataLookup:
    node1:
      nic1: "em1"
      nic2: "em2"
    node2:
      nic1: "00:50:56:2F:9F:2E"
      nic2: "em2"
```

得られた設定は、**os-net-config** により適用されます。それぞれノードで、適用された設定が **/etc/os-net-config/mapping.yaml** ファイルの **interface_mapping** セクションに表示されます。

注記

**NetConfigDataLookup** パラメーターは、事前にプロビジョニングされたノードへのデプロイメント時に適用されません。事前にプロビジョニングされたノードでカスタムインターフェースマッピングを使用する場合は、デプロイメントの前に各ノードに **/etc/os-net-config/mapping.yaml** ファイルを作成する必要があります。以下に示す **/etc/os-net-config/mapping.yaml** ファイルのインターフェースマッピングの例を使用します。

```
interface_mapping:
  nic1: em1
  nic2: em2
```

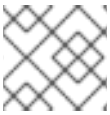
#### 10.5.3.2. ルートおよびデフォルトルートの設定

ホストのデフォルトルートは、2 つの方法のどちらかで設定することができます。インターフェースが DHCP を使用しており、DHCP サーバーがゲートウェイアドレスを提供している場合には、システムはそのゲートウェイのデフォルトルートを使用します。それ以外の場合には、固定 IP が指定されたインターフェースにデフォルトのルートを設定することができます。

Linux カーネルは複数のデフォルトゲートウェイをサポートしますが、最も低いメトリックのゲートウェイだけを使用します。複数の DHCP インターフェースがある場合には、どのデフォルトゲートウェイが使用されるかが推測できなくなります。このような場合には、デフォルトルートを使用しないインターフェースに **defroute: false** を設定することを推奨します。

たとえば、DHCP インターフェース (**nic3**) をデフォルトのルートに指定する場合があります。そのためには、以下の YAML スニペットを使用して別の DHCP インターフェース (**nic2**) 上のデフォルトルートを無効にします。

```
# No default route on this DHCP interface
- type: interface
  name: nic2
  use_dhcp: true
  defroute: false
# Instead use this DHCP interface as the default route
- type: interface
  name: nic3
  use_dhcp: true
```



注記

**defroute** パラメーターは、DHCP で取得したルートにのみ適用されます。

固定 IP が指定されたインターフェースに静的なルートを設定するには、サブネットへのルートを指定します。たとえば、Internal API ネットワーク上のゲートウェイ 172.17.0.1 を経由してサブネット 10.1.2.0/24 へのルートを設定します。

```
- type: vlan
  device: bond1
  vlan_id:
    get_param: InternalApiNetworkVlanID
```

```
addresses:
- ip_netmask:
    get_param: InternalApiIpSubnet
routes:
- ip_netmask: 10.1.2.0/24
  next_hop: 172.17.0.1
```

#### 10.5.3.3. ポリシーベースのルーティングの設定

**重要**

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能で、実稼働環境にデプロイすべきではありません。テクノロジープレビュー機能についての詳しい情報は、「対象範囲の詳細」を参照してください。

コントローラーノードで、異なるネットワークからの無制限のアクセスを設定するには、ポリシーベースのルーティングを設定します。複数のインターフェースを持つホストでは、ポリシーベースのルーティングはルーティングテーブルを使用し、送信元のアドレスに応じて特定のインターフェース経由でトラフィックを送信することができます。送信先が同じであっても、異なる送信元からのパケットを異なるネットワークにルーティングすることができます。

たとえば、デフォルトのルートが External ネットワークの場合でも、パケットの送信元アドレスに基づいてトラフィックを Internal API ネットワークに送信するようにルートを設定することができます。インターフェースごとに特定のルーティングルールを定義することもできます。

Red Hat OpenStack Platform では **os-net-config** ツールを使用してオーバークラウドノードのネットワーク属性を設定します。**os-net-config** ツールは、コントローラーノードの以下のネットワークルーティングを管理します。

- **/etc/iproute2/rt_tables** ファイルのルーティングテーブル
- **/etc/sysconfig/network-scripts/rule-{ifname}** ファイルの IPv4 ルール
- **/etc/sysconfig/network-scripts/rule6-{ifname}** ファイルの IPv6 ルール
- **/etc/sysconfig/network-scripts/route-{ifname}** のルーティングテーブル固有のルート

**前提条件**

- アンダークラウドが正常にインストールされていること。詳しい情報は、**『director のインストールと使用方法』**の「director のインストール」を参照してください。
- **openstack-tripleo-heat-templates** ディレクトリーからのデフォルトの **.j2** ネットワークインターフェーステンプレートをレンダリングしていること。詳細は、「カスタマイズのためのデフォルトネットワークインターフェーステンプレートのレンダリング」を参照してください。

**手順**

1. **~/templates/custom-nics** ディレクトリーからのカスタム NIC テンプレートに **route_table** および **interface** エントリーを作成し、インターフェースのルートを定義し、デプロイメントに関連するルールを定義します。

```
$network_config:
```

```
network_config:

- type: route_table
  name: <custom>
  table_id: 200

- type: interface
  name: em1
  use_dhcp: false
  addresses:
    - ip_netmask: {get_param: ExternalIpSubnet}
  routes:
    - ip_netmask: 10.1.3.0/24
      next_hop: {get_param: ExternalInterfaceDefaultRoute}
      table: 200
  rules:
    - rule: "iif em1 table 200"
      comment: "Route incoming traffic to em1 with table 200"
    - rule: "from 192.0.2.0/24 table 200"
      comment: "Route all traffic from 192.0.2.0/24 with table 200"
    - rule: "add blackhole from 172.19.40.0/24 table 200"
    - rule: "add unreachable iif em1 from 192.168.1.0/24"
```

2. **run-os-net-config.sh** スクリプトの場所を、作成する各カスタム NIC テンプレートの絶対パスに設定します。スクリプトは、アンダークラウドの **/usr/share/openstack-tripleo-heat-templates/network/scripts/** ディレクトリーにあります。

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-
config.sh
```

3. ご自分のデプロイメントに該当するその他の環境ファイルと共に、カスタム NIC 設定およびネットワーク環境ファイルをデプロイメントコマンドに追加します。

```
$ openstack overcloud deploy --templates \
-e ~/templates/<custom-nic-template>
-e <OTHER_ENVIRONMENT_FILES>
```

**検証手順**

- コントローラーノードで以下のコマンドを入力して、ルーティング設定が正しく機能していることを確認します。

```
$ cat /etc/iproute2/rt_tables
$ ip route
$ ip rule
```

#### 10.5.3.4. ジャンボフレームの設定

最大伝送単位 (MTU) の設定は、単一の Ethernet フレームで転送されるデータの最大量を決定します。各フレームはヘッダー形式でデータを追加するため、より大きい値を指定すると、オーバーヘッドが少なくなります。デフォルト値が 1500 で、1500 より高い値を使用する場合には、ジャンボフレームをサポートするスイッチポートの設定が必要になります。大半のスイッチは、9000 以上の MTU 値をサポートしていますが、それらの多くはデフォルトで 1500 に指定されています。

VLAN の MTU は、物理インターフェースの MTU を超えることができません。ボンディングまたはインターフェースで MTU 値を含めるようにしてください。

ジャンボフレームは、Storage、Storage Management、Internal API、Tenant ネットワークのすべてにメリットをもたらします。

**警告**

ルーターは、通常レイヤー 3 の境界を超えてジャンボフレームでのデータを転送することができません。接続性の問題を回避するには、プロビジョニングインターフェース、外部インターフェース、および Floating IP インターフェースのデフォルト MTU を変更しないでください。

```
- type: ovs_bond
  name: bond1
  mtu: 9000
  ovs_options: {get_param: BondInterfaceOvsOptions}
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000

# The external interface should stay at default
- type: vlan
  device: bond1
  vlan_id:
    get_param: ExternalNetworkVlanID
  addresses:
    - ip_netmask:
        get_param: ExternalIpSubnet
  routes:
    - ip_netmask: 0.0.0.0/0
      next_hop:
        get_param: ExternalInterfaceDefaultRoute

# MTU 9000 for Internal API, Storage, and Storage Management
- type: vlan
  device: bond1
  mtu: 9000
```

```
vlan_id:
  get_param: InternalApiNetworkVlanID
addresses:
- ip_netmask:
    get_param: InternalApiIpSubnet
```

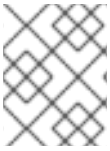
#### 10.5.3.5. ジャンボフレームを細分化するための ML2/OVN ノースバウンドパス MTU 検出の設定

内部ネットワーク上の仮想マシンがジャンボフレームを外部ネットワークに送信する場合、内部ネットワークの最大伝送単位 (MTU) が外部ネットワークの MTU より大きいと、ノースバウンドフレームが外部ネットワークの容量を容易に超過してしまいます。

ML2/OVS はこのオーバーサイズパケットの問題を自動的に処理し、ML2/OVN は TCP パケットについてこの問題を自動的に処理します。

ただし、ML2/OVN メカニズムドライバーを使用するデプロイメントで、オーバーズのノースバウンド UDP パケットを適切に処理するには、追加の設定手順を実施する必要があります。

以下の手順により、ML2/OVN ルーターが ICMP の「fragmentation needed」パケットを送信元の仮想マシンに返すように設定します。この場合、送信元アプリケーションはペイロードをより小さなパケットに分割することができます。



**注記**

East-West トラフィックでは、OVN は East-West パスの最少 MTU を超えるパケットの断片化をサポートしていません。

**前提条件**

- RHEL 8.2.0.4 以降 (kernel-4.18.0-193.20.1.el8_2 以降)

**手順**

1. カーネルバージョンを確認します。

```
ovs-appctl -t ovs-vswitchd dpif/show-dp-features br-int
```

2. 出力に **Check pkt length action: No** の文字列が含まれる場合、または **Check pkt length action** の文字列が含まれない場合は、RHEL 8.2.0.4 またはそれ以降にアップグレードしてください。あるいは、MTU がより小さい外部ネットワークにジャンボフレームを送信しないでください。

3. 出力に **Check pkt length action: Yes** の文字列が含まれる場合は、ml2_conf.ini の [ovn] セクションに以下の値を設定します。

```
ovn_emit_need_to_frag = True
```

#### 10.5.3.6. Floating IP のためのネイティブ VLAN の設定

Networking サービス (neutron) は、外部ブリッジのマッピングにデフォルトの空の文字列を使用します。これにより、物理インターフェースは **br-ex** の代わりに **br-int** を使用して直接マッピングされます。このモデルでは、複数の Floating IP ネットワークが VLAN または複数の物理接続のいずれかを使用することができます。

ネットワーク分離環境ファイルの **parameter_defaults** セクションで **NeutronExternalNetworkBridge** パラメーターを使用します。

```
parameter_defaults:
  # Set to "br-ex" when using floating IPs on the native VLAN
  NeutronExternalNetworkBridge: """"
```

ブリッジのネイティブ VLAN 上で使用する Floating IP ネットワークが 1 つのみの場合には、オプションで Neutron の外部ブリッジを設定できます。これにより、パケットが通過するブリッジは 2 つではなく 1 つだけになり、Floating IP ネットワーク上でトラフィックを渡す際の CPU の使用率がやや低くなる可能性があります。

#### 10.5.3.7. トランキングされたインターフェースでのネイティブ VLAN の設定

トランキングされたインターフェースまたはボンディングに、ネイティブ VLAN を使用したネットワークがある場合には、IP アドレスはブリッジに直接割り当てられ、VLAN インターフェースはありません。

たとえば、External ネットワークがネイティブ VLAN に存在する場合には、ボンディングの設定は以下のようになります。

```
network_config:
  - type: ovs_bridge
    name: bridge_name
    dns_servers:
      get_param: DnsServers
    addresses:
      - ip_netmask:
          get_param: ExternalIpSubnet
    routes:
      - ip_netmask: 0.0.0.0/0
        next_hop:
          get_param: ExternalInterfaceDefaultRoute
    members:
      - type: ovs_bond
        name: bond1
        ovs_options:
          get_param: BondInterfaceOvsOptions
        members:
          - type: interface
            name: nic3
            primary: true
          - type: interface
            name: nic4
```

**注記**

アドレスまたはルートのステートメントをブリッジに移動する場合は、対応する VLAN インターフェースをそのブリッジから削除します。該当する全ロールに変更を加えます。External ネットワークはコントローラーのみに存在するため、変更する必要があるのはコントローラーのテンプレートだけです。Storage ネットワークは全ロールにアタッチされているため、Storage ネットワークがデフォルトの VLAN の場合には、全ロールを変更する必要があります。

### 10.5.4. ネットワークインターフェースボンディング

カスタムネットワーク設定では、さまざまなボンディングオプションを使用することができます。

#### 10.5.4.1. ネットワークインターフェースボンディングおよび Link Aggregation Control Protocol (LACP)

複数の物理 NIC をバンドルして、単一の論理チャネルを形成することができます。この構成はボンディングとも呼ばれます。ボンディングを設定して、高可用性システム用の冗長性またはスループットの向上を実現することができます。

Red Hat OpenStack Platform では、Linux ボンディング、Open vSwitch (OVS) カーネルボンディング、および OVS-DPDK ボンディングがサポートされます。

ボンディングを、オプションの Link Aggregation Control Protocol (LACP) と共に使用することができます。LACP は動的ボンディングを作成するネゴシエーションプロトコルで、これにより負荷分散機能および耐障害性を持たせることができます。

**重要**

Red Hat では、OVS カーネルボンディング (**type: ovs_bond**) ではなく Linux カーネルボンディング (type: linux_bond) を使用することを推奨します。ユーザーモードボンディング (**type: ovs_dpdk_bond**) は、カーネルモードブリッジ (**type: ovs_bridge**) ではなくユーザーモードブリッジ (**type: ovs_user_bridge**) と共に使用します。ただし、**ovs_bridge** と **ovs_user_bridge** を同じノード上で組み合わせないでください。

コントロールネットワークおよびストレージネットワークの場合、Red Hat では VLAN を使用する Linux ボンディングを LACP と組み合わせて使用することを推奨します。OVS ボンディングを使用すると、更新、ホットフィックス等の理由により OVS または neutron エージェントが再起動すると、コントロールプレーンの中断が生じる可能性があるためです。Linux ボンディング/LACP/VLAN の構成であれば、OVS の中断を懸念することなく NIC を管理できます。

1 つの VLAN を使用する Linux ボンディングを設定するには、以下の例を使用します。

```
params:
        $network_config:
          network_config:

          - type: linux_bond
            name: bond_api
            bonding_options: "mode=active-backup"
            use_dhcp: false
            dns_servers:
              get_param: DnsServers
            members:
            - type: interface
              name: nic3
              primary: true
            - type: interface
              name: nic4

          - type: vlan
            vlan_id:
              get_param: InternalApiNetworkVlanID
            device: bond_api
```

```
            addresses:
            - ip_netmask:
                get_param: InternalApiIpSubnet
```

OVS ブリッジにプラグインした Linux ボンディングを設定するには、以下の例を使用します。

```
params:
        $network_config:
          network_config:

        -  type: ovs_bridge
           name: br-tenant
           use_dhcp: false
           mtu: 9000
           members:
            - type: linux_bond
              name: bond_tenant
              bonding_options: "mode=802.3ad updelay=1000 miimon=100"
              use_dhcp: false
              dns_servers:
                get_param: DnsServers
              members:
              - type: interface
                name: p1p1
                primary: true
              - type: interface
                name: p1p2
            - type: vlan
              device: bond_tenant
              vlan_id: {get_param: TenantNetworkVlanID}
              addresses:
                -
                  ip_netmask: {get_param: TenantIpSubnet}
```

OVS ユーザースペースブリッジを設定するには、以下の例を使用します。

```
params:
        $network_config:
          network_config:

      -    type: ovs_user_bridge
           name: br-ex
           use_dhcp: false
           members:
           - type: ovs_dpdk_bond
             name: dpdkbond0
             mtu: 2140
             ovs_options: {get_param: BondInterfaceOvsOptions}
             #ovs_extra:
             #- set interface dpdk0 mtu_request=$MTU
             #- set interface dpdk1 mtu_request=$MTU
             rx_queue:
               get_param: NumDpdkInterfaceRxQueues
             members:
             - type: ovs_dpdk_port
```

```
      name: dpdk0
      mtu: 2140
      members:
      - type: interface
        name: p1p1
    - type: ovs_dpdk_port
      name: dpdk1
      mtu: 2140
      members:
      - type: interface
        name: p1p2
```

#### 10.5.4.2. Open vSwitch ボンディングのオプション

オーバークラウドは、Open vSwtich (OVS) を介してネットワークを提供します。以下の表を使用して、ボンディングされたインターフェースに関する OVS カーネルおよび OVS-DPDK のサポート互換性について説明します。OVS/OVS-DPDK balance-tcp モードは、テクノロジープレビューとしてのみ利用可能です。

**注記**

このサポートには Open vSwitch 2.11 以降が必要です。

| OVS ボンディングモード | 用途 | 備考 | 互換性のある LACP オプション |
|---|---|---|---|
| active-backup | 高可用性 (active-passive) | | off |
| balance-slb | スループットの向上 (active-active) | • パフォーマンスは、パケットあたりの追加パース量の影響を受けます。<br>• vhost-user ロック競合が生じる可能性があります。 | active、passive、または off |

| | | | |
|---|---|---|---|
| balance-tcp (テクノロジープレビューのみ) | 推奨されない (active-active) | • L4 ハッシュに必要な再循環が、パフォーマンスに影響を及ぼします。<br>• balance-slb と同様に、パフォーマンスはパケットあたりの追加パース量の影響を受け、vhost-user ロック競合が生じる可能性があります。<br>• LACP を有効にする必要があります。 | active または passive |

ネットワーク環境ファイルの **BondInterfaceOvsOptions** パラメーターを使用して、ボンディングされたインターフェースを設定することができます。

```
parameter_defaults:
  BondInterfaceOvsOptions: "bond_mode=balance-slb"
```

#### 10.5.4.3. Linux ボンディングのオプション

ネットワークインターフェースのテンプレートにおいて、LACP を Linux ボンディングで使用することができます。

```
    - type: linux_bond
      name: bond1
      members:
      - type: interface
        name: nic2
      - type: interface
        name: nic3
      bonding_options: "mode=802.3ad lacp_rate=[fast|slow] updelay=1000 miimon=100"
```

- **mode**: LACP を有効にします。
- **lacp_rate**: LACP パケットの送信間隔を 1 秒または 30 秒に定義します。
- **updelay**: インターフェースをトラフィックに使用する前にそのインターフェースがアクティブである必要のある最低限の時間を定義します。この最小設定は、ポートフラッピングによる停止を軽減するのに役立ちます。
- **miimon**: ドライバーの MIIMON 機能を使用してポートの状態を監視する間隔 (ミリ秒単位)

#### 10.5.4.4. 一般的なボンディングオプション

以下の表には、これらのオプションについての説明と、ハードウェアに応じた代替手段を記載しています。

**表10.7 ボンディングオプション**

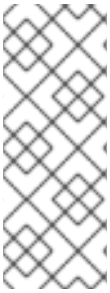
| | |
|---|---|
| **bond_mode=balance-slb** | 送信元の MAC アドレスと出力の VLAN に基づいてフローのバランスを取り、トラフィックパターンの変化に応じて定期的にリバランスを行います。ボンディングを **balance-slb** に設定することにより、リモートスイッチについての知識や協力なしに限定された形態の負荷分散が可能となります。SLB は送信元 MAC アドレスと VLAN の各ペアをリンクに割り当て、そのリンクを介して、対象の MAC アドレスと LAN からのパケットをすべて伝送します。このモードはトラフィックパターンの変化に応じて定期的にリバランスを行う、送信元 MAC アドレスと VLAN の番号に基づいた、簡単なハッシュアルゴリズムを使用します。このモードは、Linux ボンディングドライバーで使用されているモード 2 のボンディングに類似しています。このモードを使用すると、スイッチが LACP を使用するように設定されていない場合でも、負荷分散機能を有効にすることができます。 |
| **bond_mode=active-backup** | このモードは、アクティブな接続が失敗した場合にスタンバイの NIC がネットワーク操作を再開するアクティブ/スタンバイ構成のフェイルオーバーを提供します。物理スイッチに提示される MAC アドレスは 1 つのみです。このモードには、特別なスイッチのサポートや設定は必要なく、リンクが別のスイッチに接続された際に機能します。このモードは、負荷分散機能は提供しません。 |
| **lacp=[active\|passive\|off]** | Link Aggregation Control Protocol (LACP) の動作を制御します。LACP をサポートしているのは特定のスイッチのみです。お使いのスイッチが LACP に対応していない場合には **bond_mode=balance-slb** または **bond_mode=active-backup** を使用してください。 |
| **other-config:lacp-fallback-ab=true** | フォールバックとして **bond_mode=active-backup** に切り替わるように LACP の動作を設定します。 |
| **other_config:lacp-time=[fast\|slow]** | LACP のハートビートを 1 秒 (高速) または 30 秒 (低速) に設定します。デフォルトは低速です。 |
| **other_config:bond-detect-mode=[miimon\|carrier]** | リンク検出に miimon ハートビート (miimon) またはモニターキャリア (carrier) を設定します。デフォルトは carrier です。 |
| **other_config:bond-miimon-interval=100** | miimon を使用する場合には、ハートビートの間隔をミリ秒単位で設定します。 |

| | |
|---|---|
| **other_config:bond_updelay=1000** | フラッピングを防ぐためにアクティブ化してリンクが Up の状態である必要のある時間 (ミリ秒単位) |
| **other_config:bond-rebalance-interval=10000** | ボンディングメンバー間のリバランシングフローの間隔 (ミリ秒単位)。ボンディングメンバー間のリバランシングフローを無効にするには、この値をゼロに設定します。 |

### 10.5.5. ノード配置の制御

デフォルトでは、director はノードのプロファイルタグに従って、それぞれのロールのノードを無作為に選択します。ただし、特定のノード配置を定義することもできます。これは、以下のシナリオで役に立ちます。

- 特定のノード ID (例: **controller-0**、**controller-1**) を割り当てる
- カスタムのホスト名を割り当てる
- 特定の IP アドレスを割り当てる
- 特定の仮想 IP アドレスを割り当てる



**注記**

予測可能な IP アドレス、仮想 IP アドレス、ネットワークのポートを手動で設定すると、割り当てプールの必要性が軽減されます。ただし、新規ノードがスケーリングされた場合に対応できるように、各ネットワーク用の割り当てプールは維持することを推奨します。定義された固定 IP アドレスは、必ず割り当てプール外となるようにしてください。割り当てプールの設定に関する詳しい情報は、「カスタムネットワーク環境ファイル」を参照してください。

#### 10.5.5.1. 特定のノード ID の割り当て

特定のノードにノード ID を割り当てることができます (例: **controller-0**、**controller-1**、**compute-0**、および **compute-1**)。

**手順**

1. デプロイメント時に Compute スケジューラーが照合するノード別ケイパビリティーとして、この ID を割り当てます。

```
openstack baremetal node set --property capabilities='node:controller-0,boot_option:local'
<id>
```

このコマンドにより、**node:controller-0** のケイパビリティーがノードに割り当てられます。0 から始まる一意の連番のインデックスを使用して、すべてのノードに対してこのパターンを繰り返します。指定したロール (Controller、Compute、各ストレージロール) のすべてのノードが同じようにタグ付けされるようにします。このようにタグ付けしないと、Compute スケジューラーはこのケイパビリティーを正しく照合することができません。

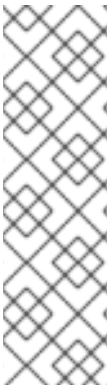
2. heat 環境ファイル (例: **scheduler_hints_env.yaml**) を作成します。このファイルは、スケジューラーヒントを使用して、各ノードのケイパビリティーと照合します。

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
```

以下のパラメーターを使用して、他のロール種別のスケジューラーヒントを設定します。

- コントローラーノードの場合: **ControllerSchedulerHints**
- コンピュートノードの場合: **ComputeSchedulerHints**
- Block Storage ノードの場合: **BlockStorageSchedulerHints**
- Object Storage ノードの場合: **ObjectStorageSchedulerHints**
- Ceph Storage ノードの場合: **CephStorageSchedulerHints**
- カスタムロールの場合: **[ROLE]SchedulerHints** (**[ROLE]** はロール名に置き換えます)

3. **overcloud deploy** コマンドに **scheduler_hints_env.yaml** 環境ファイルを追加します。



**注記**

プロファイルの照合よりもノード配置が優先されます。スケジューリングが機能しなくならないように、プロファイル照合用に設計されたフレーバー (**compute**、**control**) ではなく、デプロイメント用のデフォルトの **baremetal** フレーバーを使用します。環境ファイルで、それぞれのフレーバーパラメーターを baremetal に設定します。

```
parameter_defaults:
  OvercloudControllerFlavor: baremetal
  OvercloudComputeFlavor: baremetal
```

#### 10.5.5.2. カスタムのホスト名の割り当て

「特定のノード ID の割り当て」のノード ID の設定と組み合わせて、director は特定のカスタムホスト名を各ノードに割り当てることもできます。システムの場所 (例: **rack2-row12**) を定義する必要がある場合や、インベントリー ID を照合する必要がある場合、またはカスタムのホスト名が必要となるその他の状況において、カスタムのホスト名は便利です。

**手順**

- 「特定のノード ID の割り当て」で作成した **scheduler_hints_env.yaml** ファイル等の環境ファイルの **HostnameMap** パラメーターを使用します。

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
  ComputeSchedulerHints:
    'capabilities:node': 'compute-%index%'
  HostnameMap:
    overcloud-controller-0: overcloud-controller-prod-123-0
    overcloud-controller-1: overcloud-controller-prod-456-0
    overcloud-controller-2: overcloud-controller-prod-789-0
    overcloud-novacompute-0: overcloud-compute-prod-abc-0
```

**parameter_defaults** セクションで **HostnameMap** を定義し、それぞれのマッピングには、**HostnameFormat** パラメーターで heat が定義する元のホスト名 (例: **overcloud-controller-0**) と、2 番目の値としてそのノードに指定するカスタムのホスト名 (**overcloud-controller-prod-123-0**) を設定します。

ノード ID の配置と合わせてこの手法を使用して、各ノードにカスタムのホスト名が指定されるようにします。

#### 10.5.5.3. 予測可能な IP の割り当て

作成される環境をより細かく制御するために、director はそれぞれのネットワークにおいてオーバークラウドノードに特定の IP アドレスを割り当てることができます。

**手順**

1. 予測可能な IP アドレス設定を定義する環境ファイルを作成します。

```
$ touch ~/templates/predictive_ips.yaml
```

2. **~/templates/predictive_ips.yaml** ファイルに **parameter_defaults** セクションを作成し、以下の構文を使用してネットワークごとに各ノードの予測可能な IP アドレス設定を定義します。

```
parameter_defaults:
  <role_name>IPs:
    <network>:
    - <IP_address>
    <network>:
    - <IP_address>
```

各ノードロールには固有のパラメーターがあります。**<role_name>IPs** を該当するパラメーターに置き換えます。

- コントローラーノードの場合: **ControllerIPs**
- コンピュートノードの場合: **ComputeIPs**
- Ceph Storage ノードの場合: **CephStorageIPs**
- Block Storage ノードの場合: **BlockStorageIPs**
- Object Storage ノードの場合: **SwiftStorageIPs**
- カスタムロールの場合: **[ROLE]IPs** (**[ROLE]** はロール名に置き換えます)
  各パラメーターは、アドレスの一覧へのネットワーク名のマッピングです。各ネットワーク種別には、最低でもそのネットワークにあるノード数と同じ数のアドレスが必要です。director はアドレスを順番に割り当てます。各種別の最初のノードは、適切な一覧にある最初のアドレスが割り当てられ、2 番目のノードは 2 番目のアドレスというように割り当てられていきます。

  たとえば、予測可能な IP アドレスを持つ 3 つの Ceph Storage ノードをオーバークラウドにデプロイする場合は、以下の構文例を使用します。

```
parameter_defaults:
  CephStorageIPs:
    storage:
```

```
    - 172.16.1.100
    - 172.16.1.101
    - 172.16.1.102
    storage_mgmt:
    - 172.16.3.100
    - 172.16.3.101
    - 172.16.3.102
```

最初の Ceph Storage ノードは 172.16.1.100 と 172.16.3.100 の 2 つのアドレスを取得します。2 番目は 172.16.1.101 と 172.16.3.101、3 番目は 172.16.1.102 と 172.16.3.102 を取得します。他のノード種別でも同じパターンが適用されます。

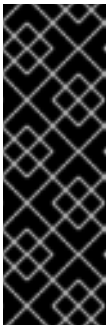
コントロールプレーンに予測可能な IP アドレスを設定するには、**/usr/share/openstack-tripleo-heat-templates/environments/ips-from-pool-ctlplane.yaml** ファイルを **stack** ユーザーの **templates** ディレクトリーにコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/ips-from-pool-
ctlplane.yaml ~/templates/.
```

以下の例に示すパラメーターで、新たな **ips-from-pool-ctlplane.yaml** ファイルを設定します。コントロールプレーンの IP アドレスの宣言に他のネットワークの IP アドレスの宣言を組み合わせ、1 つのファイルだけを使用してすべてのロールの全ネットワークの IP アドレスを宣言することができます。また、スパイン/リーフ型ネットワークに予測可能な IP アドレスを使用することもできます。それぞれのノードには、正しいサブネットからの IP アドレスを設定する必要があります。

```
parameter_defaults:
  ControllerIPs:
    ctlplane:
    - 192.168.24.10
    - 192.168.24.11
    - 192.168.24.12
    internal_api:
    - 172.16.1.20
    - 172.16.1.21
    - 172.16.1.22
    external:
    - 10.0.0.40
    - 10.0.0.57
    - 10.0.0.104
  ComputeLeaf1IPs:
    ctlplane:
    - 192.168.25.100
    - 192.168.25.101
    internal_api:
    - 172.16.2.100
    - 172.16.2.101
  ComputeLeaf2IPs:
    ctlplane:
    - 192.168.26.100
    - 192.168.26.101
    internal_api:
    - 172.16.3.100
    - 172.16.3.101
```

選択する IP アドレスは、ネットワーク環境ファイルで定義する各ネットワークの割り当てプールの範囲に入らないようにしてください (「カスタムネットワーク環境ファイル」を参照)。たとえば、**internal_api** の割り当ては **InternalApiAllocationPools** の範囲外にし、自動的に選択される IP との競合を避けるようにします。また、標準の予測可能な仮想 IP 配置 (「予測可能な仮想 IP の割り当て」を参照) または外部の負荷分散 (「外部の負荷分散機能の設定」を参照) のいずれでも、IP アドレスの割り当てが仮想 IP 設定と競合しないようにしてください。



**重要**

オーバークラウドノードが削除された場合に、そのノードのエントリーを IP の一覧から削除しないでください。IP の一覧は、下層の heat インデックスをベースとしています。このインデックスは、ノードを削除した場合でも変更されません。IP の一覧で特定のエントリーが使用されなくなったことを示すには、IP の値を **DELETED** または **UNUSED** 等に置き換えてください。エントリーは変更または追加するのみとし、IP の一覧から決して削除すべきではありません。

3. デプロイメント中にこの設定を適用するには、**openstack overcloud deploy** コマンドで **predictive_ips.yaml** 環境ファイルを指定します。

**重要**

ネットワーク分離の機能を使用する場合には、**network-isolation.yaml** ファイルの後に **predictive_ips.yaml** ファイルを追加してください。

```
$ openstack overcloud deploy --templates \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/predictive_ips.yaml \
  [OTHER OPTIONS]
```

#### 10.5.5.4. 予測可能な仮想 IP の割り当て

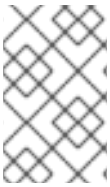
各ノードへの予測可能な IP アドレスの定義に加えて、クラスター化されたサービス向けに予測可能な仮想 IP (VIP) を定義することもできます。

**手順**

- 「カスタムネットワーク環境ファイル」で作成したネットワークの環境ファイルを編集して、**parameter_defaults** セクションに仮想 IP のパラメーターを追加します。

```
parameter_defaults:
  ...
  # Predictable VIPs
  ControlFixedIPs: [{'ip_address':'192.168.201.101'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.16.0.9'}]
  PublicVirtualFixedIPs: [{'ip_address':'10.1.1.9'}]
  StorageVirtualFixedIPs: [{'ip_address':'172.18.0.9'}]
  StorageMgmtVirtualFixedIPs: [{'ip_address':'172.19.0.9'}]
  RedisVirtualFixedIPs: [{'ip_address':'172.16.0.8'}]
  OVNDBsVirtualFixedIPs: [{'ip_address':'172.16.0.7'}]
```

それぞれの割り当てプール範囲外の IP アドレスを選択します。たとえば、**InternalApiAllocationPools** の範囲外から、**InternalApiVirtualFixedIPs** の IP アドレスを 1 つ選択します。



**注記**

このステップは、デフォルトの内部負荷分散設定を使用するオーバークラウドのみが対象です。外部のロードバランサーを使用して仮想 IP を割り当てる場合には、『External Load Balancing for the Overcloud』に記載の専用の手順を使用してください。

## 10.5.6. オーバークラウドのパブリックエンドポイントでの SSL/TLS の有効化

デフォルトでは、オーバークラウドはオーバークラウドサービスに暗号化されていないエンドポイントを使用します。オーバークラウドで SSL/TLS を有効にするには、SSL/TLS 証明書を設定し、それをオーバークラウドのデプロイメントコマンドに追加する必要があります。

**注記**

このプロセスで有効になるのは、パブリック API エンドポイント向けの SSL/TLS だけです。Internal API や Admin API は暗号化されません。

**前提条件**

- パブリック API のエンドポイントを定義するためのネットワーク分離
- **openssl-perl** パッケージがインストールされていること

### 10.5.6.1. 署名ホストの初期化

署名ホストとは、認証局を使用して新規証明書を生成し署名するホストです。選択した署名ホスト上で SSL 証明書を作成したことがない場合には、ホストを初期化して新規証明書に署名できるようにする必要がある可能性があります。

**手順**

1. すべての署名済み証明書の記録は、**/etc/pki/CA/index.txt** ファイルに含まれます。このファイルが存在しているかどうかを確認してください。存在していない場合は、必要に応じてディレクトリーのパスを作成してから、空のファイル **index.txt** を作成します。

```
$ sudo mkdir -p /etc/pki/CA
$ sudo touch /etc/pki/CA/index.txt
```

2. **/etc/pki/CA/serial** ファイルは、次に署名する証明書に使用する次のシリアル番号を特定します。このファイルが存在しているかどうかを確認してください。存在していない場合は、新規ファイル **serial** を作成して開始値を **1000** に指定します。

```
$ echo '1000' | sudo tee /etc/pki/CA/serial
```

### 10.5.6.2. 認証局の作成

通常、SSL/TLS 証明書の署名には、外部の認証局を使用します。場合によっては、独自の認証局を使用する場合もあります。たとえば、内部のみの認証局を使用するように設定する場合などです。

**手順**

**手順**

1. 鍵と証明書のペアを生成して、認証局として機能するようにします。

```
$ openssl genrsa -out ca.key.pem 4096
$ openssl req  -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -out ca.crt.pem
```

2. **openssl req** コマンドは、認証局に関する特定の情報を要求します。要求されたら、それらの情報を入力してください。

これらのコマンドにより、**ca.crt.pem** という名前の認証局ファイルが作成されます。

#### 10.5.6.3. クライアントへの認証局の追加

SSL/TLS を使用して通信する必要のある任意の外部クライアントに、認証局を追加することができます。

**手順**

1. Red Hat OpenStack Platform 環境にアクセスする必要のある各クライアントに認証局ファイルをコピーします。

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
```

2. 各クライアントに認証局ファイルをコピーしたら、それぞれのクライアントで以下のコマンドを実行し、証明書を認証局のトラストバンドルに追加します。

```
$ sudo update-ca-trust extract
```

たとえば、アンダークラウドには、作成中にオーバークラウドのエンドポイントと通信できるようにするために、認証局ファイルのコピーが必要です。

#### 10.5.6.4. SSL/TLS 鍵の作成

以下のコマンドを実行して、SSL/TLS 鍵 (**server.key.pem**) を生成します。さまざまな段階でこの鍵を使用して、アンダークラウドまたはオーバークラウドの証明書を生成します。

```
$ openssl genrsa -out server.key.pem 2048
```

#### 10.5.6.5. SSL/TLS 証明書署名要求の作成

オーバークラウドの証明書署名要求を作成します。

**手順**

1. カスタマイズを実施できるように、デフォルトの OpenSSL 設定ファイルをコピーします。

```
$ cp /etc/pki/tls/openssl.cnf .
```

2. カスタムの **openssl.cnf** ファイルを編集し、オーバークラウドに使用する SSL パラメーターを設定します。たとえば、以下のパラメーターを変更します。

```
[req]
```

```
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]
countryName = Country Name (2 letter code)
countryName_default = AU
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Queensland
localityName = Locality Name (eg, city)
localityName_default = Brisbane
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = Red Hat
commonName = Common Name
commonName_default = 10.0.0.1
commonName_max = 64

[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
IP.1 = 10.0.0.1
DNS.1 = 10.0.0.1
DNS.2 = myovercloud.example.com
```

**commonName_default** は以下のいずれか 1 つに設定します。

- SSL/TLS でアクセスするのに IP を使用する場合には、パブリック API に仮想 IP を使用します。この仮想 IP は、環境ファイルで **PublicVirtualFixedIPs** パラメーターを使用して設定します。詳細は、「予測可能な仮想 IP の割り当て」を参照してください。予測可能な仮想 IP を使用していない場合には、director は **ExternalAllocationPools** パラメーターで定義する範囲から最初の IP アドレスを割り当てます。

- SSL/TLS でアクセスするのに完全修飾ドメイン名を使用する場合には、代わりにドメイン名を使用します。
  **alt_names** セクションの IP エントリーおよび DNS エントリーとして、同じパブリック API の IP アドレスを追加します。DNS も使用する場合は、同じセクションに DNS エントリーとしてそのサーバーのホスト名を追加します。**openssl.cnf** に関する詳しい情報については、**man openssl.cnf** を実行します。

3. 以下のコマンドを実行し、証明書署名要求 (**server.csr.pem**) を生成します。

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out server.csr.pem
```

「SSL/TLS 鍵の作成」で作成した SSL/TLS 鍵を、**-key** オプションで追加するようにしてください。

次の項では、この **server.csr.pem** ファイルを使用して SSL/TLS 証明書を作成します。

#### 10.5.6.6. SSL/TLS 証明書の作成

以下のコマンドを実行し、アンダークラウドまたはオーバークラウドの証明書を作成します。

```
$ sudo openssl ca -config openssl.cnf -extensions v3_req -days 3650 -in server.csr.pem -out
server.crt.pem -cert ca.crt.pem -keyfile ca.key.pem
```

**注記**

**openssl-perl** パッケージがインストールされていない場合、このコマンドの実行に失敗し **/etc/pki/CA/newcerts: No such file or directory** というエラーメッセージが表示されます。

上記のコマンドでは、以下のオプションを使用しています。

- v3 拡張機能を指定する設定ファイル。**-config** オプションを使って設定ファイルを追加します。
- 認証局を使用して証明書を生成し署名するために「SSL/TLS 証明書署名要求の作成」で設定した証明書署名要求。**-in** オプションを使って証明書署名要求を追加します。
- 証明書への署名を行う、「認証局の作成」で作成した認証局。**-cert** オプションを使って認証局を追加します。
- 「認証局の作成」で作成した認証局の秘密鍵。**-keyfile** オプションを使って秘密鍵を追加します。

上記のコマンドにより、**server.crt.pem** という名前の新規証明書が作成されます。「SSL/TLS 鍵の作成」で作成した SSL/TLS キーと共にこの証明書を使用して、SSL/TLS を有効にします。

### 10.5.6.7. SSL/TLS の有効化

**手順**

1. heat テンプレートコレクションから **enable-tls.yaml** 環境ファイルをコピーします。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-tls.yaml
~/templates/.
```

2. このファイルを編集して、下記のパラメーターに以下の変更を加えます。

**SSLCertificate**

証明書ファイル (**server.crt.pem**) の内容を **SSLCertificate** パラメーターにコピーします。

```
parameter_defaults:
  SSLCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGS
    ...
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQ
    -----END CERTIFICATE-----
```

**重要**

この証明書の内容に新しく追加する行は、すべて同じレベルにインデントする必要があります。

SSLIntermediateCertificate

中間証明書がある場合、中間証明書のコンテンツを **SSLIntermediateCertificate** パラメーターにコピーします。

```
parameter_defaults:
  SSLIntermediateCertificate: |
    -----BEGIN CERTIFICATE-----
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvpBCwUAMFgxCzAJB
    ...
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQE
    -----END CERTIFICATE-----
```

**重要**

この証明書の内容に新しく追加する行は、すべて同じレベルにインデントする必要があります。

SSLKey

秘密鍵 (**server.key.pem**) の内容を **SSLKey** パラメーターにコピーします。

```
parameter_defaults:
  ...
  SSLKey: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0IVO
    ...
    ctlKn3rAAdyumi4JDjESAXHIKFjJNOLrBmpQyES4X
    -----END RSA PRIVATE KEY-----
```

**重要**

この秘密鍵の内容に新しく追加する行は、すべて同じレベルにインデントする必要があります。

### 10.5.6.8. ルート証明書の注入

証明書の署名者がオーバークラウドのイメージにあるデフォルトのトラストストアに含まれない場合には、オーバークラウドのイメージに認証局を注入する必要があります。

**手順**

1. heat テンプレートコレクションから **inject-trust-anchor-hiera.yaml** 環境ファイルをコピーします。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/ssl/inject-trust-anchor-hiera.yaml ~/templates/.
```

このファイルを編集して、下記のパラメーターに以下の変更を加えます。

CAMap

オーバークラウドに注入する各認証局 (CA) の内容を一覧にして定義します。オーバークラウドには、アンダークラウドとオーバークラウド両方の証明書を署名するのに使用する CA ファイルが必要です。ルート認証局ファイル (**ca.crt.pem**) の内容をエントリーにコピーします。**CAMap** パラメーターの例を以下に示します。

```
parameter_defaults:
  CAMap:
    ...
    undercloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDITCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCS
        BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBw
        UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBA
        -----END CERTIFICATE-----
    overcloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDBzCCAe+gAwIBAgIJAIc75A7FD++DMA0GCS
        BAMMD3d3dy5leGFtcGxlLmNvbTAeFw0xOTAxMz
        Um54yGCARyp3LpkxvyfMXX1DokpS1uKi7s6CkF
        -----END CERTIFICATE-----
```

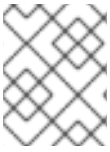
**重要**

この認証局の内容に新しく追加する行は、すべて同じレベルにインデントする必要があります。

**CAMap** パラメーターを使用して、別の CA を注入することもできます。

### 10.5.6.9. DNS エンドポイントの設定

SSL/TLS でオーバークラウドにアクセスするのに DNS ホスト名を使用する場合には、**/usr/share/tripleo-heat-templates/environments/predictable-placement/custom-domain.yaml** ファイルを **/home/stack/templates** ディレクトリーにコピーします。



**注記**

この環境ファイルが初回のデプロイメントに含まれていない場合は、TLS-everywhere のアーキテクチャーで再デプロイすることはできません。

すべてのフィールドのホスト名およびドメイン名を設定し、必要に応じてカスタムネットワークのパラメーターを追加します。

**CloudDomain**

ホストの DNS ドメイン

**CloudName**

オーバークラウドエンドポイントの DNS ホスト名

**CloudNameCtlplane**

プロビジョニングネットワークエンドポイントの DNS 名

**CloudNameInternal**

Internal API エンドポイントの DNS 名

**CloudNameStorage**

ストレージエンドポイントの DNS 名

**DnsServers**

使用する DNS サーバーの一覧。設定済みの DNS サーバーには、パブリック API の IP アドレスに一致する設定済みの **CloudName** へのエントリーが含まれていなければなりません。

**CloudNameStorageManagement**

ストレージ管理エンドポイントの DNS 名

1. 新規または既存の環境ファイルのいずれかで、パラメーターのデフォルトセクションに使用する DNS サーバーの一覧を追加します。

```
parameter_defaults:
  DnsServers: ["10.0.0.254"]
  ....
```

### 10.5.6.10. オーバークラウド作成時の環境ファイルの追加

デプロイメントコマンド **openstack overcloud deploy** で **-e** オプションを使用して、デプロイメントプロセスに環境ファイルを追加します。本項の環境ファイルは、以下の順序で追加します。

- SSL/TLS を有効化する環境ファイル (**enable-tls.yaml**)
- DNS ホスト名を設定する環境ファイル (**cloudname.yaml**)
- ルート認証局を注入する環境ファイル (**inject-trust-anchor-hiera.yaml**)
- パブリックエンドポイントのマッピングを設定するための環境ファイル:
  - パブリックエンドポイントへのアクセスに DNS 名を使用する場合には、**/usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml** を使用します。
  - パブリックエンドポイントへのアクセスに IP アドレスを使用する場合には、**/usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-ip.yaml** を使用します。

デプロイメントコマンドの例:

```
$ openstack overcloud deploy --templates \
[...]
-e /home/stack/templates/enable-tls.yaml \
-e ~/templates/cloudname.yaml \
-e ~/templates/inject-trust-anchor-hiera.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml
```

### 10.5.6.11. SSL/TLS 証明書の更新

これ以降、証明書を更新する必要がある場合:

- **enable-tls.yaml** ファイルを編集して、**SSLCertificate**、**SSLKey**、**SSLIntermediateCertificate** のパラメーターを更新してください。
- 認証局が変更された場合には、**inject-trust-anchor-hiera.yaml** ファイルを編集して、**CAMap** パラメーターを更新してください。

新しい証明書の内容が記載されたら、デプロイメントコマンドを再度実行します。

```
$ openstack overcloud deploy --templates \
[...]
-e /home/stack/templates/enable-tls.yaml \
-e ~/templates/cloudname.yaml \
-e ~/templates/inject-trust-anchor-hiera.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml
```

## 10.5.7. Identity Management を使用した内部およびパブリックエンドポイントでの SSL/TLS の有効化

特定のオーバークラウドエンドポイントで SSL/TLS を有効化することができます。多数の証明書数が必要となるため、director は Red Hat Identity Management (IdM) サーバーと統合して認証局として機能し、オーバークラウドの証明書を管理します。このプロセスでは、**novajoin** を使用してオーバークラウドノードを IdM サーバーに登録します。

OpenStack 全コンポーネントの TLS サポートのステータスを確認するには、「TLS Enablement status matrix」を参照してください。

### 10.5.7.1. CA へのアンダークラウドの追加

オーバークラウドをデプロイする前に、アンダークラウドを認証局 (CA) に追加する必要があります。

**手順**

1. アンダークラウドノードで、**python3-novajoin** パッケージをインストールします。

```
$ sudo dnf install python3-novajoin
```

2. アンダークラウドノードで **novajoin-ipa-setup** スクリプトを実行します。値はデプロイメントに応じて調整します。

```
$ sudo /usr/libexec/novajoin-ipa-setup \
    --principal admin \
    --password <IdM admin password> \
    --server <IdM server hostname> \
    --realm <overcloud cloud domain (in upper case)> \
    --domain <overcloud cloud domain> \
    --hostname <undercloud hostname> \
    --precreate
```

ここで設定したワンタイムパスワード (OTP) を使用して、アンダークラウドを登録します。

### 10.5.7.2. IdM へのアンダークラウドの追加

アンダークラウドを IdM に登録して novajoin を設定します。**undercloud.conf** ファイルの **[DEFAULT]** セクションで、以下の設定を行います。

**手順**

1. **novajoin** サービスを有効にします。

```
[DEFAULT]
enable_novajoin = true
```

2. アンダークラウドノードを IdM に登録できるように、ワンタイムパスワード (OTP) を設定します。

```
ipa_otp = <otp>
```

3. neutron の DHCP サーバーが提供するオーバークラウドのドメイン名が IdM ドメインと一致するようにします。たとえば、小文字の kerberos レルム。

```
overcloud_domain_name = <domain>
```

4. アンダークラウドに適切なホスト名を設定します。

```
undercloud_hostname = <undercloud FQDN>
```

5. アンダークラウドのネームサーバーとして IdM を設定します。

```
undercloud_nameservers = <IdM IP>
```

6. より大規模な環境の場合には、novajoin の接続タイムアウト値を見直します。**undercloud.conf** ファイルで、**undercloud-timeout.yaml** という名前の新規ファイルへの参照を追加します。

```
hieradata_override = /home/stack/undercloud-timeout.yaml
```

**undercloud-timeout.yaml** に以下のオプションを追加します。タイムアウト値は秒単位で指定することができます (例: **5**)。

```
nova::api::vendordata_dynamic_connect_timeout: <timeout value>
nova::api::vendordata_dynamic_read_timeout: <timeout value>
```

7. **undercloud.conf** ファイルを保存します。

8. アンダークラウドのデプロイコマンドを実行して、既存のアンダークラウドに変更を適用します。

```
$ openstack undercloud install
```

### 10.5.7.3. オーバークラウド DNS の設定

IdM 環境を自動検出して、登録をより簡単にするには、IdM を DNS サーバーとして使用することができます。

**手順**

**手順**

1. アンダークラウドに接続します。

```
$ source ~/stackrc
```

2. DNS ネームサーバーとして IdM を使用するようにコントロールプレーンサブネットを設定します。

```
$ openstack subnet set ctlplane-subnet --dns-nameserver  <idm_server_address>
```

3. IdM サーバーを使用するように環境ファイルの **DnsServers** パラメーターを設定します。

```
parameter_defaults:
  DnsServers: ["<idm_server_address>"]
```

このパラメーターは、通常カスタムの **network-environment.yaml** ファイルで定義されます。

#### 10.5.7.4. novajoin を使用するためのオーバークラウドの設定

**手順**

1. IdM 統合を有効化するには、**/usr/share/openstack-tripleo-heat-templates/environments/predictable-placement/custom-domain.yaml** 環境ファイルのコピーを作成します。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/predictable-
placement/custom-domain.yaml \
  /home/stack/templates/custom-domain.yaml
```

2. **/home/stack/templates/custom-domain.yaml** 環境ファイルを編集して、デプロイメントに適した **CloudDomain** と **CloudName*** の値を設定します。

```
parameter_defaults:
  CloudDomain: lab.local
  CloudName: overcloud.lab.local
  CloudNameInternal: overcloud.internalapi.lab.local
  CloudNameStorage: overcloud.storage.lab.local
  CloudNameStorageManagement: overcloud.storagemgmt.lab.local
  CloudNameCtlplane: overcloud.ctlplane.lab.local
```

3. オーバークラウドのデプロイプロセスで以下の環境ファイルを追加します。

- **/usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml**
- **/usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml**
- **/home/stack/templates/custom-domain.yaml**
  以下に例を示します。

```
openstack overcloud deploy \
  --templates \
   -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-
```

```
tls.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e /home/stack/templates/custom-domain.yaml \
```

デプロイされたオーバークラウドノードは、自動的に IdM に登録されます。

4. これで設定されるのは、内部エンドポイント向けの TLS だけです。外部エンドポイントには、**/usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-tls.yaml** 環境ファイル (カスタムの証明書と鍵を追加するために編集する必要あり) で TLS を追加する通常の方法を使用することができます。

```
openstack overcloud deploy \
 --templates \
 -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml \
 -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml \
 -e /home/stack/templates/custom-domain.yaml \
 -e /home/stack/templates/enable-tls.yaml
```

5. また、IdM を使用して公開証明書を発行することもできます。その場合には、**/usr/share/openstack-tripleo-heat-templates/environments/services/haproxy-public-tls-certmonger.yaml** 環境ファイルを使用する必要があります。

```
openstack overcloud deploy \
 --templates \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e /home/stack/templates/custom-domain.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/haproxy-public-tls-certmonger.yaml
```

**注記**

novajoin を使用して、既存のデプロイメントに TLS everywhere (TLS-e) を実装することができなくなりました。TLS-e を使用して Red Hat OpenStack Platform の既存のデプロイメントを強化する方法は、「「Ansible を使用した TLS-e の実装」」を参照してください。

### 10.5.8. Ansible を使用した TLS-e の実装

Red Hat では、アンダークラウドおよびオーバークラウドに TLS-e を設定するのに、**novajoin** を使用するデフォルトの方式よりも、新たな Ansible ベースの **tripleo-ipa** を使用する方式を推奨しています。以下の手順を使用して、Red Hat OpenStack Platform の新規インストール、または TLS-e の設定が必要な既存のデプロイメントのいずれかに、TLS-e を実装することができます。事前にプロビジョニングされたノードに TLS-e を設定した Red Hat OpenStack Platform をデプロイする場合は、この方式を使用する必要があります。

**注記**

既存の環境に TLS-e を実装する場合は、引き続き **openstack undercloud install**、**openstack overcloud deploy** コマンド等のコマンドを実行する必要があります。これらの手順はべき等性を持ち、更新されたテンプレートおよび設定ファイルと一致するように既存のデプロイメント設定を調整するだけです。

#### 10.5.8.1. アンダークラウドでの TLS-e の設定

**前提条件**

stack ユーザーの作成など、アンダークラウドの設定手順がすべて完了していること。詳細は、『director のインストールと使用方法』を参照してください。

**手順**

1. ホストファイルを設定します。
   アンダークラウドの **/etc/resolv.conf** に、適切な検索ドメインおよびネームサーバーを設定します。たとえば、デプロイメントドメインが **example.com** で FreeIPA サーバーのドメインが **bigcorp.com** の場合、以下の行を /etc/resolv.conf に追加します。

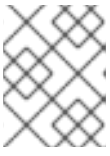
```
search example.com bigcorp.com
nameserver $IDM_SERVER_IP_ADDR
```

2. 必要なソフトウェアをインストールします。

```
sudo yum install -y python3-ipalib python3-ipaclient krb5-devel
```

3. ご自分の環境に固有の値で環境変数をエクスポートします。

```
export IPA_DOMAIN=bigcorp.com
export IPA_REALM=BIGCORP.COM
export IPA_ADMIN_USER=$IPA_USER
export IPA_ADMIN_PASSWORD=$IPA_PASSWORD
export IPA_SERVER_HOSTNAME=ipa.bigcorp.com
export UNDERCLOUD_FQDN=undercloud.example.com
export USER=stack
export CLOUD_DOMAIN=example.com
```



**注記**

IdM のユーザー認証情報は、新しいホストおよびサービスを追加できる管理ユーザーでなければなりません。

4. アンダークラウドで **undercloud-ipa-install.yaml** Ansible Playbook を実行します。

```
ansible-playbook \
--ssh-extra-args "-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null" \
/usr/share/ansible/tripleo-playbooks/undercloud-ipa-install.yaml
```

5. undercloud.conf に以下のパラメーターを追加します。

```
undercloud_nameservers = $IDM_SERVER_IP_ADDR
overcloud_domain_name = example.com
```

6. アンダークラウドをデプロイします。

```
openstack undercloud install
```

**検証**

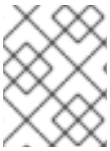
以下の手順を実施して、アンダークラウドが正しく登録されたことを確認します。

1. IdM のホストを一覧表示します。

```
$ kinit admin
$ ipa host-find
```

2. アンダークラウドに **/etc/novajoin/krb5.keytab** が存在することを確認します。

```
ls /etc/novajoin/krb5.keytab
```

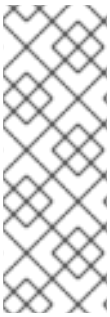


**注記**

**novajoin** というディレクトリー名は、従来の方式に対応させる目的でのみ使用されています。

#### 10.5.8.2. オーバークラウドでの TLS-e の設定

TLS everywhere (TLS-e) を設定したオーバークラウドをデプロイする場合、アンダークラウドおよびオーバークラウドの IP アドレスは自動的に IdM に登録されます。



**注記**

IP アドレスの自動登録を無効にするには、**IDMModifyDNS** heat パラメーターを false に設定します。

```
parameter_defaults:
  ....
  IdMModifyDNS: false
```

1. オーバークラウドをデプロイする前に、以下のような内容で YAML ファイル **tls-parameters.yaml** を作成します。お使いの環境に固有の値を選択してください。

   - **DnsServers** パラメーターの値は、IdM サーバーの IP アドレスを反映させる必要があります。
   - IdM サーバーのドメインがクラウドのドメインと異なる場合は、**DnsSearchDomains** パラメーターに追加します。たとえば、**DnsSearchDomains: ["example.com", "bigcorp.com"]** のように設定します。
   - 事前にプロビジョニングされたノードが無い限り、**IDMInstallClientPackages** パラメーターの値を **false** に設定する必要があります。
   - **OS::TripleO::Services::IpaClient** パラメータに示す値は、**enable-internal-tls.yaml** ファ

イルのデフォルト設定を上書きします。**openstack overcloud deploy** コマンドで、**enable-internal-tls.yaml** の後に **tls-parameters.yaml** ファイルを指定するようにします。

- アクティブ/アクティブとして設定された cinder と共に分散コンピュートノード (DCN) アーキテクチャーを実行している場合は、**EnableEtcdInternalTLS** パラメーターを追加して **true** に設定する必要があります。

```
parameter_defaults:
  DnsSearchDomains: ["example.com"]
  DnsServers: ["192.168.1.13"]
  CloudDomain: example.com
  CloudName: overcloud.example.com
  CloudNameInternal: overcloud.internalapi.example.com
  CloudNameStorage: overcloud.storage.example.com
  CloudNameStorageManagement: overcloud.storagemgmt.example.com
  CloudNameCtlplane: overcloud.ctlplane.example.com
  IdMServer: freeipa-0.redhat.local
  IdMDomain: redhat.local
  IdMInstallClientPackages: False

resource_registry:
    OS::TripleO::Services::IpaClient: /usr/share/openstack-tripleo-heat-
templates/deployment/ipa/ipaservices-baremetal-ansible.yaml
```

2. オーバークラウドをデプロイします。デプロイメントコマンドに tls-parameters.yaml を追加する必要があります。

```
DEFAULT_TEMPLATES=/usr/share/openstack-tripleo-heat-templates/
CUSTOM_TEMPLATES=/home/stack/templates

openstack overcloud deploy \
-e ${DEFAULT_TEMPLATES}/environments/ssl/tls-everywhere-endpoints-dns.yaml \
-e ${DEFAULT_TEMPLATES}/environments/services/haproxy-public-tls-certmonger.yaml \
-e ${DEFAULT_TEMPLATES}/environments/ssl/enable-internal-tls.yaml \
-e ${CUSTOM_TEMPLATES}/tls-parameters.yaml \
...
```

3. keystone にエンドポイント一覧のクエリーを行い、各エンドポイントが HTTPS を使用していることを確認します。

```
openstack overcloud endpoint list
```

### 10.5.9. デバッグモード

オーバークラウド内の特定のサービスに **DEBUG** レベルロギングモードを有効化または無効化することができます。

サービスのデバッグモードを設定するには、それぞれのデバッグパラメーターを設定します。たとえば、OpenStack Identity (keystone) は **KeystoneDebug** パラメーターを使用します。

- 環境ファイルの **parameter_defaults** セクションでパラメーターを設定します。

```
parameter_defaults:
  KeystoneDebug: True
```

**KeystoneDebug** パラメーターを **True** に設定すると、標準の keystone ログファイル **/var/log/containers/keystone/keystone.log** が **DEBUG** レベルのログで更新されます。

デバッグパラメーターの全一覧は、**『オーバークラウドのパラメーター』**の「デバッグパラメーター」を参照してください。

### 10.5.10. ポリシー

オーバークラウド内の特定のサービスに対してアクセスポリシーを設定することができます。サービスに対してポリシーを設定するには、そのサービスのポリシーが含まれるハッシュ値でそれぞれのポリシーのパラメーターを設定します。

- OpenStack Identity (keystone) には **KeystonePolicies** パラメーターを使用します。このパラメーターを環境ファイルの **parameter_defaults** セクションで設定します。

```
parameter_defaults:
  KeystonePolicies: { keystone-context_is_admin: { key: context_is_admin, value: 'role:admin'
} }
```

- OpenStack Compute (nova) には **NovaApiPolicies** パラメーターを使用します。このパラメーターを環境ファイルの **parameter_defaults** セクションで設定します。

```
parameter_defaults:
  NovaApiPolicies: { nova-context_is_admin: { key: 'compute:get_all', value: '@' } }
```

ポリシーパラメーターの全一覧は、**『オーバークラウドのパラメーター』**の「ポリシーパラメーター」を参照してください。

### 10.5.11. ストレージの設定

本章では、オーバークラウドのストレージオプションを設定するためのさまざまな方法について説明します。

**重要**

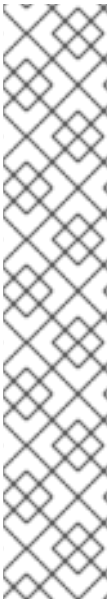
オーバークラウドは、デフォルトのストレージオプションにローカルおよび LVM のストレージを使用します。これらのオプションはエンタープライズレベルのオーバークラウドではサポートされないので、本章で説明するストレージオプションの 1 つを使用するようにオーバークラウドを設定する必要があります。

#### 10.5.11.1. NFS ストレージの設定

NFS 共有を使用するようにオーバークラウドを設定するには、コア heat テンプレートコレクションの既存の環境ファイルを変更する必要があります。

**重要**

Red Hat では、認定済みのストレージバックエンドおよびドライバーを使用することを推奨します。Red Hat では、汎用 NFS バックエンドの NFS を使用することを推奨していません。認定済みのストレージバックエンドおよびドライバーと比較すると、その機能に制限があるためです。たとえば、汎用 NFS バックエンドは、ボリュームの暗号化やボリュームのマルチアタッチなどの機能をサポートしません。サポート対象のドライバーについての情報は、Red Hat Ecosystem Catalog を参照してください。



**注記**

director のさまざまな heat パラメーターにより、NFS バックエンドまたは NetApp NFS Block Storage バックエンドが NetApp 機能 (NAS secure と呼ばれる) をサポートするかどうかが制御されます。

- CinderNetappNasSecureFileOperations
- CinderNetappNasSecureFilePermissions
- CinderNasSecureFileOperations
- CinderNasSecureFilePermissions

通常のボリューム操作に干渉するため、Red Hat では、この機能を有効にすることを推奨していません。director はデフォルトでこの機能を無効にするため、Red Hat OpenStack Platform はこの機能をサポートしません。

**注記**

Block Storage (cinder) サービスおよび Compute (nova) サービスには、NFS バージョン 4.0 以降を使用する必要があります。

コア heat テンプレートコレクションの **/usr/share/openstack-tripleo-heat-templates/environments/** には、一連の環境ファイルが格納されています。これらの環境ファイルを使用すると、director が作成したオーバークラウドにおいて、一部のサポート対象機能のカスタム設定を作成することができます。これには、ストレージを設定するための環境ファイルが含まれます。このファイルは、**/usr/share/openstack-tripleo-heat-templates/environments/storage-environment.yaml** に保管されています。

**手順**

1. ファイルを **stack** ユーザーのホームディレクトリーにあるテンプレートディレクトリーにコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/storage-environment.yaml ~/templates/.
```

2. 以下のパラメーターを変更してください。

**CinderEnableIscsiBackend**
iSCSI バックエンドを有効にするパラメーター。この値を **false** に設定します。

**CinderEnableRbdBackend**
Ceph Storage バックエンドを有効にするパラメーター。この値を **false** に設定します。

**CinderEnableNfsBackend**

NFS バックエンドを有効にするパラメーター。この値を **true** に設定します。

NovaEnableRbdBackend

Nova エフェメラルストレージ用に Ceph Storage を有効にするパラメーター。この値を **false** に設定します。

GlanceBackend

Image サービスに使用するバックエンドを定義するパラメーター。イメージ用にファイルベースのストレージを使用するには、この値を **file** に設定します。オーバークラウドは、Image サービス (glance) 用にマウントされた NFS 共有にこれらのファイルを保存します。

CinderNfsMountOptions

ボリュームストレージ用の NFS マウントオプション

CinderNfsServers

ボリュームストレージ用にマウントする NFS 共有。たとえば、192.168.122.1:/export/cinder と設定します。

GlanceNfsEnabled

**GlanceBackend** を **file** に設定した場合、**GlanceNfsEnabled** パラメーターを使用して、イメージが NFS 経由で共有の場所に保管されるのを有効にします。これにより、すべてのコントローラーノードがイメージにアクセスすることができます。**GlanceNfsEnabled** パラメーターを false に設定すると、オーバークラウドはイメージをコントローラーノードのファイルシステムに保管します。この値を **true** に設定します。

GlanceNfsShare

イメージストレージ用にマウントする NFS 共有。たとえば、192.168.122.1:/export/glance と設定します。

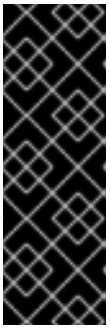
GlanceNfsOptions

イメージストレージ用の NFS マウントオプション

**storage-environment.yaml** 環境ファイルには、Red Hat OpenStack Platform (RHOSP) の Block Storage (cinder) サービスおよび Image (glance) サービス用に、さまざまなストレージオプションを設定するためのパラメーターが含まれます。以下の例で、オーバークラウドが NFS 共有を使用するように設定する方法を説明します。

```
parameter_defaults:
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: false
  CinderEnableNfsBackend: true
  NovaEnableRbdBackend: false
  GlanceBackend: file

  CinderNfsServers: 192.0.2.230:/cinder

  GlanceNfsEnabled: true
  GlanceNfsShare: 192.0.2.230:/glance
```

これらのパラメーターは、heat テンプレートコレクションの一部として統合されます。この例を使用して、使用する Block Storage サービスおよび Image サービス用に NFS マウントポイントを 2 つ作成します。



**重要**

Image サービスが **/var/lib** ディレクトリーにアクセスできるようにするには、**GlanceNfsOptions** パラメーターに **_netdev,bg,intr,context=system_u:object_r:container_file_t:s0** オプションを追加します。この SELinux コンテンツを設定しないと、Image サービスはマウントポイントに書き込みを行うことができません。同じ NFS サーバーを共有するように複数のサービスを設定する際に問題が発生した場合は、Red Hat サポートにお問い合わせください。

3. オーバークラウドをデプロイする際に、**storage-environment.yaml** 環境ファイルを追加します。

### 10.5.11.2. Ceph Storage の設定

以下のいずれかの方法を使用して、Red Hat Ceph Storage を Red Hat OpenStack Platform のオーバークラウドに統合します。

**固有の Ceph Storage Cluster を持つオーバークラウドの作成**

オーバークラウドの作成中に Ceph Storage クラスターを作成することができます。director は、データの格納に Ceph OSD を使用する Ceph Storage ノードのセットを作成します。director は、オーバークラウドのコントローラーノードに Ceph Monitor サービスもインストールします。このため、組織が 3 台の高可用性コントローラーノードで構成されるオーバークラウドを作成する場合には、Ceph Monitor も高可用性サービスになります。詳しい情報は、『コンテナー化された Red Hat Ceph を持つオーバークラウドのデプロイ』を参照してください。

**既存 Ceph Storage クラスターのオーバークラウドへの統合**

既存の Ceph Storage クラスターがある場合には、デプロイメント中にこのクラスターを Red Hat OpenStack Platform のオーバークラウドに統合することができます。これは、オーバークラウドの設定とは独立して、クラスターの管理やスケーリングが可能であることを意味します。詳しい情報は、『オーバークラウドの既存 Red Hat Ceph クラスターとの統合』 を参照してください。

### 10.5.11.3. 外部の Object Storage クラスターの使用

コントローラーノードでデフォルトの Object Storage サービスのデプロイメントを無効にすることによって、外部の OpenStack Object Storage (swift) クラスターを再利用することができます。これにより、Object Storage のプロキシーとストレージサービスの両方が無効になり、haproxy と OpenStack Identify (keystone) が特定の外部 Object Storage エンドポイントを使用するように設定されます。

**注記**

外部 Object Storage (swift) クラスター上のユーザーアカウントを手動で管理する必要があります。

**前提条件**

- 外部の Object Storage クラスターのエンドポイントの IP アドレスに加えて、外部の Object Storage クラスターの **proxy-server.conf** ファイルの **authtoken** パスワードも必要です。この情報は、**openstack endpoint list** コマンドを使用して確認することができます。

**手順**

1. 以下の内容を記載した **swift-external-params.yaml** という名前の新しいファイルを作成します。

- **EXTERNAL.IP:PORT** は、外部プロキシーの IP アドレスとポートに置き換えます。
- **SwiftPassword** の行の **AUTHTOKEN** は、外部プロキシーの **authtoken** パスワードに置き換えます。

```
parameter_defaults:
  ExternalPublicUrl: 'https://EXTERNAL.IP:PORT/v1/AUTH_%(tenant_id)s'
  ExternalInternalUrl: 'http://192.168.24.9:8080/v1/AUTH_%(tenant_id)s'
  ExternalAdminUrl: 'http://192.168.24.9:8080'
  ExternalSwiftUserTenant: 'service'
  SwiftPassword: AUTHTOKEN
```

2. このファイルを **swift-external-params.yaml** として保存します。
3. デプロイメントに該当するその他の環境ファイルと共に、以下の外部 Object Storage サービスの環境ファイルを指定して、オーバークラウドをデプロイします。

```
openstack overcloud deploy --templates \
-e [your environment files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/swift-external.yaml \
-e swift-external-params.yaml
```

### 10.5.11.4. 外部の Ceph Object Gateway を使用するための Ceph Object Store の設定

Red Hat OpenStack Platform (RHOSP) director は、外部の Ceph Object Gateway (RGW) を Object Store サービスとして設定することをサポートしています。外部 RGW サービスで認証するには、Identity サービス (keystone) のユーザーとそのロールを確認するように RGW を設定する必要があります。

外部 Ceph Object Gateway の設定方法に関する詳細は、**『Ceph Object Gateway ガイドでの Keystone の使用』**の「Keystone 認証を使用するように Ceph Object Gateway を設定」 を参照してください。

**手順**

1. カスタム環境ファイル (**swift-external-params.yaml** 等) に以下の **parameter_defaults** を追加し、実際のデプロイメントに合わせて値を調整します。

```
parameter_defaults:
  ExternalSwiftPublicUrl: 'http://<Public RGW endpoint or
loadbalancer>:8080/swift/v1/AUTH_%(project_id)s'
  ExternalSwiftInternalUrl: 'http://<Internal RGW endpoint>:8080/swift/v1/AUTH_%
(project_id)s'
  ExternalSwiftAdminUrl: 'http://<Admin RGW endpoint>:8080/swift/v1/AUTH_%(project_id)s'
  ExternalSwiftUserTenant: 'service'
  SwiftPassword: 'choose_a_random_password'
```

**注記**

サンプルコードスニペットには、お使いの環境で使用する値とは異なるパラメーター値が含まれる場合があります。

- リモート RGW インスタンスがリッスンするデフォルトのポートは **8080** です。外部 RGW の設定方法によっては、ポートが異なる場合があります。
- オーバークラウドで作成した **swift** ユーザーは、**SwiftPassword** パラメーターで定義したパスワードを使用します。**rgw_keystone_admin_password** を使用し、Identity サービスに対する認証に同じパスワードを使用するように外部 RGW インスタンスを設定する必要があります。

2. Ceph 設定ファイルに以下のコードを追加して、Identity サービスを使用するように RGW を設定します。変数の値を実際の環境に応じて調整します。

```
rgw_keystone_api_version: 3
rgw_keystone_url: http://<public Keystone endpoint>:5000/
rgw_keystone_accepted_roles: 'member, Member, admin'
rgw_keystone_accepted_admin_roles: ResellerAdmin, swiftoperator
rgw_keystone_admin_domain: default
rgw_keystone_admin_project: service
rgw_keystone_admin_user: swift
rgw_keystone_admin_password: <Password as defined in the environment parameters>
rgw_keystone_implicit_tenants: 'true'
rgw_keystone_revocation_interval: '0'
rgw_s3_auth_use_keystone: 'true'
rgw_swift_versioning_enabled: 'true'
rgw_swift_account_in_url: 'true'
```

**注記**

デフォルトでは、director は Identity サービスに以下のロールとユーザーを作成します。

- rgw_keystone_accepted_admin_roles: ResellerAdmin, swiftoperator
- rgw_keystone_admin_domain: default
- rgw_keystone_admin_project: service
- rgw_keystone_admin_user: swift

3. 追加の環境ファイルを指定してオーバークラウドをデプロイします。

```
openstack overcloud deploy --templates \
-e <your environment files>
-e /usr/share/openstack-tripleo-heat-templates/environments/swift-external.yaml
-e swift-external-params.yaml
```

### 10.5.11.5. イメージのインポート法および共有ステージングエリアの設定

OpenStack Image サービス (glance) のデフォルト設定は、Red Hat OpenStack Platform のインストール時に使用する heat テンプレートで定義されます。Image サービスの heat テンプレートは **deployment/glance/glance-api-container-puppet.yaml** です。

以下の方法でイメージをインポートすることができます。

**web-download**

**web-download** メソッドを使用して、URL からイメージをインポートする。

**glance-direct**

**glance-direct** メソッドを使用して、ローカルボリュームからイメージをインポートする。

#### 10.5.11.5.1. glance-settings.yaml ファイルの作成およびデプロイメント

カスタム環境ファイルを使用して、インポートパラメーターを設定します。これらのパラメーターは、コア heat テンプレートコレクションのデフォルト値を上書きします。以下の環境コンテンツは、相互運用可能なイメージのインポート用パラメーターの例です。

```
parameter_defaults:
  # Configure NFS backend
  GlanceBackend: file
  GlanceNfsEnabled: true
  GlanceNfsShare: 192.168.122.1:/export/glance

  # Enable glance-direct import method
  GlanceEnabledImportMethods: glance-direct,web-download

  # Configure NFS staging area (required for glance-direct import method)
  GlanceStagingNfsShare: 192.168.122.1:/export/glance-staging
```

**GlanceBackend**、**GlanceNfsEnabled**、および **GlanceNfsShare** パラメーターについては、**『オーバークラウドの高度なカスタマイズ』**の「ストレージの設定」の章に説明があります。

相互運用可能なイメージのインポートに関する 2 つの新たなパラメーターを使用して、インポート法および共有 FNS ステージングエリアを定義します。

**GlanceEnabledImportMethods**

利用可能なインポート法として web-download (デフォルト) および glance-direct を定義します。このパラメーターが必要になるのは、web-download に加えて別の方法を有効にする場合に限ります。

**GlanceStagingNfsShare**

glance-direct インポート法で使用する NFS ステージングエリアを設定します。この領域は、高可用性クラスター設定のノード間で共有することができます。このパラメーターを使用する場合は、**GlanceNfsEnabled** パラメーターを **true** に設定する必要もあります。

**手順**

1. 新規ファイルを作成します (例: **glance-settings.yaml**)。サンプルの構文を使用して、このファイルを設定します。
2. デプロイメントに該当するその他の環境ファイルと共に、**glance-settings.yaml** ファイルを **openstack overcloud deploy** コマンドに追加します。

```
$ openstack overcloud deploy --templates -e glance-settings.yaml
```

環境ファイルの使用に関する詳細については、**『オーバークラウドの高度なカスタマイズ』**の「オーバークラウド作成時の環境ファイルの追加」セクションを参照してください。

#### 10.5.11.5.2. イメージの Web インポートソースの制御

Web インポートによるイメージダウンロードのソースを制限することができます。そのためには、オプションの **glance-image-import.conf** ファイルに URI のブラックリストおよびホワイトリストを追加します。

3 段階のレベルで、イメージソースの URI をホワイトリスト登録またはブラックリスト登録することができます。

- スキームレベル (allowed_schemes、disallowed_schemes)
- ホストレベル (allowed_hosts、disallowed_hosts)
- ポートレベル (allowed_ports、disallowed_ports)

レベルにかかわらず、ホワイトリストとブラックリストの両方を指定した場合には、ホワイトリストが優先されブラックリストは無視されます。

Image サービスは、以下の判断ロジックを使用してイメージソースの URI を検証します。

1. スキームを確認する。
    a. スキームが定義されていない場合: 拒否する。
    b. ホワイトリストがあり、そのスキームがホワイトリストに記載されていない場合: 拒否する。記載されている場合: c 項をスキップして 2 項に進む。
    c. ブラックリストがあり、そのスキームがブラックリストに記載されている場合: 拒否する。
2. ホスト名を確認する。
    a. ホスト名が定義されていない場合: 拒否する。
    b. ホワイトリストがあり、そのホスト名がホワイトリストに記載されていない場合: 拒否する。記載されている場合: c 項をスキップして 3 項に進む。
    c. ブラックリストがあり、そのホスト名がブラックリストに記載されている場合: 拒否する。
3. URI にポートが含まれていれば、ポートを確認する。
    a. ホワイトリストがあり、そのポートがホワイトリストに記載されていない場合: 拒否する。記載されている場合: b 項をスキップして 4 項に進む。
    b. ブラックリストがあり、そのポートがブラックリストに記載されている場合: 拒否する。
4. 有効な URI として受け入れる。

(ホワイトリストに登録する、あるいはブラックリストに登録しないことにより) スキームを許可した場合には、URI にポートを含めないことでそのスキームのデフォルトポートを使用する URI は、すべて許可されます。URI にポートが含まれている場合には、URI はデフォルトの判断ロジックに従って検証されます。

##### 10.5.11.5.2.1. 例

たとえば、FTP のデフォルトポートは 21 です。**ftp** はホワイトリストに登録されたスキームなので、URL ftp://example.org/some/resource は許可されます。しかし、21 はポートのホワイトリストに含まれていないので、同じリソースへの URL であっても ftp://example.org:21/some/resource は拒否されます。

```
allowed_schemes = [http,https,ftp]
disallowed_schemes = []
allowed_hosts = []
disallowed_hosts = []
allowed_ports = [80,443]
disallowed_ports = []
```

#### 10.5.11.5.2.2. イメージのインポートに関するブラックリストおよびホワイトリストのデフォルト設定

**glance-image-import.conf** ファイルは、以下のデフォルトオプションが含まれるオプションのファイルです。

- allowed_schemes: [**http**, **https**]
- disallowed_schemes: ブランク
- allowed_hosts: ブランク
- disallowed_hosts: ブランク
- allowed_ports: [80, 443]
- disallowed_ports: ブランク

デフォルトの設定を使用する場合、エンドユーザーは **http** または **https** スキームを使用する URI にしかアクセスすることができません。ユーザーが指定することのできるポートは、**80** および **443** だけです。ユーザーはポートを指定する必要はありませんが、指定する場合には **80** または **443** のどちらかでなければなりません。

**glance-image-import.conf** ファイルは、Image サービスのソースコードツリーの **etc/** サブディレクトリーにあります。お使いの Red Hat OpenStack Platform のリリースに対応する正しいブランチを使用してください。

#### 10.5.11.5.3. イメージインポート時のメタデータ注入による仮想マシン起動場所の制御

エンドユーザーは Image サービスにイメージをアップロードし、それらのイメージを使用して仮想マシンを起動することができます。これらのユーザーの提供する (非管理者) イメージは、特定のコンピュートノードセットで起動する必要があります。インスタンスのコンピュートノードへの割り当ては、イメージメタデータ属性で制御されます。

Image Property Injection プラグインにより、メタデータ属性がインポート時にイメージに注入されます。属性を指定するには、**glance-image-import.conf** ファイルの [image_import_opts] および [inject_metadata_properties] セクションを編集します。

Image Property Injection プラグインを有効にするには、以下の行を **[image_import_opts]** セクションに追加します。

```
[image_import_opts]
image_import_plugins = [inject_image_metadata]
```

メタデータの注入を特定ユーザーが提供したイメージに制限するには、**ignore_user_roles** パラメーターを設定します。たとえば、**property1** に関する 1 つの値および **property2** に関する 2 つの値を任意の非管理者ユーザーがダウンロードしたイメージに注入するには、以下の設定を使用します。

```
[DEFAULT]
[image_conversion]
[image_import_opts]
image_import_plugins = [inject_image_metadata]
[import_filtering_opts]
[inject_metadata_properties]
ignore_user_roles = admin
inject = PROPERTY1:value,PROPERTY2:value;another value
```

パラメーター **ignore_user_roles** は、プラグインが無視する Keystone ロールのコンマ区切りリストです。つまり、イメージインポートの呼び出しを行うユーザーにこれらのロールが設定されている場合、プラグインはイメージに属性を注入しません。

パラメーター **inject** は、インポートされたイメージのイメージレコードに注入される属性と値のコンマ区切りリストです。それぞれの属性と値は、コロン **(「:」)** で区切る必要があります。

**glance-image-import.conf** ファイルは、Image サービスのソースコードツリーの **etc/** サブディレクトリーにあります。お使いの Red Hat OpenStack Platform のリリースに対応する正しいブランチを使用してください。

#### 10.5.11.6. Image サービス用 cinder バックエンドの設定

**GlanceBackend** パラメーターを使用して、Image サービスがイメージを保管するのに使用するバックエンドを設定します。

**重要**

デフォルトでは、1 つのプロジェクトで作成可能な最大のボリューム数は 10 です。

**手順**

1. Image サービスのバックエンドに **cinder** を設定するには、以下の行を環境ファイルに追加します。

```
parameter_defaults:
  GlanceBackend: cinder
```

2. **cinder** バックエンドが有効な場合には、デフォルトで以下のパラメーターおよび値が設定されます。

```
cinder_store_auth_address = http://172.17.1.19:5000/v3
cinder_store_project_name = service
cinder_store_user_name = glance
cinder_store_password = ****secret****
```

3. **cinder_store_** パラメーターにカスタムのユーザー名またはその他のカスタムの値を使用するには、**parameter_defaults** に **ExtraConfig** パラメーターを追加してカスタムの値を指定します。

```
ExtraConfig:
```

```
glance::config::api_config:
  glance_store/cinder_store_auth_address:
    value: "%{hiera('glance::api::authtoken::auth_url')}/v3"
  glance_store/cinder_store_user_name:
    value: <user-name>
  glance_store/cinder_store_password:
    value: "%{hiera('glance::api::authtoken::password')}"
  glance_store/cinder_store_project_name:
    value: "%{hiera('glance::api::authtoken::project_name')}"
```

#### 10.5.11.7. 1つのインスタンスにアタッチすることのできる最大ストレージデバイス数の設定

デフォルトでは、1つのインスタンスにアタッチすることのできるストレージデバイスの数に制限はありません。デバイスの最大数を制限するには、コンピュート環境ファイルに **max_disk_devices_to_attach** パラメーターを追加します。以下の例を使用して、**max_disk_devices_to_attach** の値を「30」に変更します。

```
parameter_defaults:
  ComputeExtraConfig:
      nova::config::nova_config:
        compute/max_disk_devices_to_attach:
          value: '30'
```

**ガイドラインおよび留意事項**

- インスタンスがサポートするストレージディスクの数は、ディスクが使用するバスにより異なります。たとえば、IDE ディスクバスでは、アタッチされるデバイスは 4 つに制限されます。
- アクティブなインスタンスを持つコンピュートノードで **max_disk_devices_to_attach** を変更した場合に、最大数がインスタンスにすでにアタッチされているデバイスの数より小さいと、再ビルドが失敗する可能性があります。たとえば、インスタンス **A** に 26 のデバイスがアタッチされている場合に、**max_disk_devices_to_attach** を 20 に変更すると、インスタンス **A** を再ビルドする要求は失敗します。
- コールドマイグレーション時には、設定されたストレージデバイスの最大数は、移行する元のインスタンスにのみ適用されます。移動前に移行先が確認されることはありません。つまり、コンピュートノード A に 26 のディスクデバイスがアタッチされていて、コンピュートノード B の最大ディスクデバイスアタッチ数が 20 に設定されている場合に、26 のデバイスがアタッチされたインスタンスをコンピュートノード A からコンピュートノード B に移行するコールドマイグレーションの操作は成功します。ただし、これ以降、コンピュートノード B でインスタンスを再ビルドする要求は失敗します。すでにアタッチされているデバイスの数 26 が、設定された最大値の 20 を超えているためです。
- 設定された最大値は、退避オフロード中のインスタンスには適用されません。これらのインスタンスはコンピュートノードを持たないためです。
- 多数のディスクデバイスをインスタンスにアタッチすると、インスタンスのパフォーマンスが低下する可能性があります。お使いの環境がサポートすることのできる限度に基づいて、最大数を調整します。
- マシン種別 が Q35 のインスタンスは、最大で 500 のディスクデバイスをアタッチすることができます。

#### 10.5.11.8. Image サービスのキャッシュ機能を使用したスケーラビリティーの向上

glance-api キャッシュメカニズムを使用して、Image サービス (glance) API サーバーにイメージのコピーを保存し、それらを自動的に取得してスケーラビリティーを向上させます。Image サービスのキャッシュ機能を使用することで、複数のホスト上で glance-api を実行することができます。つまり、同じイメージをバックエンドストレージから何度も取得する必要はありません。Image サービスのキャッシュ機能は、Image サービスの動作には一切影響を与えません。

Red Hat OpenStack Platform director (tripleo) heat テンプレートを使用して、Image サービスのキャッシュ機能を設定します。

**手順**

1. 環境ファイルの **GlanceCacheEnabled** パラメーターの値を **true** に設定します。これにより、**glance-api.conf** heat テンプレートの **flavor** の値が自動的に **keystone+cachemanagement** に設定されます。

```
parameter_defaults:
  GlanceCacheEnabled: true
```

2. オーバークラウドを再デプロイする際に、**openstack overcloud deploy** コマンドにその環境ファイルを追加します。

3. オプション: オーバークラウドを再デプロイする際に、**glance_cache_pruner** を異なる頻度に調節します。5 分間の頻度の例を以下に示します。

```
parameter_defaults:
  ControllerExtraConfig:
    glance::cache::pruner::minute: '*/5'
```

ファイルシステムを使い果たす状況を回避するために、ご自分のニーズに合わせて頻度を調節します。異なる頻度を選択する場合は、以下の要素を考慮に入れます。

- 実際の環境でキャッシュするファイルのサイズ
- 利用可能なファイルシステムの容量
- 環境がイメージをキャッシュする頻度

### 10.5.11.9. サードパーティーのストレージの設定

以下の環境ファイルは、コア heat テンプレートコレクション **/usr/share/openstack-tripleo-heat-templates** にあります。

**Dell EMC Storage Center**

Block Storage (cinder) サービス用に単一の Dell EMC Storage Center バックエンドをデプロイします。
環境ファイルは **/usr/share/openstack-tripleo-heat-templates/environments/cinder-dellsc-config.yaml** にあります。

設定に関する詳しい情報は、『Dell Storage Center Back End Guide』 を参照してください。

**Dell EMC PS Series**

Block Storage (cinder) サービス用に単一の Dell EMC PS Series バックエンドをデプロイします。
環境ファイルは **/usr/share/openstack-tripleo-heat-templates/environments/cinder-dellps-config.yaml** にあります。

設定に関する詳しい情報は、『Dell EMC PS Series Back End Guide』を参照してください。

**NetApp ブロックストレージ**

Block Storage (cinder) サービス用に NetApp ストレージアプライアンスをバックエンドとしてデプロイします。
環境ファイルは **/usr/share/openstack-tripleo-heat-templates/environments/storage/cinder-netapp-config.yaml** にあります。

NetApp Block Storage についての詳しい情報は、NetApp の Block Storage Service (Cinder) に関するドキュメントを参照してください。

## 10.5.12. セキュリティーの強化

以下の項では、オーバークラウドのセキュリティーを強化するための推奨事項について説明します。

### 10.5.12.1. オーバークラウドのファイアウォールの管理

OpenStack Platform の各コアサービスには、それぞれのコンポーザブルサービステンプレートにファイアウォールルールが含まれています。これにより、各オーバークラウドノードにファイアウォールルールのデフォルトセットが自動的に作成されます。

オーバークラウドの heat テンプレートには、追加のファイアウォール管理に役立つパラメーターのセットが含まれています。

**ManageFirewall**

ファイアウォールルールを自動管理するかどうかを定義します。このパラメーターを **true** に設定すると、Puppet は各ノードでファイアウォールを自動的に設定することができます。ファイアウォールを手動で管理する場合には **false** に設定してください。デフォルトは **true** です。

**PurgeFirewallRules**

ファイアウォールルールを新規設定する前に、デフォルトの Linux ファイアウォールルールを完全削除するかどうかを定義します。デフォルトは **false** です。

**ManageFirewall** パラメーターを **true** に設定すると、デプロイメントに追加のファイアウォールルールを作成することができます。オーバークラウドの環境ファイルで、設定フックを使用して (「Puppet: ロール用 hieradata のカスタマイズ」を参照) **tripleo::firewall::firewall_rules** hieradata を設定します。この hieradata は、ファイアウォールルール名とそれぞれのパラメーター (すべてオプション) を鍵として記載したハッシュです。

**port**

ルールに関連付けられたポート

**dport**

ルールに関連付けられた宛先ポート

**sport**

ルールに関連付けられた送信元ポート

**proto**

ルールに関連付けられたプロトコル。デフォルトは **tcp** です。

**action**

ルールに関連付けられたアクションポリシー。デフォルトは **accept** です。

**jump**

ジャンプ先のチェーン。設定されている場合には **action** を上書きします。

**state**

ルールに関連付けられた状態の配列。デフォルトは **['NEW']** です。

**source**

ルールに関連付けられた送信元の IP アドレス

**iniface**

ルールに関連付けられたネットワークインターフェース

**chain**

ルールに関連付けられたチェーン。デフォルトは **INPUT** です。

**destination**

ルールに関連付けられた宛先の CIDR

以下の例は、ファイアウォールルールの形式の構文を示しています。

```
ExtraConfig:
  tripleo::firewall::firewall_rules:
    '300 allow custom application 1':
      port: 999
      proto: udp
      action: accept
    '301 allow custom application 2':
      port: 8081
      proto: tcp
      action: accept
```

この設定では、**ExtraConfig** により、追加で 2 つのファイアウォールルールが 全ノードに適用されます。

**注記**

各ルール名はそれぞれの **iptables** ルールのコメントになります。各ルール名は 3 桁のプレフィックスで始まり、Puppet が最終の **iptables** ファイルで定義済みの全ルールの順序を設定するのに役立ちます。デフォルトの Red Hat OpenStack Platform ルールでは、000 から 200 までの範囲のプレフィックスを使用します。

#### 10.5.12.2. Simple Network Management Protocol (SNMP) 文字列の変更

director は、オーバークラウド向けのデフォルトの読み取り専用 SNMP 設定を提供します。SNMP の文字列を変更して、権限のないユーザーがネットワークデバイスに関する情報を取得するリスクを軽減することを推奨します。

**注記**

文字列パラメーターを使用して **ExtraConfig** インターフェースを設定する場合には、heat および Hiera が文字列をブール値と解釈しないように、**'"<VALUE>"'** の構文を使用する必要があります。

オーバークラウドの環境ファイルで **ExtraConfig** フックを使用して、以下の hieradata を設定します。

**SNMP の従来のアクセス制御設定**

**snmp::ro_community**

IPv4 の読み取り専用 SNMP コミュニティー文字列。デフォルト値は **public** です。

**snmp::ro_community6**

IPv6 の読み取り専用 SNMP コミュニティー文字列。デフォルト値は **public** です。

**snmp::ro_network**

デーモンへの **RO クエリー** が許可されるネットワーク。この値は文字列または配列のいずれかです。デフォルト値は **127.0.0.1** です。

**snmp::ro_network6**

デーモンへの IPv6 **RO クエリー** が許可されるネットワーク。この値は文字列または配列のいずれかです。デフォルト値は **::1/128** です。

**tripleo::profile::base::snmp::snmpd_config**

安全弁として **snmpd.conf** ファイルに追加する行の配列。デフォルト値は **[]** です。利用できるすべてのオプションについては、SNMP 設定ファイル に関する Web ページを参照してください。

以下に例を示します。

```
parameter_defaults:
  ExtraConfig:
    snmp::ro_community: mysecurestring
    snmp::ro_community6: myv6securestring
```

これにより、全ノードで、読み取り専用の SNMP コミュニティー文字列が変更されます。

#### SNMP のビューベースのアクセス制御設定 (VACM)

**snmp::com2sec**

IPv4 セキュリティー名

**snmp::com2sec6**

IPv6 セキュリティー名

以下に例を示します。

```
parameter_defaults:
  ExtraConfig:
    snmp::com2sec: mysecurestring
    snmp::com2sec6: myv6securestring
```

これにより、全ノードで、読み取り専用の SNMP コミュニティー文字列が変更されます。

詳細は、**snmpd.conf** の man ページを参照してください。

### 10.5.12.3. HAProxy の SSL/TLS の暗号およびルールの変更

オーバークラウドで SSL/TLS を有効化した場合には (「オーバークラウドのパブリックエンドポイントでの SSL/TLS の有効化」を参照)、HAProxy 設定を使用する SSL/TLS の暗号とルールを強化することをお勧めします。これにより、POODLE TLS 脆弱性 などの SSL/TLS の脆弱性を回避することができます。

オーバークラウドの環境ファイルで **ExtraConfig** フックを使用して、以下の hieradata を設定します。

**tripleo::haproxy::ssl_cipher_suite**

HAProxy で使用する暗号スイート

**tripleo::haproxy::ssl_options**

HAProxy で使用する SSL/TLS ルール

たとえば、以下のような暗号およびルールを使用することができます。

- 暗号: **ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS**
- ルール: **no-sslv3 no-tls-tickets**

環境ファイルを作成して、以下の内容を記載します。

```
parameter_defaults:
  ExtraConfig:
    tripleo::haproxy::ssl_cipher_suite: ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS
    tripleo::haproxy::ssl_options: no-sslv3 no-tls-tickets
```

**注記**

暗号のコレクションは、改行せずに 1 行に記述します。

オーバークラウドの作成時にこの環境ファイルを追加します。

### 10.5.12.4. Open vSwitch ファイアウォールの使用

Red Hat OpenStack Platform director の Open vSwitch (OVS) ファイアウォールドライバーを使用するように、セキュリティーグループを設定することができます。**NeutronOVSFirewallDriver** パラメーターを使用して、使用するファイアウォールドライバーを指定します。

- **iptables_hybrid**: Networking サービス (neutron) が iptables/ハイブリッドベースの実装を使用するように設定します。
- **openvswitch**: Networking サービスが OVS ファイアウォールのフローベースのドライバーを使用するように設定します。

**openvswitch** ファイアウォールドライバーはパフォーマンスがより高く、ゲストをプロジェクトネットワークに接続するためのインターフェースとブリッジの数を削減します。

**重要**

Open vSwitch (OVS) ファイアウォールドライバーによるマルチキャストトラフィックの処理は、iptables ファイアウォールドライバーの場合とは異なります。iptables の場合、デフォルトでは VRRP トラフィックは拒否されます。したがって、VRRP トラフィックがエンドポイントに到達できるようにするには、セキュリティーグループルールで VRRP を有効にする必要があります。OVS の場合、すべてのポートが同じ OpenFlow コンテキストを共有し、ポートごとに個別にマルチキャストトラフィックを処理することはできません。セキュリティーグループはすべてのポート (ルーター上のポートなど) には適用されないため、OVS は RFC 4541 の定義に従って **NORMAL** アクションを使用してマルチキャストトラフィックを全ポートに転送します。

**注記**

**iptables_hybrid** オプションには、OVS-DPDK との互換性はありません。**openvswitch** オプションには、OVS ハードウェアオフロードとの互換性はありません。

**network-environment.yaml** ファイルで **NeutronOVSFirewallDriver** パラメーターを設定します。

```
NeutronOVSFirewallDriver: openvswitch
```

- **NeutronOVSFirewallDriver**: セキュリティーグループを実装する時に使用するファイアウォールドライバーの名前を設定します。設定可能な値は、お使いのシステム構成により異なります。例としては、**noop**、**openvswitch**、および **iptables_hybrid** 等が挙げられます。デフォルト値である空の文字列を指定すると、サポートされている構成となります。

### 10.5.12.5. セキュアな root ユーザーアクセスの使用

オーバークラウドのイメージでは、**root** ユーザーのセキュリティー強化機能が自動的に含まれます。たとえば、デプロイされる各オーバークラウドノードでは、**root** ユーザーへの直接の SSH アクセスを自動的に無効にされます。ただし、オーバークラウドノードの **root** ユーザーにアクセスすることはできます。

1. アンダークラウドノードに **stack** ユーザーとしてログインします。
2. 各オーバークラウドノードには **heat-admin** ユーザーアカウントがあります。このユーザーアカウントにはアンダークラウドのパブリック SSH 鍵が含まれており、アンダークラウドからオーバークラウドノードへのパスワード無しの SSH アクセスが可能です。アンダークラウドノードで、**heat-admin** ユーザーとして SSH 経由でオーバークラウドノードにログインします。
3. **sudo -i** で **root** ユーザーに切り替えます。

#### root ユーザーのセキュリティーレベルの引き下げ

状況によっては、**root** ユーザーへの直接 SSH アクセスが必要な場合があります。このような場合には、各オーバークラウドノードで **root** ユーザーの SSH 制限を引き下げることができます。

**警告**

この方法は、デバッグのみを目的としています。したがって、実稼働環境での使用は推奨されません。

この方法では、初回ブートの設定フック (「初回ブート: 初回ブート設定のカスタマイズ」を参照) を使用します。環境ファイルに以下の内容を入力してください。

```
resource_registry:
  OS::TripleO::NodeUserData: /usr/share/openstack-tripleo-heat-
templates/firstboot/userdata_root_password.yaml

parameter_defaults:
  NodeRootPassword: "p@55w0rd!"
```

以下の点に注意してください。

- **OS::TripleO::NodeUserData** リソースは、初回ブートの **cloud-init** 段階に **root** ユーザーを設定するテンプレートを参照します。
- **NodeRootPassword** パラメーターは **root** ユーザーのパスワードを設定します。このパラメーターの値は、任意の値に変更してください。環境ファイルにはパスワードがプレーンテキスト形式の文字列として記載されるので、セキュリティーリスクと見なされます。

オーバークラウドを作成する際に、**openstack overcloud deploy** コマンドにこの環境ファイルを追加します。

### 10.5.13. モニタリングツールの設定

モニタリングツールは、可用性のモニタリングと集中ロギングに使用できるオプションのツールスイートです。可用性のモニタリングを使用して全コンポーネントの機能を監視し、集中ロギングを使用して Red Hat OpenStack Platform 環境全体のすべてのログを一箇所で確認します。

モニタリングツールの設定に関する詳しい情報は、**『監視ツール設定ガイド』**を参照してください。

### 10.5.14. ネットワークプラグインの設定

director には、サードパーティーのネットワークプラグインを設定する際に使用できる環境ファイルが含まれています。

#### 10.5.14.1. Fujitsu Converged Fabric (C-Fabric)

**/usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-cfab.yaml** にある環境ファイルを使用して、Fujitsu Converged Fabric (C-Fabric) プラグインを有効にすることができます。

1. 環境ファイルを **templates** サブディレクトリーにコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-cfab.yaml
/home/stack/templates/
```

2. **resource_registry** で絶対パスを使用するように編集します。

```
resource_registry:
  OS::TripleO::Services::NeutronML2FujitsuCfab: /usr/share/openstack-tripleo-heat-templates/puppet/services/neutron-plugin-ml2-fujitsu-cfab.yaml
```

3. **/home/stack/templates/neutron-ml2-fujitsu-cfab.yaml** の **parameter_defaults** を確認します。

- **NeutronFujitsuCfabAddress**: C-Fabric の telnet IP アドレス (文字列)
- **NeutronFujitsuCfabUserName**: 使用する C-Fabric ユーザー名 (文字列)
- **NeutronFujitsuCfabPassword**: C-Fabric ユーザーアカウントのパスワード (文字列)
- **NeutronFujitsuCfabPhysicalNetworks**: **physical_network** 名と対応する vfab ID を指定する **<physical_network>:<vfab_id>** タプルの一覧 (コンマ区切りリスト)
- **NeutronFujitsuCfabSharePprofile**: 同じ VLAN ID を使用する neutron ポート間で C-Fabric pprofile を共有するかどうかを決定するパラメーター (ブール値)
- **NeutronFujitsuCfabPprofilePrefix**: pprofile 名のプレフィックス文字列 (文字列)
- **NeutronFujitsuCfabSaveConfig**: 設定を保存するかどうかを決定するパラメーター (ブール値)

4. デプロイメントにテンプレートを適用するには、**openstack overcloud deploy** コマンドで環境ファイルを指定します。

```
$ openstack overcloud deploy --templates -e /home/stack/templates/neutron-ml2-fujitsu-cfab.yaml [OTHER OPTIONS] ...
```

#### 10.5.14.2. Fujitsu FOS Switch

**/usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-fossw.yaml** にある環境ファイルを使用して、Fujitsu FOS Switch プラグインを有効にすることができます。

**手順**

1. 環境ファイルを **templates** サブディレクトリーにコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-fossw.yaml /home/stack/templates/
```

2. **resource_registry** で絶対パスを使用するように編集します。

```
resource_registry:
  OS::TripleO::Services::NeutronML2FujitsuFossw: /usr/share/openstack-tripleo-heat-templates/puppet/services/neutron-plugin-ml2-fujitsu-fossw.yaml
```

3. **/home/stack/templates/neutron-ml2-fujitsu-fossw.yaml** の **parameter_defaults** を確認します。

- **NeutronFujitsuFosswIps**: 全 FOS スイッチの IP アドレス (コンマ区切りリスト)

- **NeutronFujitsuFosswUserName**: 使用する FOS ユーザー名 (文字列)
- **NeutronFujitsuFosswPassword**: FOS ユーザーアカウントのパスワード (文字列)
- **NeutronFujitsuFosswPort**: SSH 接続に使用するポート番号 (数値)
- **NeutronFujitsuFosswTimeout**: SSH 接続のタイムアウト時間 (数値)
- **NeutronFujitsuFosswUdpDestPort**: FOS スイッチ上の VXLAN UDP 宛先のポート番号 (数値)
- **NeutronFujitsuFosswOvsdbVlanidRangeMin**: VNI および物理ポートのバインディングに使用する範囲内の最小の VLAN ID (数値)
- **NeutronFujitsuFosswOvsdbPort**: FOS スイッチ上の OVSDB サーバー用のポート番号 (数値)

4. デプロイメントにテンプレートを適用するには、**openstack overcloud deploy** コマンドで環境ファイルを指定します。

```
$ openstack overcloud deploy --templates -e /home/stack/templates/neutron-ml2-fujitsu-fossw.yaml [OTHER OPTIONS] ...
```

### 10.5.15. Identity の設定

director には、Identity サービス (keystone) の設定に役立つパラメーターが含まれています。

#### 10.5.15.1. リージョン名

デフォルトでは、オーバークラウドのリージョンは、**regionOne** という名前です。環境ファイルに **KeystoneRegion** エントリーを追加することによって変更できます。オーバークラウドのデプロイ後に、この値を変更することはできません。

```
parameter_defaults:
  KeystoneRegion: 'SampleRegion'
```

### 10.5.16. その他の設定

#### 10.5.16.1. オーバークラウドノードカーネルの設定

Red Hat OpenStack Platform director には、オーバークラウドノードのカーネルを設定するパラメーターが含まれています。

**ExtraKernelModules**

読み込むカーネルモジュール。モジュール名は、空の値を持つハッシュキーとして一覧表示されます。

```
ExtraKernelModules:
  <MODULE_NAME>: {}
```

**ExtraKernelPackages**

**ExtraKernelModules** で定義されるカーネルモジュールを読み込む前にインストールするカーネル関連のパッケージ。パッケージ名は、空の値を持つハッシュキーとして一覧表示されます。

```
ExtraKernelPackages:
  <PACKAGE_NAME>: {}
```

**ExtraSysctlSettings**

適用する sysctl 設定のハッシュ。**value** キーを使用して、各パラメーターの値を設定します。

```
ExtraSysctlSettings:
  <KERNEL_PARAMETER>:
    value: <VALUE>
```

環境ファイルのこれらのパラメーターの構文例を以下に示します。

```
parameter_defaults:
  ExtraKernelModules:
    iscsi_target_mod: {}
  ExtraKernelPackages:
    iscsi-initiator-utils: {}
  ExtraSysctlSettings:
    dev.scsi.logging_level:
      value: 1
```

### 10.5.16.2. サーバーコンソールの設定

オーバークラウドノードからのコンソール出力は、常にサーバーコンソールに送信される訳ではありません。サーバーコンソールでこの出力を表示するには、ハードウェアの正しいコンソールを使用するようにオーバークラウドを設定する必要があります。この設定を行うには、以下のいずれかの方法を使用します。

- オーバークラウドロールごとに **KernelArgs** heat パラメータを変更する
- オーバークラウドノードをプロビジョニングするのに director が使用する **overcloud-full.qcow2** イメージをカスタマイズする

**前提条件**

- アンダークラウドの正常なインストール。詳しくは、『director のインストールと使用方法』を参照してください。
- デプロイ可能なオーバークラウドノード

**デプロイメント時の heat を使用した KernelArgs の変更**

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. source コマンドで **stackrc** 認証情報ファイルを読み込みます。

```
$ source stackrc
```

3. 以下の内容で環境ファイル **overcloud-console.yaml** を作成します。

```
parameter_defaults:
  <role>Parameters:
    KernelArgs: "console=<console-name>"
```

**<role>** を設定するオーバークラウドロールの名前に置き換え、**<console-name>** を使用するコンソールの ID に置き換えます。たとえば、デフォルトロールのすべてのオーバークラウドノードが **tty0** を使用するように設定するには、以下のスニペットを使用します。

```
parameter_defaults:
  ControllerParameters:
    KernelArgs: "console=tty0"
  ComputeParameters:
    KernelArgs: "console=tty0"
  BlockStorageParameters:
    KernelArgs: "console=tty0"
  ObjectStorageParameters:
    KernelArgs: "console=tty0"
  CephStorageParameters:
    KernelArgs: "console=tty0"
```

4. **-e** オプションを使用して、**overcloud-console-tty0.yaml** ファイルをデプロイメントコマンドに追加します。

**overcloud-full.qcow2 イメージの変更**

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. source コマンドで **stackrc** 認証情報ファイルを読み込みます。

```
$ source stackrc
```

3. **overcloud-full.qcow2** イメージのカーネル引数を変更して、ハードウェアの正しいコンソールを設定します。たとえば、コンソールを **tty0** に設定します。

```
$ virt-customize --selinux-relabel -a overcloud-full.qcow2 --run-command 'grubby --update-kernel=ALL --args="console=tty0"'
```

4. イメージを director にインポートします。

```
$ openstack overcloud image upload --image-path overcloud-full.qcow2
```

5. オーバークラウドをデプロイします。

**検証**

1. アンダークラウドからオーバークラウドノードにログインします。

```
$ ssh heat-admin@<IP-address>
```

**<IP-address>** をオーバークラウドノードの IP アドレスに置き換えます。

2. **/proc/cmdline** ファイルの内容を調べ、**console=** パラメータが使用するコンソールの値に設定されていることを確認します。

```
[heat-admin@controller-0 ~]$ cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos2)/boot/vmlinuz-4.18.0-193.29.1.el8_2.x86_64
root=UUID=0ec3dea5-f293-4729-b676-5d38a611ce81 ro console=tty0
console=ttyS0,115200n81 no_timer_check crashkernel=auto rhgb quiet
```

#### 10.5.16.3. 外部の負荷分散機能の設定

オーバークラウドは、複数のコントローラーを合わせて、高可用性クラスターとして使用し、OpenStack サービスのオペレーションパフォーマンスを最大限に保つようにします。さらに、クラスターにより、OpenStack サービスへのアクセスの負荷分散が行われ、コントローラーノードに均等にトラフィックを分配して、各ノードのサーバーで過剰負荷を軽減します。外部のロードバランサーを使用して、この負荷分散を実行することも可能です。たとえば、専用のハードウェアベースのロードバランサーを使用して、コントローラーノードへのトラフィックの分散処理を行うことができます。

外部の負荷分散機能の設定に関する詳しい情報は、『External Load Balancing for the Overcloud』を参照してください。

#### 10.5.16.4. IPv6 ネットワークの設定

デフォルトでは、オーバークラウドは、インターネットプロトコルのバージョン 4 (IPv4) を使用してサービスのエンドポイントを設定します。ただし、オーバークラウドはインターネットプロトコルのバージョン 6 (IPv6) のエンドポイントもサポートします。これは、IPv6 のインフラストラクチャーをサポートする組織には便利です。director には、IPv6 ベースのオーバークラウドを作成するための環境ファイルのセットが含まれています。

オーバークラウドでの IPv6 の設定に関する詳しい情報は、全手順が記載されている専用の『IPv6 Networking for the Overcloud』を参照してください。