Red Hat OpenStack Platform 15

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: 2022-09-15

Red Hat OpenStack Platform 15 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

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

法律上の通知

概要

本ガイドでは、Red Hat OpenStack Platform director を使用して、Red Hat OpenStack Platform 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する方法について説明します。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

第1章 はじめに

Red Hat OpenStack Platform director は、オーバークラウドとしても知られる、完全な機能を実装した OpenStack 環境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。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

本ガイドでは、director を使用してオーバークラウドの機能を拡張する方法について説明します。ここでは、director でのノードの登録が完了済みで、かつオーバークラウドの作成に必要なサービスが設定済みであることを前提としています。本ガイドの手順を使用して、オーバークラウドをカスタマイズすることができます。

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

本ガイドのカスタム設定では、Heat テンプレートと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、Heat が作成するリソースのコレクションである **スタック** およびリソースの設定を定義/作成することです。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 OpenStack のオブジェクトを指します。

注記

Heat テンプレートファイルの拡張子は、**.yaml** または **.template** にする必要があります。そうでないと、カスタムテンプレートリソースとして処理されません。

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。

パラメーター

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。

リソース

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。OpenStack には全コンポーネントに対応するコアのリソース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。

アウトプット

これらは、スタックの作成後に Heat から渡される値です。これらの値には、Heat API またはクライアントツールを使用してアクセスすることができます。これらがテンプレートの **output** セクションで定義されます。

以下に、基本的な Heat テンプレートの例を示します。

```
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、特定のフレーバー、イメージ、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したメインのスタックに基づいた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、Heat テンプレートをカスタマイズする特別な種類のテンプレートです。このファイルは、3 つの主要な部分で構成されます。

リソースレジストリー

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。この設定は、環境ファイルの **resource_registry** セクションで定義されます。

パラメーター

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など)、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。これらの設定は、環境ファイルの **parameters** セクションで定義します。

パラメーターのデフォルト

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等)、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。パラメーターのデフォルト値は、環境ファイルの **parameter_defaults** セクションで定義します。

重要

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** を使用することを推奨します。パラメーターが、オーバークラウドの全スタックテンプレートに適用されるためです。

以下に基本的な環境ファイルの例を示します。

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

parameter_defaults:
  NetworkName: my_network

parameters:
  MyIP: 192.168.0.1
```

たとえば、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、**my_template.yaml** のパラメーターにのみ適用します。

NetworkName はメインの Heat テンプレート (上記の例では **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** リソースとその **myserver.yaml** テンプレート) の両方に適用されます。

注記

環境ファイルの拡張子は、**.yaml** または **.template** にする必要があります。そうでないと、カスタムテンプレートリソースとして処理されません。

2.3. オーバークラウドのコア HEAT テンプレート

director には、オーバークラウドのコア Heat テンプレート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。

このコレクションには、多数の Heat テンプレートおよび環境ファイルが含まれます。このテンプレートコレクションで特に注意すべき主要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の作成に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の作成に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

オーバークラウド内のロールを定義して、サービスを各ロールにマッピングするファイル。

network_data.yaml

サブネット、割り当てプール、VIP ステータスなどのオーバークラウド内のネットワークとそれらのプロパティーを定義するファイル。デフォルトの **network_data** 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** 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

オーバークラウドプラン用のメタデータを定義するファイル。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

オーバークラウドプラン用の環境ファイルのマッピング。director の Web UI で環境ファイルを記述および有効化するには、このファイルを使用します。オーバークラウドプラン内の **environments** ディレクトリーで検出されるカスタムの環境ファイルの中で、**capabilities-map.yaml** では定義されていないファイルは、Web UI の **2 デプロイメントの設定の指定 > 全体の設定** の **Other** サブタブに一覧表示されます。

environments

オーバークラウドの作成に使用可能な Heat 環境ファイルを追加で含む。これらの環境ファイルは、作成された OpenStack 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 (**cinder-netapp-config.yaml**) を有効にする環境ファイルが含まれています。**capabilities-map.yaml** ファイルでは定義されていない、このディレクトリーで検出される環境ファイルはいずれも、director の Web UI の **2 デプロイメントの設定の指定 > 全体の設定** の **Other** サブタブにリストされます。

network

分離ネットワークおよびポートの作成に役立つ Heat テンプレートセット。

puppet

主に Puppet を使用した設定により動作するテンプレート。前述した **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

コンポーザブルサービスアーキテクチャー内の全サービス用の Heat テンプレートが含まれるディレクトリー。

extraconfig

追加の機能を有効化するために使用するテンプレート。

firstboot

ノードを最初に作成する際に director が使用する **first_boot** スクリプトの例を提供します。

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により、オーバークラウドプランに関するメタデータを定義することができます。この情報は、オーバークラウドプランのインポートとエクスポートに使用されるのに加えて、プランからオーバークラウドを作成する際にも使用されます。

プランの環境ファイル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と、プランファイルの保管に使用する 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** (**overcloud.yaml.j2** テンプレートをレンダリングしたバージョン) です。

environments

使用する環境ファイルの一覧を定義します。各環境ファイルのパスは **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、このセクションには director が無作為に生成したパスワードが自動的に設定されます。

workflow_parameters

OpenStack Workflow (mistral) の名前空間にパラメーターのセットを指定することができます。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
```

openstack overcloud deploy コマンドに **-p** オプションを使用して、プランの環境メタデータファイルを指定することができます。以下に例を示します。

```
(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

デプロイメントのコマンド (**openstack overcloud deploy**) で **-e** オプションを使用して、オーバークラウドをカスタマイズするための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、後で実行される環境ファイルで定義されてい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。たとえば、以下に示す 2 つの環境ファイルを作成したとします。

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'
```

次に両環境ファイルを指定してデプロイを実行します。

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

この例では、両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**) が含まれています。**openstack overcloud deploy** コマンドは、以下のプロセスを順に実行します。

1. **--template** オプションで指定したコア Heat テンプレート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** で指定されているとおりに **/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** で指定されているとおりに **Hongkong** に設定されます。
- **environment-file-1.yaml** で指定されているとおりに、**RabbitFDLimit** パラメーターは **65536** に設定されます。**environment-file-2.yaml** はこの値を変更しません。

これにより、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に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には、git ワークフローで変更をトラッキングして更新をマージしてください。以下の git プロセスを使用すると、カスタムテンプレートコレクションの管理に役立ちます。

カスタムテンプレートコレクションの初期化

以下の手順に従って、Heat テンプレートコレクションを格納する初期 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 **stack** ユーザーのディレクトリーにコピーします。以下の例では、コレクションを **~/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 openstack-tripleo-heat-templates
$ git init .
```

3. 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 *
```

4. 初期コミットを作成します。

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

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** ブランチを作成します。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、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

アンダークラウドの更新時には、**openstack-tripleo-heat-templates** パッケージも更新される可能性があります。このような場合には、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更新してください。

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。

リベース中に git で競合が発生した場合には、以下の手順を実行します。

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

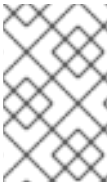
以下の手順に従って、カスタムテンプレートコレクションをデプロイします。

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
```

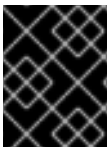
2. **openstack overcloud deploy** コマンドに **--templates** オプションを付けて、ローカルのテンプレートディレクトリーを指定して実行します。

```
$ 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** の拡張子が付いた多数のファイルが含まれています。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** で終わる等価な静的 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ます (**--templates**)。director は **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 ovecloud 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ます (**--templates /home/stack/tripleo-heat-templates**)。director はカスタムコアテンプレートの **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 ovecloud 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

この場合、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。

第3章 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を定義します。これには、以下が含まれます。

- **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。

次の数項には、**deployment** ディレクトリー内にあるサービスの特定のパラメーターを設定する方法について、具体的な例を挙げて説明します。

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 (**deployment/time/timezone-baremetal-ansible.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タイムゾーンを **UTC** に設定します。director はタイムゾーンデータベース **/usr/share/zoneinfo/** で定義済みの標準タイムゾーン名を認識します。たとえば、タイムゾーンを **日本** に設定するには、**/usr/share/zoneinfo** の内容を確認して適切なエントリーを特定します。

```
$ ls /usr/share/zoneinfo/
Africa  Asia  Canada  Cuba  EST  GB  GMT-0  HST  iso3166.tab  Kwajalein  MST
NZ-CHAT  posix  right  Turkey  UTC  Zulu
America  Atlantic  CET  EET  EST5EDT  GB-Eire  GMT+0  Iceland  Israel  Libya
MST7MDT  Pacific  posixrules  ROC  UCT  WET
Antarctica  Australia  Chile  Egypt  Etc  GMT  Greenwich  Indian  Jamaica  MET  Navajo
Poland  PRC  ROK  Universal  W-SU
Arctic  Brazil  CST6CDT  Eire  Europe  GMT0  Hongkong  Iran  Japan  Mexico  NZ
Portugal  PST8PDT  Singapore  US  zone.tab
```

上記の出力には、タイムゾーンファイルと、追加のタイムゾーンファイルを格納するディレクトリーが一覧表示されています。たとえば、**Japan** はこの結果では個別のタイムゾーンファイルですが、**Africa** は追加のタイムゾーンファイルを格納するディレクトリーです。

```
$ ls /usr/share/zoneinfo/Africa/
Abidjan  Algiers  Bamako  Bissau  Bujumbura  Ceuta  Dar_es_Salaam  El_Aaiun  Harare
Kampala  Kinshasa  Lome  Lusaka  Maseru  Monrovia  Niamey  Porto-Novo  Tripoli
Accra  Asmara  Bangui  Blantyre  Cairo  Conakry  Djibouti  Freetown  Johannesburg
Khartoum  Lagos  Luanda  Malabo  Mbabane  Nairobi  Nouakchott  Sao_Tome  Tunis
Addis_Ababa  Asmera  Banjul  Brazzaville  Casablanca  Dakar  Douala  Gaborone  Juba
Kigali  Libreville  Lubumbashi  Maputo  Mogadishu  Ndjamena  Ouagadougou  Timbuktu
Windhoek
```

環境ファイルにエントリーを追加して、タイムゾーンを **Japan** に設定します。

```
parameter_defaults:
  TimeZone: 'Japan'
```

3.2. 例 2: NETWORKING 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、環境ファイルで以下のように設定します。

```
parameter_defaults:
  NeutronEnableDVR: true
```

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** の Heat テンプレートを使用して **RabbitFDLimit** パラメーターを必要な上限値に設定することができます。以下の設定を環境ファイルに追加します。

```
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、以下のパラメーターを追加します。

```
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。そうではなく、パラメーターに空の値を設定します。

```
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 **logrotate** 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 **logrotate** 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 logrotate パラメーターを設定します。

```
parameter_defaults:
  ControllerParameters:
    LogrotateMaxsize: 10M
    LogrotatePurgeAfterDays: 30
  ComputeParameters:
    LogrotateMaxsize: 20M
    LogrotatePurgeAfterDays: 15
```

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、設定すべき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 で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プしてください。

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. director のコア 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 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

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. 設定すべき 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。以下に例を示します。

   ```
   $ 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。以下に例を示します。

```
$ 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 を設定していると表示されます。

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

設定フックは、オーバークラウドのデプロイメントプロセスに独自の設定関数を挿入する手段を提供します。これには、メインのオーバークラウドサービスの設定の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が含まれます。

4.1. 初回起動: 初回起動設定のカスタマイズ

director は、オーバークラウドの初回作成時に全ノードに対して設定を行います。そのために、director は **OS::TripleO::NodeUserData** リソース種別を使用して呼び出すことのできる **cloud-init** を使用します。

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。各ノードの **resolv.conf** に特定のネームサーバーを追加するスクリプトを実行するために、まず基本的な Heat テンプレート (/**home/stack/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}
```

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 (/**home/stack/templates/firstboot.yaml**) を作成します。

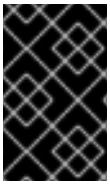
```
resource_registry:
  OS::TripleO::NodeUserData: /home/stack/templates/nameserver.yaml
```

初回起動の設定を追加するには、最初にオーバークラウドを作成する際に、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。以下に例を示します。

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

-e を使用して、オーバークラウドスタックに環境ファイルを適用します。

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

OS::TripleO::NodeUserData 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。

4.2. 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。director の 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 にはフックのセットが用意されており、初回起動が完了してコア設定が開始する前に、特定ノードロールのカスタム設定が提供されます。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、ノードの **resolv.conf** 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ために、まず基本的な Heat テンプレート (**/home/stack/templates/nameserver.yaml**) を作成します。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

  parameters:
    server:
```

```
    type: json
  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。

- **config** パラメーターは、適用する設定を Heat が理解できるように **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、それ以降のオーバークラウド更新に必ず

リソースが再度適用されます。

Heat テンプレートをロールベースのリソース種別として登録する環境ファイル (**/home/stack/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。

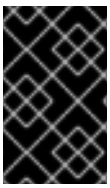
```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

この設定を適用するには、オーバークラウドの作成時または更新時に、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。以下に例を示します。

```
$ 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 にはフックが用意されており、初回起動が完了してコア設定が開始する前に、すべてのノード種別が設定されます。

OS::TripleO::NodeExtraConfig

Puppet のコア設定前に全ノードに適用される追加の設定

以下の例では、各ノードの **resolv.conf** に変数のネームサーバーを追加するスクリプトを実行するために、基本的な Heat テンプレート (**/home/stack/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。

- **config** パラメーターは、適用する設定を Heat が理解できるように **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、それ以降のオーバークラウド更新に必ずリソースが再度適用されます。

次に、**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 (**/home/stack/templates/pre_config.yaml**) を作成します。

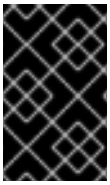
```
resource_registry:
  OS::TripleO::NodeExtraConfig: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

この設定を適用するには、オーバークラウドの作成時または更新時に、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。以下に例を示します。

```
$ 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。director の 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** に変数のネームサーバーを追加するスクリプトを実行するために、まず基本的な Heat テンプレート (**/home/stack/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。

- **config** パラメーターは、適用する設定を Heat が理解できるように **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、それ以降のオーバークラウド更新に必ずリソースが再度適用されます。

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 (**/home/stack/templates/post_config.yaml**) を作成します。

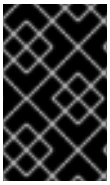
```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

この設定を適用するには、オーバークラウドの作成時または更新時に、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。以下に例を示します。

```
$ 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、特定のノード種別に追加の設定を渡すためのパラメーターセットが含まれています。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。これらのパラメーターは以下のとおりです。

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

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、以下のように設定します。

```
parameter_defaults:
  ComputeExtraConfig:
```

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

openstack overcloud deploy を実行する際に、この環境ファイルを指定します。

重要

それぞれのパラメーターを定義できるのは一度だけです。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。そのためには、ノードのイントロスペクションデータの一部として保存されているシステム UUID を取得します。

```
$ 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 を出力します。以下に例を示します。

```
"F5055C6C-477F-47FB-AFE5-95C6928C407F"
```

このシステム UUID は、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で使用します。以下に例を示します。

```
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" ]}}'
```

openstack overcloud deploy を実行する際に、この環境ファイルを指定します。

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 heiradata ファイルのセットを生成して、定義した hieradata を含めます。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** フックに従って) 全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。

これにより、特定の要件に応じて各ノードを調整することができます。

NodeDataLookup の詳細情報は、『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」セクション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必要がある場合があります。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とします。そ

のためには、まず Puppet 設定を起動する Heat テンプレート (**/home/stack/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}
```

これは、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** ファイル自体には、**motd** のインストールと設定を行うための Puppet クラスが含まれています。

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 (**/home/stack/templates/puppet_post_config.yaml**) を作成します。

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

オーバークラウドのスタックを作成または更新する際に、その他の環境ファイルと共にこの環境ファイルを含めます。

```
$ 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 6 サーバーに登録します。

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は、Ansible を介してオーバークラウドノードを登録する方法を提供します。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/extraconfig/services/rhsm.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。これにより、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。以下に例を示します。

```
parameter_defaults:
  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-2.8-for-rhel-8-x86_64-rpms
      - advanced-virt-for-rhel-8-x86_64-rpms
      - openstack-15-for-rhel-8-x86_64-rpms
      - rhceph-4-osd-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"
```

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。

次の項には、**rhsm** コンポーザブルサービスで使う **RhsmVars** パラメーターと共に使用することのできるサブパラメーターの一覧を記載します。

5.2. RHSMVARS サブパラメーター

すべての Ansible パラメーターを把握するには、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	説明
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** 値を使用します。
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には、このパラメーターを使用してください。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="*OpenStack*"** を実行して、出力される **Pool ID** 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_repos** が設定されている場合には機能しません。
rhsm_autosubscribe	互換性のあるサブスクリプションをこのシステムに自動的にアタッチします。有効にするには、**true** に設定します。
rhsm_satellite_url	オーバークラウドノードを登録する Satellite Server のベース URL。
rhsm_repos	有効にするリポジトリーの一覧。**rhsm_activation_key** が設定されている場合には機能しません。
rhsm_username	登録用のユーザー名。可能な場合には、登録用の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_rhsm_proxy_hostname	HTTP プロキシーのホスト名。たとえば、**proxy.example.com**。
rhsm_rhsm_proxy_port	HTTP プロキシー通信用のポート。たとえば、**8080**。
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

rhsm コンポーザブルサービスがどのように機能し、どのように設定するかを理解したので、以下の手順に従って独自の登録情報を設定することができます。

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

以下の手順に従って、**rhsm** コンポーザブルサービスを有効化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 (**templates/rhsm.yml**) を作成します。
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/extraconfig/services/rhsm.yaml
parameter_defaults:
  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-2.8-for-rhel-8-x86_64-rpms
      - advanced-virt-for-rhel-8-x86_64-rpms
      - openstack-15-for-rhel-8-x86_64-rpms
      - rhceph-4-osd-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: "1a85f9223e3d5e43013e3d6e8ff506fd"
    rhsm_method: "portal"
```

   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。

特定のオーバークラウドロールに対して登録情報を提供することもできます。次の項では、その例を説明します。

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に 1 つの設定セットを適用し、コンピュート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 (**templates/rhsm.yml**) を作成します。
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/extraconfig/services/rhsm.yaml
parameter_defaults:
  ControllerParameters:
    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-2.8-for-rhel-8-x86_64-rpms
      - advanced-virt-for-rhel-8-x86_64-rpms
      - openstack-15-for-rhel-8-x86_64-rpms
      - rhceph-4-osd-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: "1a85f9223e3d5e43013e3d6e8ff506fd"
    rhsm_method: "portal"
ComputeParameters:
  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-2.8-for-rhel-8-x86_64-rpms
      - advanced-virt-for-rhel-8-x86_64-rpms
      - openstack-15-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: "1a85f9223e3d5e43013e3d6e8ff506fd"
    rhsm_method: "portal"
```

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 と **ComputeParameters** はいずれも、独自の **RhsmVars** パラメーターを使用して、サブスクリプションの情報をそれぞれのロールに渡します。

3. 環境ファイルを保存します。

これらの手順は、オーバークラウドで **rhsm** を有効化して設定します。ただし、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。従来の **rhel-registration** メソッドから Ansible ベースのメソッドに切り替えるには、以下の手順に従ってください。

5.5. 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。このメソッド用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、サービスを有効化する Heat リソースを変更します。

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

必要に応じて、以下を行ってください。

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/extraconfig/services/rhsm.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。

rhel-registration メソッドから **rhsm** メソッド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その値をマッピングします。

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

rhel-registration	rhsm / RhsmVars
rhel_reg_method	**rhsm_method**
rhel_reg_org	**rhsm_org_id**
rhel_reg_pool_id	**rhsm_pool_ids**

rhel-registration	rhsm / RhsmVars
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_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**

これで、**rhsm** サービスの環境ファイルの設定が完了し、オーバークラウドの次のデプロイメント操作で追加することができます。

5.7. 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。

本項では、**rhsm** の設定をオーバークラウドに適用する方法について説明します。

手順

1. **openstack overcloud deploy** に **rhsm.yml** 環境ファイルを追加します。

```
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
```

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。

5.8. 手動による ANSIBLE ベースの登録の実行

デプロイしたオーバークラウドで、手動による Ansible ベースの登録を実施することができます。そのためには、director の動的インベントリー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。以下の例で、Playbook を使用してコントローラーノードを手動で登録する方法を説明します。

手順

1. ノードを登録するための **redhat_subscription** モジュールを使用して、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-rpms
      - rhel-8-for-x86_64-appstream-rpms
      - rhel-8-for-x86_64-highavailability-rpms
      - ansible-2.8-for-rhel-8-x86_64-rpms
      - advanced-virt-for-rhel-8-x86_64-rpms
      - openstack-15-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
        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。
     - アクティベーションキーを使用してノードを登録する。
     - 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は、以下のコマンドを実行して、Ansible がオーバークラウドに対して Playbook (**ansible-osp-registration.yml**) 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、以下の処理が行われます。

- 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、コンピュートノード、異なるストレージノード種別など、事前定義されたロールの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、コア Heat テンプレートのアーキテクチャーにより、以下のようなタスクを行うことができます。

- カスタムロールの作成
- 各ロールへのサービスの追加と削除

これにより、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。

アーキテクチャー 1: 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。

アーキテクチャー 2: 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。「サポートされるカスタムロール」 を参照してください。

アーキテクチャー 3: 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

オーバークラウドの作成プロセスでは、**roles_data** ファイルを使用して、そのオーバークラウドのロールを定義します。**roles_data** ファイルには、YAML 形式のロール一覧が含まれます。**roles_data** 構文の短い例を以下に示します。

```
- 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** ファイルが **/usr/share/openstack-tripleo-heat-templates/roles_data.yaml** に含まれています。デフォルトのファイルは、以下のロール種別を定義します。

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

openstack overcloud deploy コマンドにより、デプロイ中にこのファイルが追加されます。このファイルは、**-r** 引数を使用して、カスタムの **roles_data** ファイルで上書きすることができます。以下に例を示します。

```
$ 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。

デフォルトロールのテンプレートを一覧表示するには、**openstack overcloud roles list** コマンドを使用します。

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

ロールの YAML 定義を確認するには、**openstack overcloud roles show** コマンドを使用します。

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

カスタムの **roles_data** ファイルを生成するには、**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ロールに統合します。たとえば、以下のコマンドは、**Controller**、**Compute**、**Networker** のロールを単一のファイルに統合します。

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

-o は、作成するファイルの名前を定義します。

これにより、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。これは、ネットワークサービスが **Controller** から **Networker** ロールにスケーリングされることを意味します。オーバークラウドは、**コントローラー** ノードと **Networker** ノードの間で、ネットワークサービスの負荷のバランスを取ります。

この **Networker** ロールをスタンドアロンにするには、独自のカスタム **Controller** ロールと、その他の必要なロールを作成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。

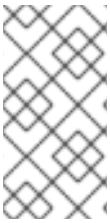
このディレクトリーを、コア Heat テンプレートコレクションから **stack** ユーザーのホームディレクトリーにコピーします。

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

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、前述したロールのサブコマンドに **--roles-path** オプションを指定します。以下に例を示します。

```
$ 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**、**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** および **NovaVcpuPinSet** 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**
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**
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 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のパラメーター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"
```

6.2.5. ロールパラメーターの考察

各ロールは、以下のパラメーターを使用します。

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。

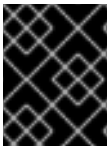
description

(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、最初に定義された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 **サブネット** 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 のデプロイ時に 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。事前プロビジョニング済みのノードでオーバークラウドをデプロイする場合に使用します。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。詳しくは、「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は、新規ロールの作成方法を指定するのに加えて、追加するサービスを定義します。

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。

overcloud.j2.yaml Heat テンプレートの各ロールのリソースの定義は、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

以下の例は、OpenStack Dashboard (**horizon**) のみをホストする新しい **Horizon** ロールを作成することを目的としています。このような場合には、新規ロールの情報が含まれるカスタムの **roles** ディレクトリーを作成します。

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 ~/.
```

~/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 サービスが含まれた **Horizon** ロールを新規作成します。以下に例を示します。

```
- 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
```

CountDefault を **1** に設定して、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すると良いでしょう。

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には、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
    ...
```

roles ディレクトリーをソースに使用して、新しい **roles_data** 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
```

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
```

以下のコマンドを実行して、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
```

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
```

openstack overcloud deploy コマンドの実行時には、新しい **roles_data** ファイルと環境ファイルを指定します。以下に例を示します。

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

デプロイメントが完了すると、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

コンポーザブルノード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 Platform 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。以下に例を示します。

```
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。この場合には、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 ~/.
```

~/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
```

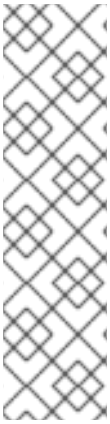
新しい **roles_data** ファイルを生成します。以下に例を示します。

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

openstack overcloud deploy コマンドを実行する際には、この新しい **roles_data** ファイルを指定します。以下に例を示します。

```
$ 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、以下のような内容を含む **environments/cinder-backup.yaml** ファイルを使用します。

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

これにより、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
```

ヒント

サービスの有効化/無効化の方法についてのその他の例は、『OpenStack Data Processing』の「Installation」セクションを参照してください。この項には、オーバークラウドで OpenStack Data Processing サービス (**sahara**) を有効にする手順が記載されています。

6.3.5. サービスなしの汎用ノードの作成

Red Hat OpenStack Platform では、OpenStack サービスを一切設定しない汎用の Red Hat Enterprise Linux 8 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 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、OpenStack Platform は Kibana や Sensu（『監視ツール 設定ガイド』を参照）などのモニタリングツールとの統合を提供します。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 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。

汎用ノードを作成するには、**ServicesDefault** 一覧なしの新規ロールが必要です。

```
- name: Generic
```

カスタムの **roles_data** ファイル (**roles_data_with_generic.yaml**) にそのロールを追加します。既存の **Controller** ロールと **Compute** ロールは必ず維持してください。

また、プロビジョニングするノードを選択する際には、必要な汎用 Red Hat Enterprise Linux 8 ノード数とフレーバーを指定する環境ファイル (**generic-node-params.yaml**) も追加することができます。以下に例を示します。

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

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。以下に例を示します。

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

このコマンドにより、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ノードではすべて、**OS::TripleO::Services::Podman** サービスを有効化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、ベースコンポーザブルサービスとともに **OS::TripleO::Services::Podman** サービスをコンテナー化されたサービスとして追加します。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、director の 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。これらのタグは、Puppet 実行をサービスの特定の設定リソースに制限するために OpenStack Platform で使用されます。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設定には、イメージの取得先およびその保存方法を定義するための初期レジストリーの設定が必要です。コンテナーイメージを準備するための環境ファイルを生成およびカスタマイズするには、以下の手順を実施します。

手順

1. アンダークラウドホストに stack ユーザーとしてログインします。

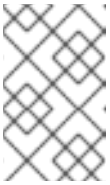
2. デフォルトのコンテナーイメージ準備ファイルを生成します。

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

上記のコマンドでは、以下の追加オプションを使用しています。

- **--local-push-destination**: コンテナーイメージの保管場所として、アンダークラウド上のレジストリーを設定します。つまり、director は必要なイメージを Red Hat Container Catalog からプルし、それをアンダークラウド上のレジストリーにプッシュします。director はこのレジストリーをコンテナーイメージのソースとして使用します。Red Hat Container Catalog から直接プルする場合には、このオプションを省略します。

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



注記

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

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

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	設定から除外するイメージの名前に含まれる文字列のリスト

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

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

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

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

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

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

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

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

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

7.5. イメージ準備エントリーの階層化

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

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

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

7.6. 準備プロセスにおけるイメージの変更

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

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

準備プロセス中にイメージを変更するには、変更する各イメージで Ansible ロールを呼び出します。ロールはソースイメージを取得して必要な変更を行い、その結果をタグ付けします。prepare コマンド

でイメージを目的のレジストリーにプッシュし、変更したイメージを参照するように Heat パラメーターを設定することができます。

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

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

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

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

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

7.7. コンテナーイメージの既存パッケージの更新

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

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

7.8. コンテナーイメージへの追加 RPM ファイルのインストール

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

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
```

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

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

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

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

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

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

USER "root"

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

USER "nova"
```

第8章 基本的なネットワーク分離

本章では、標準的なネットワーク分離構成のオーバークラウドを設定する方法について説明します。これには、以下の設定項目が含まれます。

- ネットワーク分離を有効にするためのレンダリング済み環境ファイル(**/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にコピーした環境ファイル (**/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml**)
- IP 範囲、サブネット、および仮想 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。

本章の以下のセクションでは、これらの各項目を定義する方法を説明します。

8.1. 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。director のコア 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。以下に例を示します。

```
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 (「9章 *カスタムコンポーザブルネットワーク*」および「10章 *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、このネットワーク分離の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ファイルの名前を含める必要があります。以下に例を示します。

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

8.2. 分離ネットワーク設定の変更

network_data ファイルで、デフォルトの分離ネットワーク設定を定義します。本手順では、カスタム **network_data** ファイルを作成し、そのファイルをネットワーク要件に応じて設定する方法について説明します。

手順

1. デフォルトの **network_data** ファイルのコピーします。

   ```
   $ 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、以下の項目を編集します。

- **vlan** は、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** は、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** は、ネットワークのゲートウェイを設定します。主に外部ネットワークのデフォルトルートを定義するために使用されますが、必要であれば他のネットワークに使用することもできます。

-n オプションを使用して、カスタム **network_data** 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。

8.3. ネットワークインターフェースのテンプレート

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 構文のメリットを生かしてテンプレートを容易に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、「10章 *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。

8.4. 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ために使用す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** だけ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**

NIC ディレクトリー	説明	環境ファイル
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**
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** 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ネットワーク用の環境ファイル (例: **net-bond-with-vlans-no-external.yaml**) や IPv6 を使用するネットワーク用の環境ファイル (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ファイルをさらに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**

8.5. 基本的なネットワーク分離の有効化

以下の手順で、デフォルト NIC テンプレートの 1 つを使用して基本的なネットワーク分離を有効にする方法について説明します。ここでは、単一 NIC および VLAN のテンプレート (**single-nic-vlans**) を用いています。

手順

1. **openstack overcloud deploy** コマンドを実行する際に、以下に示すファイルのレンダリング済み環境ファイル名を含めるようにしてください。
   - カスタム **network_data** 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
   - 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 \
  ...
```

第9章 カスタムコンポーザブルネットワーク

本章では、「*8章 基本的なネットワーク分離*」で説明した概念および手順に続いて、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方法について説明します。これには、以下のファイルおよびテンプレートの設定が含まれます。

- 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。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。

本章の以下のセクションでは、これらの各項目を定義する方法を説明します。

9.1. 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
- ストレージ
- 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段階中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は、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、デプロイされ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ファイル (**/usr/share/openstack-tripleo-heat-templates/network_data.yaml**) が使用されます。

9.2. コンポーザブルネットワークの追加

以下の手順で、オーバークラウドにさらにコンポーザブルネットワークを追加する方法について説明します。

手順

1. デフォルトの **network_data** ファイルのコピーします。

```
$ 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。以下に例を示します。

```
- 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 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** 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。構文の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、上記の **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頻繁に使用されます。詳しくは、『Spine Leaf Networking』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** ファイルをデプロイメントに含めます。**-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。

9.3. 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。

以下の手順で、オーバークラウドのロールにコンポーザブルネットワークを追加する方法について説明します。

手順

1. カスタム **roles_data** 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** ファイルを編集します。

3. コンポーザブルネットワークを追加するロールまでスクロールし、**networks** の一覧にネットワーク名を追加します。たとえば、ネットワークを Ceph Storage ロールに追加するには、以下のスニペットをガイドとして使用します。

```
- 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 **roles_data** 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。

9.4. 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。

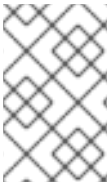
たとえば、ハイライトしたセクションを変更して、Storage Management ネットワークサービスを Storage Backup ネットワークに再割り当てすることができます。

```
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** パラメーターの定義を **ServiceNetMapDefaults** から取得した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** 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

デフォルトのサービスの全一覧は、**/usr/share/openstack-tripleo-heat-templates/network/service_net_map.j2.yaml** 内の **ServiceNetMapDefaults** パラメーターの箇所に記載されています。

9.5. カスタムコンポーザブルネットワークの有効化

以下の手順で、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する方法について説明します。ここでは、単一 NIC および VLAN (**single-nic-vlans**) を用いています。

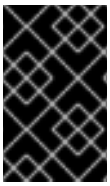
手順

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** 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** 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。

第10章 カスタムネットワークインターフェーステンプレート

本章では、「*8章 基本的なネットワーク分離*」で説明した概念および手順に続いて、ご自分の環境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方法について説明します。これには、以下の設定項目が含まれます。

- 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。この場合、カスタムテンプレートのベースとしてデフォルトをレンダリングします。
- 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。
- ネットワークのカスタマイズを使用する場合には、カスタム **network_data** 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** ファイルおよびカスタム **roles_data** ファイル

10.1. 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。この場合、サービス/NIC 間の割り当ては以下のマッピングとなります。

- 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

10.2. 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ために、以下の手順で、Jinja2 構文のデフォルト NIC テンプレートをレンダリングする方法について説明します。これにより、レンダリングしたテンプレートをカスタム設定のベースとして使用することができます。

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。以下に例を示します。

```
$ ./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することができます。

10.3. ネットワークインターフェースのアーキテクチャー

本項では、**custom-nics** のカスタム NIC テンプレートのアーキテクチャーおよびテンプレートを編集する際の推奨事項について説明します。

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、編集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:
```

これは、**os-net-config** がノードでネットワークプロパティーを設定するのに使用する設定ファイルを作成するスクリプト (**run-os-net-config.sh**) を実行します。**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。

10.4. ネットワークインターフェースの参照

以下のセクションでは、ネットワークインターフェースの種別および各ネットワークインターフェースで使用されるパラメーターを定義します。

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) または番号付きのインターフェース (nic1、nic2、nic3) を使用して各インターフェースを定義します。

以下に例を示します。

```
- 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。

オプション	デフォルト	説明
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
routes		インターフェース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。
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

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

オプション	デフォルト	説明
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。「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。「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
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	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。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: 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、設定データを取得するために Neutron サーバーに接続します。OpenStack の制御トラフィック (通常はコントロールプレーンと Internal API のネットワーク) が OVS ブリッジに配置されると、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、ダウンタイムが生じます。このような状況でダウンタイムが許されない場合には、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べきです。

- 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 上に配置し、OVS ブリッジを 2 番目のインターフェースに配置すると、最小の設定にすることができます。
- ボンディングを使用する場合には、最小で 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は Linux ボンディング (Linux ブリッジ) に配置すべきです。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。「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、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"
```

nic2 が **primary: true** と設定されている点に注意してください。これにより、ボンディングが必ず **nic2** の MAC アドレスを使用するようになります。

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
use_dhcp	False	DHCP を使用して IP アドレスを取得します。

オプション	デフォルト	説明
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。「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、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. ネットワークインターフェースレイアウトの例

以下のコントローラーノード NIC テンプレートのスニペットは、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ため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、タグ付けられた複数の VLAN デバイスを、番号付きのインターフェース (**nic1** から **nic4**) に割り当てます。**nic3** では、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

10.6. 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固定のテンプレートに、**network_data** および **roles_data** ファイルで定義した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 テンプレートで、テンプレートが含まれていることを確認します。

- カスタムロールを含む各ロールの静的ファイル
- 各ロールのそれぞれの固定ファイルに、正しいネットワーク定義が含まれている

カスタムネットワークがロールで使用されなくても、各固定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ていることを確認してください。たとえば、**StorageBackup** ネットワークが Ceph ノードだけに追加される場合でも、すべてのロールで NIC 設定テンプレートの **parameters** セクションに以下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
```

10.7. 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。

10.8. 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、デフォルトでは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、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)

パラメーター	説明	タイプ
***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には、**network_data** ファイルで規定するネットワークの **ip_subnet** が自動的に設定されます。例: **InternalApiNetCidr: 172.16.0.0/24**	文字列
***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には、**network_data** ファイルで規定するネットワークの **allocation_pools** が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**	特定の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には、**network_data** ファイルで規定するネットワークの **vlan** が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または他のネットワークへのルートとして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には、**network_data** ファイルで規定するネットワークの **gateway_ip** が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

パラメーター	説明	タイプ
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。コントロールプレーン上のアンダークラウドの IP アドレスに設定されます。	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
NeutronExternalNetworkBridge	OpenStack Networking (neutron) に使用する外部ブリッジ名のレガシー値。**NeutronBridgeMappings** で複数の物理ブリッジを定義することができるように、この値はデフォルトでは空欄になっています。通常は、この値を上書きしないでください。	文字列
NeutronFlatNetworks	フラットなネットワークが neutron プラグインで設定されるように定義します。External ネットワークを作成できるようにデフォルトは「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。ホストの外部ブリッジ (**br-ex**) を物理名 (**datacentre**) にマッピングするようにデフォルト設定されています。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

パラメーター	説明	タイプ
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インターフェースを **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
NeutronNetworkType	OpenStack Networking (neutron) のテナントネットワークタイプ。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、最初に指定したタイプが使用されます。その後、次のタイプが使用されます。例: **NeutronNetworkType: "vxlan"**	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'	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できる 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できる 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全くない場合を除き、このパラメーターは有効のままにしてください。デフォルトでは有効に設定されています。	ブール値

パラメーター	説明	タイプ
NeutronNetworkVLANRanges	サポートされる 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 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

10.9. 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: """
```

10.10. カスタム NIC を使用したネットワーク分離の有効化

以下の手順で、カスタム NIC テンプレートを使用してネットワーク分離を有効にする方法について説明します。

手順

1. **openstack overcloud deploy** コマンドを実行する際に、以下に示すファイルを含めるようにしてください。
   - カスタム **network_data** 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** および **roles_data** ファイルを含めます。

第11章 その他のネットワーク設定

本章では、「10章 *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。

11.1. 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 アドレスでインフラストラクチャーネットワークへ接続するための 2 つ目の NIC、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
```

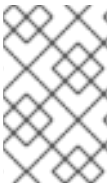
ネットワークインターフェースのテンプレートは、実際のインターフェース名 (**eth0**、**eth1**、**enp0s25**) または番号付きのインターフェース (**nic1**、**nic2**、**nic3**) のいずれかを使用します。名前付きのインターフェース (**eth0**、**eno2** など) ではなく、番号付きのインターフェース (**nic1**、**nic2** など) を使用した場合には、ロール内のホストのネットワークインターフェースは、全く同じである必要はありません。たとえば、あるホストに **em1** と **em2** のインターフェースが指定されており、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。

番号付きのインターフェースの順序は、名前付きのネットワークインターフェースのタイプの順序と同じです。

- **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。

番号付きの NIC スキームは、ライブのインターフェース (例: スイッチに接続されているケーブル) のみ考慮します。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に、各ホストで **nic1** から **nic4** を使用してケーブル 4 本のみを結線します。

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。これにより、**nic1**、**nic2**・・・としてマッピングする物理 NIC を事前に定義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。これぞれのノードで、適用された設定が **/etc/os-net-config/mapping.yaml** の **interface_mapping** に表示されます。

11.2. 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を設定するには、2 とおりの方法があります。インターフェースが DHCP を使用しており、DHCP がゲートウェイアドレスを提供している場合には、システムは対象のゲートウェイに対してデフォルトルートを使用します。それ以外の場合には、静的な IP を使用する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が指定されたゲートウェイのみを使用します。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 を使用して別の DHCP インターフェース (**nic2**) 上のデフォルトの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 で取得したルートのみに適用されます。

静的な 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、サブネットに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 を経由するサブネット 10.1.2.0/24 にルートを設定します。

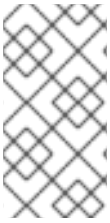
```
  - type: vlan
    device: bond1
    vlan_id:
      get_param: InternalApiNetworkVlanID
    addresses:
    - ip_netmask:
        get_param: InternalApiIpSubnet
    routes:
    - ip_netmask: 10.1.2.0/24
      next_hop: 172.17.0.1
```

11.3. 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。テストでは、VXLAN トンネルと合わせてジャンボフレームを使用した場合に、プロジェクトのネットワークスループットが大幅に向上しました。



注記

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、Floating IP インターフェースの MTU はデフォルトの 1500 のままにしておくことを推奨します。変更すると、接続性の問題が発生する可能性があります。これは、ルーターが通常レイヤー 3 の境界を超えてジャンボフレームでのデータ転送ができないのが理由です。

```
- 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
```

11.4. FLOATING IP のためのネイティブ VLAN の設定

Neutron は、Neutron の外部のブリッジ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。このモデルにより、VLAN または複数の物理接続のいずれかを使用した複数の Floating IP ネットワークが可能となります。

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 つとなり、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。

11.5. 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には、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。反対に、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。

第12章 ネットワークインターフェースボンディング

本章では、カスタムネットワーク設定で使用することのできるボンディングのオプションを定義します。

12.1. 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。ボンディング構成にすることにより、高可用性システムに冗長性を持たせたり、スループットを向上させたり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。

ボンディングは、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。

仮想マシンインスタンスと直接データをやり取りするネットワークについては、Red Hat では LACP と共に OVS カーネルボンディング (ボンディングタイプ: ovs_bond) または OVS-DPDK ボンディング (ボンディングタイプ: ovs_dpdk_bond) を使用することを推奨します。ただし、OVS カーネルボンディングと OVS-DPDK ボンディング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は、Red Hat では VLAN を使用する Linux ボンディングと LACP の組み合わせを推奨します。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 ボンディングの設定例を以下に示します。

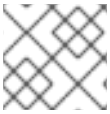
```
params:
          $network_config:
            network_config:

            - 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
```

12.2. 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。以下の表は、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 のサポートについてまとめています。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。 • 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。 • 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。 • LACP を有効にする必要があります。	active または passive

以下の例に示すように、ネットワークの環境ファイルで 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"
```

12.3. 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。以下に例を示します。

```
    - 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 (これは、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)。

12.4. 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12.1 ボンディングオプション

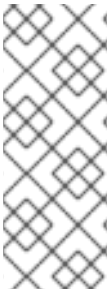
bond_mode=balance-slb	送信元の MAC アドレスと出力の VLAN に基づいてフローのバランスを取り、トラフィックパターンの変化に応じて定期的にリバランスを行います。ボンディングを **balance-slb** に設定することにより、リモートスイッチについての知識や協力なしに限定された形態の負荷分散が可能となります。SLB は送信元 MAC アドレスと VLAN の各ペアをリンクに割り当て、そのリンクを介して、対象の MAC アドレスと LAN からのパケットをすべて伝送します。このモードはトラフィックパターンの変化に応じて定期的にリバランスを行う、送信元 MAC アドレスと 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)。無効にするにはゼロに設定します。

第13章 ノード配置の制御

director のデフォルトの動作では、通常プロファイルタグに基づいて、各ロールにノードが無作為に選択されます。ただし、director には特定のノード配置を定義する機能も備えられています。この機能は、以下の場合に役立ちます。

- **controller-0**、**controller-1** などの特定のノード ID 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。静的に定義された 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。

13.1. 特定のノード ID の割り当て

以下の手順では、特定のノードにノード ID を割り当てます。ノード ID には、**controller-0**、**controller-1**、**compute-0**、**compute-1** などがあります。

最初のステップでは、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。以下に例を示します。

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

これにより、**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 (コントローラー、コンピュート、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しません。

次のステップでは、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。以下に例を示します。

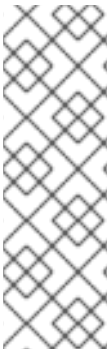
```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
```

これらのスケジューラーヒントを使用するには、オーバークラウドの作成時に、**overcloud deploy** コマンドに scheduler_hints_env.yaml 環境ファイルを追加します。

これらのパラメーターを使用してロールごとに、同じアプローチを使用することができます。

- コントローラーノードの場合は **ControllerSchedulerHints**。
- コンピュートノードの場合は **ComputeSchedulerHints**。

- Block Storage ノードの場合は **BlockStorageSchedulerHints**。
- Object Storage ノードの場合は **ObjectStorageSchedulerHints**。
- Ceph Storage ノードの場合は **CephStorageSchedulerHints**。
- カスタムロールの場合は **[ROLE]SchedulerHints**。**[ROLE]** はロール名に置き換えます。



注記

プロファイル照合よりもノードの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** など) ではなく、デプロイメントにデフォルトの **baremetal** フレーバーを使用します。以下に例を示します。

```
$ openstack overcloud deploy ... --control-flavor baremetal --compute-flavor baremetal
...
```

13.2. 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。

ノードのホスト名をカスタマイズするには、「特定のノード ID の割り当て」 で作成した scheduler_hints_env.yaml ファイルなどの環境ファイルの **HostnameMap** パラメーターを使用します。以下に例を示します。

```
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-compute-0: overcloud-compute-prod-abc-0
```

parameter_defaults セクションで **HostnameMap** を定義し、各マッピングは、**HostnameFormat** パラメーターを使用して Heat が定義する元のホスト名に設定します (例: **overcloud-controller-0**)。また、2 つ目の値は、ノードに指定するカスタムのホスト名 (例: **overcloud-controller-prod-123-0**) にします。

ノード ID の配置と合わせてこの手法を使用することで、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。

13.3. 予測可能な IP の割り当て

作成された環境でさらに制御を行う場合には、director はオーバークラウドノードに各ネットワークの固有の IP を割り当てることもできます。コア Heat テンプレートコレクションにある **environments/ips-from-pool-all.yaml** 環境ファイルを使用します。このファイルを **stack** ユーザーの **templates** ディレクトリーにコピーしてください。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/ips-from-pool-all.yaml ~/templates/.
```

ips-from-pool-all.yaml ファイルには、2 つの主要セクションがあります。

1 番目のセクションは、デフォルトよりも優先される **resource_registry** の参照セットです。この参照では、director に対して、ノード種別のある特定のポートに特定の IP を使用するように指示を出します。適切なテンプレートの絶対パスを使用するように各リソースを編集してください。以下に例を示します。

```
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/external_from_pool.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/internal_api_from_pool.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/storage_from_pool.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/storage_mgmt_from_pool.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/tenant_from_pool.yaml
```

デフォルトの設定では、全ノード種別上にあるすべてのネットワークが、事前に割り当てられた IP を使用するように設定します。特定のネットワークやノード種別がデフォルトの IP 割り当てを使用するように許可するには、環境ファイルからノード種別やネットワークに関連する **resource_registry** のエントリーを削除するだけです。

2 番目のセクションは、実際の IP アドレスを割り当てる parameter_defaults です。各ノード種別には、関連するパラメーターが指定されます。

- コントローラーノードの **ControllerIPs**
- コンピュートノードの場合は **ComputeIPs**。
- Ceph Storage ノードの場合は **CephStorageIPs**。
- Block Storage ノードの場合は **BlockStorageIPs**。
- Object Storage ノードの場合は **SwiftStorageIPs**。
- カスタムロールの場合は **[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、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最低でも必要です。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、オーバークラウドに 3 つの Ceph Storage ノードが含まれる場合には、**CephStorageIPs** パラメーターは以下のようになります。

```
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
```

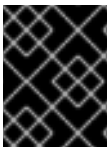
選択した IP アドレスは、ネットワーク環境ファイルで定義されている各ネットワークの割り当てプールの範囲に入らないようにしてください (「カスタムネットワーク環境ファイル」を参照)。たとえば、**internal_api** の割り当ては **InternalApiAllocationPools** の範囲外となるようにします。これにより、自動的に選択される IP アドレスと競合が発生しないようになります。また同様に、標準の予測可

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** など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。

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 **ips-from-pool-all.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 **ips-from-pool-all.yaml** ファイルを追加してください。

以下に例を示します。

```
$ openstack overcloud deploy --templates \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/ips-from-pool-all.yaml \
  [OTHER OPTIONS]
```

13.4. 予測可能な仮想 IP の割り当て

director は、各ノードの予測可能な 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) を定義する同様の機能も提供します。そのためには、「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。

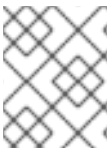
```
parameter_defaults:
  ...
  # Predictable VIPs
  ControlFixedIPs: [{'ip_address':'192.168.201.101'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.16.0.9'}]
  PublicVirtualFixedIPs: [{'ip_address':'10.1.1.9'}]
  StorageVirtualFixedIPs: [{'ip_address':'172.18.0.9'}]
  StorageMgmtVirtualFixedIPs: [{'ip_address':'172.19.0.9'}]
  RedisVirtualFixedIPs: [{'ip_address':'172.16.0.8'}]
```

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。

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。

第14章 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。これは、オーバークラウドの設定に、パブリック API エンドポイントに SSL/TLS を有効化するための追加の環境ファイルが必要であることを意味します。本章では、SSL/TLS 証明書を設定して、オーバークラウドの作成の一部として追加する方法を説明します。



注記

このプロセスでは、パブリック API のエンドポイントの SSL/TLS のみを有効化します。Internal API や Admin API は暗号化されません。

このプロセスには、パブリック API のエンドポイントを定義するネットワークの分離が必要です。

14.1. 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。

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には、空のファイルを作成します。

```
$ sudo touch /etc/pki/CA/index.txt
```

/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。ファイルが存在しない場合には、新規ファイルを作成して新しい開始値を指定します。

```
$ echo '1000' | sudo tee /etc/pki/CA/serial
```

14.2. 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。

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
```

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。

14.3. 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外部クライアントについては、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/
```

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。

14.4. 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
```

14.5. SSL/TLS 証明書署名要求の作成

次の手順では、オーバークラウドの証明書署名要求を作成します。カスタマイズするデフォルトの OpenSSL 設定ファイルをコピーします。

```
$ cp /etc/pki/tls/openssl.cnf .
```

カスタムの **openssl.cnf** ファイルを編集して、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。変更するパラメーターの種別には以下のような例が含まれます。

```
[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 でアクセスするために 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されている範囲から最初の IP アドレスを割り当てます。
- 完全修飾ドメイン名を使用して SSL/TLS でアクセスする場合には、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。

次のコマンドを実行し、手順 1 で作成したキーストアより公開鍵を使用して証明書署名要求を生成します (**server.csr.pem**)。

```
$ 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** オプションで必ず指定してください。

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。

14.6. 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
```

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。

14.7. SSL/TLS の有効化

Heat テンプレートコレクションから **enable-tls.yaml** の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/.
```

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**)のコンテンツを **SSLCertificate** パラメーターにコピーします。以下に例を示します。

```
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。

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。以下に例を示します。

```
parameter_defaults:
  ...
  SSLKey: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0lVO
    ...
    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。

14.8. 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。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、アンダークラウドおよびオーバークラウドの証明書を署名するのに使用する両方の 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-----
        MIIDlTCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCS
        BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBw
        UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBA
        -----END CERTIFICATE-----
    overcloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDBzCCAe+gAwIBAgIJAIc75A7FD++DMA0GCS
        BAMMD3d3dy5leGFtcGxlLmNvbTAeFw0xOTAxMz
        Um54yGCARyp3LpkxvyfMXX1DokpS1uKi7s6CkF
        -----END CERTIFICATE-----
```

重要

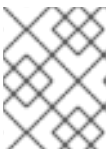
この認証局の内容に新しく追加する行は、すべて同じレベルにインデントする必要があります。

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。

14.9. DNS エンドポイントの設定

DNS ホスト名を使用して SSL/TLS でオーバークラウドにアクセスする場合は、**custom-domain.yaml** ファイルを **/home/stack/templates** にコピーする必要があります。このファイルは、**/usr/share/tripleo-heat-templates/environments/predictable-placement/** にあります。

1. 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。



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。

```
# title: Custom Domain Name
# description: |
#   This environment contains the parameters that need to be set in order to
#   use a custom domain name and have all of the various FQDNs reflect it.
parameter_defaults:
```

```
  # The DNS domain used for the hosts. This must match the overcloud_domain_name
configured on the undercloud.
  # Type: string
  CloudDomain: localdomain

  # The DNS name of this cloud. E.g. ci-overcloud.tripleo.org
  # Type: string
  CloudName: overcloud.localdomain

  # The DNS name of this cloud's provisioning network endpoint. E.g. 'ci-
overcloud.ctlplane.tripleo.org'.
  # Type: string
  CloudNameCtlplane: overcloud.ctlplane.localdomain

  # The DNS name of this cloud's internal_api endpoint. E.g. 'ci-
overcloud.internalapi.tripleo.org'.
  # Type: string
  CloudNameInternal: overcloud.internalapi.localdomain

  # The DNS name of this cloud's storage endpoint. E.g. 'ci-overcloud.storage.tripleo.org'.
  # Type: string
  CloudNameStorage: overcloud.storage.localdomain

  # The DNS name of this cloud's storage_mgmt endpoint. E.g. 'ci-
overcloud.storagemgmt.tripleo.org'.
  # Type: string
  CloudNameStorageManagement: overcloud.storagemgmt.localdomain
```

2. 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"]
  ....
```

14.10. 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**) は、**-e** オプションを使用して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。

以下に例を示します。

```
$ 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
```

14.11. SSL/TLS 証明書の更新

将来に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.yaml** ファイルを編集して、**SSLRootCertificate** パラメーターを更新してください。

新規証明書の内容が記載されたら、デプロイメントを再度実行します。以下に例を示します。

```
$ openstack overcloud deploy --templates [...] -e /home/stack/templates/enable-tls.yaml -e
~/templates/cloudname.yaml -e ~/templates/inject-trust-anchor.yaml -e /usr/share/openstack-tripleo-
heat-templates/environments/ssl/tls-endpoints-public-dns.yaml
```

第15章 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。このプロセスには、**novajoin** を使用してオーバークラウドノードを IdM サーバーに登録するプロセスが必要です。

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。

15.1. 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は、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。

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
```

以下の項では、ここで設定され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。

15.2. IDM へのアンダークラウドの追加

以下の手順では、アンダークラウドを IdM に登録して novajoin を設定します。**undercloud.conf** で以下の設定を行います (**[DEFAULT]** セクション内)。

1. novajoin サービスは、デフォルトで無効にされます。有効にするには、以下のように設定します。

```
[DEFAULT]
enable_novajoin = true
```

2. アンダークラウドノードをIdM に登録するためのワンタイムパスワード (OTP) を設定する必要があります。

```
ipa_otp = <otp>
```

3. neutron の DHCP サーバーにより提供されるように、オーバークラウドのドメイン名を設定します。

```
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. より大きな環境の場合には、novajoin の接続タイムアウト値を確認する必要があります。**undercloud.conf** 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
```

検証

1. アンダークラウドのキーエントリーについて **keytab** ファイルを確認します。

```
[root@undercloud-0 ~]# klist -kt
Keytab name: FILE:/etc/krb5.keytab
KVNO Timestamp           Principal
---- ------------------- ------------------------------------------------------
   1 04/28/2020 12:22:06 host/undercloud-0.redhat.local@REDHAT.LOCAL
   1 04/28/2020 12:22:06 host/undercloud-0.redhat.local@REDHAT.LOCAL


[root@undercloud-0 ~]# klist -kt /etc/novajoin/krb5.keytab
Keytab name: FILE:/etc/novajoin/krb5.keytab
KVNO Timestamp           Principal
---- ------------------- ------------------------------------------------------
   1 04/28/2020 12:22:26 nova/undercloud-0.redhat.local@REDHAT.LOCAL
   1 04/28/2020 12:22:26 nova/undercloud-0.redhat.local@REDHAT.LOCAL
```

2. ホストプリンシパルを使用して、システム **/etc/krb.keytab** ファイルをテストします。

```
[root@undercloud-0 ~]# kinit -k
[root@undercloud-0 ~]# klist
Ticket cache: KEYRING:persistent:0:0
```

```
Default principal: host/undercloud-0.redhat.local@REDHAT.LOCAL

Valid starting       Expires              Service principal
05/04/2020 10:34:30  05/05/2020 10:34:30  krbtgt/REDHAT.LOCAL@REDHAT.LOCAL

[root@undercloud-0 ~]# kdestroy
Other credential caches present, use -A to destroy all
```

3. nova プリンシパルを使用して novajoin **/etc/novajoin/krb.keytab** ファイルをテストします。

```
[root@undercloud-0 ~]# kinit -kt /etc/novajoin/krb5.keytab 'nova/undercloud-
0.redhat.local@REDHAT.LOCAL'
[root@undercloud-0 ~]# klist
Ticket cache: KEYRING:persistent:0:0
Default principal: nova/undercloud-0.redhat.local@REDHAT.LOCAL

Valid starting       Expires              Service principal
05/04/2020 10:39:14  05/05/2020 10:39:14  krbtgt/REDHAT.LOCAL@REDHAT.LOCAL
```

15.3. 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を検討してください。

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。

15.4. NOVAJOIN を使用するためのオーバークラウドの設定

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。以下に例を示します。

```
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 \
     ```

     その結果、デプロイされるオーバークラウドノードは自動的に IdM で登録されるようになります。

4. これで設定されるのは、内部エンドポイント向けの TLS のみです。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キーを追加するよう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。そのため、**openstack deploy** コマンドは以下のようになります。

   ```
   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 /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。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 \
      -e /usr/share/openstack-tripleo-heat-templates/environments/services/haproxy-public-tls-certmonger.yaml
   ```

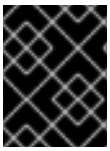
第16章 既存デプロイメントの TLS を使用するデプロイメントへの変換

既存のオーバークラウドおよびアンダークラウドエンドポイントを、TLS 暗号化を使用するように設定することができます。この方法では、**novajoin** を使用してデプロイメントを Red Hat Identity Management(IdM)と統合し、DNS、Kerberos、および certmonger へのアクセスが可能になります。それぞれのオーバークラウドノードは、certmonger クライアントを使用して各サービスの証明書を取得します。

TLS についての詳細は、『Security and Hardening Guide 』を参照してください。

16.1. 要件

- 既存の IdM デプロイメントが必要です。また、OpenStack デプロイメントに DNS サービスを提供する必要もあります。
- 既存のデプロイメントは、パブリックエンドポイントに FQDN を使用している必要があります。デフォルトの設定では、IP アドレスベースのエンドポイントが使用されている可能性があり、したがって IP アドレスベースの証明書が生成されます。以下の手順に進む前に、これを FQDN に変更する必要があります。



重要

この手順の実施中、オーバークラウドおよびアンダークラウドのサービスが利用できなくなります。

16.2. エンドポイントの確認

デフォルトでは、既存の Red Hat OpenStack Platform オーバークラウドは特定のエンドポイントを TLS で暗号化しません。たとえば、以下の出力には、**https** の代わりに **http** を使用する URL が含まれますが、これは暗号化されません。

```
+----------------------------------+-----------+--------------+--------------+---------+-----------+----------------------------------+
| ID                               | Region    | Service Name | Service Type | Enabled | Interface | URL                              |
+----------------------------------+-----------+--------------+--------------+---------+-----------+----------------------------------+
| 0ad11e943e1f4ff988650cfba57b4031 | regionOne | nova         | compute      | True    | internal  | http://172.16.2.17:8774/v2.1                        |
| 1413eb9ef38a45b8bee1bee1b0dfe744 | regionOne | swift        | object-store | True    | public    | https://overcloud.lab.local:13808/v1/AUTH_%(tenant_id)s |
| 1a54f13f212044b0a20468861cd06f85 | regionOne | neutron      | network      | True    | public    | https://overcloud.lab.local:13696                   |
| 3477a3a052d2445697bb6642a8c26a91 | regionOne | placement    | placement    | True    | internal  | http://172.16.2.17:8778/placement                   |
| 3f56445c0dd14721ac830d6afb2c2cd4 | regionOne | nova         | compute      | True    | admin     | http://172.16.2.17:8774/v2.1                        |
| 425b1773a55c4245bcbe3d051772ebba | regionOne | glance       | image        | True    | internal  | http://172.16.2.17:9292                             |
| 57cf09fa33ed446f8736d4228bdfa881 | regionOne | placement    | placement    | True    | public    | https://overcloud.lab.local:13778/placement         |
| 58600f3751e54f7e9d0a50ba618e4c54 | regionOne | glance       | image        | True    | public    | https://overcloud.lab.local:13292                   |
```

```
| 5c52f273c3284b068f2dc885c77174ca | regionOne | neutron      | network      | True    | internal  |
http://172.16.2.17:9696                          |
| 8792a4dd8bbb456d9dea4643e57c43dc | regionOne | nova         | compute      | True    | public    |
https://overcloud.lab.local:13774/v2.1           |
| 94bbea97580a4c4b844478aad5a85e84 | regionOne | keystone     | identity     | True    | public    |
https://overcloud.lab.local:13000                |
| acbf11b5c76d44198af49e3b78ffedcd | regionOne | swift        | object-store | True    | internal  |
http://172.16.1.9:8080/v1/AUTH_%(tenant_id)s     |
| d4a1344f02a74f7ab0a50c5a7c13ca5c | regionOne | keystone     | identity     | True    | internal  |
http://172.16.2.17:5000                          |
| d86c241dc97642419ddc12533447d73d | regionOne | placement    | placement    | True    | admin
| http://172.16.2.17:8778/placement                |
| de7d6c34533e4298a2752852427a7030 | regionOne | glance       | image        | True    | admin     |
http://172.16.2.17:9292                          |
| e82086062ebd4d4b9e03c7f1544bdd3b | regionOne | swift        | object-store | True    | admin     |
http://172.16.1.9:8080                           |
| f8134cd9746247bca6a06389b563c743 | regionOne | keystone     | identity     | True    | admin     |
http://192.168.24.6:35357                        |
| fe29177bd29545ca8fdc0c777a7cf03f | regionOne | neutron      | network      | True    | admin     |
http://172.16.2.17:9696                          |
+----------------------------------+-----------+--------------+--------------+---------+-----------+------------------
-----------------------------------+
```

以降のセクションで、これらのエンドポイントを TLS を使用して暗号化する方法について説明します。

16.3. 既知の問題に対する回避策の適用

現在、TLS Everywhere のその場アップグレードには、オーバークラウドノードを IdM に登録することができないという既知の問題があります。回避策としては、overcloud deploy を実行する前に、すべてのオーバークラウドノードから **/etc/ipa/ca.crt/** を削除する方法があります。詳細は、Bug 1732564 を参照してください。

回避策を適用する 1 つの方法として、以下のスクリプトが挙げられます。デプロイメントに合わせてこのスクリプトを変更しなければならない場合があります。

```
[stack@undercloud-0 ~]$ vi rm-ca.crt-dir.sh
#!/bin/bash

source /home/stack/stackrc
NODES=$(openstack server list -f value -c Networks|sed s/ctlplane=//g)

for NODE in $NODES
do
    ssh heat-admin@$NODE sudo rm -rf /etc/ipa/ca.crt/
Done

[stack@undercloud-0 ~]$ bash rm-ca.crt-dir.sh
```

16.4. TLS を使用するエンドポイントの設定

本セクションでは、既存のデプロイメントで TLS によるエンドポイントの暗号化を有効にする方法、さらにエンドポイントが正しく設定されていることを確認する方法について説明します。

TLS everywhere を有効にする場合、ドメインの構造化方法に応じて、さまざまなアップグレードパスを利用することができます。以下の例では、サンプルドメイン名を使用してアップグレードパスを説明します。

- オーバークラウドドメイン(**lab.local**)が IdM ドメイン(**lab.local**)と一致する **内部** エンドポイントおよび **admin** エンドポイントで、既存のパブリックエンドポイント証明書を再利用して、TLS everywhere を有効にします。
- オーバークラウドドメイン(**lab.local**)が IdM ドメイン(**lab.local**)と一致する **内部** および **admin** エンドポイントで、IdM が新しいパブリックエンドポイント証明書を発行できるようにし、TLS everywhere を有効にすることが可能です。
- オーバークラウドドメイン(**site1.lab.local)が IdM ドメイン(lab.local**)のサブドメインである **内部** および **admin** エンドポイントで、既存のパブリックエンドポイント証明書を再利用して、TLS everywhere を有効にします。
- オーバークラウドドメイン(**site1.lab.local)が IdM ドメイン(lab.local**)のサブドメインである場合に、IdM が新しいパブリックエンドポイント証明書を発行し、**内部** および **admin** エンドポイントで TLS everywhere を有効にすることを許可します。

本セクションの以降の手順で、上記のさまざまな組み合わせを使用して、この統合を設定する方法について説明します。

16.4.1. IdM と同じドメインを使用するデプロイメントのアンダークラウドインテグレーションの設定

以下の手順では、IdM と同じドメインを使用するデプロイメントのアンダークラウドインテグレーションを設定する方法について説明します。

Red Hat OpenStack Platform は **novajoin** を使用して Red Hat Identity Management(IdM)と統合し、暗号化証明書を発行および管理します。以下の手順では、アンダークラウドを IdM に登録し、トークンを生成し、アンダークラウド設定でトークンを有効にしてから、アンダークラウドおよびオーバークラウドのデプロイメントスクリプトを再実行します。以下に例を示します。

1. IdM と統合するために **python-novajoin** をインストールします。

```
[stack@undercloud-0 ~]$ sudo yum install python-novajoin
```

2. **novajoin** 設定スクリプトを実行し、IdM デプロイメントの設定情報を指定します。以下に例を示します。

```
[stack@undercloud-0 ~]$ sudo novajoin-ipa-setup --principal admin --password
ComplexRedactedPassword \
  --server ipa.lab.local --realm lab.local --domain lab.local \
  --hostname undercloud-0.lab.local --precreate
...
0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
```

この出力には、IdM のワンタイムパスワード (OTP) が含まれますが、この値はご自分のデプロイメントとは異なります。

3. **novajoin** を使用するようにアンダークラウドを設定し、ワンタイムパスワード(OTP)を追加し、DNS に IdM IP アドレスを使用し、オーバークラウドドメインを記述します。実際のデプロイメントに合わせて、この例を修正する必要があります。

```
[stack@undercloud ~]$ vi undercloud.conf
...
enable_novajoin = true
ipa_otp = 0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
undercloud_hostname = undercloud-0.lab.local
undercloud_nameservers = X.X.X.X
overcloud_domain_name = lab.local
...
```

4. アンダークラウドに **novajoin** サービスをインストールします。

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

5. オーバークラウドの IP アドレスを DNS に追加します。実際のデプロイメントに合わせて、この例を修正する必要があります。
注記： オーバークラウドの **network-environment.yaml** を確認し、各ネットワークの範囲内の仮想 IP アドレスを選択します。

```
[root@ipa ~]$ ipa dnsrecord-add lab.local overcloud --a-rec=10.0.0.101
[root@ipa ~]# ipa dnszone-add ctlplane.lab.local
[root@ipa ~]# ipa dnsrecord-add ctlplane.lab.local overcloud --a-rec 192.168.24.101
[root@ipa ~]# ipa dnszone-add internalapi.lab.local
[root@ipa ~]# ipa dnsrecord-add internalapi.lab.local overcloud --a-rec 172.17.1.101
[root@ipa ~]# ipa dnszone-add storage.lab.local
[root@ipa ~]# ipa dnsrecord-add storage.lab.local overcloud --a-rec 172.17.3.101
[root@ipa ~]# ipa dnszone-add storagemgmt.lab.local
[root@ipa ~]# ipa dnsrecord-add storagemgmt.lab.local overcloud --a-rec 172.17.4.101
```

6. すべてのエンドポイント用に **public_vip.yaml** マッピングを作成します。

```
Parameter_defaults:
  PublicVirtualFixedIPs: [{'ip_address':'10.0.0.101'}]
  ControlFixedIPs: [{'ip_address':'192.168.24.101'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.17.1.101'}]
  StorageVirtualFixedIPs: [{'ip_address':'172.17.3.101'}]
  StorageMgmtVirtualFixedIPs: [{'ip_address':'172.17.4.101'}]
  RedisVirtualFixedIPs: [{'ip_address':'172.17.1.102'}]
```

16.4.2. IdM と同じドメインを使用し、既存のパブリックエンドポイント証明書を維持するデプロイメントのオーバークラウドインテグレーションの設定

1. 以下のパラメーターが **openstack overcloud deploy** コマンド（有効な設定あり）に存在することを確認してから、デプロイメントコマンドを再度実行します。

   - --ntp-server` : まだ設定されていなければ、ご自分の環境に適した NTP サーバーを指定します。IdM サーバーは ntp を実行している必要があります。
   - **cloud-names.yaml**:（IP ではなく）最初のデプロイメントコマンドからの FQDN が含まれます。
   - **enable-tls.yaml**: 新しいオーバークラウド証明書が含まれます。https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml で例を確認してください。

- **public_vip.yaml**: dns が照合できるようにエンドポイントを特定の ip にマッピングします。
- **enable-internal-tls.yaml**: 内部エンドポイントの TLS を有効にします。
- **tls-everywhere-endpoints-dns.yaml**: DNS 名を使用して TLS エンドポイントを設定します。このファイルの内容を確認して、設定の範囲を把握することができます。
- **haproxy-internal-tls-certmonger.yaml** - certmonger は haproxy の内部証明書を管理します。
- **inject-trust-anchor.yaml**: ルート認証局を追加します。このパラメーターは、証明書が依存する CA チェーンがまだデフォルトで使用される共通セットの一部ではない場合にのみ必要です (たとえば、自己署名の証明書を使用する場合など)。
  以下に例を示します。

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-internal-tls-certmonger.yaml
\
  -e /home/stack/inject-trust-anchor.yaml
...
```

注記

これらの環境ファイルの例は、https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl を参照してください。

16.4.3. IdM と同じドメインを使用し、既存のパブリックエンドポイント証明書を IdM の生成する証明書に置き換えるデプロイメントのオーバークラウドインテグレーションの設定

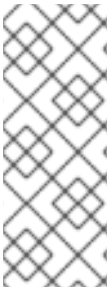
1. 以下のパラメーターが **openstack overcloud deploy** コマンド（有効な設定あり）に存在することを確認してから、デプロイメントコマンドを再度実行します。
   - --ntp-server` : まだ設定されていなければ、ご自分の環境に適した NTP サーバーを指定します。IdM サーバーは ntp を実行している必要があります。
   - **cloud-names.yaml**:（IP ではなく）最初のデプロイメントコマンドからの FQDN が含まれます。
   - **enable-tls.yaml**: 新しいオーバークラウド証明書が含まれます。https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml で例を確認してください。
   - **public_vip.yaml**: dns が照合できるようにエンドポイントを特定の ip にマッピングします。
   - **enable-internal-tls.yaml**: 内部エンドポイントの TLS を有効にします。

- **tls-everywhere-endpoints-dns.yaml**: DNS 名を使用して TLS エンドポイントを設定します。このファイルの内容を確認して、設定の範囲を把握することができます。
- **haproxy-public-tls-certmonger.yaml**: certmonger は haproxy の内部証明書およびパブリック証明書を管理します。
- **inject-trust-anchor.yaml**: ルート認証局を追加します。このパラメーターは、証明書が依存する CA チェーンがまだデフォルトで使用される共通セットの一部ではない場合にのみ必要です (たとえば、自己署名の証明書を使用する場合など)。
  以下に例を示します。

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-public-tls-certmonger.yaml \
  -e /home/stack/inject-trust-anchor.yaml
...
```

注記

これらの環境ファイルの例は、https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl を参照してください。



注記

テンプレート **enable-internal-tls.j2.yaml** は、overcloud deploy コマンドで **enable-internal-tls.yaml** として参照されます。

さらに、**enable-tls.yaml** の古いパブリックエンドポイント証明書は certmonger により **haproxy-public-tls-certmonger.yaml** に置き換えられますが、このファイルは引き続きアップグレードプロセスで参照される必要があります。

16.4.4. IdM サブドメインを使用するデプロイメントのアンダークラウドインテグレーションの設定

以下の手順では、IdM サブドメインを使用するデプロイメントのアンダークラウドインテグレーションを設定する方法について説明します。

Red Hat OpenStack Platform は **novajoin** を使用して Red Hat Identity Management(IdM)と統合し、暗号化証明書を発行および管理します。以下の手順では、アンダークラウドを IdM に登録し、トークンを生成し、アンダークラウド設定でトークンを有効にしてから、アンダークラウドおよびオーバークラウドのデプロイメントスクリプトを再実行します。以下に例を示します。

1. IdM と統合するために **python-novajoin** をインストールします。

```
[stack@undercloud-0 ~]$
```

2. **novajoin** 設定スクリプトを実行し、IdM デプロイメントの設定情報を指定します。以下に例を示します。

```
[stack@undercloud-0 ~]$ sudo novajoin-ipa-setup --principal admin --password
ComplexRedactedPassword \
  --server ipa.lab.local --realm lab.local --domain lab.local \
  --hostname undercloud-0.site1.lab.local --precreate
...
0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
```

この出力には、IdM のワンタイムパスワード (OTP) が含まれますが、この値はご自分のデプロイメントとは異なります。

3. **novajoin** を使用するようにアンダークラウドを設定し、OTP、DNS および NTP 用の IdM IP、およびオーバークラウドドメインを追加します。

```
[stack@undercloud ~]$ vi undercloud.conf
…
[DEFAULT]
undercloud_ntp_servers=X.X.X.X
hieradata_override = /home/stack/hiera_override.yaml
enable_novajoin = true
ipa_otp = 0Uvua6NyIWVkfCSTOmwbdAobsqGH2GONRJrW24MoQ4wg
undercloud_hostname = undercloud-0.site1.lab.local
undercloud_nameservers = X.X.X.X
overcloud_domain_name = site1.lab.local
...
```

4. **novajoin** を使用するようにアンダークラウドを設定し、OTP、DNS 用の IdM IP、およびオーバークラウドドメインを追加します。

```
[stack@undercloud-0 ~]$ vi hiera_override.yaml
nova::metadata::novajoin::api::ipa_domain: site1.lab.local
...
```

5. アンダークラウドに **novajoin** サービスをインストールします。

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

6. オーバークラウドの IP アドレスを DNS に追加します。実際のデプロイメントに合わせて、この例を修正する必要があります。
注記： オーバークラウドの **network-environment.yaml** を確認し、各ネットワークの範囲内の仮想 IP アドレスを選択します。

```
[root@ipa ~]$ ipa dnsrecord-add site1.lab.local overcloud --a-rec=10.0.0.101
[root@ipa ~]# ipa dnszone-add site1.ctlplane.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.ctlplane.lab.local overcloud --a-rec 192.168.24.101
[root@ipa ~]# ipa dnszone-add site1.internalapi.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.internalapi.lab.local overcloud --a-rec 172.17.1.101
[root@ipa ~]# ipa dnszone-add site1.storage.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.storage.lab.local overcloud --a-rec 172.17.3.101
[root@ipa ~]# ipa dnszone-add site1.storagemgmt.lab.local
[root@ipa ~]# ipa dnsrecord-add site1.storagemgmt.lab.local overcloud --a-rec 172.17.4.101
```

7. それぞれのエンドポイントについて **public_vip.yaml** マッピングを作成します。以下に例を示します。

```
Parameter_defaults:
  PublicVirtualFixedIPs: [{'ip_address':'10.0.0.101'}]
  ControlFixedIPs: [{'ip_address':'192.168.24.101'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.17.1.101'}]
  StorageVirtualFixedIPs: [{'ip_address':'172.17.3.101'}]
  StorageMgmtVirtualFixedIPs: [{'ip_address':'172.17.4.101'}]
  RedisVirtualFixedIPs: [{'ip_address':'172.17.1.102'}]
```

8. それぞれのエンドポイントについて **extras.yaml** マッピングを作成します。以下に例を示します。

```
parameter_defaults:
  MakeHomeDir: True
  IdMNoNtpSetup: false
  IdMDomain: redhat.local
  DnsSearchDomains: "site1.redhat.local,redhat.local"
```

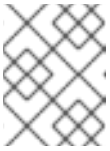
16.4.5. IdM サブドメインを使用し、既存のパブリックエンドポイント証明書を維持するデプロイメントのアンダークラウドインテグレーションの設定

以下の手順では、IdM サブドメインを使用し、引き続き既存のパブリックエンドポイント証明書を維持するデプロイメントのアンダークラウドインテグレーションを設定する方法について説明します。

1. 以下のパラメーターが **openstack overcloud deploy** コマンド（有効な設定あり）に存在することを確認してから、デプロイメントコマンドを再度実行します。

   - --ntp-server` : まだ設定されていなければ、ご自分の環境に適した NTP サーバーを指定します。IdM サーバーは ntp を実行している必要があります。
   - **cloud-names.yaml**:（IP ではなく）最初のデプロイメントコマンドからの FQDN が含まれます。
   - **enable-tls.yaml**: 新しいオーバークラウド証明書が含まれます。https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml で例を確認してください。
   - **public_vip.yaml**: dns が照合できるように特定の ip にマップするエンドポイントが含まれます。
   - `extras.yaml ` : ログイン時のホームディレクトリー作成、no ntp 設定、ベース IdM ドメイン、および resolv.conf の dns 検索に関する設定を含めます。
   - **enable-internal-tls.yaml**: 内部エンドポイントの TLS を有効にします。
   - **tls-everywhere-endpoints-dns.yaml**: DNS 名を使用して TLS エンドポイントを設定します。このファイルの内容を確認して、設定の範囲を把握することができます。
   - **haproxy-internal-tls-certmonger.yaml** - certmonger は haproxy の内部証明書を管理します。
   - **inject-trust-anchor.yaml**: ルート認証局を追加します。このパラメーターは、証明書が依存する CA チェーンがまだデフォルトで使用される共通セットの一部ではない場合にのみ必要です (たとえば、自己署名の証明書を使用する場合など)。
     以下に例を示します。

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e /home/stack/extras.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-internal-tls-certmonger.yaml
\
  -e /home/stack/inject-trust-anchor.yaml
...
```



注記

これらの環境ファイルの例は、https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl を参照してください。

16.4.6. IdM サブドメインを使用し、既存のパブリックエンドポイント証明書を IdM の生成する証明書に置き換えるデプロイメントのアンダークラウドインテグレーションの設定

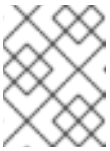
以下の手順では、IdM サブドメインを使用するデプロイメントのアンダークラウドインテグレーションを設定する方法、および既存のパブリックエンドポイント証明書を IdM の生成する証明書に置き換える方法について説明します。

1. 以下のパラメーターが **openstack overcloud deploy** コマンド（有効な設定あり）に存在することを確認してから、デプロイメントコマンドを再度実行します。

   - --ntp-server` : まだ設定されていなければ、ご自分の環境に適した NTP サーバーを指定します。IdM サーバーは ntp を実行している必要があります。
   - **cloud-names.yaml**:（IP ではなく）最初のデプロイメントコマンドからの FQDN が含まれます。
   - **enable-tls.yaml**: 新しいオーバークラウド証明書が含まれます。https://github.com/openstack/tripleo-heat-templates/blob/master/environments/ssl/enable-tls.yaml で例を確認してください。
   - **public_vip.yaml**: dns が照合できるようにエンドポイントを特定の ip にマッピングします。
   - `extras.yaml ` : ログイン時のホームディレクトリー作成、no ntp 設定、ベース IdM ドメイン、および resolv.conf の dns 検索に関する設定を含めます。
   - **enable-internal-tls.yaml**: 内部エンドポイントの TLS を有効にします。
   - **tls-everywhere-endpoints-dns.yaml**: DNS 名を使用して TLS エンドポイントを設定します。このファイルの内容を確認して、設定の範囲を把握することができます。
   - **haproxy-public-tls-certmonger.yaml**: certmonger は haproxy の内部証明書およびパブリック証明書を管理します。

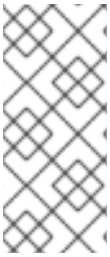
- **inject-trust-anchor.yaml**: ルート認証局を追加します。このパラメーターは、証明書が依存する CA チェーンがまだデフォルトで使用される共通セットの一部ではない場合にのみ必要です (たとえば、自己署名の証明書を使用する場合など)。
  以下に例を示します。

```
[ stack@undercloud ~]$ openstack overcloud deploy \
...
  --ntp-server 10.13.57.78 \
  -e /home/stack/cloud-names.yaml \
  -e /home/stack/enable-tls.yaml \
  -e /home/stack/public_vip.yaml \
  -e /home/stack/extras.yaml \
  -e <tripleo-heat-templates>/environments/ssl/enable-internal-tls.yaml \
  -e <tripleo-heat-templates>/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e <tripleo-heat-templates>/environments/services/haproxy-public-tls-certmonger.yaml \
  -e /home/stack/inject-trust-anchor.yaml
...
```



注記

これらの環境ファイルの例は、https://github.com/openstack/tripleo-heat-templates/tree/master/environments/ssl を参照してください。



注記

この例では、テンプレートの **enable-internal-tls.j2.yaml** は **overcloud deploy** コマンドで **enable-internal-tls.yaml** として参照されます。さらに、**enable-tls.yaml** の古いパブリックエンドポイント証明書は **haproxy-public-tls-certmonger.yaml** を使用して certmonger に置き換えられますが、このファイルは引き続きアップグレードプロセスで参照される必要があります。

16.5. TLS 暗号化の確認

オーバークラウドの再デプロイメントが完了したら、すべてのエンドポイントが TLS で暗号化されていることを確認します。以下の例では、すべてのエンドポイントが **https** を使用するように設定されています。これは、TLS 暗号化を使用していることを示しています。

```
+----------------------------------+-----------+--------------+----------------+---------+-----------+----------------
----------------------------------------------+
| ID                               | Region    | Service Name | Service Type   | Enabled | Interface | URL
|
+----------------------------------+-----------+--------------+----------------+---------+-----------+----------------
----------------------------------------------+
| 0fee4efdc4ae4310b6a139a25d9c0d9c | regionOne | neutron      | network        | True    | public    |
https://overcloud.lab.local:13696                |
| 220558ab1d2445139952425961a0c89a | regionOne | glance       | image          | True    | public    |
https://overcloud.lab.local:13292                |
| 24d966109ffa419da850da946f19c4ca | regionOne | placement    | placement      | True    | admin     |
https://overcloud.internalapi.lab.local:8778/placement        |
| 27ac9e0d22804ee5bd3cd8c0323db49c | regionOne | nova         | compute        | True    | internal  |
https://overcloud.internalapi.lab.local:8774/v2.1             |
| 31d376853bd241c2ba1a27912fc896c6 | regionOne | swift        | object-store   | True    | admin     |
https://overcloud.storage.lab.local:8080                 |
| 350806234c784332bfb8615e721057e3 | regionOne | nova         | compute        | True    | admin     |
```

```
https://overcloud.internalapi.lab.local:8774/v2.1               |
| 49c312f4db6748429d27c60164779302 | regionOne | keystone     | identity       | True    | public    |
https://overcloud.lab.local:13000                               |
| 4e535265c35e486e97bb5a8bc77708b6 | regionOne | nova         | compute        | True    | public    |
https://overcloud.lab.local:13774/v2.1                          |
| 5e93dd46b45f40fe8d91d3a5d6e847d3 | regionOne | keystone     | identity       | True    | admin     |
https://overcloud.ctlplane.lab.local:35357                      |
| 6561984a90c742a988bf3d0acf80d1b6 | regionOne | swift        | object-store   | True    | public    |
https://overcloud.lab.local:13808/v1/AUTH_%(tenant_id)s         |
| 76b8aad0bdda4313a02e4342e6a19fd6 | regionOne | placement    | placement      | True    | public
| https://overcloud.lab.local:13778/placement                     |
| 96b004d5217c4d87a38cb780607bf9fb | regionOne | placement    | placement      | True    | internal  |
https://overcloud.internalapi.lab.local:8778/placement          |
| 98489b4b107f4da596262b712c3fe883 | regionOne | glance       | image          | True    | internal  |
https://overcloud.internalapi.lab.local:9292                    |
| bb7ab36f30b14b549178ef06ec74ff84 | regionOne | glance       | image          | True    | admin     |
https://overcloud.internalapi.lab.local:9292                    |
| c1547f7bf9a14e9e85eaaaeea26413b7 | regionOne | neutron      | network        | True    | admin     |
https://overcloud.internalapi.lab.local:9696                    |
| ca66f499ec544f42838eb78a515d9f1e | regionOne | keystone     | identity       | True    | internal  |
https://overcloud.internalapi.lab.local:5000                    |
| df0181358c07431390bc66822176281d | regionOne | swift        | object-store   | True    | internal  |
https://overcloud.storage.lab.local:8080/v1/AUTH_%(tenant_id)s |
| e420350ef856460991c3edbfbae917c1 | regionOne | neutron      | network        | True    | internal  |
https://overcloud.internalapi.lab.local:9696                    |
+----------------------------------+-----------+--------------+----------------+---------+-----------+----------------
-----------------------------------------------+
```

第17章 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。デバッグパラメーターを保存する **debug.yaml** 環境ファイルを作成し、**parameter_defaults** セクションに **KeystoneDebug** 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。

第18章 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。以下に例を示します。

- 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' } }
```

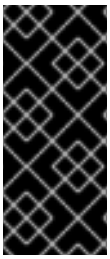
- OpenStack Compute (nova) には **NovaApiPolicies** パラメーターを使用します。このパラメーターを環境ファイルの **parameter_defaults** セクションで設定します。

```
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。

第19章 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の設定方法をいくつか説明します。



重要

デフォルトでは、オーバークラウドは OpenStack Compute (nova) の提供するローカル一時ストレージおよび OpenStack Storage (cinder) の提供する LVM ブロックストレージを使用します。ただし、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ません。代わりに、本章のストレージオプションのいずれかを使用してください。

19.1. NFS ストレージの設定

本項では、NFS 共有を使用するオーバークラウドの設定方法について説明します。インストールおよび設定のプロセスは、コア heat テンプレートコレクション内にすでに存在する環境ファイルの変更がベースとなります。

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 サービスおよび Compute サービスには、NFS バージョン 4.1 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。

1. このファイルを **stack** ユーザーの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。**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。**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。**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。**false** に設定します。

GlanceBackend

glance 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ストレージを使用するには **file** に設定します。オーバークラウドは、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をマウントするための NFS 共有。たとえば、192.168.122.1:/export/cinder と設定します。

GlanceNfsEnabled

GlanceBackend を **file** に設定すると、**GlanceNfsEnabled** によりイメージが NFS 経由で共有の場所に保管されます。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。無効にすると、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。**true** に設定します。

GlanceNfsShare

イメージストレージをマウントするための 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
この環境ファイルには、Red Hat OpenStack Platform 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。

環境ファイルのオプションは、以下のようになるはずです。

```
parameter_defaults:
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: false
```

```
CinderEnableNfsBackend: true
NovaEnableRbdBackend: false
GlanceBackend: 'file'

CinderNfsMountOptions: 'rw,sync'
CinderNfsServers: '192.0.2.230:/cinder'

GlanceNfsEnabled: true
GlanceNfsShare: '192.0.2.230:/glance'
GlanceNfsOptions: 'rw,sync,context=system_u:object_r:glance_var_lib_t:s0'
```

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。パラメーターを上記のコード例のように設定すると、使用する Block Storage サービスおよび Image サービス用に NFS マウントポイントが 2 つ作成されます。

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 **context=system_u:object_r:glance_var_lib_t:s0** オプションを追加します。この SELinux コンテンツがないと、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。

3. オーバークラウドをデプロイする際に、このファイルを追加します。

19.2. CEPH STORAGE の設定

director では、Red Hat Ceph Storage のオーバークラウドへの統合には主に 2 つの方法を提供します。

Ceph Storage Cluster でのオーバークラウドの作成

director には、オーバークラウドの作成中に Ceph Storage Cluster を作成する機能があります。director は、データの格納に Ceph OSD を使用する Ceph Storage ノードセットを作成します。さらに、director は、オーバークラウドのコントローラーノードに Ceph Monitor サービスをインストールします。このため、組織が高可用性のコントローラーノード 3 台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。

既存の Ceph Storage 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、オーバークラウドのデプロイメント時に統合できます。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。

19.3. 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 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 keystone が特定の外部 Swift エンドポイントを使用するように設定されます。

注記

Object Storage (swift) クラスター上のユーザーアカウントは手動で管理する必要があります。

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。

外部の Swift クラスターを使用して director をデプロイする場合:

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. これらの追加の環境ファイルを使用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
```

19.4. 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、OpenStack のインストール時に使用される 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。

相互運用可能なイメージのインポートにより、以下の 2 とおり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
- glance-direct

web-download 法では、URL からイメージをインポートします。**glance-direct** 法では、ローカルボリュームからイメージをインポートします。

19.4.1. glance-settings.yaml ファイルの作成およびデプロイメント

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、Heat テンプレートで定義した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**」セクションを参照してください。

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で、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。この行が必要になるのは、web-download に加えて別の方法を有効にする場合だけです。

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。GlanceNfsEnabled を true に設定する必要があります。

設定を行うには、以下の手順を実施します。

1. 新規ファイルを作成します (例: glance-settings.yaml)。このファイルの内容は、上記の例のようにする必要があります。
2. **openstack overcloud deploy** コマンドを使用して、ファイルをご自分の OpenStack 環境に追加します。

```
$ 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。

19.4.2. 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、そのスキームがそこに含まれていない場合: 拒否する。含まれている場合: iii 項をスキップして 2 項に進む。
    c. ブラックリストがあり、そのスキームがそこに含まれている場合: 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、そのホスト名がそこに含まれていない場合: 拒否する。含まれている場合: iii 項をスキップして 3 項に進む。
    c. ブラックリストがあり、そのホスト名がそこに含まれている場合: 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、そのポートがそこに含まれていない場合: 拒否する。含まれている場合: ii 項をスキップして 4 項に進む。
    b. ブラックリストがあり、そのポートがそこに含まれている場合: 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 にポートが含まれていないため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 は上記のルールに従って検証されます。

19.4.2.1. URI 検証の例

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 = []
```

詳細な情報は、**『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。

19.4.2.2. 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 はオプションのファイルです。オプションのデフォルト設定を以下に示します。

- 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には、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 と 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。使用している OpenStack 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。

19.4.3. 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追加し、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、[image_import_opts] セクションに以下の行を追加します。

```
[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、以下の設定では、property1 に関する 1 つの値および property2 に関する 2 つの値が、任意の非管理者ユーザーによってダウンロードされたイメージに注入されます。

```
[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、イメージのインポートをコールするユーザーがこれらのロールを持つ場合には、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、上記の例に示すようにコロン (「:」) 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。使用している OpenStack 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。

19.5. IMAGE サービス用 CINDER バックエンドの設定

GlanceBackend パラメーターにより、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。Image サービスのバックエンドに **cinder** を設定するには、以下の設定を環境ファイルに追加します。

```
parameter_defaults:
  GlanceBackend: cinder
```

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****
```

カスタムのユーザー名または **cinder_store_** パラメーターにカスタムの値を使用する場合には、**parameter_defaults** に ExtraConfig 設定を追加してカスタムの値を渡します。以下に例を示します。

```
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')}"
```

19.6. 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。以下の例は、**max_disk_devices_to_attach** の値を「30」に変更する方法を示しています。

```
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、インスタンスの移行元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、最大数を調整する必要があります。
- 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。

19.7. 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、ローカルマシンにイメージのコピーを保存し、イメージ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。

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するには、以下の手順を実施します。

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。

19.8. サードパーティーのストレージの設定

director には、サードパーティーのストレージプロバイダーの設定に役立つ複数の環境ファイルが含まれています。これには、以下の設定項目が含まれます。

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/cinder-netapp-config.yaml** にあります。

設定に関する詳しい情報は、『NetApp Block Storage Back End Guide』を参照してください。

第20章 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。

20.1. 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。**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 が **true** に設定されている場合には、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
```

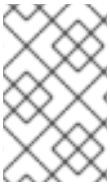
この設定では、**ExtraConfig** により、追加で 2 つのファイアウォールルールが 全ノードに適用されます。

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る点に注意してください。このプレフィックスは、Puppet が最終の **iptables** ファイルに記載されている定義済みの全ルールを順序付けるのに役立ちます。デフォルトの OpenStack Platform ルールは、000 から 200 までの範囲のプレフィックスを使用します。

20.2. 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、未承認のユーザーがネットワークデバイスに関する情報にアクセス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 man ページの **snmpd.conf** を参照してください。

20.3. 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 (「14章*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。

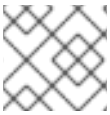
- 暗号: **ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-**

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**

以下の内容で環境ファイルを作成します。

```
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。

20.4. OPEN VSWITCH ファイアウォールの使用

Red Hat OpenStack Platform director で Open vSwitch (OVS) ファイアウォールドライバーを使用するためのセキュリティーグループを設定することができます。**NeutronOVSFirewallDriver** パラメーターで、使用するファイアウォールドライバーを指定することができます。

- **iptables_hybrid**: 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**: neutron 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。

注記

iptables_hybrid オプションは、OVS-DPDK 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の実装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**、**iptables_hybrid** 等が挙げられます。デフォルト値である空の文字列は **iptables_hybrid** と等価です。

20.5. 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化されます。以下の方法を使用すると、オーバークラウドで **root** ユーザーにアクセスすることが引き続き可能となります。

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 キーが含まれており、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 を介して選択した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 ユーザーセキュリティーの軽減

状況によっては、**root** ユーザーに直接 SSH アクセスする必要がある可能性があります。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 制限を軽減することが可能です。

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 (「初回起動: 初回起動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記載してください。

```
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、パスワードは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る点に注意してください。

オーバークラウドの作成時に、**openstack overcloud deploy** コマンドにこの環境ファイルを追加します。

第21章 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中央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により、全コンポーネントの機能を監視できます。また、中央集中ロギングにより、OpenStack 環境全体の全ログを一箇所で確認できます。

モニタリングツールの設定に関する詳しい情報は、『専用のモニタリング ツール設定ガイド』 を参照してください。

第22章 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の設定に役立つ環境ファイルが含まれています。

22.1. 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。以下に例を示します。

```
$ openstack overcloud deploy --templates -e /home/stack/templates/neutron-ml2-fujitsu-cfab.yaml [OTHER OPTIONS] ...
```

22.2. 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。

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。以下に例を示します。

```
$ openstack overcloud deploy --templates -e /home/stack/templates/neutron-ml2-fujitsu-fossw.yaml [OTHER OPTIONS] ...
```

第23章 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。

23.1. 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** という名前になります。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。この設定は、デプロイ後には変更できません。

```
parameter_defaults:
  KeystoneRegion: 'SampleRegion'
```

第24章 その他の設定

24.1. オーバークラウドノードカーネルの設定

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
```

24.2. 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。また、外部のロードバランサーを使用して、この分散を実行することも可能です。たとえば、組織で、コントローラーノードへのトラフィックの分散処理に、ハードウェアベースのロードバランサーを使用する場合などです。

外部の負荷分散機能の設定に関する詳しい情報は、全手順について『External Load Balancing for the Overcloud』を参照してください。

24.3. 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の作成に役立つ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 設定に関する詳しい情報は、全手順について『専用の IPv6 Networking for the Overcloud 』を参照してください。