



Red Hat OpenStack Platform 13

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

Red Hat OpenStack Platform 13 オーバークラウドの高度なカスタマイズ

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

OpenStack Team
rhos-docs@redhat.com

法律上の通知

Copyright © 2018 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

本ガイドでは、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	5
第2章 HEAT テンプレートの理解	6
2.1. HEAT テンプレート	6
2.2. 環境ファイル	7
2.3. オーバークラウドのコア HEAT テンプレート	8
2.4. プランの環境メタデータ	9
2.5. ケイパビリティマップ	10
2.6. オーバークラウド作成時の環境ファイルの追加	12
2.7. カスタムのコア HEAT テンプレートの使用	13
第3章 PARAMETERS	17
3.1. 例 1: タイムゾーンの設定	17
3.2. 例 2: LAYER 3 HIGH AVAILABILITY (L3HA) の無効化	18
3.3. 例 3: TELEMETRY DISPATCHER の設定	18
3.4. 例 4: RABBITMQ ファイル記述子の上限の設定	18
3.5. 変更するパラメーターの特定	18
第4章 設定フック	21
4.1. 初回起動: 初回起動時の設定のカスタマイズ	21
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	22
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	24
4.4. 設定後: 全オーバークラウドロールのカスタマイズ	26
4.5. PUPPET: ロール用の HIERADATA のカスタマイズ	28
4.6. PUPPET: 個別のノードの HIERADATA のカスタマイズ	29
4.7. PUPPET: カスタムのマニフェストの適用	29
第5章 オーバークラウドの登録	31
5.1. 環境ファイルを使用したオーバークラウドの登録	31
5.2. 例 1: カスタマーポータルへの登録	33
5.3. 例 2: RED HAT SATELLITE 6 サーバーへの登録	34
5.4. 例 3: RED HAT SATELLITE 5 サーバーへの登録	34
5.5. 例 4: HTTP プロキシを介した登録	35
5.6. 高度な登録方法	35
第6章 ANSIBLE ベースのオーバークラウド登録	38
6.1. RHSM コンポーザブルサービス	38
6.2. RHSMVARS サブパラメーター	38
6.3. 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	39
6.4. 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	40
6.5. RHSM コンポーザブルサービスへの切り替え	41
6.6. RHEL-REGISTRATION から RHSM へのマッピング	42
6.7. 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。	43
第7章 コンポーザブルサービスとカスタムロール	44
7.1. サポートされているカスタムロールアーキテクチャー	44
7.2. ガイドラインおよび制限事項	44
7.3. ロール	45
7.3.1. roles_data ファイルの検証	45
7.3.2. roles_data ファイルの作成	46
7.3.3. ロールパラメーターの考察	47
7.3.4. 新規ロールの作成	49
7.4. コンポーザブルサービス	51

7.4.1. コンポーザブルサービスアーキテクチャーの考察	51
7.4.2. ロールへのサービスの追加と削除	53
7.4.3. 無効化されたサービスの有効化	54
7.4.4. サービスなしの汎用ノードの作成	54
7.5. アーキテクチャー	55
7.5.1. サービスアーキテクチャー: モノリシックコントローラー	55
7.5.2. サービスアーキテクチャー: 分割コントローラー	57
7.5.3. サービスアーキテクチャー: スタンドアロンロール	59
7.6. コンポーザブルサービスのリファレンス	74
第8章 コンテナ化されたサービス	84
8.1. コンテナ化されたサービスのアーキテクチャー	84
8.2. コンテナ化されたサービスのパラメーター	85
8.3. OPENSTACK PLATFORM コンテナの修正	85
第9章 ネットワークの分離	88
9.1. カスタムのインターフェーステンプレートの作成	89
9.2. ネットワーク環境ファイルの作成	96
9.3. OPENSTACK サービスの分離ネットワークへの割り当て	98
9.4. デプロイするネットワークの選択	99
第10章 コンポーザブルネットワークの使用	105
10.1. コンポーザブルネットワークの定義	105
10.1.1. コンポーザブルネットワーク用のネットワークインターフェース設定の定義	106
10.1.2. サービスに対するコンポーザブルネットワークの割り当て	106
10.1.3. ルーティングネットワークの定義	107
第11章 ノード配置の制御	109
11.1. 特定のノード ID の割り当て	109
11.2. カスタムのホスト名の割り当て	110
11.3. 予測可能な IP の割り当て	110
11.4. 予測可能な仮想 IP の割り当て	112
第12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	114
12.1. 署名ホストの初期化	114
12.2. 認証局の作成	114
12.3. クライアントへの認証局の追加	114
12.4. SSL/TLS キーの作成	115
12.5. SSL/TLS 証明書署名要求の作成	115
12.6. SSL/TLS 証明書の作成	116
12.7. SSL/TLS の有効化	116
12.8. ルート証明書の注入	117
12.9. DNS エンドポイントの設定	118
12.10. オーバークラウド作成時の環境ファイルの追加	119
12.11. SSL/TLS 証明書の更新	119
第13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	120
13.1. CA へのアンダークラウド追加	120
13.2. アンダークラウドを IDM に追加します。	120
13.3. オーバークラウド DNS の設定	121
13.4. NOVAJOIN を使用するためのオーバークラウドの設定	121
第14章 デバッグモード	124
第15章 ポリシー	125

第16章 ストレージの設定	126
16.1. NFS ストレージの設定	126
16.2. CEPH STORAGE の設定	127
16.3. 外部の OBJECT STORAGE クラスターの使用	128
16.4. サードパーティーのストレージの設定	128
第17章 セキュリティーの強化	130
17.1. オーバークラウドのファイアウォールの管理	130
17.2. SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP) のコミュニティ文字列の変更	131
17.3. HAPROXY の SSL/TLS の暗号およびルールの変更	131
17.4. OPEN VSWITCH ファイアウォールの使用	132
17.5. セキュアな ROOT ユーザーアクセスの使用	133
第18章 コントローラーノードのフェンシング	135
18.1. 前提条件の確認	135
18.2. フェンシングの有効化	135
18.3. フェンシングのテスト	137
第19章 モニタリングツールの設定	138
第20章 ネットワークプラグインの設定	139
20.1. FUJITSU CONVERGED FABRIC (C-FABRIC)	139
20.2. FUJITSU FOS SWITCH	139
第21章 IDENTITY の設定	141
21.1. リージョン名	141
第22章 その他の設定	142
22.1. 外部の負荷分散機能の設定	142
22.2. IPV6 ネットワークの設定	142
付録A ネットワーク環境のオプション	143
付録B ネットワークインターフェースのテンプレート例	146
B.1. インターフェースの設定	146
B.2. ルートおよびデフォルトルートの設定	146
B.3. FLOATING IP のためのネイティブ VLAN の使用	147
B.4. 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 の使用	148
B.5. ジャンボフレームの設定	148
第23章 ネットワークインターフェースのパラメーター	150
23.1. インターフェースのオプション	150
23.2. VLAN のオプション	150
23.3. OVS ボンディングのオプション	151
23.4. OVS ブリッジのオプション	152
23.5. LINUX ボンディングのオプション	153
23.6. LINUX BRIDGE のオプション	154
付録C OPEN VSWITCH ボンディングのオプション	156
C.1. ボンディングモードの選択	156
C.2. ボンディングオプション	157

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](#)』に記載していますが、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。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には、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必要があります。



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のみを使用してください。

第2章 HEAT テンプレートの理解

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の多くは、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と、リソースの設定が含まれる **スタック** を定義して作成することです。リソースとは、コンピュータリソース、ネットワーク設定、セキュリティグループ、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。

Heat テンプレートは、3 つの主要なセクションで構成されます。

parameters

parameters は Heat に渡される設定で、値を指定せずにパラメーターのデフォルト値やスタックをカスタマイズする方法を提供します。これらは、テンプレートの **parameters** セクションで定義されます。

resources

resources 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。これらの設定は、テンプレートの **resources** セクションで定義されます。

output

output 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。これらは、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** と呼ばれるインスタンスを作成します。このスタックは、**My Cirros Instance** と呼ばれる **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。

```
$ heat stack-list --show-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。

resource registry

このセクションでは、他の Heat テンプレートに関連付けられたカスタムのリソース名を定義します。これは実質的に、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方法を提供します。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。

parameters

これらは、最上位のテンプレートのパラメーターに適用する共通の設定です。たとえば、リソースレジストリーマッピングなどのネストされたスタックをデプロイするテンプレートがある場合には、パラメーターは最上位のテンプレートにのみ適用され、ネストされたリソースのテンプレートには適用されません。パラメーターは、環境ファイルの **parameters** セクションで定義されます。

parameter defaults

これらのパラメーターは、すべての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、リソースレジストリーマッピングなどのネストされたスタックをデプロイするテンプレートがある場合には、パラメーターのデフォルト値は、最上位のテンプレートとすべてのネストされたリソースを定義するテンプレートなど、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** セクションで定義されます。



重要

オーバークラウドの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。これは、パラメーターがオーバークラウドのスタックテンプレートすべて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場合に、このような環境ファイル (**my_env.yaml**) を追加することができます。**my_env.yaml** ファイルにより、**OS::Nova::Server::MyServer** と呼ばれる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して、組み込まれている種別を上書きする Heat テンプレートです。**my_template.yaml** ファイルに **OS::Nova::Server::MyServer** リソースを追加することができます。

MyIP は、この環境ファイルと一緒にデプロイされるメインの Heat テンプレートにのみパラメーターを適用します。上記の例では、**my_template.yaml** のパラメーターにのみ適用されます。

NetworkName はメインの Heat テンプレート (上記の例では **my_template.yaml**) とメインのテンプレート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。

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。

このテンプレートコレクションには、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が、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

これは、オーバークラウド環境の作成に使用されるメインのテンプレートファイルです。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のセクションを反復し、カスタムロールを作成します。Jinja2 形式はオーバークラウドのデプロイメント処理中に YAML にレンダリングされます。

overcloud-resource-registry-puppet.j2.yaml

これは、オーバークラウド環境の作成に使用する主要な環境ファイルで、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に Puppet の設定を開始します。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のセクションを反復し、カスタムロールを作成します。Jinja2 形式はオーバークラウドのデプロイメント処理中に 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、IP 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のみ (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、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 環境ファイルが追加で含まれます。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

大部分は Puppet を使用した設定によって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たディレクトリー。

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。たとえば、director が提供する **extraconfig/pre_deploy/rhel-registration** は、ノードの Red Hat Enterprise Linux オペレーティングシステムを Red Hat コンテンツ配信ネットワークまたは Red Hat Satellite サーバーに登録できるようにします。

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を自動的に設定します。

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/docker.yaml
- path: environments/docker-ha.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. ケイパビリティーマップ

ケイパビリティーマップは、プラン内の環境ファイルとそれらの依存関係のマッピングを提供します。

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で検出されるカスタムの環境ファイルの中で、**capabilities-map.yaml** にリスト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。

デフォルトのファイルは、**/usr/share/openstack-tripleo-heat-templates/capabilities-map.yaml** にあります。

ケイパビリティーマップの構文の例を以下に示します。

```
topics: ❶
  - title: My Parent Section
    description: This contains a main section for different environment
files
    environment_groups: ❷
      - name: my-environment-group
        title: My Environment Group
        description: A list of environment files grouped together
        environments: ❸
          - file: environment_file_1.yaml
            title: Environment File 1
            description: Enables environment file 1
            requires: ❹
              - dependent_environment_file.yaml
          - file: environment_file_2.yaml
            title: Environment File 2
            description: Enables environment file 2
            requires: ❺
              - dependent_environment_file.yaml
          - file: dependent_environment_file.yaml
            title: Dependent Environment File
            description: Enables the dependent environment file
```

❶ **topics** パラメーターには、UI のデプロイメント設定内のセクションの一覧が含まれます。各トピックは、環境オプションの単一画面として表示され、複数の環境グループが含まれます。これは、**environment_groups** パラメーターで定義することができます。各トピックには、プレーンテキストの **title** と **description** を記述することができます。

❷ **environment_groups** パラメーターには、UI のデプロイメント設定内の環境ファイルのグループを一覧表示します。たとえば、ストレージトピックでは、Ceph 関係の環境ファイルの環境グループがある可能性があります。各環境グループには、プレーンテキストの **title** と **description** を記述することができます。

❸ **environments** パラメーターには、環境グループに属する環境ファイルがすべて表示されます。**file** パラメーターは、環境ファイルの場所です。各環境エントリーには、プレーンテキストで **title** と **description** を記述することができます。

❹ ❺ **requires** パラメーターは、環境ファイルの依存関係の一覧です。この例では、**environment_file_1.yaml** と **environment_file_2.yaml** にはいずれも **dependent_environment_file.yaml** を有効化する必要もあります。



注記

Red Hat OpenStack Platform は、このファイルを使用して director UI の機能へのアクセスを追加します。Red Hat OpenStack Platform のバージョンが新しくなると、このファイルは上書きされる可能性があるため、編集しないことを推奨します。

2.6. 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が、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、環境ファイルの順序は重要です。以下の一覧は、環境ファイルの順序の例です。

environment-file-1.yaml

```
resource_registry:
  OS::Triple0::NodeExtraConfigPost: /home/stack/templates/template-1.yaml

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

environment-file-2.yaml

```
resource_registry:
  OS::Triple0::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::Triple0::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::Triple0::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。

- **environment-file-2.yaml** で指定されているとおりに、**TimeZone** パラメーターは **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。この値は、**environment-file-2.yaml** によっては変更されません。

この設定は、複数の環境ファイルによって競合が発生することなくカスタム設定を定義する手段を提供します。

2.7. 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、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. 初期コミットに全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. **my-customization** ブランチに切り替わっていることを確認します。

```
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。

第3章 PARAMETERS

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、以下のパラメーターが含まれます。

- **overcloud.j2.yaml**: デフォルトのベースパラメーター
- **roles_data.yaml**: コンポーザブルロールのデフォルトパラメーター
- **puppet/services/*.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、**puppet/services** 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 (**puppet/services/time/timezone.yaml**) には **TimeZone** パラメーターが含まれています。**TimeZone** パラメーターの値を空白のままにすると、オーバークラウドはデフォルトで時刻を **UTC** に設定します。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 **Japan** 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: LAYER 3 HIGH AVAILABILITY (L3HA) の無効化

OpenStack Networking (neutron) API 用の Heat テンプレート (**puppet/services/neutron-api.yaml**) には、Layer 3 High Availability (L3HA) 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** ですが、環境ファイルで以下の設定を使用して有効化することができます。

```

parameter_defaults:
    NeutronL3HA: true

```

3.3. 例 3: TELEMETRY DISPATCHER の設定

OpenStack Telemetry (**ceilometer**) サービスには、時系列データストレージ向けのコンポーネント (**gnocchi**) が含まれています。 **puppet/services/ceilometer-base.yaml** の Heat テンプレートにより、**gnocchi** と標準のデータベースを切り替えることができます。これは、**CeilometerMeterDispatcher** パラメーターを次のいずれかの値に設定して切り替えます。

- **gnocchi**: Ceilometer dispatcher に新しい時系列データベースを使用します。これは、デフォルトのオプションです。
- **database**: Ceilometer dispatcher に標準のデータベースを使用します。

標準のデータベースに切り替えるには、以下の設定を環境ファイルに追加します。

```

parameter_defaults:
    CeilometerMeterDispatcher: database

```

3.4. 例 4: 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。 **puppet/services/rabbitmq.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

```

3.5. 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。
 - Puppet モジュールに選択したオプションを制御する変数が含まれている場合には、次のステップに進んでください。
 - Puppet モジュールに選択したオプションを制御する変数が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。**puppet/services/*** 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、**puppet/services/keystone.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、設定フックを使用して、環境ファイルを使用する 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: ロール用の 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必要がある場合があります。ワークフローを使用して、以下の操作を行います。

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/puppet/services/keystone.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

これは、オーバークラウドの環境ファイルの **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ることを意味します。

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: 初回起動時の設定のカスタマイズ

director は、オーバークラウドの初期設定時に全ノードに設定を行うメカニズムを提供し、**cloud-init** でこの設定をアーカイブします。アーカイブした内容は、**OS::TripleO::NodeUserData** リソース種別を使用して呼び出すことが可能です。

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。まず基本的な Heat テンプレート (**/home/stack/templates/nameserver.yaml**) を作成する必要があります。このテンプレートは、固有のネームサーバーが指定された各ノードの **resolv.conf** を追加するスクリプトを実行します。**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}
```

次に、Heat テンプレートを登録する環境ファイル (**/home/stack/templates/firstboot.yaml**) を **OS::TripleO::NodeUserData** リソース種別として作成します。

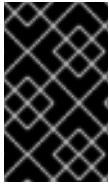
```
resource_registry:
  OS::TripleO::NodeUserData: /home/stack/templates/nameserver.yaml
```

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、この環境ファイルをスタックに追加します。たとえば、以下のコマンドを実行します。

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

-e は、オーバークラウドのスタックに環境ファイルを適用します。

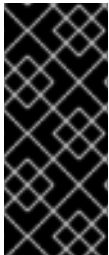
これにより、初回作成/起動時に、全ノードに設定が追加されます。オーバークラウドのスタックの更新など、これらのテンプレートを後で追加しても、このスクリプトは実行されません。



重要

OS::TripleO::NodeUserData は、1 つの Heat テンプレートに対してのみ登録することが可能です。それ以外に使用すると、以前の Heat テンプレートの内容が上書きされてしまいます。

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

本ガイドの以前のバージョンでは、**OS::TripleO::Tasks::*PreConfig** リソースで、ロールごとに事前設定フックを指定していましたが、director の Heat テンプレートコレクションにはこれらのフックを専用で使用する必要があるため、カスタムには使用すべきではありません。このリソースの代わりに、以下に記載する **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 は、初回のブートが完了してコア設定が開始する前に、特定のノードロール向けのカスタム設定を指定するフックのセットを提供します。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 Object Storage 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 Block Storage 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する追加の設定。**[ROLE]** はコンポーザブルロール名に置き換えます。

以下の例では、まず基本的な Heat テンプレート (`/home/stack/templates/nameserver.yaml`) を作成します。このテンプレートは、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]}

```

この例では、**resources** セクションに以下が含まれています。

CustomExtraConfigPre

これは、ソフトウェアの設定を定義します。上記の例では、Bash **script** を定義しており、Heat は **_NAMESERVER_IP_** を **nameserver_ip** パラメーターに保存されている値に置き換えます。

CustomExtraDeploymentPre

これは、**CustomExtraConfigPre** リソースのソフトウェア設定で指定されているソフトウェアの設定を実行します。次の点に注意してください。

- **config** パラメーターは、**CustomExtraConfigPre** リソースへの参照を作成して、適用する設定を Heat が認識するようにします。
- **server** パラメーターはオーバークラウドノードのマップを取得します。このパラメーターは親テンプレートにより提供され、このフックを使用するテンプレートでは必須です。
- **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。この場合は、オーバークラウドが作成された時にのみ設定を適用します。実行可能なアクションは **CREATE**、**UPDATE**、**DELETE**、**SUSPEND** および **RESUME** です。
- **input_values** には **deploy_identifier** と呼ばれるパラメーターが含まれます。これは、親テンプレートからの **DeployIdentifier** を保存します。このパラメーターは、デプロイメントが更新される度にリソースにタイムスタンプを付けます。これにより、そのリソースは以降のオーバークラウドの更新に再度適用されるようになります。

次に、Heat テンプレートをロールベースのリソース種別に登録する環境ファイル (`/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のみに適用するには、**ControllerExtraConfigPre** フックを使用します。

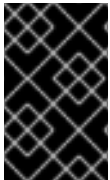
```
resource_registry:
    OS::TripleO::ControllerExtraConfigPre:
        /home/stack/templates/nameserver.yaml

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スタックにこの環境ファイルを追加します。たとえば、以下のコマンドを実行します。

```
$ 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の後の更新時にコア設定が開始する前に、カスタム設定が全コントローラーノードに適用されます。



重要

各リソースは、1 フックあたり 1 つの Heat テンプレートにしか登録できません。その後、別の Heat テンプレートを使用すると、最初に登録した Heat テンプレートは上書きされます。

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 は、初回のブート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を設定するフックを用意します。

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、まず基本的な Heat テンプレート (`/home/stack/templates/nameserver.yaml`) を作成します。このテンプレートは、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_identififier: {get_param: DeployIdentifier}

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

```

この例では、**resources** セクションに以下が含まれています。

CustomExtraConfigPre

これは、ソフトウェアの設定を定義します。上記の例では、Bash **script** を定義しており、Heat は **_NAMESERVER_IP_** を **nameserver_ip** パラメーターに保存されている値に置き換えます。

CustomExtraDeploymentPre

これは、**CustomExtraConfigPre** リソースのソフトウェア設定で指定されているソフトウェアの設定を実行します。次の点に注意してください。

- **config** パラメーターは、**CustomExtraConfigPre** リソースへの参照を作成して、適用する設定を Heat が認識するようにします。
- **server** パラメーターはオーバークラウドノードのマップを取得します。このパラメーターは親テンプレートにより提供され、このフックを使用するテンプレートでは必須です。
- **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。この場合は、オーバークラウドが作成された時にのみ設定を適用します。実行可能なアクションは **CREATE**、**UPDATE**、**DELETE**、**SUSPEND** および **RESUME** です。
- **input_values** パラメーターには **deploy_identififier** と呼ばれるサブパラメーターが含まれます。これは、親テンプレートからの **DeployIdentifier** を保存します。このパラメーターは、デプロイメントが更新される度にリソースにタイムスタンプを付けます。これにより、そのリソースは以降のオーバークラウドの更新に再度適用されるようになります。

次に、**OS::Triple0::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::Triple0::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スタックにこの環境ファイルを追加します。たとえば、以下のコマンドを実行します。

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

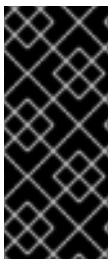
このコマンドにより、オーバークラウドの初期作成またはその後の更新時にコア設定が開始する前に、全ノードに設定が適用されます。



重要

OS::TripleO::NodeExtraConfig は 1 つの Heat テンプレートにしか登録できません。その後に別のテンプレートを使用すると、最初に登録した Heat テンプレートは上書きされます。

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

本ガイドの以前のバージョンでは、**OS::TripleO::Tasks::*PostConfig** リソースで、ロールごとに設定後のフックを指定していましたが、director の Heat テンプレートコレクションにはこれらのフックを専用で使用する必要があるため、カスタムには使用すべきではありません。このリソースの代わりに、以下に記載する **OS::TripleO::NodeExtraConfigPost** フックを使用してください。

オーバークラウドの作成完了後に、最初に作成したオーバークラウドまたは次回の更新で、追加設定を全ロールに追加する必要がある状況が発生する可能性があります。そのような場合には、以下のような設定後のフックを使用します。

OS::TripleO::NodeExtraConfigPost

Puppet のコア設定後に全ノードに適用される追加の設定

以下の例では、まず基本的な Heat テンプレート (`/home/stack/templates/nameserver.yaml`) を作成します。このテンプレートは、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します。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  servers:
    type: json
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string
```

```
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}
```

この例では、**resources** セクションに以下が含まれています。

CustomExtraConfig

これは、ソフトウェアの設定を定義します。上記の例では、Bash **script** を定義しており、Heat は **_NAMESERVER_IP_** を **nameserver_ip** パラメーターに保存されている値に置き換えます。

CustomExtraDeployments

これは、**CustomExtraConfig** リソースのソフトウェア設定で指定されているソフトウェアの設定を実行します。次の点に注意してください。

- **config** パラメーターは、**CustomExtraConfig** リソースへの参照を作成して、適用する設定を Heat が認識するようにします。
- **servers** パラメーターはオーバークラウドノードのマップを取得します。このパラメーターは親テンプレートにより提供され、このフックを使用するテンプレートでは必須です。
- **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。この場合は、オーバークラウドが作成された時にのみ設定を適用します。実行可能なアクションは **CREATE**、**UPDATE**、**DELETE**、**SUSPEND** および **RESUME** です。
- **input_values** には **deploy_identifier** と呼ばれるパラメーターが含まれます。これは、親テンプレートからの **DeployIdentifier** を保存します。このパラメーターは、デプロイメントが更新される度にリソースにタイムスタンプを付けます。これにより、そのリソースは以降のオーバークラウドの更新に再度適用されるようになります。

次に、**OS::Triple0::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。

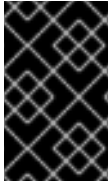
```
resource_registry:
  OS::Triple0::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml

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スタックにこの環境ファイルを追加します。たとえば、以下のコマンドを実行します。

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

このコマンドにより、オーバークラウドの初期作成またはその後の更新時にコア設定が完了した後に、全ノードに設定が適用されます。



重要

OS::TripleO::NodeExtraConfigPost は、1 つの Heat テンプレートに対してのみ登録することが可能です。複数で使用すると、使用する Heat テンプレートが上書きされます。

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

Heat テンプレートコレクションには、追加の設定を特定のノードタイプに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は、ノードの Puppet の設定用 hieradata として設定を保存します。これには、以下のパラメーターが含まれます。

ControllerExtraConfig

コントローラーノードに追加する設定

ComputeExtraConfig

コンピュートノードに追加する設定

BlockStorageExtraConfig

Block Storage ノードに追加する設定

ObjectStorageExtraConfig

Object Storage ノードに追加する設定

CephStorageExtraConfig

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]** はコンポーザブルロール名に置き換えます。

ExtraConfig

全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 に増やして、VNC キーマップを日本語に設定するには、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含めます。

**重要**

各パラメーターは 1 回のみ定義することが可能です。その後に使用すると、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 を取得する必要があります。

```
$ 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含めます。

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 hieradata 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方法が提供されます。

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ります。これには、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。

次に、**OS::Triple0::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::Triple0::NodeExtraConfigPost:
    /home/stack/templates/custom_puppet_config.yaml
```

最後に、オープンクラウドのスタックが作成または更新されたら、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** からの設定がオープンクラウド内の全ノードに適用されます。

第5章 オーバークラウドの登録

オーバークラウドでは、Red Hat コンテンツ配信ネットワーク、Red Hat Satellite 5 サーバー、Red Hat Satellite 6 サーバーのいずれかにノードを登録することができます。

5.1. 環境ファイルを使用したオーバークラウドの登録

登録ファイルを Heat テンプレートコレクションからコピーします。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration ~/templates/.
```

`~/templates/rhel-registration/environment-rhel-registration.yaml` を編集し、登録の方法と詳細に応じて以下の値を変更します。

一般的なパラメーター

`rhel_reg_method`

登録の方法を選択します。**portal**、**satellite**、**disable** のいずれかです。

`rhel_reg_type`

登録するユニットの種別。**system** として登録するには空欄のままにします。

`rhel_reg_auto_attach`

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。この機能を無効にするには、このパラメーターを環境ファイルから削除します。

`rhel_reg_service_level`

自動アタッチメントに使用するサービスレベル

`rhel_reg_release`

このパラメーターを使用して、自動アタッチメント用のリリースバージョンを設定します。Red Hat サブスクリプションマネージャーからのデフォルトを使用するには、空欄のままにします。

`rhel_reg_pool_id`

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。

`rhel_reg_sat_url`

オーバークラウドノードを登録する Satellite サーバーのベース URL。このパラメーターには、HTTPS URL ではなく、Satellite の HTTP URL を使用します。たとえば、<https://satellite.example.com> ではなく <http://satellite.example.com> を使用します。オーバークラウドの作成プロセスではこの URL を使用して、どのサーバーが Red Hat Satellite 5 または Red Hat Satellite 6 サーバーであるかを判断します。Red Hat Satellite 6 サーバーの場合は、オーバークラウドは **katello-ca-consumer-latest.noarch.rpm** ファイルを取得して **subscription-manager** に登録し、**katello-agent** をインストールします。Red Hat Satellite 5 サーバーの場合はオーバークラウドは、**RHN-ORG-TRUSTED-SSL-CERT** ファイルを取得して **rhgreg_ks** に登録します。

`rhel_reg_server_url`

使用するサブスクリプションサービスのホスト名を指定します。デフォルトは、カスタマーポータルのサブスクリプション管理「subscription.rhn.redhat.com」です。このオプションを使用しない場合、システムはカスタマーポータルのサブスクリプション管理に登録されます。サブスクリプシ

ンサーバーの URL は、<https://hostname:port/prefix> の形式を使用します。

rhel_reg_base_url

更新を受信するためのコンテンツ配信サーバーのホスト名を指定します。デフォルトは <https://cdn.redhat.com> です。Satellite 6 は独自のコンテンツをホストするため、URL は Satellite 6 で登録されているシステムに使用する必要があります。コンテンツのベース URL <https://hostname:port/prefix> の形式を使用します。

rhel_reg_org

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。

rhel_reg_environment

選択した組織内で使用する環境

rhel_reg_repos

有効化するリポジトリのコンマ区切りリスト

rhel_reg_activation_key

登録に使用するアクティベーションキー

rhel_reg_user、rhel_reg_password

登録用のユーザー名およびパスワード。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。

rhel_reg_machine_name

マシン名。ノードのホスト名を使用するには、空欄のままにします。

rhel_reg_force

登録のオプションを強制するには **true** に設定します (例: ノードの再登録時など)。

rhel_reg_sat_repo

Red Hat Satellite 6 の管理ツール (**katello-agent** など) が含まれているリポジトリ。リポジトリ名が Red Hat Satellite のバージョンに対応した正しい名前であることを確認し、また Satellite サーバーと同期されていることをチェックします。たとえば、**rhel-7-server-satellite-tools-6.2-rpms** は Red Hat Satellite 6.2 に対応します。

アップグレードのパラメーター

UpdateOnRHELRegistration

True に設定すると、登録が完了した後にオーバークラウドパッケージの更新がトリガーされます。デフォルトでは **False** に設定されます。

HTTP プロキシのパラメーター

rhel_reg_http_proxy_host

HTTP プロキシのホスト名。例: **proxy.example.com**

rhel_reg_http_proxy_port

HTTP プロキシ通信用のポート。例: **8080**。

rhel_reg_http_proxy_username

HTTP プロキシにアクセスするためのユーザー名

rhel_reg_http_proxy_password

HTTP プロキシにアクセスするためのパスワード

**重要**

プロキシサーバーを使用する場合には、`rhel_reg_http_proxy_host` パラメーターで定義されているホストへのルートが、全オーバークラウドノードにあることを確認してください。このホストへのルートがなければ、`subscription-manager` がタイムアウトして、デプロイメントが失敗する原因となります。

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。~/templates/rhel-registration/environment-rhel-registration.yaml と ~/templates/rhel-registration/rhel-registration-resource-registry.yaml の両方を追加します。以下に例を示します。

```
$ openstack overcloud deploy --templates [...] -e
/home/stack/templates/rhel-registration/environment-rhel-registration.yaml
-e /home/stack/templates/rhel-registration/rhel-registration-resource-
registry.yaml
```

**重要**

登録は、`OS::TripleO::NodeExtraConfig` Heat リソースとして設定されます。これは、このリソースを登録のみに使用できることを意味します。詳しくは、[「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」](#)を参照してください。

5.2. 例 1: カスタマーポータルへの登録

以下の設定は、`my-openstack` アクティベーションキー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に登録し、`1a85f9223e3d5e43013e3d6e8ff506fd` のプールをサブスクライブします。

```
parameter_defaults:
  rhel_reg_auto_attach: ""
  rhel_reg_activation_key: "my-openstack"
  rhel_reg_org: "1234567"
  rhel_reg_pool_id: "1a85f9223e3d5e43013e3d6e8ff506fd"
  rhel_reg_repos: "rhel-7-server-rpms,rhel-7-server-extras-rpms,rhel-7-
server-rh-common-rpms,rhel-ha-for-rhel-7-server-rpms,rhel-7-server-
openstack-13-rpms,rhel-7-server-rhceph-3-osd-rpms,rhel-7-server-rhceph-3-
mon-rpms,rhel-7-server-rhceph-3-tools-rpms"
  rhel_reg_method: "portal"
  rhel_reg_sat_repo: ""
  rhel_reg_base_url: ""
  rhel_reg_environment: ""
  rhel_reg_force: ""
  rhel_reg_machine_name: ""
  rhel_reg_password: ""
  rhel_reg_release: ""
  rhel_reg_sat_url: ""
  rhel_reg_server_url: ""
  rhel_reg_service_level: ""
  rhel_reg_user: ""
  rhel_reg_type: ""
  rhel_reg_http_proxy_host: ""
```

```
rhel_reg_http_proxy_port: ""
rhel_reg_http_proxy_username: ""
rhel_reg_http_proxy_password: ""
```

5.3. 例 2: RED HAT SATELLITE 6 サーバーへの登録

以下の設定は、**my-openstack** アクティベーションキー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に登録し、**1a85f9223e3d5e43013e3d6e8ff506fd** のプールをサブスクライブします。この場合は、アクティベーションキーで有効化するレポジトリも指定します。

```
parameter_defaults:
  rhel_reg_activation_key: "my-openstack"
  rhel_reg_org: "1"
  rhel_reg_pool_id: "1a85f9223e3d5e43013e3d6e8ff506fd"
  rhel_reg_method: "satellite"
  rhel_reg_sat_url: "http://sat6.example.com"
  rhel_reg_sat_repo: "rhel-7-server-satellite-tools-6.2-rpms"
  rhel_reg_repos: ""
  rhel_reg_auto_attach: ""
  rhel_reg_base_url: ""
  rhel_reg_environment: ""
  rhel_reg_force: ""
  rhel_reg_machine_name: ""
  rhel_reg_password: ""
  rhel_reg_release: ""
  rhel_reg_server_url: ""
  rhel_reg_service_level: ""
  rhel_reg_user: ""
  rhel_reg_type: ""
  rhel_reg_http_proxy_host: ""
  rhel_reg_http_proxy_port: ""
  rhel_reg_http_proxy_username: ""
  rhel_reg_http_proxy_password: ""
```

5.4. 例 3: RED HAT SATELLITE 5 サーバーへの登録

以下の設定は、**my-openstack** アクティベーションキーを使用してオーバークラウドノードを sat5.example.com にある Red Hat Satellite 5 サーバーに登録し、サブスクリプションを自動的にアタッチします。この場合は、アクティベーションキーで有効化するレポジトリも指定します。

```
parameter_defaults:
  rhel_reg_auto_attach: ""
  rhel_reg_activation_key: "my-openstack"
  rhel_reg_org: "1"
  rhel_reg_method: "satellite"
  rhel_reg_sat_url: "http://sat5.example.com"
  rhel_reg_repos: ""
  rhel_reg_base_url: ""
  rhel_reg_environment: ""
  rhel_reg_force: ""
  rhel_reg_machine_name: ""
  rhel_reg_password: ""
  rhel_reg_pool_id: ""
  rhel_reg_release: ""
```

```

rhel_reg_server_url: ""
rhel_reg_service_level: ""
rhel_reg_user: ""
rhel_reg_type: ""
rhel_reg_sat_repo: ""
rhel_reg_http_proxy_host: ""
rhel_reg_http_proxy_port: ""
rhel_reg_http_proxy_username: ""
rhel_reg_http_proxy_password: ""

```

5.5. 例 4: HTTP プロキシを介した登録

以下のサンプルパラメーターは、必要な登録方法向けの HTTP プロキシの設定値を設定します。

```

parameter_defaults:
...
rhel_reg_http_proxy_host: "proxy.example.com"
rhel_reg_http_proxy_port: "8080"
rhel_reg_http_proxy_username: "proxyuser"
rhel_reg_http_proxy_password: "p@55w0rd!"
...

```

5.6. 高度な登録方法

一部の状況では、異なるサブスクリプションタイプに異なるロールを登録する必要がある場合があります。たとえば、コントローラーノードのみを OpenStack Platform サブスクリプションにサブスクライブして、Ceph Storage ノードを Ceph Storage サブスクリプションにするとします。本項では、ロールごとに別々のサブスクリプションを割り当てるのに役立つ高度な登録方法をいくつか紹介します。

設定フック

その1つとして、ロール固有のスクリプトを記述して、ロール固有のフックと共に追加する方法があります。たとえば、以下のスニペットを **OS::TripleO::ControllerExtraConfigPre** リソースのテンプレートに追加することができます。これにより、コントローラーノードのみがサブスクリプションの情報を受け取るようになります。

```

ControllerRegistrationConfig:
  type: OS::Heat::SoftwareConfig
  properties:
    group: script
    config:
      str_replace:
        template: |
          #!/bin/sh
          sudo subscription-manager register --org 1234567 \
            --activationkey "my-openstack"
          sudo subscription-manager attach --pool
1a85f9223e3d5e43013e3d6e8ff506fd
          sudo subscription-manager repos --enable rhel-7-server-rpms \
            --enable rhel-7-server-extras-rpms \
            --enable rhel-7-server-rh-common-rpms \
            --enable rhel-ha-for-rhel-7-server-rpms \
            --enable rhel-7-server-openstack-13-rpms \
            --enable rhel-7-server-rhceph-3-mon-rpms \

```

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

このスクリプトでは、一式の **subscription-manager** コマンドを使用してシステムを登録し、サブスクリプションをアタッチして、必要なリポジトリを有効化します。

フックについての詳しい情報は、「[4章 設定フック](#)」を参照してください。

Ansible ベースの設定

director の動的インベントリースクリプトを使用して、特定のロールで Ansible ベース登録を実行することができます。たとえば、以下のプレイを使用してコントローラーノードを登録します。

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-7-server-rpms
      - rhel-7-server-extras-rpms
      - rhel-7-server-rh-common-rpms
      - rhel-ha-for-rhel-7-server-rpms
      - rhel-7-server-openstack-13-rpms
      - rhel-7-server-rhceph-3-mon-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        activationkey: my-openstack
        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、アクティベーションキーを使用したノードの登録、自動的に有効化されたリポジトリの無効化、コントローラーノードに関連したリポジトリのみの有効化の3つのタスクが含まれています。リポジトリは **repos** 変数でリストされます。

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
```


このコマンドは、動的インベントリスクリプトを実行して、ホストとそのグループの一覧を取得し、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 **Controller** グループ) 内のノードに、その Playbook のタスクを適用します。

オーバークラウドでの Ansible 自動化の実行に関する詳しい情報は、『**director** のインストールと使用方法』ガイドの「[Ansible 自動化の実行](#)」を参照してください。

第6章 ANSIBLE ベースのオーバークラウド登録



重要

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

「[5章 オーバークラウドの登録](#)」に記載の **rhel-registration** の方法の代わりとして、director で Ansible ベースの方法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 サーバーに登録することができます。この方法を使用するには、オーバークラウドで Ansible ベースの設定 (**config-download**) を有効化する必要があります。

6.1. 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に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::Triple0::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。以下に例を示します。

```
resource_registry:
  OS::Triple0::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、登録に関連した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-7-server-rpms
      - rhel-7-server-extras-rpms
      - rhel-7-server-rh-common-rpms
      - rhel-ha-for-rhel-7-server-rpms
      - rhel-7-server-openstack-13-rpms
      - rhel-7-server-rhceph-3-osd-rpms
      - rhel-7-server-rhceph-3-mon-rpms
      - rhel-7-server-rhceph-3-tools-rpms
    rhsm_activation_key: "my-openstack"
    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。

6.2. RHSMVARS サブパラメーター

rhsm	説明
rhsm_method	登録の方法を選択します。 portal 、 satellite 、 disable のいずれかです。
rhsm_org_id	登録に使用する組織。この ID を特定するには、アンダークラウドノードから sudo subscription-manager orgs を実行します。プロンプトが表示されたら、Red Hat の認証情報を入力して、出力される Key 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_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ための Satellite サーバーのベース URL
rhsm_repos	有効化するリポジトリの一覧
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、登録用のアクティベーションキーを使用します。
rhsm_rhsm_proxy_hostname	HTTP プロキシのホスト名。例: proxy.example.com
rhsm_rhsm_proxy_port	HTTP プロキシ通信用のポート。例: 8080 。
rhsm_rhsm_proxy_user	HTTP プロキシにアクセスするためのユーザー名
rhsm_rhsm_proxy_password	HTTP プロキシ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。

6.3. 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-7-server-rpms
      - rhel-7-server-extras-rpms
      - rhel-7-server-rh-common-rpms
      - rhel-ha-for-rhel-7-server-rpms
      - rhel-7-server-openstack-13-rpms
      - rhel-7-server-rhceph-3-osd-rpms
      - rhel-7-server-rhceph-3-mon-rpms
      - rhel-7-server-rhceph-3-tools-rpms
    rhsm_activation_key: "my-openstack"
    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。

6.4. 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はロールベースで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 つのセットを適用し、コンピューターノードに異なる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-7-server-rpms
  - rhel-7-server-extras-rpms
  - rhel-7-server-rh-common-rpms
  - rhel-ha-for-rhel-7-server-rpms
  - rhel-7-server-openstack-13-rpms
  - rhel-7-server-rhceph-3-osd-rpms
  - rhel-7-server-rhceph-3-mon-rpms
  - rhel-7-server-rhceph-3-tools-rpms
rhsm_activation_key: "my-openstack"
rhsm_org_id: "1234567"
rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
rhsm_method: "portal"
ComputeParameters:
  RhsmVars:
    rhsm_repos:
      - rhel-7-server-rpms
      - rhel-7-server-extras-rpms
      - rhel-7-server-rh-common-rpms
      - rhel-ha-for-rhel-7-server-rpms
      - rhel-7-server-openstack-13-rpms
      - rhel-7-server-rhceph-3-tools-rpms
    rhsm_activation_key: "my-openstack"
    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、「[5章 オーバークラウドの登録](#)」の **rhel-registration** メソッドを使用する場合には、Ansible ベースのメソッドに切り替えるために **rhsm** を無効化する必要があります。**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。

6.5. RHSM コンポーザブルサービスへの切り替え

バッシュスクリプトを実行してオーバークラウドの登録を処理する **rhel-registration** メソッド。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。

この手順では、**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方法を説明します。

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。大半の場合、これは以下のファイルです。

- **rhel-registration/environment-rhel-registration.yaml**
- **rhel-registration/rhel-registration-resource-registry.yaml**

2. **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
```

上記の行を以下のように変更します。

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

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。

6.6. 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_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-registration	rhsm / RhsmVars
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。

6.7. 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。

このプロセスでは、**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** コマンドの実行時には、**config-download** のオプションおよび環境ファイルと、**rhsm.yaml** 環境ファイルを含めてください。

```
openstack overcloud deploy \
  <other cli args> \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/config-download-environment.yaml \
  --config-download \
  -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
```

第7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 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は、以下のような設定を行う手段を提供します。

- 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。

7.1. サポートされているカスタムロールアーキテクチャー

テスト/検証済みのコンポーザブルサービスの組み合わせはごく限られています。Red Hat は、カスタムロールとコンポーザブルを使用する場合に、以下のアーキテクチャーをサポートしています。

アーキテクチャー 1: モノリシックコントローラー

すべてのコントローラーサービスが単一の Controller ロールに含まれます。これはデフォルトのアーキテクチャーです。詳しくは、「[サービスアーキテクチャー: モノリシックコントローラー](#)」を参照してください。

アーキテクチャー 2: 分割コントローラー

コントローラーサービスが2つのロールに分割されます。

- Controller PCMK: データベースやロードバランシングなど、Pacemaker の管理対象のコアサービス
- コントローラー Systemd: systemd の管理対象の OpenStack Platform サービス

詳しくは、「[サービスアーキテクチャー: 分割コントローラー](#)」を参照してください。

アーキテクチャー 3: スタンドアロンロール

OpenStack Platform のサービスを分割する以外は、アーキテクチャー 1 またはアーキテクチャー 2 を使用します。詳しくは、「[サービスアーキテクチャー: スタンドアロンロール](#)」を参照してください。

7.2. 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。

systemd サービスの場合:

- サポートされているスタンドアロンのカスタムロールに **systemd** の管理対象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 **systemd** サービスをスケーリングすることができます。

Pacemaker の管理対象サービスの場合:

- サポートされている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::Triple0::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::Triple0::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てす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::Triple0::Services::Pacemaker**) を追加しないでください。
- **OS::Triple0::Services::Pacemaker** または **OS::Triple0::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

- Red Hat OpenStack Platform 12 から 13 へ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。
- オーバー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の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。

7.3. ロール

7.3.1. 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::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CephClient
    ...
- name: Compute
  description: |
    Basic Compute Node role
  ServicesDefault:
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::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
```

7.3.2. 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 role list** コマンドを使用します。

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

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

```
$ openstack overcloud role 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** ロールにスケールアップされることを意味します。オーバークラウドは、**Controller** 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、**Networker**、**Messaging**、**Database** のロール用のサービスが含まれていない **ControllerOpenStack** ロールも含まれていません。**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。

7.3.3. 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。ロール名についての推奨事項は、「[サービスアーキテクチャー：分割コントローラー](#)」に記載の例を参照してください。

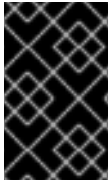
description

(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。このロールがコントローラー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一覧。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

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』ガイドの「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」の章](#)を参照してください。

disable_upgrade_deployment

(任意) 特定のロールのアップグレードを無効にするかどうかを定義します。これにより、1つのロールのノードを個別にアップグレードしてサービスの可用性を確保する方法が提供されます。たとえば、Compute と Swift Storage のロールにこのパラメーターを使用します。

upgrade_batch_size

(任意) アップグレード中に 1 回にまとめて実行するタスクの数を定義します。1つのタスクは、1ノードあたりの 1 アップグレードステップとして数えられます。デフォルトのバッチサイズは 1 です。これは、アップグレードプロセスによって各ノードで 1 回に実行されるアップグレードステップは 1 つであることを意味します。バッチサイズを大きくすると、ノードで同時に実行されるタスクの数が増えます。

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** パラメータを使用して、対応する各 **OS::Heat::ResourceGroup** リソースを命名します。

7.3.4. 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

また、**CountDefault** を **1** に設定して、デフォルトのオープンクラウドには常に **Horizon** ノードが含まれるようにした方がよいでしょう。

既存のオープン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。新規オープンクラウドを作成して、OpenStack Dashboard がスタンドアロン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
```

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、コンピューターノードが 1 台、Networker 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
```

7.4. コンポーザブルサービス

7.4.1. コンポーザブルサービスアーキテクチャーの考察

Heat のコア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が 2 セット含まれています。

- **puppet/services** には、コンポーザブルサービスを設定するためのベーステンプレートが含まれています。
- **docker/services** には、主要な OpenStack Platform サービスのコンテナ化されたテンプレートが含まれます。これらのテンプレートは、一部のベーステンプレートの拡張として機能し、そのベーステンプレートを参照します。

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、**ntp.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、コンポーザブルサービスのベーステンプレートを直接使用します。以下に例を示します。

```
resource_registry:
  ...
  OS::TripleO::Services::Ntp: puppet/services/time/ntp.yaml
  ...
```

ただし、コアサービスにはコンテナが必要なため、コンテナサービステンプレートを使用してください。たとえば、**keystone** のコンテナ化されたサービスには、以下を使用します。

```
resource_registry:
    ...
    OS::TripleO::Services::Keystone: docker/services/keystone.yaml
    ...
```

これらのコンテナ化されたテンプレートは、通常、Puppet の設定を含めるためにベーステンプレートを参照します。たとえば、**docker/services/keystone.yaml** テンプレートには、**KeystoneBase** パラメーター内のベーステンプレートの出力が保管されます。

```
KeystoneBase:
    type: ../../puppet/services/keystone.yaml
```

コンテナ化されたテンプレートには、ベーステンプレートからの機能とデータを組み入れることができます。

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。

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。

7.4.2. 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
```

7.4.3. 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、以下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](#)』ガイドの「[Installation](#)」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。

7.4.4. 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 7 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 (『[Monitoring Tools Configuration Guide](#)』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 7 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 7 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 7 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。

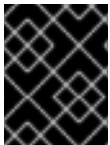
```
$ openstack overcloud deploy --templates -r
~/templates/roles_data_with_generic.yaml -e ~/templates/generic-node-
params.yaml
```

このコマンドにより、コントローラーノードが 1 台、コンピューターノードが 1 台、汎用 Red Hat Enterprise Linux 7 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。

7.5. アーキテクチャー

7.5.1. サービスアーキテクチャー：モノリシックコントローラー

コンポーザブルサービスのデフォルトのアーキテクチャーは、Red Hat OpenStack Platform のコアサービスを含むモノリシックなコントローラーを使用します。これらのデフォルトサービスは、director の Heat テンプレートコレクション (**/usr/share/openstack-tripleo-heat-templates/roles_data.yaml**) に含まれるロールファイルで定義されます。



重要

一部のサービスはデフォルトで無効化されています。それらのサービスを有効化するための情報は、[「無効化されたサービスの有効化」](#)を参照してください。

```
- name: Controller # the 'primary' role goes first
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMds
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRbdMirror
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::CinderBackendDellPs
    - OS::TripleO::Services::CinderBackendDellSc
    - OS::TripleO::Services::CinderBackendNetApp
    - OS::TripleO::Services::CinderBackendScaleIO
    - OS::TripleO::Services::Congress
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::GlanceApi
```

- OS::Triple0::Services::HeatApi
- OS::Triple0::Services::HeatApiCfn
- OS::Triple0::Services::HeatApiCloudwatch
- OS::Triple0::Services::HeatEngine
- OS::Triple0::Services::MySQL
- OS::Triple0::Services::MySQLClient
- OS::Triple0::Services::NeutronDhcpAgent
- OS::Triple0::Services::NeutronL3Agent
- OS::Triple0::Services::NeutronMetadataAgent
- OS::Triple0::Services::NeutronApi
- OS::Triple0::Services::NeutronCorePlugin
- OS::Triple0::Services::NeutronOvsAgent
- OS::Triple0::Services::RabbitMQ
- OS::Triple0::Services::HAproxy
- OS::Triple0::Services::Keepalived
- OS::Triple0::Services::Memcached
- OS::Triple0::Services::Pacemaker
- OS::Triple0::Services::Redis
- OS::Triple0::Services::NovaConductor
- OS::Triple0::Services::MongoDb
- OS::Triple0::Services::NovaApi
- OS::Triple0::Services::NovaPlacement
- OS::Triple0::Services::NovaMetadata
- OS::Triple0::Services::NovaScheduler
- OS::Triple0::Services::NovaConsoleauth
- OS::Triple0::Services::NovaVncProxy
- OS::Triple0::Services::Ec2Api
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::SwiftProxy
- OS::Triple0::Services::SwiftStorage
- OS::Triple0::Services::SwiftRingBuilder
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Sshd
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::CeilometerApi
- OS::Triple0::Services::CeilometerCollector
- OS::Triple0::Services::CeilometerExpirer
- OS::Triple0::Services::CeilometerAgentCentral
- OS::Triple0::Services::CeilometerAgentNotification
- OS::Triple0::Services::Horizon
- OS::Triple0::Services::GnocchiApi
- OS::Triple0::Services::GnocchiMetricd
- OS::Triple0::Services::GnocchiStatsd
- OS::Triple0::Services::ManilaApi
- OS::Triple0::Services::ManilaScheduler
- OS::Triple0::Services::ManilaBackendGeneric
- OS::Triple0::Services::ManilaBackendNetapp
- OS::Triple0::Services::ManilaBackendCephFs
- OS::Triple0::Services::ManilaShare
- OS::Triple0::Services::AodhApi
- OS::Triple0::Services::AodhEvaluator
- OS::Triple0::Services::AodhNotifier
- OS::Triple0::Services::AodhListener
- OS::Triple0::Services::SaharaApi
- OS::Triple0::Services::SaharaEngine
- OS::Triple0::Services::IronicApi

```

- OS::Triple0::Services::IronicConductor
- OS::Triple0::Services::NovaIronic
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::OpenDaylightApi
- OS::Triple0::Services::OpenDaylightOvs
- OS::Triple0::Services::SensuClient
- OS::Triple0::Services::FluentdClient
- OS::Triple0::Services::Collectd
- OS::Triple0::Services::BarbicanApi
- OS::Triple0::Services::PankoApi
- OS::Triple0::Services::Tacker
- OS::Triple0::Services::Zaqar
- OS::Triple0::Services::OVNDBs
- OS::Triple0::Services::NeutronML2FujitsuCfab
- OS::Triple0::Services::NeutronML2FujitsuFossw
- OS::Triple0::Services::CinderHPELeftHandISCSI
- OS::Triple0::Services::Etcd
- OS::Triple0::Services::AuditD
- OS::Triple0::Services::OctaviaApi
- OS::Triple0::Services::OctaviaHealthManager
- OS::Triple0::Services::OctaviaHousekeeping
- OS::Triple0::Services::OctaviaWorker

```

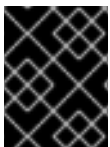
7.5.2. サービスアーキテクチャー：分割コントローラー

コントローラーノード上のサービスを2つのロールに分割することができます。

- **Controller PCMK:** Pacemaker が管理するコアサービスのみを含みます (データベース、ロードバランシングなど)。
- **Controller systemd:** 全 OpenStack サービスを含みます。

残りのデフォルトロール (Compute、Ceph Storage、Object Storage、Block Storage) は引き続き影響を受けません。

分割コントローラーのアーキテクチャーを作成するには、以下の表を参考にしてください。



重要

一部のサービスはデフォルトで無効化されています。それらのサービスを有効化するための情報は、[「無効化されたサービスの有効化」](#)を参照してください。

Controller PCMK

以下のサービスは、Controller PCMK ロールが必要とする最小限のサービスです。

```

- name: Controller
  ServicesDefault:
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Sshd
    - OS::Triple0::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::CephClient
- OS::TripleO::Services::CephExternal
- OS::TripleO::Services::CinderBackup
- OS::TripleO::Services::CinderVolume
- OS::TripleO::Services::HAproxy
- OS::TripleO::Services::Keepalived
- OS::TripleO::Services::ManilaBackendGeneric
- OS::TripleO::Services::ManilaBackendNetapp
- OS::TripleO::Services::ManilaBackendCephFs
- OS::TripleO::Services::ManilaShare
- OS::TripleO::Services::Memcached
- OS::TripleO::Services::MySQL
- OS::TripleO::Services::Pacemaker
- OS::TripleO::Services::RabbitMQ
- OS::TripleO::Services::Redis

Controller systemd

以下の表には、Controller systemd ロールで利用可能なサービスをまとめています。

- name: ControllerSystemd
 - 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::AodhApi
 - OS::TripleO::Services::AodhEvaluator
 - OS::TripleO::Services::AodhListener
 - OS::TripleO::Services::AodhNotifier
 - OS::TripleO::Services::CeilometerAgentCentral
 - OS::TripleO::Services::CeilometerAgentNotification
 - OS::TripleO::Services::CeilometerApi
 - OS::TripleO::Services::CeilometerCollector
 - OS::TripleO::Services::CeilometerExpirer
 - OS::TripleO::Services::CephClient
 - OS::TripleO::Services::CephExternal
 - OS::TripleO::Services::CephMon
 - OS::TripleO::Services::CephRgw

```

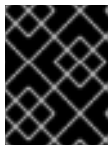
- OS::Triple0::Services::CinderApi
- OS::Triple0::Services::CinderScheduler
- OS::Triple0::Services::GlanceApi
- OS::Triple0::Services::GnocchiApi
- OS::Triple0::Services::GnocchiMetricd
- OS::Triple0::Services::GnocchiStatsd
- OS::Triple0::Services::HeatApi
- OS::Triple0::Services::HeatApiCfn
- OS::Triple0::Services::HeatApiCloudwatch
- OS::Triple0::Services::HeatEngine
- OS::Triple0::Services::Horizon
- OS::Triple0::Services::IronicApi
- OS::Triple0::Services::IronicConductor
- OS::Triple0::Services::Keystone
- OS::Triple0::Services::ManilaApi
- OS::Triple0::Services::ManilaScheduler
- OS::Triple0::Services::MongoDb
- OS::Triple0::Services::MySQLClient
- OS::Triple0::Services::NeutronApi
- OS::Triple0::Services::NeutronCorePlugin
- OS::Triple0::Services::NeutronCorePluginML2OVN
- OS::Triple0::Services::NeutronCorePluginMidonet
- OS::Triple0::Services::NeutronCorePluginNuage
- OS::Triple0::Services::NeutronCorePluginOpencontrail
- OS::Triple0::Services::NeutronCorePluginPlumgrid
- OS::Triple0::Services::NeutronDhcpAgent
- OS::Triple0::Services::NeutronL3Agent
- OS::Triple0::Services::NeutronMetadataAgent
- OS::Triple0::Services::NeutronOvsAgent
- OS::Triple0::Services::NovaApi
- OS::Triple0::Services::NovaConductor
- OS::Triple0::Services::NovaConsoleauth
- OS::Triple0::Services::NovaIronic
- OS::Triple0::Services::NovaPlacement
- OS::Triple0::Services::NovaScheduler
- OS::Triple0::Services::NovaVncProxy
- OS::Triple0::Services::OpenDaylightApi
- OS::Triple0::Services::OpenDaylightOvs
- OS::Triple0::Services::PankoApi
- OS::Triple0::Services::SaharaApi
- OS::Triple0::Services::SaharaEngine
- OS::Triple0::Services::SwiftProxy
- OS::Triple0::Services::SwiftRingBuilder

```

7.5.3. サービスアーキテクチャー: スタンドアロンロール

以下の表は、Red Hat OpenStack Platform のコンポーザブルサービスアーキテクチャーで作成/スケールリングが可能なサポート対象のカスタムロールを一覧にまとめています。これらのコレクションを個別のロールとしてまとめて、以前のアーキテクチャーと組み合わせてサービスを分離/分割するのに使用してください。

- 「サービスアーキテクチャー: モノリシックコントローラー」
- 「サービスアーキテクチャー: 分割コントローラー」



重要

一部のサービスはデフォルトで無効化されています。それらのサービスを有効化するための情報は、[「無効化されたサービスの有効化」](#)を参照してください。

すべてのロールは、以下を含む **共通のサービス** セットを使用する点に注意してください。

- **OS::TripleO::Services::AuditD**
- **OS::TripleO::Services::CACerts**
- **OS::TripleO::Services::CertmongerUser**
- **OS::TripleO::Services::Collectd**
- **OS::TripleO::Services::ContainersLogrotateCron**
- **OS::TripleO::Services::Docker**
- **OS::TripleO::Services::FluentdClient**
- **OS::TripleO::Services::Kernel**
- **OS::TripleO::Services::Ntp**
- **OS::TripleO::Services::SensuClient**
- **OS::TripleO::Services::Snmp**
- **OS::TripleO::Services::Timezone**
- **OS::TripleO::Services::TripleoFirewall**
- **OS::TripleO::Services::TripleoPackages**
- **OS::TripleO::Services::Tuned**

オーバークラウドに追加するロールを選択したら、メインの Controller ロールから関連付けられたサービスを削除します (**共通のサービス** は除く)。たとえば、スタンドアロンの [Keystone](#) ロールを作成する場合は、Controller ノードから **OS::TripleO::Services::Apache** および

OS::TripleO::Services::Keystone サービスを削除します。唯一の例外は、カスタムロールサポートが限定されているサービスです ([表7.1「カスタムロールのサポート」](#)を参照)。

以下の表でロールをクリックすると、そのロールに関連付けられているサービスが表示されます。

表7.1 カスタムロールのサポート

ロール	サポートの状態
Ceph Storage Monitor	対応
Ceph Storage OSD	対応

ロール	サポートの状態
Ceph Storage RadosGW	限定。分割する場合には、このサービスは Controller systemd ロールの一部にする必要があります。
Cinder API	対応
Controller PCMK	対応
Database	対応
Glance	対応
Heat	対応
Horizon	対応
Ironic	対応
Keystone	対応
Load Balancer	対応
Manila	限定。分割する場合には、このサービスは Controller systemd ロールの一部にする必要があります。
Message Bus	対応
Networker	対応
Neutron API	対応
Nova	対応
Nova Compute	対応
OpenDaylight	テクノロジープレビュー
Redis	対応
Sahara	限定。分割する場合には、このサービスは Controller systemd ロールの一部にする必要があります。

ロール	サポートの状態
Swift API	対応
Swift Storage	対応
Telemetry	対応

Ceph Storage Monitor

以下のサービスは、Ceph Storage Monitor を構成します。

```
- name: CephMon
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CertmongerUser
    - OS::Triple0::Services::Collectd
    - OS::Triple0::Services::Docker
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::ContainersLogrotateCronD
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::Tuned
    # Role-Specific Services
    - OS::Triple0::Services::CephMon
```

Ceph Storage OSD

以下のサービスは、Ceph Storage OSD を構成します。

```
- name: CephStorage
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CertmongerUser
    - OS::Triple0::Services::Collectd
    - OS::Triple0::Services::Docker
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::ContainersLogrotateCronD
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
```

```

- OS::TripleO::Services::TripleoPackages
- OS::TripleO::Services::Tuned
# Role-Specific Services
- OS::TripleO::Services::CephOSD

```

Ceph Storage RadosGW

以下のサービスは、Ceph Storage RadosGW を構成します。これらのサービスを分離する場合には、**Controller systemd** ロールの一部にする必要があります。

```

# Common Services
- OS::TripleO::Services::AuditD
- OS::TripleO::Services::CACerts
- OS::TripleO::Services::CertmongerUser
- OS::TripleO::Services::Collectd
- OS::TripleO::Services::Docker
- OS::TripleO::Services::FluentdClient
- OS::TripleO::Services::Kernel
- OS::TripleO::Services::Ntp
- OS::TripleO::Services::ContainersLogrotateCronD
- OS::TripleO::Services::SensuClient
- OS::TripleO::Services::Snmp
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::TripleoPackages
- OS::TripleO::Services::Tuned
# Role-Specific Services
- OS::TripleO::Services::CephRgw

```

Cinder API

以下のサービスは、OpenStack Block Storage API を構成します。

```

- name: CinderApi
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCronD
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderScheduler

```

Controller PCMK

以下のサービスは、Controller PCMK がスタンドアロンロールとして必要とする最小限のサービスです。

```
- name: Controller
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCron
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Keepalived
    - OS::TripleO::Services::ManilaBackendGeneric
    - OS::TripleO::Services::ManilaBackendNetapp
    - OS::TripleO::Services::ManilaBackendCephFs
    - OS::TripleO::Services::ManilaShare
    - OS::TripleO::Services::Memcached
    - OS::TripleO::Services::Pacemaker
```

これは、分割コントローラーのアーキテクチャーにおける [Controller PCMK](#) ロールと同じですが、以下の高可用性サービスをスタンドアロンロールに分割できる点が異なります。

- [Database](#)
- [Load Balancer](#)
- [Message Bus](#)
- [Redis](#)

上記のサービスのスタンドアロンロールを作成しない場合には、それらのロールのサービスは Controller PCMK スタンドアロンロールにマージします。

Database

以下のサービスは、メインのデータベースを構成します。このデータベースは MariaDB によって、Pacemaker を使用する Galera クラスターとして管理されます。

```
- name: Database
  ServicesDefault:
    # Common Services
```

```

- OS::Triple0::Services::AuditD
- OS::Triple0::Services::CACerts
- OS::Triple0::Services::CertmongerUser
- OS::Triple0::Services::Collectd
- OS::Triple0::Services::Docker
- OS::Triple0::Services::FluentdClient
- OS::Triple0::Services::Kernel
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::ContainersLogrotateCronD
- OS::Triple0::Services::SensuClient
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::Tuned
# Role-Specific Services
- OS::Triple0::Services::Pacemaker
- OS::Triple0::Services::MySQL

```

Glance

以下のサービスは、OpenStack Image サービスを構成します。

```

- name: Glance
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CertmongerUser
    - OS::Triple0::Services::Collectd
    - OS::Triple0::Services::Docker
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::ContainersLogrotateCronD
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::Tuned
    # Role-Specific Services
    - OS::Triple0::Services::CephClient
    - OS::Triple0::Services::CephExternal
    - OS::Triple0::Services::GlanceApi

```

Heat

以下のサービスは、OpenStack Orchestration サービスを構成します。

```

- name: Heat
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CertmongerUser

```

```

- OS::TripleO::Services::Collectd
- OS::TripleO::Services::Docker
- OS::TripleO::Services::FluentdClient
- OS::TripleO::Services::Kernel
- OS::TripleO::Services::Ntp
- OS::TripleO::Services::ContainersLogrotateCron
- OS::TripleO::Services::SensuClient
- OS::TripleO::Services::Snmp
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::TripleoPackages
- OS::TripleO::Services::Tuned
# Role-Specific Services
- OS::TripleO::Services::HeatApi
- OS::TripleO::Services::HeatApiCfn
- OS::TripleO::Services::HeatApiCloudwatch
- OS::TripleO::Services::HeatEngine

```

Horizon

以下のサービスは、OpenStack Dashboard を構成します。

```

- name: Horizon
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCron
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::Horizon

```

Ironic

以下のサービスは、OpenStack Bare Metal Provisioning サービスを構成します。

```

- name: Ironic
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker

```

```

- OS::TripleO::Services::FluentdClient
- OS::TripleO::Services::Kernel
- OS::TripleO::Services::Ntp
- OS::TripleO::Services::ContainersLogrotateCron
- OS::TripleO::Services::SensuClient
- OS::TripleO::Services::Snmp
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::TripleoPackages
- OS::TripleO::Services::Tuned
# Role-Specific Services
- OS::TripleO::Services::IronicApi
- OS::TripleO::Services::IronicConductor
- OS::TripleO::Services::IronicPxe

```

以下の点に注意してください。

- Storage ネットワークへのアクセスが必要です。
- **OS::TripleO::Services::IronicApi** サービスは、要件に応じて、**Ironic** ロールまたは **Controller** ロールに割り当てることができます。
- **Controller** ロールに **OS::TripleO::Services::NovaIronic** サービスが割り当てられている必要があります。

Keystone

以下のサービスは、OpenStack Identity サービスを構成します。マイナーな更新を実行する際には、他のサービスを更新する前にこのロールを必ず更新してください。

```

- name: Keystone
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCron
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::Keystone

```

Load Balancer

以下のサービスは、オーバークラウドのロードバランサーを構成します。ロードバランサーは、Pacemaker を使用して管理される HAProxy です。

```
- name: LoadBalancer
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CertmongerUser
    - OS::Triple0::Services::Collectd
    - OS::Triple0::Services::Docker
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::ContainersLogrotateCronD
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::Tuned
    # Role-Specific Services
    - OS::Triple0::Services::Pacemaker
    - OS::Triple0::Services::HAproxy
```

Manila

以下のサービスは、OpenStack Shared File System サービスを構成します。これらのサービスを分離する場合には、**Controller systemd** ロールの一部にする必要があります。

```
# Common Services
- OS::Triple0::Services::AuditD
- OS::Triple0::Services::CACerts
- OS::Triple0::Services::CertmongerUser
- OS::Triple0::Services::Collectd
- OS::Triple0::Services::Docker
- OS::Triple0::Services::FluentdClient
- OS::Triple0::Services::Kernel
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::ContainersLogrotateCronD
- OS::Triple0::Services::SensuClient
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::Tuned
# Role-Specific Services
- OS::Triple0::Services::ManilaApi
- OS::Triple0::Services::ManilaScheduler
```

Message Bus

以下のサービスは、メッセージングキューを構成します。メッセージングキューは、Pacemaker で管理される RabbitMQ です。

```
- name: MessageBus
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
```



```

- OS::Triple0::Services::CACerts
- OS::Triple0::Services::CertmongerUser
- OS::Triple0::Services::Collectd
- OS::Triple0::Services::Docker
- OS::Triple0::Services::FluentdClient
- OS::Triple0::Services::Kernel
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::ContainersLogrotateCron
- OS::Triple0::Services::SensuClient
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::Tuned
# Role-Specific Services
- OS::Triple0::Services::Pacemaker
- OS::Triple0::Services::RabbitMQ

```

Networker

以下のサービスは、OpenStack Networking エージェントを構成します。

```

- name: Networker
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CertmongerUser
    - OS::Triple0::Services::Collectd
    - OS::Triple0::Services::Docker
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::ContainersLogrotateCron
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::Tuned
    # Role-Specific Services
    - OS::Triple0::Services::NeutronDhcpAgent
    - OS::Triple0::Services::NeutronL3Agent
    - OS::Triple0::Services::NeutronMetadataAgent
    - OS::Triple0::Services::NeutronOvsAgent

```

Neutron API

以下のサービスは、OpenStack Networking API を構成します。

```

- name: NeutronApi
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CertmongerUser

```

```

- OS::TripleO::Services::Collectd
- OS::TripleO::Services::Docker
- OS::TripleO::Services::FluentdClient
- OS::TripleO::Services::Kernel
- OS::TripleO::Services::Ntp
- OS::TripleO::Services::ContainersLogrotateCron
- OS::TripleO::Services::SensuClient
- OS::TripleO::Services::Snmp
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::TripleoPackages
- OS::TripleO::Services::Tuned
# Role-Specific Services
- OS::TripleO::Services::NeutronApi
- OS::TripleO::Services::NeutronCorePlugin
- OS::TripleO::Services::NeutronCorePluginML2OVN
- OS::TripleO::Services::NeutronCorePluginMidonet
- OS::TripleO::Services::NeutronCorePluginNuage
- OS::TripleO::Services::NeutronCorePluginOpencontrail
- OS::TripleO::Services::NeutronCorePluginPlumgrid

```

Nova

以下のサービスは、OpenStack Compute サービスを構成します。

```

- name: Nova
  ServicesDefault:
    # Common Services
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCron
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    # Role-Specific Services
    - OS::TripleO::Services::NovaApi
    - OS::TripleO::Services::NovaConductor
    - OS::TripleO::Services::NovaConsoleauth
    - OS::TripleO::Services::NovaScheduler
    - OS::TripleO::Services::NovaPlacement
    - OS::TripleO::Services::NovaVncProxy

```

Nova Compute

以下のサービスは、OpenStack Compute ノードを構成します。

```

- name: Compute

```

```

ServicesDefault:
  # Common Services
  - OS::Triple0::Services::AuditD
  - OS::Triple0::Services::CACerts
  - OS::Triple0::Services::CertmongerUser
  - OS::Triple0::Services::Collectd
  - OS::Triple0::Services::Docker
  - OS::Triple0::Services::FluentdClient
  - OS::Triple0::Services::Kernel
  - OS::Triple0::Services::Ntp
  - OS::Triple0::Services::ContainersLogrotateCrond
  - OS::Triple0::Services::SensuClient
  - OS::Triple0::Services::Snmp
  - OS::Triple0::Services::Timezone
  - OS::Triple0::Services::TripleoFirewall
  - OS::Triple0::Services::TripleoPackages
  - OS::Triple0::Services::Tuned
  # Role-Specific Services
  - OS::Triple0::Services::CephClient
  - OS::Triple0::Services::CephExternal
  - OS::Triple0::Services::ComputeCeilometerAgent
  - OS::Triple0::Services::ComputeNeutronCorePlugin
  - OS::Triple0::Services::ComputeNeutronL3Agent
  - OS::Triple0::Services::ComputeNeutronMetadataAgent
  - OS::Triple0::Services::ComputeNeutronOvsAgent
  - OS::Triple0::Services::NeutronOvsAgent
  - OS::Triple0::Services::NeutronSriovAgent
  - OS::Triple0::Services::NovaCompute
  - OS::Triple0::Services::NovaLibvirt
  - OS::Triple0::Services::OpenDaylightOvs

```

OpenDaylight

以下のサービスは、OpenDayLight を構成します。これらのサービスは、**Red Hat OpenStack Platform 11** ではテクノロジープレビューとして提供しています。

```

- name: Opendaylight
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CertmongerUser
    - OS::Triple0::Services::Collectd
    - OS::Triple0::Services::Docker
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::ContainersLogrotateCrond
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::Tuned

```

```
# Role-Specific Services
- OS::Triple0::Services::OpenDaylightApi
- OS::Triple0::Services::OpenDaylightOvs
```

Redis

以下のサービスは、Pacemaker で管理される Redis を構成します。

```
- name: Redis
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CertmongerUser
    - OS::Triple0::Services::Collectd
    - OS::Triple0::Services::Docker
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::ContainersLogrotateCronD
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::Tuned
    # Role-Specific Services
    - OS::Triple0::Services::Pacemaker
    - OS::Triple0::Services::Redis
```

Sahara

以下のサービスは、OpenStack Clustering サービスを構成します。これらのサービスを分離する場合には、**Controller systemd** ロールの一部にする必要があります。

```
# Common Services
- OS::Triple0::Services::AuditD
- OS::Triple0::Services::CACerts
- OS::Triple0::Services::CertmongerUser
- OS::Triple0::Services::Collectd
- OS::Triple0::Services::Docker
- OS::Triple0::Services::FluentdClient
- OS::Triple0::Services::Kernel
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::ContainersLogrotateCronD
- OS::Triple0::Services::SensuClient
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::Tuned
# Role-Specific Services
- OS::Triple0::Services::SaharaApi
- OS::Triple0::Services::SaharaEngine
```

Swift API

以下のサービスは、OpenStack Object Storage API を構成します。

```
- name: SwiftApi
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CertmongerUser
    - OS::Triple0::Services::Collectd
    - OS::Triple0::Services::Docker
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::ContainersLogrotateCronD
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::Tuned
    # Role-Specific Services
    - OS::Triple0::Services::SwiftProxy
    - OS::Triple0::Services::SwiftRingBuilder
```

Swift Storage

以下のサービスは、OpenStack Object Storage サービスを構成します。

```
- name: ObjectStorage
  ServicesDefault:
    # Common Services
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CertmongerUser
    - OS::Triple0::Services::Collectd
    - OS::Triple0::Services::Docker
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::ContainersLogrotateCronD
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::Tuned
    # Role-Specific Services
    - OS::Triple0::Services::SwiftRingBuilder
    - OS::Triple0::Services::SwiftStorage
```

Telemetry

以下のサービスは、OpenStack Telemetry サービスを構成します。

```
- name: Telemetry
  ServicesDefault:
```

```
# Common Services
- OS::TripleO::Services::AuditD
- OS::TripleO::Services::CACerts
- OS::TripleO::Services::CertmongerUser
- OS::TripleO::Services::Collectd
- OS::TripleO::Services::Docker
- OS::TripleO::Services::FluentdClient
- OS::TripleO::Services::Kernel
- OS::TripleO::Services::Ntp
- OS::TripleO::Services::ContainersLogrotateCron
- OS::TripleO::Services::SensuClient
- OS::TripleO::Services::Snmp
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::TripleoPackages
- OS::TripleO::Services::Tuned
# Role-Specific Services
- OS::TripleO::Services::Apache
- OS::TripleO::Services::AodhApi
- OS::TripleO::Services::AodhEvaluator
- OS::TripleO::Services::AodhListener
- OS::TripleO::Services::AodhNotifier
- OS::TripleO::Services::CeilometerAgentCentral
- OS::TripleO::Services::CeilometerAgentNotification
- OS::TripleO::Services::CeilometerApi
- OS::TripleO::Services::CeilometerCollector
- OS::TripleO::Services::CeilometerExpirer
- OS::TripleO::Services::GnocchiApi
- OS::TripleO::Services::GnocchiMetricd
- OS::TripleO::Services::GnocchiStatsd
- OS::TripleO::Services::MongoDb
- OS::TripleO::Services::PankoApi
```

7.6. コンポーザブルサービスのリファレンス

以下の表には、Red Hat OpenStack Platform 13 で利用可能な全コンポーザブルサービスをまとめています。

- [表7.2 「以前のバージョンから引き続き提供されているサービス」](#)
- [表7.3 「Red Hat OpenStack Platform 13 の新しいサービス」](#)



重要

一部のサービスはデフォルトで無効化されています。それらのサービスを有効化するための情報は、「[無効化されたサービスの有効化](#)」を参照してください。

表7.2 以前のバージョンから引き続き提供されているサービス

サービス	説明
OS::TripleO::Services::AodhApi	Puppet で設定される OpenStack Telemetry Alarming (aodh) API サービス

サービス	説明
OS::Triple0::Services::AodhEvaluator	Puppet で設定される OpenStack Telemetry Alarming (aodh) Evaluator サービス
OS::Triple0::Services::AodhListener	Puppet で設定される OpenStack Telemetry Alarming (aodh) Listener サービス
OS::Triple0::Services::AodhNotifier	Puppet で設定される OpenStack Telemetry Alarming (aodh) Notifier サービス
OS::Triple0::Services::Apache	Puppet で設定される Apache サービス。通常このサービスは、Apache で実行されるサービスには自動的に含まれる点に注意してください。
OS::Triple0::Services::CACerts	Puppet で設定される HAProxy サービス
OS::Triple0::Services::CeilometerAgentCentral	Puppet で設定される OpenStack Telemetry (ceilometer) Central Agent サービス
OS::Triple0::Services::CeilometerAgentNotification	Puppet で設定される OpenStack Telemetry (ceilometer) Notification Agent サービス
OS::Triple0::Services::CeilometerApi	Puppet で設定される OpenStack Telemetry (ceilometer) API サービス
OS::Triple0::Services::CeilometerCollector	Puppet で設定される OpenStack Telemetry (ceilometer) Collector サービス
OS::Triple0::Services::CeilometerExpirer	Puppet で設定される OpenStack Telemetry (ceilometer) Expirer サービス
OS::Triple0::Services::CephClient	(デフォルトでは無効) Ceph Client サービス
OS::Triple0::Services::CephExternal	(デフォルトでは無効) Ceph External サービス
OS::Triple0::Services::CephMon	(デフォルトでは無効) Ceph Monitor サービス
OS::Triple0::Services::CephOSD	(デフォルトでは無効) Ceph OSD サービス
OS::Triple0::Services::CinderApi	Puppet で設定される OpenStack Block Storage (cinder) API サービス
OS::Triple0::Services::CinderBackup	(デフォルトでは無効) Puppet で設定される OpenStack Block Storage (cinder) Backup サービス

サービス	説明
OS::TripleO::Services::CinderScheduler	Puppet で設定される OpenStack Block Storage (cinder) Scheduler サービス
OS::TripleO::Services::CinderVolume	Puppet で設定される OpenStack Block Storage (cinder) Volume サービス (Pacemaker の管理対象)
OS::TripleO::Services::ComputeCeilometerAgent	Puppet で設定される OpenStack Telemetry (ceilometer) Compute Agent サービス
OS::TripleO::Services::ComputeNeutronCorePlugin	Puppet で設定される OpenStack Networking (neutron) ML2 プラグイン
OS::TripleO::Services::ComputeNeutronL3Agent	(デフォルトでは無効) Puppet で設定される、DVR 対応のコンピュータード用 OpenStack Networking (neutron) L3 エージェント
OS::TripleO::Services::ComputeNeutronMetadataAgent	(デフォルトでは無効) Puppet で設定される OpenStack Networking (neutron) Metadata エージェント
OS::TripleO::Services::ComputeNeutronOvsAgent	Puppet で設定される OpenStack Networking (neutron) OVS エージェント
OS::TripleO::Services::FluentdClient	(デフォルトでは無効) Puppet で設定される Fluentd クライアント
OS::TripleO::Services::GlanceApi	Puppet で設定される OpenStack Image (glance) API サービス
OS::TripleO::Services::GnocchiApi	Puppet で設定される OpenStack Telemetry Metrics (gnocchi) サービス
OS::TripleO::Services::GnocchiMetricd	Puppet で設定される OpenStack Telemetry Metrics (gnocchi) サービス
OS::TripleO::Services::GnocchiStatsd	Puppet で設定される OpenStack Telemetry Metrics (gnocchi) サービス
OS::TripleO::Services::HAproxy	Puppet で設定される HAProxy サービス (Pacemaker の管理対象)
OS::TripleO::Services::HeatApi	Puppet で設定される OpenStack Orchestration (heat) API サービス
OS::TripleO::Services::HeatApiCfn	Puppet で設定される OpenStack Orchestration (heat) CloudFormation API サービス

サービス	説明
OS::Triple0::Services::HeatApiCloudwatch	Puppet で設定される OpenStack Orchestration (heat) CloudWatch API サービス
OS::Triple0::Services::HeatEngine	Puppet で設定される OpenStack Orchestration (heat) Engine サービス
OS::Triple0::Services::Horizon	Puppet で設定される OpenStack Dashboard (horizon) サービス
OS::Triple0::Services::IronicApi	(デフォルトでは無効) Puppet で設定される OpenStack Bare Metal Provisioning (ironic) API
OS::Triple0::Services::IronicConductor	(デフォルトでは無効) Puppet で設定される OpenStack Bare Metal Provisioning (ironic) コンダクター
OS::Triple0::Services::Keepalived	Puppet で設定される Keepalived サービス
OS::Triple0::Services::Kernel	kmod でカーネルモジュールを読み込み、sysctl でカーネルオプションを設定
OS::Triple0::Services::Keystone	Puppet で設定される OpenStack Identity (keystone) サービス
OS::Triple0::Services::ManilaApi	(デフォルトでは無効) Puppet で設定される OpenStack Shared File Systems (manila) API サービス
OS::Triple0::Services::ManilaScheduler	(デフォルトでは無効) Puppet で設定される OpenStack Shared File Systems (manila) Scheduler サービス
OS::Triple0::Services::ManilaShare	(デフォルトでは無効) Puppet で設定される OpenStack Shared File Systems (manila) Share サービス
OS::Triple0::Services::Memcached	Puppet で設定される Memcached サービス
OS::Triple0::Services::MongoDb	Puppet を使用した MongoDB サービスのデプロイメント
OS::Triple0::Services::MySQL	Puppet を使用する MySQL (Pacemaker の管理対象) サービスのデプロイメント
OS::Triple0::Services::NeutronApi	Puppet で設定される OpenStack Networking (neutron) サーバー

サービス	説明
OS::Triple0::Services::NeutronCorePlugin	Puppet で設定される OpenStack Networking (neutron) ML2 プラグイン
OS::Triple0::Services::NeutronCorePluginML2OVN	Puppet で設定される OpenStack Networking (neutron) ML2/OVN プラグイン
OS::Triple0::Services::NeutronCorePluginMidonet	OpenStack Networking (neutron) Midonet プラグインおよびサービス
OS::Triple0::Services::NeutronCorePluginNuage	OpenStack Networking (neutron) Nuage プラグイン
OS::Triple0::Services::NeutronCorePluginOpencontrail	OpenStack Networking (neutron) Opencontrail プラグイン
OS::Triple0::Services::NeutronCorePluginPlumgrid	OpenStack Networking (neutron) Plumgrid プラグイン
OS::Triple0::Services::NeutronDhcpAgent	Puppet で設定される OpenStack Networking (neutron) DHCP エージェント
OS::Triple0::Services::NeutronL3Agent	Puppet で設定される OpenStack Networking (neutron) L3 エージェント
OS::Triple0::Services::NeutronMetadataAgent	Puppet で設定される OpenStack Networking (neutron) Metadata エージェント
OS::Triple0::Services::NeutronOvsAgent	Puppet で設定される OpenStack Networking (neutron) OVS エージェント
OS::Triple0::Services::NeutronServer	Puppet で設定される OpenStack Networking (neutron) サーバー
OS::Triple0::Services::NeutronSriovAgent	(デフォルトでは無効) Puppet で設定される OpenStack Neutron SR-IOV nic エージェント
OS::Triple0::Services::NovaApi	Puppet で設定される OpenStack Compute (nova) API サービス
OS::Triple0::Services::NovaCompute	Puppet で設定される OpenStack Compute (nova) Compute サービス
OS::Triple0::Services::NovaConductor	Puppet で設定される OpenStack Compute (nova) Conductor サービス

サービス	説明
OS::Triple0::Services::NovaConsoleauth	Puppet で設定される OpenStack Compute (nova) Consoleauth サービス
OS::Triple0::Services::NovaIronic	(デフォルトでは無効) Puppet で設定される、Ironic を使用する OpenStack Compute (nova) サービス
OS::Triple0::Services::NovaLibvirt	Puppet で設定される Libvirt サービス
OS::Triple0::Services::NovaScheduler	Puppet で設定される OpenStack Compute (nova) Scheduler サービス
OS::Triple0::Services::NovaVncProxy	Puppet で設定される OpenStack Compute (nova) Vncproxy サービス
OS::Triple0::Services::Ntp	Puppet を使用した NTP サービスのデプロイメント
OS::Triple0::Services::OpenDaylightApi	(デフォルトでは無効) OpenDaylight SDN のコントローラー
OS::Triple0::Services::OpenDaylightOvs	(デフォルトでは無効) OpenDaylight OVS の設定
OS::Triple0::Services::Pacemaker	Puppet で設定される Pacemaker サービス
OS::Triple0::Services::RabbitMQ	Puppet で設定される RabbitMQ サービス (Pacemaker の管理対象)
OS::Triple0::Services::Redis	Puppet で設定される OpenStack Redis サービス
OS::Triple0::Services::SaharaApi	(デフォルトでは無効) Puppet で設定される OpenStack Clustering (sahara) API サービス
OS::Triple0::Services::SaharaEngine	(デフォルトでは無効) Puppet で設定される OpenStack Clustering (sahara) Engine サービス
OS::Triple0::Services::SensuClient	(デフォルトでは無効) Puppet で設定される Sensu クライアント
OS::Triple0::Services::Snmp	Puppet で設定される SNMP クライアント。アンダークラウドでの Ceilometer のハードウェアモニタリングを円滑にします。このサービスは、ハードウェアモニタリングを有効化するのに必要です。
OS::Triple0::Services::SwiftProxy	Puppet で設定される OpenStack Object Storage (swift) Proxy サービス

サービス	説明
OS::TripleO::Services::SwiftRingBuilder	OpenStack Object Storage (swift) Ringbuilder
OS::TripleO::Services::SwiftStorage	Puppet で設定される OpenStack Object Storage (swift) サービス
OS::TripleO::Services::Timezone	コンポーザブルな Timezone サービス
OS::TripleO::Services::TripleoFirewall	ファイアウォールの設定
OS::TripleO::Services::TripleoPackages	パッケージのインストールの設定

表7.3 Red Hat OpenStack Platform 13 の新しいサービス

サービス	説明
OS::TripleO::Services::ApacheTLS	(デフォルトでは無効) TLS/SSL 対応の Apache サービス。このサービスは、Certmonger ベースの TLS/SSL 設定 (environments/enable-internal-tls.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::AuditD	(デフォルトでは無効) 監査サービスを実装します。監査サービスの環境ファイル (environments/auditd.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::CephMds	(デフォルトでは無効) Ceph Metadata Server (MDS)。Ceph MDS の環境ファイル (environments/services/ceph-mds.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::CephRbdMirror	(デフォルトでは無効) Ceph Storage RBD Mirroring サービス。RBD ミラーリングの環境ファイル (environments/services/ceph-rbdmirror.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::CephRgw	(デフォルトでは無効) Ceph Storage Object Gateway (radosgw)。RadosGW の環境ファイル (environments/ceph-radosgw.yaml) が含まれている場合に有効化されます。また、この環境ファイルにより、OpenStack Object Storage (swift) サービスが無効化されます。

サービス	説明
OS::Triple0::Services::CinderHPELeftHandISCSI	(デフォルトでは無効) Cinder HPE LeftHand iSCSI バックエンド。LeftHand iSCSI の環境ファイル (environments/cinder-hpelefthand-config.yaml) が含まれている場合に有効化されます。
OS::Triple0::Services::Collectd	(デフォルトでは無効) 統計コレクションデーモン。Collectd の環境ファイル (environments/collectd-environment.yaml) が含まれている場合に有効化されます。
OS::Triple0::Services::Congress	(デフォルトでは無効) OpenStack Policy-as-a-Service (Congress)。Congress の環境ファイル (environments/enable_congress.yaml) が含まれている場合に有効化されます。
OS::Triple0::Services::Etcd	(デフォルトでは無効) Etcd のキーと値のストレージ。etcd の環境ファイル (environments/services/etcd.yaml) が含まれている場合に有効化されます。
OS::Triple0::Services::HAProxyInternalTLS	(デフォルトでは無効) TLS/SSL 対応の HAProxy サービス用の内部ネットワーク。このサービスは Certmonger ベースの TLS/SSL 設定 (environments/enable-internal-tls.yaml) が含まれている場合に有効化されます。
OS::Triple0::Services::HAProxyPublicTLS	(デフォルトでは無効) TLS/SSL 対応の HAProxy サービス用の External ネットワーク。このサービスは、Certmonger ベースの TLS/SSL 設定 (environments/services/haproxy-public-tls-certmonger.yaml) が含まれている場合に有効化されます。
OS::Triple0::Services::ManilaBackendCephFs	(デフォルトでは無効) Ceph Storage 用の Manila バックエンド。各バックエンドの環境ファイル (environments/manila-cephfsnative-config.yaml) が含まれている場合に有効化されます。
OS::Triple0::Services::ManilaBackendGeneric	(デフォルトでは無効) 汎用の Manila バックエンド。各バックエンドの環境ファイル (environments/manila-generic-config.yaml) が含まれている場合に有効化されます。

サービス	説明
OS::TripleO::Services::ManilaBackendNetapp	(デフォルトでは無効) NetApp 用の Manila バックエンド。各バックエンドの環境ファイル (environments/manila-netapp-config.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::MistralApi	(デフォルトでは無効) OpenStack Workflow サービス (mistral) API。mistral の環境ファイル (environments/services/mistral.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::MistralEngine	(デフォルトでは無効) OpenStack Workflow サービス (mistral) エンジン。mistral の環境ファイル (environments/services/mistral.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::MistralExecutor	(デフォルトでは無効) OpenStack Workflow サービス (mistral) 実行サーバー。mistral の環境ファイル (environments/services/mistral.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::MySQLClient	データベースクライアント
OS::TripleO::Services::MySQLTLS	(デフォルトでは無効) TLS/SSL 対応のデータベースサービス。このサービスは、Certmonger ベースの TLS/SSL 設定 (environments/enable-internal-tls.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::NeutronML2FujitsuCfab	(デフォルトでは無効) OpenStack Networking (neutron) 向けの Fujitsu C-Fabric プラグイン。C-Fabric の環境ファイル (environments/neutron-ml2-fujitsu-cfab.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::NeutronML2FujitsuFossw	(デフォルトでは無効) OpenStack Networking (neutron) 向けの Fujitsu fossw プラグイン。fossw の環境ファイル (environments/neutron-ml2-fujitsu-fossw.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::NovaMetadata	OpenStack Compute (nova) Metadata サービス
OS::TripleO::Services::NovaPlacement	OpenStack Compute (nova) Placement サービス

サービス	説明
OS::TripleO::Services::OctaviaApi	(デフォルトでは無効) OpenStack Load Balancing-as-a-Service (octavia) API。octavia の環境ファイル (environments/services/octavia.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::OctaviaHealth Manager	(デフォルトでは無効) OpenStack Load Balancing-as-a-Service (octavia) Health Manager。octavia の環境ファイル (environments/services/octavia.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::OctaviaHousekeeping	(デフォルトでは無効) OpenStack Load Balancing-as-a-Service (octavia) Housekeeping サービス。octavia の環境ファイル (environments/services/octavia.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::OctaviaWorker	(デフォルトでは無効) OpenStack Load Balancing-as-a-Service (octavia) Worker サービス。octavia の環境ファイル (environments/services/octavia.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::OVNDBs	(デフォルトでは無効) OVN データベース。OVN の拡張機能 (environments/neutron-m12-ovn.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::PankoApi	OpenStack Telemetry Event Storage (panko)
OS::TripleO::Services::Sshd	(デフォルトでは無効) SSH デーモンの設定。デフォルトのサービスとして含まれます。
OS::TripleO::Services::Tacker	(デフォルトでは無効) OpenStack NFV Orchestration (tacker)。tacker の環境ファイル (environments/enable_tacker.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::TLSProxyBase	(デフォルトでは無効) TLS/SSL を設定するための基本サービス。このサービスは、Certmonger ベースの TLS/SSL 設定 (environments/enable-internal-tls.yaml) が含まれている場合に有効化されます。
OS::TripleO::Services::Zaqar	(デフォルトでは無効) OpenStack Messaging (zaqar)。zaqar の環境ファイル (environments/services/zaqar.yaml) が含まれている場合に有効化されます。

第8章 コンテナ化されたサービス

director は、OpenStack Platform のコアサービスをオープンクラウド上にコンテナとしてインストールします。本項では、コンテナ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。

8.1. コンテナ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プンクラウド上にコンテナとしてインストールします。コンテナ化されたサービス用のテンプレートは、`/usr/share/openstack-tripleo-heat-templates/docker/services/` にあります。これらのテンプレートは、それぞれのコンポーザブルサービステンプレートを参照します。たとえば、OpenStack Identity (keystone) のコンテナ化されたサービスのテンプレート (`docker/services/keystone.yaml`) には、以下のリソースが含まれます。

```
KeystoneBase:
  type: ../../puppet/services/keystone.yaml
  properties:
    EndpointMap: {get_param: EndpointMap}
    ServiceData: {get_param: ServiceData}
    ServiceNetMap: {get_param: ServiceNetMap}
    DefaultPasswords: {get_param: DefaultPasswords}
    RoleName: {get_param: RoleName}
    RoleParameters: {get_param: RoleParameters}
```

type は、それぞれの OpenStack Identity (keystone) コンポーザブルサービスを参照して、そのテンプレートから **outputs** データをプルします。コンテナ化されたサービスは、独自のコンテナ固有データにこのデータをマージします。

コンテナ化されたサービスを使用するノードではすべて、**OS::TripleO::Services::Docker** 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::Docker** サービスをコンテナ化されたサービスとして追加します。たとえば、**Keystone** ロールには、以下の定義を使用します。

```
- name: Keystone
  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::Docker
    - OS::TripleO::Services::Keystone
```


8.2. コンテナ化されたサービスのパラメーター

コンテナ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 docker ボリューム
- **puppet_tags**: 設定中に Puppet に渡すタグ。これらのタグは、特定のサービスの設定リソースに対する Puppet の実行を制限します。たとえば、OpenStack Identity (keystone) のコンテナ化されたサービスは、**keystone_config** タグを使用してすべての必要な **keystone_config** Puppet リソースのみが設定コンテナで実行されるようにします。
- **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上で実行するコマンドを定義するコンテナ固有のデータセット

docker_config

サービスの設定コンテナで実行するタスク。全タスクは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。ステップは以下のとおりです。

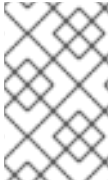
- **ステップ 1**: ロードバランサーの設定
- **ステップ 2**: コアサービス (データベース、Redis)
- **ステップ 3**: OpenStack Platform サービスの初期設定
- **ステップ 4**: OpenStack Platform サービスの全般設定
- **ステップ 5**: サービスのアクティブ化

host_prep_tasks

ベアメタルノードがコンテナ化されたサービスに対応するための準備タスク

8.3. OPENSTACK PLATFORM コンテナの修正

Red Hat は、Red Hat Container Catalog (registry.access.redhat.com) で、事前にビルドされたコンテナイメージを提供しています。これらのイメージを修正して、さらにレイヤーを追加することができます。これは、認定済みのサードパーティードライバーの RPM をコンテナに追加する場合に役立ちます。



注記

修正された OpenStack Platform コンテナイメージが継続的にサポートされるには、修正後のイメージが「[Red Hat Container Support Policy](#)」を順守することを確認してください。

この例には、最新の **openstack-keystone** イメージをカスタマイズする方法を示していますが、これらの手順は、他のイメージにも適用することができます。

1. 編集するイメージをプルします。たとえば、**openstack-keystone** イメージの場合には、以下のコマンドを実行します。

```
$ sudo docker pull registry.access.redhat.com/rhosp12/openstack-keystone:latest
```

2. 元のイメージで、デフォルトのユーザーを確認します。たとえば、**openstack-keystone** イメージの場合には、以下のコマンドを実行します。

```
$ sudo docker run -it registry.access.redhat.com/rhosp12/openstack-keystone:latest whoami
root
```



注記

openstack-keystone イメージは、**root** をデフォルトユーザーとして使用します。その他のイメージは、特定のユーザーを使用します。たとえば、**openstack-glance-api** はデフォルトユーザーに **glance** を使用します。

3. **Dockerfile** を作成して、既存のコンテナイメージ上に追加のレイヤーを構築します。Container Catalog から最新の OpenStack Identity (keystone) イメージをプルして、カスタムの RPM ファイルをイメージにインストールする例を以下に示します。

```
FROM registry.access.redhat.com/rhosp12/openstack-keystone
MAINTAINER Acme
LABEL name="rhosp12/openstack-keystone-acme" vendor="Acme"
version="2.1" release="1"

# switch to root and install a custom RPM, etc.
USER root
COPY custom.rpm /tmp
RUN rpm -ivh /tmp/custom.rpm

# switch the container back to the default user
USER root
```

4. 新規イメージをビルドして、タグ付けします。たとえば、**/home/stack/keystone** ディレクトリーに保管されたローカルの **Dockerfile** でビルドして、アンダークラウドのローカルレジストリーにタグ付けするには、以下のコマンドを実行します。

```
$ docker build /home/stack/keystone -t
"192.168.24.1:8787/rhosp12/openstack-keystone-acme:rev1"
```

5. 編集が終わったイメージをアンダークラウドのローカルレジストリーにプッシュします。

```
$ docker push 192.168.24.1:8787/rhosp12/openstack-keystone-acme:rev1
```

6. オーバークラウドコンテナイメージの環境ファイル (通常は **overcloud_images.yaml**) を編集して、カスタムのコンテナイメージを使用するための適切なパラメーターを変更します。



警告

Container Catalog は、コンテナイメージに完全なソフトウェアスタックを組み込んでパブリッシュします。更新およびセキュリティ問題の修正を含むコンテナイメージを Container Catalog がリリースする際には、既存のカスタムコンテナには、それらの更新は **含まれない** ので、カタログからのイメージを使用して再ビルドする必要があります。

第9章 ネットワークの分離

director は、分離したオーバークラウドネットワークを設定する方法を提供します。つまり、オーバークラウド環境はネットワークトラフィック種別を異なるネットワークに分離して、個別のネットワークインターフェースまたはボンディングにネットワークトラフィックを割り当てます。分離ネットワークを設定した後に、director は OpenStack サービスが分離ネットワークを使用するように設定します。分離ネットワークが設定されていない場合には、サービスはすべて、プロビジョニングネットワーク上で実行されます。

この例では、サービスごとに別のネットワークを使用します。

- NIC1 (プロビジョニング):
 - Provisioning (別名: 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 ネットワーク (VXLAN トンネリング)
 - テナント VLAN / プロバイダー VLAN
 - Storage
 - External VLAN (Floating IP/SNAT)
- NIC4 (管理)
 - Management

この例では、各オーバークラウドノードは、タグ付けられた VLAN でネットワークを提供するために、ボンディング内の残りのネットワークインターフェース 2 つを使用します。以下のネットワーク割り当ては、このボンディングに適用されます。

表9.1 ネットワークサブネットおよび VLAN 割り当て

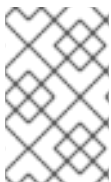
ネットワーク種別	サブネット	VLAN	NIC/グループ
Internal API	172.16.0.0/24	201	NIC2 (コントロール)
Tenant	172.17.0.0/24	202	NIC3 (データ)
Storage	172.18.0.0/24	203	NIC3 (データ)
Storage Management	172.19.0.0/24	204	NIC2 (コントロール)
Management	172.20.0.0/24	205	NIC4 (管理)

ネットワーク種別	サブネット	VLAN	NIC/グループ
External / Floating IP	10.1.1.0/24	100	NIC2 (コントロール) NIC3 (データ)

9.1. カスタムのインターフェーステンプレートの作成

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をカスタマイズして、ロールごとにノードのインターフェースを設定します。このテンプレートは YAML 形式の標準の Heat テンプレート (「[Heat テンプレート](#)」を参照) で、director にはすぐに使用開始できるように、テンプレートサンプルが含まれています。

- **/usr/share/openstack-tripleo-heat-templates/network/config/single-nic-vlans:** このディレクトリーには、ロールごとに VLAN が設定された単一 NIC のテンプレートが含まれます。
- **/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans:** このディレクトリーには、ロール別のボンディング NIC 設定のテンプレートが含まれます。
- **/usr/share/openstack-tripleo-heat-templates/network/config/multiple-nics:** このディレクトリーには、ロール毎に NIC を 1 つ使用して複数の NIC 設定を行うためのテンプレートが含まれています。
- **/usr/share/openstack-tripleo-heat-templates/network/config/single-nic-linux-bridge-vlans:** このディレクトリーには、Open vSwitch ブリッジの代わりに Linux ブリッジを使用してロールベースで単一の NIC に複数の VLAN が接続される構成のテンプレートが含まれます。



注記

これらの例には、デフォルトロールのテンプレートのみが含まれています。カスタムロールのネットワークインターフェース設定を定義するには、これらのテンプレートをベースとして使用してください。

この例では、デフォルトの複数の NIC の設定サンプルをベースとして使用します。**/usr/share/openstack-tripleo-heat-templates/network/config/multiple-nics**にあるバージョンをコピーします。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/network/config/multiple-nics ~/templates/nic-configs
```

このコマンドにより、各ロールに複数の NIC を使用するネットワークインターフェース設定を定義するローカルの Heat テンプレートセットが作成されます。各テンプレートには、標準の **parameters**、**resources**、**output** セクションが含まれます。

parameters

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必要があります。

ただし、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。

パラメーター	説明	種別
ControlPlaneIp	ノードの IP アドレスと Control Plane/Provisioning ネットワーク上のサブネット	文字列
ExternalIpSubnet	ノードの IP アドレスと External ネットワーク上のサブネット	文字列
InternalApiIpSubnet	ノードの IP アドレスと Internal API ネットワーク上のサブネット	文字列
StorageIpSubnet	ノードの IP アドレスと Storage ネットワーク上のサブネット	文字列
StorageMgmtIpSubnet	ノードの IP アドレスと Storage Management ネットワーク上のサブネット	文字列
TenantIpSubnet	ノードの IP アドレスと Tenant ネットワーク上のサブネット	文字列
ManagementIpSubnet	ノードの IP アドレスと Management ネットワーク上のサブネット。 environments/network-management.yaml を追加した場合にのみ設定されます。	文字列
ExternalNetworkVlanID	External ネットワークトラフィック用のノードの VLAN ID	数値
InternalApiNetworkVlanID	Internal API ネットワークトラフィック用のノードの VLAN ID	数値
StorageNetworkVlanID	Storage ネットワークトラフィック用のノードの VLAN ID	数値
StorageMgmtNetworkVlanID	Storage Management ネットワークトラフィック用のノードの VLAN ID	数値
TenantNetworkVlanID	Tenant ネットワークトラフィック用のノードの VLAN ID	数値
ManagementNetworkVlanID	Management ネットワークトラフィック用のノードの VLAN ID	数値

パラメーター	説明	種別
ControlPlaneDefaultRoute	Control Plane/Provisioning ネットワークのデフォルトルート。この設定は、環境ファイルの parameter_defaults セクションで上書きします。	文字列
ExternalInterfaceDefaultRoute	External ネットワーク用のデフォルトルート	文字列
ManagementInterfaceDefaultRoute	Management ネットワークのデフォルトルート	文字列
ControlPlaneSubnetCidr	Control Plane/Provisioning ネットワークのサブネット CIDR。この設定は、環境ファイルの parameter_defaults セクションで上書きします。	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。この設定は、環境ファイルの parameter_defaults セクションで上書きします。	コンマ区切りリスト
EC2MetadataIp	EC2 メタデータサーバーの IP アドレス。この値は、環境ファイルの parameter_defaults セクションで上書きします。	文字列

resources

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: ../../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。これには、以下が含まれます

interface

単一のネットワークインターフェースを定義します。この設定では、実際のインターフェース名 (eth0、eth1、enp0s25) または番号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。

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

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。

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

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。

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

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。

```
- 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) は OVS ブリッジに配置され、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て、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。

linux_bond

複数の **interface** を結合する Linux ボンディングを定義します。これにより、冗長性が向上し、帯域幅が増大します。**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。Linux ボンディングのオプションに関する詳しい情報は、『Red Hat Enterprise Linux 7 ネットワークガイド』の「[4.5.1. ボンディングモジュールのディレクトリ](#)」のセクションを参照してください。

```

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

```

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。

```

- 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}

```

各項目のパラメーターの全一覧は「[23章 ネットワークインターフェースのパラメーター](#)」を参照してください。

以下の設定は、`/home/stack/templates/nic-configs/controller.yaml` ファイルのデフォルトのコントローラーテンプレートに基づいています。ネットワーク (**network-config**) は前述した推奨事項に従って、コントロールグループを OVS ブリッジと分離するように設定されています。

```

resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: ../../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
      device: nic3
      vlan_id:
        get_param: StorageNetworkVlanID
      addresses:
        - ip_netmask:
            get_param: StorageIpSubnet
    - type: vlan
      device: nic3
      vlan_id:
        get_param: TenantNetworkVlanID
      addresses:
        - ip_netmask:
            get_param: TenantIpSubnet

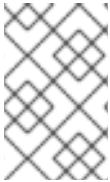
# NIC 4 - Management
- type: interface

```

```

name: nic4
use_dhcp: false
addresses:
- ip_netmask: {get_param: ManagementIpSubnet}
routes:
- default: true
  next_hop: {get_param:
ManagementInterfaceDefaultRoute}

```

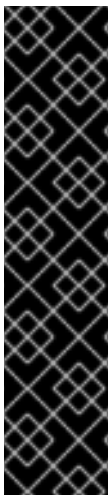


注記

Management ネットワークのセクションは、ネットワークインターフェースの Heat テンプレートにコメントアウトされて含まれています。このセクションをアンコメントして、Management ネットワークを有効化します。

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。

ネットワークインターフェーステンプレートの他のサンプルについては「[付録B ネットワークインターフェースのテンプレート例](#)」を参照してください。



重要

使用していないインターフェースは、不要なデフォルトルートとネットワークループの原因となる可能性があります。たとえば、テンプレートにはネットワークインターフェース (**nic4**) が含まれる可能性があり、このインターフェースは OpenStack のサービス用の IP 割り当てを使用しませんが、DHCP やデフォルトルートを使用します。ネットワークの競合を回避するには、使用していないインターフェースを **ovs_bridge** デバイスから削除し、DHCP とデフォルトのルート設定を無効にします。

```

- type: interface
  name: nic4
  use_dhcp: false
  defroute: false

```

9.2. ネットワーク環境ファイルの作成

ネットワーク環境ファイルは 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、前のセクションの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環境用にカスタマイズします。

director には、すぐに使用開始できるように、環境ファイルのサンプルセットが含まれています。各環境ファイルは、**/usr/share/openstack-tripleo-heat-templates/network/config/** のネットワークインターフェースファイルの例と同じです。

- **/usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml: single-nic-vlans** ネットワークインターフェースディレクトリ内の VLAN 設定が含まれる単一 NIC の環境ファイルサンプルです。External ネットワークの有効化 (**net-single-nic-with-vlans-no-external.yaml**)、または IPv6 の有効化 (**net-single-nic-with-vlans-v6.yaml**) 向けの環境ファイルもあります。

- **/usr/share/openstack-tripleo-heat-templates/environments/net-bond-with-vlans.yaml: bond-with-vlans** ネットワークインターフェースディレクトリー内の VLAN 設定が含まれる単一 NIC の環境ファイルサンプルです。External ネットワークの無効化 (**net-bond-with-vlans-no-external.yaml**)、または IPv6 の有効化 (**net-bond-with-vlans-v6.yaml**) 向けの環境ファイルもあります。
- **/usr/share/openstack-tripleo-heat-templates/environments/net-multiple-nics.yaml: multiple-nics** ネットワークインターフェースディレクトリー内の VLAN 設定が含まれる単一 NIC の環境ファイルサンプルです。IPv6 の有効化 (**net-multiple-nics-v6.yaml**) 向けの環境ファイルもあります。
- **/usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-linux-bridge-with-vlans.yaml: Open vSwitch** ブリッジではなく Linux ブリッジを使用して VLAN 設定を行う単一 NIC の環境ファイルサンプルです。これは、**single-nic-linux-bridge-vlans** ネットワークインターフェースディレクトリーを使用します。

このシナリオでは、**/usr/share/openstack-tripleo-heat-templates/environments/net-multiple-nics.yaml** ファイルの変更版を使用します。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ます。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/net-multiple-nics.yaml /home/stack/templates/network-environment.yaml
```

この環境ファイルには、以下のように変更されたセクションが含まれます。

```
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:
  InternalApiNetCidr: 172.16.0.0/24
  TenantNetCidr: 172.17.0.0/24
  StorageNetCidr: 172.18.0.0/24
  StorageMgmtNetCidr: 172.19.0.0/24
  ManagementNetCidr: 172.20.0.0/24
  ExternalNetCidr: 10.1.1.0/24
  InternalApiAllocationPools: [{'start': '172.16.0.10', 'end': '172.16.0.200'}]
  TenantAllocationPools: [{'start': '172.17.0.10', 'end': '172.17.0.200'}]
  StorageAllocationPools: [{'start': '172.18.0.10', 'end': '172.18.0.200'}]
  StorageMgmtAllocationPools: [{'start': '172.19.0.10', 'end': '172.19.0.200'}]
  ManagementAllocationPools: [{'start': '172.20.0.10', 'end': '172.20.0.200'}]
  # Leave room for floating IPs in the External allocation pool
  ExternalAllocationPools: [{'start': '10.1.1.10', 'end': '10.1.1.50'}]
```

```
# Set to the router gateway on the external network
ExternalInterfaceDefaultRoute: 10.1.1.1
# 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"]
InternalApiNetworkVlanID: 201
StorageNetworkVlanID: 202
StorageMgmtNetworkVlanID: 203
TenantNetworkVlanID: 204
ManagementNetworkVlanID: 205
ExternalNetworkVlanID: 100
NeutronExternalNetworkBridge: ""
```

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変更されたリンクが含まれます。また、このセクションにカスタムロールのネットワークインターフェーステンプレートへのリンクを追加するには、以下の形式を使用します。

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

[ROLE] はロール名に、**[FILE]** はネットワークインターフェースのテンプレートの場所に置き換えます。

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。これらのオプションについての詳しい参考情報は「[付録A ネットワーク環境のオプション](#)」を参照してください。

このシナリオでは、各ネットワークのオプションを定義します。すべてのネットワークの種別で、ホストと仮想 IP への IP アドレス割り当てに使われた個別の VLAN とサブネットを使用します。上記の例では、Internal API ネットワークの割り当てプールは、172.16.0.10 から開始し、172.16.0.200 で終了し、VLAN 201を使用します。これにより、静的な仮想 IP は 172.16.0.10 から 172.16.0.200 までの範囲内で割り当てられる一方で、環境では VLAN 201 が使用されます。

External ネットワークは、Horizon Dashboard とパブリック API をホストします。クラウドの管理と Floating IP の両方に External ネットワークを使用する場合には、仮想マシンインスタンス用の Floating IP として IP アドレスのプールを使用する余裕があることを確認します。本ガイドの例では、10.1.1.10 から 10.1.1.50 までの IP アドレスのみを External ネットワークに割り当て、10.1.1.51 以上は Floating IP アドレスに自由に使用できます。または、Floating IP ネットワークを別の VLAN に配置し、作成後にオーバークラウドを設定してそのネットワークを使用するようにします。

ボンディングされた OVS インターフェースを使用する場合には、**BondInterface0vsOptions** で追加のオプションを設定することができます。詳しい情報は、「[付録C Open vSwitch ボンディングのオプション](#)」を参照してください。



重要

オーバークラウドの作成後にネットワーク設定を変更すると、リソースの可用性が原因で設定に問題が発生する可能性があります。たとえば、ネットワーク分離テンプレートでネットワークのサブネット範囲を変更した場合に、サブネットがすでに使用されているため、再設定が失敗してしまう可能性があります。

9.3. OPENSTACK サービスの分離ネットワークへの割り当て

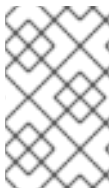
各 OpenStack サービスは、リソースレジストリーでデフォルトのネットワーク種別に割り当てられます。これらのサービスは、そのネットワーク種別に割り当てられたネットワーク内の IP アドレスにバインドされます。OpenStack サービスはこれらのネットワークに分割されますが、実際の物理ネットワーク数はネットワーク環境ファイルに定義されている数と異なる可能性があります。ネットワーク環境ファイル (`/home/stack/templates/network-environment.yaml`) で新たにネットワークマッピングを定義することで、OpenStack サービスを異なるネットワーク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 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 # Changed from storage_mgmt
    CephClusterNetwork: storage # Changed from storage_mgmt
```

これらのパラメーターを **storage** 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 ネットワークに割り当てられます。つまり、**parameter_defaults** セットを Storage Management ネットワークではなく Storage ネットワーク向けに定義するだけで設定することができます。

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 は次に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。



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。

9.4. デプロイするネットワークの選択

通常、ネットワークとポートの環境ファイルにある **resource_registry** セクションは変更する必要はありません。ネットワークの一覧は、ネットワークのサブセットを使用する場合のみ変更してください。



注記

カスタムのネットワークとポートを指定する場合には、デプロイメントのコマンドラインで **environments/network-isolation.yaml** は追加せずに、ネットワークの環境ファイルにネットワークとポートをすべて指定してください。

分離ネットワークを使用するには、各ネットワークのサーバーに IP アドレスを指定する必要があります。分離ネットワーク上の IP アドレスは、アンダークラウドで Neutron を使用して管理できるため、ネットワークごとに Neutron でのポート作成を有効化する必要があります。また、環境ファイルのリソースレジストリーを上書きすることができます。

まず最初に、デプロイ可能なロールごとのデフォルトのネットワークとポートの完全なセットを以下に示します。

```

resource_registry:
    # This section is usually not modified, if in doubt stick to the
    defaults
    # TripleO overcloud networks
    OS::TripleO::Network::External: /usr/share/openstack-tripleo-heat-
templates/network/external.yaml
    OS::TripleO::Network::InternalApi: /usr/share/openstack-tripleo-heat-
templates/network/internal_api.yaml
    OS::TripleO::Network::StorageMgmt: /usr/share/openstack-tripleo-heat-
templates/network/storage_mgmt.yaml
    OS::TripleO::Network::Storage: /usr/share/openstack-tripleo-heat-
templates/network/storage.yaml
    OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-
templates/network/tenant.yaml
    OS::TripleO::Network::Management: /usr/share/openstack-tripleo-heat-
templates/network/management.yaml

    # Port assignments for the VIPs
    OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
    OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
    OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
    OS::TripleO::Network::Ports::StorageMgmtVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
    OS::TripleO::Network::Ports::TenantVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
    OS::TripleO::Network::Ports::ManagementVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml
    OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/vip.yaml

    # Port assignments for the controller role
    OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
    OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
    OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
    OS::TripleO::Controller::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
    OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
    OS::TripleO::Controller::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml

    # Port assignments for the compute role
    OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
    OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-tripleo-
heat-templates/network/ports/storage.yaml
    OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/tenant.yaml
    OS::TripleO::Compute::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml

```



```

# Port assignments for the ceph storage role
OS::TripleO::CephStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
OS::TripleO::CephStorage::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
OS::TripleO::CephStorage::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml

# Port assignments for the swift storage role
OS::TripleO::SwiftStorage::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
OS::TripleO::SwiftStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
OS::TripleO::SwiftStorage::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
OS::TripleO::SwiftStorage::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml

# Port assignments for the block storage role
OS::TripleO::BlockStorage::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
OS::TripleO::BlockStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
OS::TripleO::BlockStorage::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
OS::TripleO::BlockStorage::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.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。

デフォルトのファイルには、デフォルトロールのポート割り当てのみが記載されています。ポートの割り当てをカスタムロールに設定するには、他のリソース定義と同じ規則を使用して、**network/ports** ディレクトリー内の適切な Heat テンプレートにリンクします。

- **OS::TripleO::[ROLE]::Ports::ExternalPort:** /usr/share/openstack-tripleo-heat-templates/network/ports/external.yaml
- **OS::TripleO::[ROLE]::Ports::InternalApiPort:** /usr/share/openstack-tripleo-heat-templates/network/ports/internal_api.yaml
- **OS::TripleO::[ROLE]::Ports::StoragePort:** /usr/share/openstack-tripleo-heat-templates/network/ports/storage.yaml
- **OS::TripleO::[ROLE]::Ports::StorageMgmtPort:** /usr/share/openstack-tripleo-heat-templates/network/ports/storage_mgmt.yaml
- **OS::TripleO::[ROLE]::Ports::TenantPort:** /usr/share/openstack-tripleo-heat-templates/network/ports/tenant.yaml

- **OS::TripleO::[ROLE]::Ports::ManagementPort: /usr/share/openstack-tripleo-heat-templates/network/ports/management.yaml**

[ROLE] は、ロールの名前に置き換えます。

事前設定済みのネットワークの1つを指定せずにデプロイするには、ロールのネットワーク定義および対応するポートの定義を無効にします。たとえば、以下のように **storage_mgmt.yaml** への全参照を **OS::Heat::None** に置き換えることができます。

```
resource_registry:
  # This section is usually not modified, if in doubt stick to the
  defaults
  # TripleO overcloud networks
  OS::TripleO::Network::External: /usr/share/openstack-tripleo-heat-
  templates/network/external.yaml
  OS::TripleO::Network::InternalApi: /usr/share/openstack-tripleo-heat-
  templates/network/internal_api.yaml
  OS::TripleO::Network::StorageMgmt: OS::Heat::None
  OS::TripleO::Network::Storage: /usr/share/openstack-tripleo-heat-
  templates/network/storage.yaml
  OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-
  templates/network/tenant.yaml

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-
  tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-
  tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-
  tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: OS::Heat::None
  OS::TripleO::Network::Ports::TenantVipPort: /usr/share/openstack-
  tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-
  heat-templates/network/ports/vip.yaml

  # Port assignments for the controller role
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-
  tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-
  tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-
  tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: OS::Heat::None
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-
  tripleo-heat-templates/network/ports/tenant.yaml

  # Port assignments for the compute role
  OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-
  tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-tripleo-
  heat-templates/network/ports/storage.yaml
  OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-tripleo-
  heat-templates/network/ports/tenant.yaml

  # Port assignments for the ceph storage role
```

```

OS::Triple0::CephStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
OS::Triple0::CephStorage::Ports::StorageMgmtPort: OS::Heat::None

# Port assignments for the swift storage role
OS::Triple0::SwiftStorage::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
OS::Triple0::SwiftStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
OS::Triple0::SwiftStorage::Ports::StorageMgmtPort: OS::Heat::None

# Port assignments for the block storage role
OS::Triple0::BlockStorage::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
OS::Triple0::BlockStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
OS::Triple0::BlockStorage::Ports::StorageMgmtPort: OS::Heat::None

parameter_defaults:
  ServiceNetMap:
    ApacheNetwork: internal_api
    NeutronTenantNetwork: tenant
    CeilometerApiNetwork: internal_api
    ContrailAnalyticsNetwork: internal_api
    ContrailAnalyticsDatabaseNetwork: internal_api
    ContrailConfigNetwork: internal_api
    ContrailControlNetwork: internal_api
    ContrailDatabaseNetwork: internal_api
    ContrailWebuiNetwork: internal_api
    ContrailTsnNetwork: internal_api
    AodhApiNetwork: internal_api
    PankoApiNetwork: internal_api
    BarbicanApiNetwork: internal_api
    GnocchiApiNetwork: internal_api
    MongodbNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    CongressApiNetwork: internal_api
    GlanceApiNetwork: internal_api
    IronicApiNetwork: ctlplane
    IronicNetwork: ctlplane
    IronicInspectorNetwork: ctlplane
    KeystoneAdminApiNetwork: ctlplane # allows undercloud to config
endpoints
  KeystonePublicApiNetwork: internal_api
  ManilaApiNetwork: internal_api
  NeutronApiNetwork: internal_api
  OctaviaApiNetwork: internal_api
  HeatApiNetwork: internal_api
  HeatApiCfnNetwork: internal_api
  HeatApiCloudwatchNetwork: internal_api
  NovaApiNetwork: internal_api
  NovaColdMigrationNetwork: ctlplane
  NovaPlacementNetwork: internal_api
  NovaMetadataNetwork: internal_api
  NovaVncProxyNetwork: internal_api

```

```
NovaLibvirtNetwork: internal_api
Ec2ApiNetwork: internal_api
Ec2ApiMetadataNetwork: internal_api
TackerApiNetwork: internal_api
SwiftStorageNetwork: storage # Changed from storage_mgmt
SwiftProxyNetwork: storage
SaharaApiNetwork: internal_api
HorizonNetwork: internal_api
MemcachedNetwork: internal_api
RabbitmqNetwork: internal_api
QdrNetwork: internal_api
RedisNetwork: internal_api
MysqlNetwork: internal_api
CephClusterNetwork: storage # Changed from storage_mgmt
CephMonNetwork: storage
CephRgwNetwork: storage
PublicNetwork: external
OpendaylightApiNetwork: internal_api
OvnDbsNetwork: internal_api
MistralApiNetwork: internal_api
ZaqarApiNetwork: internal_api
PacemakerRemoteNetwork: internal_api
EtcdNetwork: internal_api
CephStorageHostnameResolveNetwork: storage
ControllerHostnameResolveNetwork: internal_api
ComputeHostnameResolveNetwork: internal_api
ObjectStorageHostnameResolveNetwork: internal_api
BlockStorageHostnameResolveNetwork: internal_api
```

OS::Heat::None 使用するとネットワークやポートが作成されないため、Storage Management ネットワークのサービスはプロビジョニングネットワークにデフォルト設定されます。Storage Management サービスを Storage ネットワークなどの別のネットワークに移動するには **ServiceNetMap** で変更することができます。

第10章 コンポーザブルネットワークの使用

コンポーザブルネットワークでは、事前定義済みのネットワークセグメント (Internal、Storage、Storage Management、Tenant、External、Control Plane) によって制限されなくなり、その代わりに、独自のネットワークを作成して任意のロール (デフォルトまたはカスタム) に割り当てることができます。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、複数の異なる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。

デプロイされるネットワークの一覧は、単一のデータファイル (**network_data.yaml**) で管理します。それにより、ロールの定義プロセスは、ネットワーク分離を使用して、必要なロールにネットワークを割り当てます (詳しくは「[9章 ネットワークの分離](#)」を参照)。

10.1. コンポーザブルネットワークの定義

コンポーザブルネットワークを作成するには、**/usr/share/openstack-tripleo-heat-templates/network_data.yaml** の Heat テンプレートのローカルコピーを編集します。以下に例を示します。

```
- name: StorageBackup
  vip: true
  name_lower: storage_backup
  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'
  ipv6_subnet: 'fd00:fd00:fd00:7000::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:7000::10', 'end':
'fd00:fd00:fd00:7000:ffff:ffff:ffff:fffe'}]
  gateway_ipv6: 'fd00:fd00:fd00:7000::1'
```

- **name** は、唯一の必須の値です。ただし、**name_lower** を使用して名前を正規化し、読みやすくすることができます。たとえば、**InternalApi** を **internal_api** に変更します。
- **vip:true** は仮想 IP アドレス (VIP) を新規ネットワーク上に作成します。その新規ネットワークのデフォルト値は、残りのパラメーターによって設定されます。
- **ip_subnet** と **allocation_pools** はデフォルトの IPv4 サブネットと、ネットワークの IP 範囲を設定します。
- **ipv6_subnet** および **ipv6_allocation_pools** は、ネットワークのデフォルトの IPv6 サブネットを設定します。



注記

これらのデフォルト値は、環境ファイル (通常は **network-environment.yaml** という名前) を使用して上書きすることができます。使用している director のコア Heat テンプレートのルート (**/usr/share/openstack-tripleo-heat-templates/** のローカルコピー) から以下のコマンドを実行すると、サンプルの **network-environment.yaml** ファイルを作成できます。

```
[stack@undercloud ~/templates] $ ./tools/process-templates.py
```

10.1.1. コンポーザブルネットワーク用のネットワークインターフェース設定の定義

コンポーザブルネットワークを使用する場合には、各ロール (ネットワークが使用しないロールを含む) に使用する NIC 定義テンプレートにネットワーク IP アドレスのパラメーターの定義を追加する必要があります。この NIC 設定の例は、**/usr/share/openstack-tripleo-heat-templates/network/config** のディレクトリーを参照してください。たとえば、**StorageBackup** ネットワークが Ceph ノードにのみ追加される場合には、全ロールの NIC 設定テンプレートのリソース定義に以下の定義を追加する必要があります。

```
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 のリソース定義も作成することができます。

```
StorageBackupNetworkVlanID: # Override this via parameter_defaults in
network_environment.yaml
  default: 60
  description: Vlan ID for the management network traffic.
  type: number
StorageBackupDefaultRoute: # Override this via parameter_defaults in
network_environment.yaml
  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、この例では、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
```

10.1.2. サービスに対するコンポーザブルネットワークの割り当て

カスタムのネットワーク定義で **vip: true** が指定されている場合には、**ServiceNetMap** パラメー

ターを使用してサービスをネットワークに割り当てることができます。サービスに選択されたカスタムネットワークは、サービスをホストするロール上に存在する必要があります。**network_environment.yaml** (または異なる環境ファイル) の **/usr/share/openstack-tripleo-heat-templates/network/service_net_map.j2.yaml** で定義されている **ServiceNetMap** を上書きすることによって、デフォルトのネットワークを上書きすることができます。

```
parameter_defaults:
  ServiceNetMap:
    NeutronTenantNetwork: tenant
    CeilometerApiNetwork: internal_api
    AodhApiNetwork: internal_api
    GnocchiApiNetwork: internal_api
    MongoDBNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    GlanceApiNetwork: storage
    GlanceRegistryNetwork: internal_api
    KeystoneAdminApiNetwork: ctlplane # Admin connection for Undercloud
    KeystonePublicApiNetwork: internal_api
    NeutronApiNetwork: internal_api
    HeatApiNetwork: internal_api
    NovaApiNetwork: internal_api
    NovaMetadataNetwork: internal_api
    NovaVncProxyNetwork: internal_api
    SwiftMgmtNetwork: storage_backup # Changed from storage_mgmt
    SwiftProxyNetwork: storage
    SaharaApiNetwork: internal_api
    HorizonNetwork: internal_api
    MemcachedNetwork: internal_api
    RabbitMqNetwork: internal_api
    RedisNetwork: internal_api
    MysqlNetwork: internal_api
    CephClusterNetwork: storage_backup # Changed from storage_mgmt
    CephPublicNetwork: storage
    ControllerHostnameResolveNetwork: internal_api
    ComputeHostnameResolveNetwork: internal_api
    BlockStorageHostnameResolveNetwork: internal_api
    ObjectStorageHostnameResolveNetwork: internal_api
    CephStorageHostnameResolveNetwork: storage
```

10.1.3. ルーティングネットワークの定義

コンポーザブルネットワークを使用してルーティングネットワークをデプロイする場合には、ネットワーク設定で使用するルートとルーターゲートウェイを定義します。ネットワークルートを作成して、ネットワークルートとスーパーネットルートを作成して、サブネット間でトラフィックをルーティングする際に使用するインターフェースを定義します。たとえば、Compute と Controller ロールの間でトラフィックがルーティングされるデプロイメントの場合には、分離ネットワークのセット用にスーパーネットを定義することができます。たとえば、**172.17.0.0/16** は、**172.17** で始まる全ネットワークが含まれるスーパーネットで、コントローラー上で使用される **Internal API** ネットワークには **172.17.1.0/24** を使用できます。また、コンピュータード上で使用される **Internal API** ネットワークには **172.17.2.0/24** を使用できます。ロールで使用されるネットワーク固有のルーターゲートウェイを介する **172.17.0.0/16** スーパーネットへのルートを両方のノードで定義します。

network-environment.yaml で利用可能なパラメーター

```

InternalApiSupernet:
  default: '172.17.0.0/16'
  description: Supernet that contains Internal API subnets for all
roles.
  type: string
InternalApiGateway:
  default: '172.17.1.1'
  description: Router gateway on Internal API network
  type: string
InternalApi2Gateway:
  default: '172.17.2.1'
  description: Router gateway on Internal API 2 network
  Type: string

```

これらのパラメーターは、ロールの NIC 設定テンプレートで使用できます。

コントローラーは、**controller.yaml** の **InternalApi** ネットワークのパラメーターを使用します。

```

- type: interface
  name: nic3
  use_dhcp: false
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet
  - routes:
      ip_netmask:
        get_param: InternalApiSupernet
      next_hop:
        Get_param: InternalApiGateway

```

compute ロールは、**compute.yaml** の **InternalApi2** ネットワークのパラメーターを使用します。

```

- type: interface
  name: nic3
  use_dhcp: false
  addresses:
  - ip_netmask:
      get_param: InternalApi2IpSubnet
  - routes:
      ip_netmask:
        get_param: InternalApiSupernet
      next_hop:
        Get_param: InternalApi2Gateway

```

注記

特定のネットワークルートが分離ネットワークに適用されない場合には、ローカル以外のネットワークへのトラフィックはすべてデフォルトのゲートウェイを使用します。これにより、異なる種別のトラフィックが混在し、送信トラフィックがすべて同じインターフェース上に配置されるので、通常はセキュリティとパフォーマンスの両観点から望ましくありません。また、ルーティングが非対称なので (受信用とは別のインターフェースでトラフィックが送信される)、サービスが到達不可能となる可能性があります。クライアントとサーバーの両方でスーパーネットへのルートを使用すると、トラフィックは両サイドで正しいインターフェースを使用するように送られます。

第11章 ノード配置の制御

director のデフォルトの動作は、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が、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。この手法は、以下の作業に役立ちます。

- **controller-0**、**controller-1** などの特定のノード ID の割り当て
- カスタムのホスト名の割り当て
- 特定の IP アドレスの割り当て
- 特定の仮想 IP アドレスの割り当て



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が、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、「[ネットワーク環境ファイルの作成](#)」を参照してください。

11.1. 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 Nova スケジューラーが照合するノード別ケイパビリティとしてこの 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** のケイパビリティをノードに割り当てます。0 から開始するユニークな連続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。特定のロール (コントローラー、コンピュート、各ストレージロール) に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。そうでない場合は、このケイパビリティは Nova スケジューラーにより正しく照合されません。

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 command** 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 **ControllerSchedulerHints**

- コンピュートノードの **NovaComputeSchedulerHints**
- Block Storage ノードの **BlockStorageSchedulerHints**
- Object Storage ノードの **ObjectStorageSchedulerHints**
- Ceph Storage ノードの **CephStorageSchedulerHints**
- **[ROLE]SchedulerHints** はカスタムのロールに、**[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 ...
```

11.2. 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。システムの場合 (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で **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%'
  NovaComputeSchedulerHints:
    '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なります。

11.3. 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、主に 2 つの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 **NovaComputeIPs**
- Ceph Storage ノードの **CephStorageIPs**
- Block Storage ノードの **BlockStorageIPs**
- Object Storage ノードの **SwiftStorageIPs**
- カスタムロールの **[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、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。

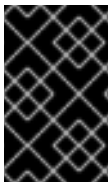
選択した IP アドレスは、ネットワーク環境ファイルで定義されている各ネットワークの割り当てプールの範囲に入らないようにしてください (「[ネットワーク環境ファイルの作成](#)」参照)。たとえば、**internal_api** の割り当ては **InternalApiAllocationPools** の範囲外となるようにします。これにより、自動的に選択される IP アドレスと競合が発生しないようになります。また同様に、IP アドレスの割り当てが標準の予測可能な仮想 IP 配置 (「[予測可能な仮想 IP の割り当て](#)」を参照) または外部のロードバランシング (「[外部の負荷分散機能の設定](#)」を参照) のいずれでも、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 (「[9章 ネットワークの分離](#)」を参照) 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]
```

11.4. 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。この定義を行うには、「[ネットワーク環境ファイルの作成](#)」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、デフォルトの内部ロードバランシング設定を使用するオーバークラウドのみが対象です。外部ロードバランシングを使用して VIP を割り当てる場合には、[『External Load Balancing for the Overcloud』](#) ガイドに記載の専用の手順を使用してください。

第12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。次の章では、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。ネットワークの分離に関する説明は、「[9章 ネットワークの分離](#)」を参照してください。

12.1. 署名ホストの初期化

署名ホストとは、新規証明書を生成し、認証局を使用して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ります。

`/etc/pki/CA/index.txt` ファイルは、すべての署名済み証明書の記録を保管します。このファイルが存在しているかどうかを確認してください。存在していない場合には、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するかどうかを確認し、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。

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

12.2. 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があります。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。

たとえば、キー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。それらの情報を指定してください。

これで、`ca.crt.pem` という名前の認証局ファイルが作成されます。

12.3. 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ことを目的としている外部のクライアントの場合は、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。クライアントへのコピーが完了したら、そのクライアントで以下のコマンドを実行して、認証局のトラストバンドルに追加します。

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ 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。

12.4. SSL/TLS キーの作成

以下のコマンドを実行して、SSL/TLS キー (**server.key.pem**) を生成します。このキーは、さまざまな段階で、アンダークラウドとオーバークラウドの証明書を生成するのに使用します。

```
$ openssl genrsa -out server.key.pem 2048
```

12.5. 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。デフォルトの OpenSSL 設定ファイルをコピーしてカスタマイズします。

```
$ 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。詳しい情報は、「[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 エントリおよび DNS エントリとして、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 エントリとしてそのサーバーのホスト名を追加します。**openssl.cnf** の詳しい情報は **man openssl.cnf** を実行してください。

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 キーの作成](#)」で作成した SSL/TLS キーを **-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。

12.6. SSL/TLS 証明書の作成

以下のコマンドで、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。この値は **-config** オプションとして追加します。
- 認証局を介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。この値は **-in** オプションとして追加します。
- 証明書への署名を行う、「[認証局の作成](#)」で作成した認証局。この値は **-cert** オプションとして追加します。
- 「[認証局の作成](#)」で作成した認証局の秘密鍵。この値は **-keyfile** オプションとして追加します。

このコマンドを実行すると、**server.crt.pem** という名前の証明書が作成されます。この証明書は、「[SSL/TLS キーの作成](#)」で作成した SSL/TLS キーとともに使用して SSL/TLS を有効にします。

12.7. 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/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-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgx CzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8m1BBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
    -----END CERTIFICATE-----
```



重要

この証明書の内容で、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。以下の例を示します。

```
parameter_defaults:
  ...
  SSLKey: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0lV09hkJZnGP6qb6wtYUoy1bVP7
    ...
    ct1Kn3rAAdyumi4JDjESAXHIKFjJN0LrBmpQyES4XpZUC7yhqPaU
    -----END RSA PRIVATE KEY-----
```



重要

この秘密鍵のコンテンツにおいて、新しく追加する行はすべて同じ ID レベルに指定する必要があります。

OS::TripleO::NodeTLSData

OS::TripleO::NodeTLSData のリソースのパスを絶対パスに変更します。

```
resource_registry:
  OS::TripleO::NodeTLSData: /usr/share/openstack-tripleo-heat-templates/puppet/extraconfig/tls/tls-cert-inject.yaml
```

12.8. ルート証明書の注入

証明書の署名者がオープン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、オープン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.yaml** 環境ファイルをコピーします。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/inject-trust-anchor.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。

SSLRootCertificate

SSLRootCertificate パラメーターにルート認証局ファイル (**ca.crt.pem**) の内容をコピーします。以下に例を示します。

```
parameter_defaults:
  SSLRootCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgx CzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8m1BBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
    -----END CERTIFICATE-----
```



重要

この認証局のコンテンツで、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。

OS::TripleO::NodeTLSCAData

OS::TripleO::NodeTLSCAData: のリソースのパスを絶対パスに変更します。

```
resource_registry:
  OS::TripleO::NodeTLSCAData: /usr/share/openstack-tripleo-heat-templates/puppet/extraconfig/tls/ca-inject.yaml
```

12.9. DNS エンドポイントの設定

DNS ホスト名を使用して SSL/TLS でオープンクラウドにアクセスする場合は、新しい環境ファイル (**~/templates/cloudname.yaml**) を作成して、オープンクラウドのエンドポイントのホスト名を定義します。以下のパラメーターを使用してください。

CloudName

オープンクラウドエンドポイントの DNS ホスト名

DnsServers

使用する DNS サーバー一覧。設定済みの DNS サーバーには、パブリック API の IP アドレスに一致する設定済みの **CloudName** へのエントリーが含まれていなければなりません。

このファイルの内容の例は以下のとおりです。

```
parameter_defaults:
  CloudName: overcloud.example.com
  DnsServers: ["10.0.0.254"]
```

12.10. オーバークラウド作成時の環境ファイルの追加

デプロイメントのコマンド (**openstack overcloud deploy**) に **-e** オプションを使用して環境ファイルを追加します。環境ファイルは、このセクションから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.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/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/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.yaml -e /usr/share/openstack-tripleo-heat-
templates/environments/tls-endpoints-public-dns.yaml
```

12.11. 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/tls-endpoints-public-dns.yaml
```

第13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

全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。

13.1. CA へのアンダークラウド追加

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、**python-novajoin** パッケージをインストールします。

```
$ sudo yum install python-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。

13.2. アンダークラウドを IDM に追加します。

この手順では、アンダークラウドを IdM に登録して novajoin を設定します。

1. novajoin サービスは、デフォルトで無効にされます。有効化するには、**undercloud.conf** にエントリーを追加します。

```
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 サーバーによって提供されるオーバークラウドのドメイン名が IdM ドメインと一致するようにします (小文字の kerberos レルム)。

```
overcloud_domain_name = <domain>
```

4. アンダークラウドに適切なホスト名を設定します。

```
undercloud_hostname = <undercloud FQDN>
```

5. アンダークラウドのネームサーバーとして IdM を設定します。

```
undercloud_nameservers = <IdM IP>
```

6. より大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
```

13.3. 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 を使用するための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。

13.4. 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/enable-internal-tls.yaml**
- **/usr/share/openstack-tripleo-heat-templates/environments/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/enable-internal-tls.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/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、**./tripleo-heat-templates/environments/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/enable-internal-tls.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/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、**./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 ./tripleo-heat-templates/environments/enable-internal-tls.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/tls-
everywhere-endpoints-dns.yaml \
  -e /home/stack/templates/custom-domain.yaml \
  -e ./tripleo-heat-templates/environments/services/haproxy-public-
tls-certmonger.yaml
```

第14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。このパラメーターは、環境ファイルの **parameter_defaults** セクションで設定してください。

```
parameter_defaults:
    KeystoneDebug: True
```

デバッグパラメーターの全一覧は、『**Overcloud Parameters**』ガイドの「[Debug Parameters](#)」を参照してください。

第15章 ポリシー

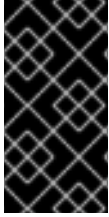
オーバークラウド内の特定のサービスに対してアクセスポリシーを設定することができます。サービスに対してポリシーを設定するには、そのサービスのポリシーが含まれるハッシュ値でそれぞれのポリシーのパラメーターを設定します。たとえば、OpenStack Identity (keystone) には **KeystonePolicies** パラメーターを使用します。このパラメーターを環境ファイルの **parameter_defaults** セクションで設定します。

```
parameter_defaults:
  KeystonePolicies: { keystone-context_is_admin: { key: context_is_admin,
value: 'role:admin' } }
```

ポリシーパラメーターの全一覧は、『**Overcloud Parameters**』ガイドの「[Policy Parameters](#)」を参照してください。

第16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

オーバークラウドは、デフォルトのストレージオプションにローカルおよび LVM のストレージを使用します。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。本章のストレージオプションの 1 つを使用することを推奨します。

16.1. 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。

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。これらは、director で作成したオーバークラウドでサポートされている一部の機能のカスタム設定に役立つ環境テンプレートです。これには、ストレージ設定に有用な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` に配置されています。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てください。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/storage-environment.yaml ~/templates/.
```

この環境ファイルには、OpenStack のブロックストレージおよびイメージストレージのコンポーネントの異なるストレージオプションを設定するのに役立つ複数のパラメーターが記載されています。この例では、オーバークラウドが NFS 共有を使用するように設定します。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用にマウントする NFS 共有 (例: 192.168.122.1:/export/cinder)

GlanceNfsEnabled

イメージストレージ用の共有を管理するための Pacemaker を有効にするパラメーター。無効に設定されている場合には、オーバークラウドはコントローラーノードのファイルシステムにイメージを保管します。**true** に設定してください。

GlanceNfsShare

イメージストレージをマウントするための NFS 共有 (例: 192.168.122.1:/export/glance)

GlanceNfsOptions

イメージストレージ用の NFS マウントオプション

環境ファイルのオプションは、以下の例のようになります。

```
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'
```

重要

Glance 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、**GlanceFilePcmkOptions** パラメーターに **context=system_u:object_r:glance_var_lib_t:s0** と記載します。この SELinux コンテキストがない場合には、Glance はマウントポイントへの書き込みに失敗します。

これらのパラメーターは、Heat テンプレートコレクションの一部として統合されます。このように設定することにより、Cinder と Glance が使用するための 2 つの NFS マウントポイントが作成されます。

このファイルを保存して、オーバークラウドの作成に含まれるようにします。

16.2. 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

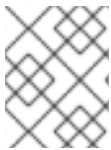
director には、オーバークラウドの作成中に Ceph Storage Cluster を作成する機能があります。director は、データの格納に Ceph OSD を使用する Ceph Storage ノードセットを作成します。さらに、director は、オーバークラウドのコントローラーノードに Ceph Monitor サービスをインストールします。このため、組織が高可用性のコントローラーノード 3 台で構成されるオーバークラウドを作成する場合には、Ceph Monitor も高可用性サービスになります。詳しい情報は、[『Deploying an Overcloud with Containerized Red Hat Ceph』](#) ガイド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 Cluster 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せん。これは、オーバークラウドの設定以外の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、『[Integrating an Overcloud with an Existing Red Hat Ceph Cluster](#)』ガイドを参照してください。

16.3. 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
```

16.4. サードパーティのストレージの設定

director には、以下のような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環境ファイルが2つ含まれています。

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17章 セキュリティーの強化

以下の項では、オープンクラウドのセキュリティを強化するための推奨事項について説明します。

17.1. オープン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、各オープン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。

オープンクラウドの 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。オープンクラウドの環境ファイルで、設定フックを使用して (「[Puppet: ロール用の 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一連の状態。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

iface

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。

17.2. SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP) のコミュニティ文字列の変更

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 のコミュニティ文字列を変更して、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。

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::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。

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。

17.3. HAPROXY の SSL/TLS の暗号およびルールの変更

オープンクラウドで SSL/TLS を有効化した場合には (「[12章 オープン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。

オープン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-sslsv3 no-tls-tickets**

環境ファイルを作成して、以下の内容を記載します。

```
parameter_defaults:
  ExtraConfig:
    tripleo::haproxy::ssl_cipher_suite: ECDHE-ECDSA-CHACHA20-
POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-
RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-
SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-
AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-
AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-
SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-
SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-
GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-
SHA:!DSS
    tripleo::haproxy::ssl_options: no-sslsv3 no-tls-tickets
```



注記

暗号のコレクションは、改行なしで 1 行に記述します。

オープンクラウドの作成時にこの環境ファイルを追加します。

17.4. 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の名前を設定します。指定可能な値は、システム構成により異なります (例: **noop**、**openvswitch**、**iptables_hybrid**)。デフォルト値である空の文字列を指定すると、サポートされている構成となります。

17.5. 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へのパスワード無しの SSH アクセスを提供します。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おり、実稼働環境での使用には推奨されません。

この方法では、初回ブートの設定フック (「[初回起動: 初回起動時の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は、**openstack overcloud deploy** コマンドでこの環境ファイルを指定します。

第18章 コントローラーノードのフェンシング

フェンシングとは、クラスターとそのリソースを保護するために、障害が発生したノードを分離するプロセスのことです。フェンシングがないと、障害のあるノードが原因でクラスター内のデータが破損する可能性があります。

director は、Pacemaker を使用して、高可用性のコントローラーノードクラスターを提供します。Pacemaker は、障害の発生したノードをフェンシングするのに STONITH というプロセスを使用します。STONITH はデフォルトでは無効化されているため、Pacemaker がクラスター内の各ノードの電源管理を制御できるように手動で設定する必要があります。

18.1. 前提条件の確認

オーバークラウドでフェンシングを設定するには、オーバークラウドがすでにデプロイ済みで稼働状態である必要があります。以下の手順では、デプロイメント内の Pacemaker と STONITH の状態を確認します。

1. director 上の **stack** ユーザーから、**heat-admin** ユーザーとして各ノードにログインします。オーバークラウドを作成すると自動的に **stack** ユーザーの SSH キーが各ノードの **heat-admin** にコピーされます。
2. 実行中のクラスターがあることを確認します。

```
$ sudo pcs status
Cluster name: openstackHA
Last updated: Wed Jun 24 12:40:27 2015
Last change: Wed Jun 24 11:36:18 2015
Stack: corosync
Current DC: lb-c1a2 (2) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
141 Resources configured
```

3. STONITH が無効化されていることを確認します。

```
$ sudo pcs property show
Cluster Properties:
cluster-infrastructure: corosync
cluster-name: openstackHA
dc-version: 1.1.12-a14efad
have-watchdog: false
stonith-enabled: false
```

18.2. フェンシングの有効化

オーバークラウドがデプロイされて機能していることを確認した後に、フェンシングを設定することができます。

1. **fencing.yaml** ファイルを生成します。

```
$ openstack overcloud generate fencing --ipmi-lanplus --ipmi-level
administrator --output fencing.yaml instackenv.json
```

- サンプルの **fencing.yaml** ファイル:

```
parameter_defaults:
  EnableFencing: true
  FencingConfig:
    devices:
      - agent: fence_ipmilan
        host_mac: 11:11:11:11:11:11
        params:
          action: reboot
          ipaddr: 10.0.0.101
          lanplus: true
          login: admin
          passwd: InsertComplexPasswordHere
          pcmk_host_list: host04
          privlvl: administrator
```

2. 以前オーバークラウドのデプロイに使用した **deploy** コマンドに、生成された **fencing.yaml** ファイルを渡します。これでデプロイメントの手順が再実行され、ホスト上でフェンシングが設定されます。

```
openstack overcloud deploy --templates -e /usr/share/openstack-
tripleo-heat-templates/environments/network-isolation.yaml -e
~/templates/network-environment.yaml -e ~/templates/storage-
environment.yaml --control-scale 3 --compute-scale 3 --ceph-storage-
scale 3 --control-flavor control --compute-flavor compute --ceph-
storage-flavor ceph-storage --ntp-server pool.ntp.org --neutron-
network-type vxlan --neutron-tunnel-types vxlan -e fencing.yaml
```

デプロイメントのコマンドは、エラーまたは例外なしで完了するはずです。

3. オーバークラウドにログインし、各コントローラーにフェンシングが設定されたことを確認します。
 - a. フェンシングリソースが Pacemaker で管理されていることを確認します。

```
$ source stackrc
$ nova list | grep controller
$ ssh heat-admin@<controller-x_ip>
$ sudo pcs status |grep fence
stonith-overcloud-controller-x (stonith:fence_ipmilan): Started
overcloud-controller-y
```

Pacemaker が、**fencing.yaml** で指定されている各コントローラーの STONITH リソースを使用するように設定されていることを確認します。**fence-resource** プロセスは、そのプロセスが制御する同じホスト上には設定すべきではありません。

- b. **pcs** でフェンシングリソースの属性を確認します。

```
$ sudo pcs stonith show <stonith-resource-controller-x>
```

STONITH が使用する値は、**fencing.yaml** に定義されている値と一致している必要があります。

18.3. フェンシングのテスト

この手順は、フェンシングが想定どおりに機能しているかどうかをテストします。

1. デプロイメント内の各コントローラーでフェンシングのアクションをトリガーします。

- a. コントローラーにログインします。

```
$ source stackrc
$ nova list |grep controller
$ ssh heat-admin@<controller-x_ip>
```

- b. root として、**iptables** を使用して全ポートを閉鎖することによって、フェンシングをトリガーします。

```
$ sudo -i
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
&&
iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT &&
iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 5016 -j ACCEPT &&
iptables -A INPUT -p udp -m state --state NEW -m udp --dport 5016 -j ACCEPT &&
iptables -A INPUT ! -i lo -j REJECT --reject-with icmp-host-prohibited &&
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT &&
iptables -A OUTPUT -p tcp --sport 5016 -j ACCEPT &&
iptables -A OUTPUT -p udp --sport 5016 -j ACCEPT &&
iptables -A OUTPUT ! -o lo -j REJECT --reject-with icmp-host-prohibited
```

その結果、接続が切断され、サーバーが再起動されるはずですが。

- c. 別のコントローラーから、Pacemaker のログファイル内のフェンシングイベントを特定します。

```
$ ssh heat-admin@<controller-x_ip>
$ less /var/log/cluster/corosync.log
(less): /fenc*
```

STONITH がそのコントローラーに対して、フェンシングのアクションを発行し、Pacemaker がログ内でイベントを発生させたことが確認できるはずですが。

- d. 再起動したコントローラーがクラスターに戻ったことを確認します。

- i. 2 番目のコントローラーから、数分待った後に **pcs status** を実行して、フェンシングされたコントローラーがクラスターに戻っているかどうかを確認します。この時間は設定によって異なります。

第19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、[『Monitoring Tools Configuration Guide』](#)に記載の詳しい手順を参照してください。

第20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。

20.1. 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]
...
```

20.2. 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 OVSDDB 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21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。

21.1. 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22章 その他の設定

22.1. 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、全手順が記載されている専用の『[External Load Balancing for the Overcloud](#)』ガイドを参照してください。

22.2. 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が、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、環境ファイルのセットが含まれており、IPv6 ベースのオーバークラウドの作成に役立ちます。

オーバークラウドでの IPv6 の設定に関する詳しい情報は、全手順が記載されている専用の『[IPv6 Networking for the Overcloud](#)』ガイドを参照してください。

付録A ネットワーク環境のオプション

表A.1 ネットワーク環境のオプション

パラメーター	説明	例
InternalApiNetCidr	Internal API ネットワークのネットワークおよびサブネット	172.17.0.0/24
StorageNetCidr	Storage ネットワークのネットワークおよびサブネット	
StorageMgmtNetCidr	Storage Management ネットワークのネットワークのおよびサブネット	
TenantNetCidr	Tenant ネットワークのネットワークおよびサブネット	
ExternalNetCidr	External ネットワークのネットワークおよびサブネット	
InternalApiAllocationPools	Internal API ネットワークの割り当てプール (タプル形式)	[{start: 172.17.0.10, end: 172.17.0.200}]
StorageAllocationPools	Storage ネットワークの割り当てプール (タプル形式)	
StorageMgmtAllocationPools	Storage Management ネットワークの割り当てプール (タプル形式)	
TenantAllocationPools	Tenant ネットワークの割り当てプール (タプル形式)	
ExternalAllocationPools	External ネットワークの割り当てプール (タプル形式)	
InternalApiNetworkVlanID	Internal API ネットワークの VLAN ID	200
StorageNetworkVlanID	Storage ネットワークの VLAN ID	
StorageMgmtNetworkVlanID	Storage Management ネットワークの VLAN ID	
TenantNetworkVlanID	Tenant ネットワークの VLAN ID	
ExternalNetworkVlanID	External ネットワークの VLAN ID	

パラメーター	説明	例
ExternalInterfaceDefaultRoute	External ネットワークのゲートウェイ IP アドレス	10.1.2.1
ControlPlaneDefaultRoute	Provisioning ネットワーク用のゲートウェイルーター (またはアンダークラウドの IP アドレス)	ControlPlaneDefaultRoute: 192.0.2.254
ControlPlaneSubnetCidr	Provisioning ネットワーク用の CIDR サブネットマスクの長さ	ControlPlaneSubnetCidr: 24
EC2Metadatalp	EC2 メタデータサーバーの IP アドレス。通常はアンダークラウドの IP アドレスです。	EC2Metadatalp: 192.0.2.1
DnsServers	オーバークラウドノード用の DNS サーバーを定義します。最大で 2 つまで指定することができます。	DnsServers: ["8.8.8.8", "8.8.4.4"]
BondInterfaceOvsOptions	ボンディングインターフェースのオプション	BondInterfaceOvsOptions: "bond_mode=balance-slb"
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。	NeutronFlatNetworks: "datacentre"
NeutronExternalNetworkBridge	各ハイパーバイザーで作成する Open vSwitch ブリッジ。通常、この値は変更する必要はありません。	NeutronExternalNetworkBridge: ""
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 (br-ex) の外部ブリッジを物理名 (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。これは、デフォルトの Floating ネットワークに使用されます。	NeutronBridgeMappings: "datacentre:br-ex"
NeutronPublicInterface	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。	NeutronPublicInterface: "eth0"

パラメーター	説明	例
NeutronNetworkType	Neutron テナントネットワークの種別	NeutronNetworkType: "vxlan"
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。	NeutronTunnelTypes: gre,vxlan
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	NeutronTunnelIdRanges "1:1000"
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	NeutronVniRanges: "1:1000"
NeutronEnableTunnelling	VLAN で区切られたネットワークまたは Neutron でのフラットネットワークを使用するためにトンネリングを有効化/無効化するかどうかを定義します。デフォルトでは有効化されます。	
NeutronNetworkVLANRanges	サポートされる Neutron 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。	NeutronNetworkVLANRanges: "datacentre:1:1000"
NeutronMechanismDrivers	neutron テナントネットワークのメカニズムドライバー。デフォルトでは、「openvswitch」に設定されており、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。	NeutronMechanismDrivers: openvswitch,l2population

付録B ネットワークインターフェースのテンプレート例

本付録では、ネットワークインターフェース設定を示す Heat テンプレート例をいくつか紹介します。

B.1. 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。

B.2. ルートおよびデフォルトルートの設定

ホストにデフォルトのルートセットを指定するには 2 つの方法があります。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=no** 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応されます。

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
```

B.3. FLOATING IP のためのネイティブ VLAN の使用

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。

B.4. 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 の使用

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。

B.5. 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、Tenant ネットワークのスループットは 300% 以上になりました。



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
```

第23章 ネットワークインターフェースのパラメーター

以下の表には、各ネットワークインターフェース種別の Heat テンプレートのパラメーターの定義をまとめています。

23.1. インターフェースのオプション

オプション	デフォルト	説明
name		インターフェース名
use_dhcp	False	DHCP を使用して IP アドレスを取得します。
use_dhcpv6	False	DHCP を使用して v6 の IP アドレスを取得します。
addresses		インターフェースに割り当てられる IP アドレスのシーケンス
routes		インターフェースに割り当てられるルートのシーケンス
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	このインターフェースをデフォルトルートとして使用します。
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

23.2. VLAN のオプション

オプション	デフォルト	説明
vlan_id		VLAN ID

device		VLAN の接続先となる VLAN の親デバイス。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 を割り当てる IP アドレスのシーケンス
routes		VLAN を割り当てるルートのシーケンス
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	このインターフェースをデフォルトルートとして使用します。
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

23.3. OVS ボンディングのオプション

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 アドレスのシーケンス
routes		ボンディングに割り当てられる ルートのシーケンス
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シーケンス
ovs_options		ボンディング作成時に OVS に渡すオプションセット
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セット
defroute	True	このインターフェースをデフォルトルートとして使用します。
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

23.4. OVS ブリッジのオプション

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 アドレスのシーケンス
routes		ブリッジに割り当てられるルートのシーケンス
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シーケンス
ovs_options		ブリッジ作成時に OVS に渡すオプションセット
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セット
defroute	True	このインターフェースをデフォルトルートとして使用します。
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

23.5. LINUX ボンディングのオプション

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 アドレスのシーケンス

routes		ボンディングに割り当てられるルートのシーケンス
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シーケンス
bonding_options		ボンディングを作成する際のオプションのセット。 nmcli ツールの使用方法についての詳細は、『Red Hat Enterprise Linux 7 ネットワークガイド』の「 4.5.1. ボンディングモジュールのディレクトリ 」を参照してください。
defroute	True	このインターフェースをデフォルトルートとして使用します。
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

23.6. 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 アドレスのシーケンス

routes		ブリッジに割り当てられるルートのシーケンス
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シーケンス
defroute	True	このインターフェースをデフォルトルートとして使用します。
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

付録C OPEN VSWITCH ボンディングのオプション

オーバークラウドは、ボンディングインターフェースのオプションを複数提供する Open vSwitch (OVS) を介してネットワークを提供します。「[ネットワーク環境ファイルの作成](#)」の項では、ネットワークの環境ファイルで以下のパラメーターを使用して、ボンディングインターフェースを設定します。

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

C.1. ボンディングモードの選択

デフォルトでは、LACP は OVS ベースのボンディングでは使用できません。この設定は、Open vSwitch の一部のバージョンで既知の問題があるためサポートされていません。その代わりに、この機能を置き換わる **bond_mode=balance-slb** を使用することを検討してください。また、ネットワークインターフェースのテンプレートでは、引き続き LACP を Linux ボンディングで 사용할 수 있습니다。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)。

それでも LACP を OVS ベースのボンディングで使いたい場合には、デプロイメントの前に、各ネットワークインターフェースの設定 (NIC) ファイルから以下の行を手動で削除することができます。

```
constraints:
- allowed_pattern: "^(?!balance.tcp).*$"
  description: |
    The balance-tcp bond mode is known to cause packet loss and
    should not be used in BondInterfaceOvsOptions.
```

各 NIC ファイルから制約を取り除いた後には、ボンディングインターフェースのパラメーターでボンディングモードを設定することができます。

```
BondInterfaceOvsOptions:
    "bond_mode=balance-tcp"
```


この制約の背後にある技術情報については、[BZ#1267291](#) を参照してください。

nmcli ツールの使用方法についての詳細は、『Red Hat Enterprise Linux 7 ネットワークガイド』の「4.5.1. ボンディングモジュールのディレクティブ」を参照してください。

C.2. 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C.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。 balance-slb とのボンディングにより、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ロードバランシングが可能となります。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。これは、Linux ボンディングドライバで使用されているモード 2 のボンディングと同様で、スイッチはボンディングで設定されているが、LACP (動的なボンディングではなく静的なボンディング) を使用するように設定されていない場合に使用されます。
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、ロードバランシングは提供しません。
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。



重要

Linux のボンディングをプロバイダーネットワークと併用してパケットのドロップやパフォーマンス上の問題が発生した場合には、スタンバイインターフェースで Large Receive Offload (LRO) を無効にすることを検討してください。ポートフラッピングが発生したり、接続を失ったりする可能性があるため、Linux ボンディングを OVS ボンディングに追加するのは避けてください。これは、スタンバイインターフェースを介したパケットループの結果です。