



Red Hat OpenStack Platform 10

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 10 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 © 2017 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. コアとなるオーバークラウドの HEAT テンプレート	8
2.4. オーバークラウド作成時の環境ファイルの追加	9
2.5. カスタムのコア HEAT テンプレートの使用	10
第3章 パラメーター	12
3.1. 例 1: タイムゾーンの設定	12
3.2. 例 2: LAYER 3 HIGH AVAILABILITY (L3HA) の無効化	13
3.3. 例 3: TELEMETRY DISPATCHER の設定	13
3.4. 例 4: RABBITMQ ファイル記述子の上限の設定	13
3.5. 変更するパラメーターの特定	13
第4章 設定フック	16
4.1. 初回起動: 初回起動時の設定のカスタマイズ	16
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	17
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	19
4.4. 設定後: 全オーバークラウドロールのカスタマイズ	21
4.5. PUPPET: ロール用の HIERADATA のカスタマイズ	23
4.6. PUPPET: 個別のノードの HIERADATA のカスタマイズ	24
4.7. PUPPET: カスタムのマニフェストの適用	24
第5章 オーバークラウドの登録	26
5.1. 環境ファイルを使用したオーバークラウドの登録	26
5.2. 例 1: カスタマーポータルへの登録	27
5.3. 例 2: RED HAT SATELLITE 6 サーバーへの登録	28
5.4. 例 3: RED HAT SATELLITE 5 サーバーへの登録	29
第6章 コンポーザブルサービスとカスタムロール	30
6.1. カスタムロールアーキテクチャーの考察	31
6.2. コンポーザブルサービスアーキテクチャーの考察	32
6.3. 無効化されたサービスの有効化	33
6.4. ロールへのサービスの追加と削除	34
6.5. 新規ロールの作成	35
6.6. サービスなしの汎用ノードの作成	37
6.7. ハイパーコンバージドの COMPUTE サービスと CEPH サービスの作成	37
6.8. 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	40
6.9. サービスアーキテクチャー: 分割コントローラー	41
6.10. サービスアーキテクチャー: スタンドアロンロール	44
6.11. コンポーザブルサービスのリファレンス	54
第7章 ネットワークの分離	60
7.1. カスタムのインターフェーステンプレートの作成	60
7.2. ネットワーク環境ファイルの作成	65
7.3. OPENSTACK サービスの分離ネットワークへの割り当て	67
7.4. デプロイするネットワークの選択	68
第8章 ノード配置の制御	74
8.1. 特定のノード ID の割り当て	74
8.2. カスタムのホスト名の割り当て	75

8.3. 予測可能な IP の割り当て	75
8.4. 予測可能な仮想 IP の割り当て	77
第9章 オーバークラウドの SSL/TLS の有効化	79
9.1. 署名ホストの初期化	79
9.2. 認証局の作成	79
9.3. クライアントへの認証局の追加	79
9.4. SSL/TLS キーの作成	80
9.5. SSL/TLS 証明書署名要求の作成	80
9.6. SSL/TLS 証明書の作成	81
9.7. SSL/TLS の有効化	81
9.8. ルート証明書の注入	82
9.9. DNS エンドポイントの設定	83
9.10. オーバークラウド作成時の環境ファイルの追加	83
9.11. SSL/TLS 証明書の更新	84
第10章 ストレージの設定	85
10.1. NFS ストレージの設定	85
10.2. CEPH STORAGE の設定	86
10.3. サードパーティーのストレージの設定	87
第11章 コンテナ化されたコンピュートノードの設定	88
11.1. スタックの深度を高くする方法	88
11.2. コンテナ化されたコンピュートの環境ファイル (DOCKER.YAML) の検証	89
11.3. ATOMIC HOST のイメージのアップロード	90
11.4. ローカルのレジストリーの使用	90
11.5. オーバークラウドのデプロイメントへの環境ファイルの追加	92
第12章 モニタリングツールの設定	94
12.1. アーキテクチャー	94
12.1.1. 集中ロギング	94
12.1.2. 可用性監視	97
12.2. クライアント側のツールのインストール	100
12.2.1. 集中ロギングのパラメーターの設定	100
12.2.2. 可用性監視クライアントのパラメーターの設定	101
12.2.3. オーバークラウドノードへの運用ツールのインストール	102
12.3. サーバー側のコンポーネントのインストール	102
12.4. OPENSTACK PLATFORM の監視	102
12.5. SENSU クライアントインストールの検証	102
12.6. ノードの状態の確認	103
12.7. OPENSTACK サービスの状態の確認	103
第13章 セキュリティーの強化	105
13.1. オーバークラウドのファイアウォールの管理	105
13.2. SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP) のコミュニティ文字列の変更	106
13.3. HAPROXY の SSL/TLS の暗号およびルールの変更	106
第14章 その他の設定	108
14.1. 外部の負荷分散機能の設定	108
14.2. IPV6 ネットワークの設定	108
付録A ネットワーク環境のオプション	109
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	112
B.1. インターフェースの設定	112

B.2. ルートおよびデフォルトルートの設定	113
B.3. FLOATING IP のためのネイティブ VLAN の使用	113
B.4. トランッ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	114
B.5. ジャンボフレームの設定	114
第15章 ネットワークインターフェースのパラメーター	116
15.1. インターフェースのオプション	116
15.2. VLAN のオプション	116
15.3. OVS ボンディングのオプション	117
15.4. OVS ブリッジのオプション	118
15.5. LINUX ボンディングのオプション	119
15.6. LINUX BRIDGE のオプション	120
付録C OPEN VSWITCH ボンディングのオプション	122
C.1. ボンディングモードの選択	122
C.2. ボンディングオプション	123

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。

パラメーター

これらは、最上位のテンプレートのパラメーターに適用する共通の設定です。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. コアとなるオーバークラウドの 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、留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。

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。

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。

network

分離ネットワークおよびポートを作成しやすくする 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. 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::TripleO::NodeExtraConfigPost**) と共通のパラメーター (**TimeZone**) が含まれています。**openstack overcloud deploy** コマンドは、以下のプロセスを順に実行します。

1. **--template** オプションで指定したコアの Heat テンプレートからデフォルト設定を読み込みます。
2. **environment-file-1.yaml** の設定を適用します。この設定により、デフォルト設定と共通している設定は上書きされます。
3. **environment-file-2.yaml** の設定を適用します。この設定により、デフォルト設定および **environment-file-1.yaml** と共通している設定は上書きされます。

これにより、オーバークラウドのデフォルト設定が以下のように変更されます。

- **OS::TripleO::NodeExtraConfigPost** リソースは、**environment-file-2.yaml** で指定されている通りに **/home/stack/templates/template-2.yaml** に設定されます。
- **environment-file-2.yaml** で指定されている通りに、**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。

2.5. 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 Heat テンプレートのコアセットを使用します。標準の Heat テンプレートをローカルディレクトリーにコピーして、オーバークラウド作成にこれらのテンプレートを使用することが可能です。

/usr/share/openstack-tripleo-heat-templates にある Heat テンプレートコレクションを **stack** ユーザーのテンプレートディレクトリーにコピーします。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates ~/templates/my-overcloud
```

これでオーバークラウドの Heat テンプレートのクローンが作成されます。**openstack overcloud deploy** のコマンドの実行時には、**--templates** オプションでローカルのテンプレートディレクトリーを指定してください。



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、今後のリリースで Heat テンプレートコレクションの更新を提供します。変更されたテンプレートコレクションを使用すると、カスタムのコピーと **/usr/share/openstack-tripleo-heat-templates** にあるオリジナルのコピーとの間に相違が生じる可能性があります。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 パラメーター

director テンプレートコレクション内の各 Heat テンプレートには、**parameters** セクションがあります。このセクションは、特定のオーバークラウドサービス固有の全パラメーターを定義します。これには、以下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。

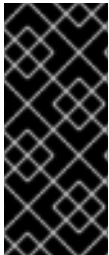
これにより、初回作成/起動時に、全ノードに設定が追加されます。オーバークラウドのスタックの更新など、これらのテンプレートを後ほど追加しても、このスクリプトは実行されません。



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。

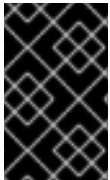
```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre:
    /home/stack/templates/nameserver.yaml

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

```

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

```



```
parameter_defaults:
  nameserver_ip: 192.168.1.1
```

この設定を適用するには、オーバークラウドの作成時または更新時に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
```

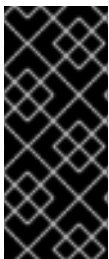
このコマンドにより、オーバークラウドの初期作成またはその後の更新時にコア設定が開始する前に、全ノードに設定が適用されます。



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

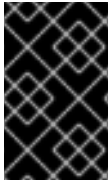
```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml
```

```
parameter_defaults:
  nameserver_ip: 192.168.1.1
```

この設定を適用するには、オーバークラウドの作成時または更新時に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

NovaComputeExtraConfig

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。

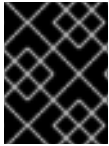
ExtraConfig

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:
  NovaComputeExtraConfig:
    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る場合があります。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、オーバー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** からの設定がオーバー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。

`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。

`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 **rhncfg** 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。サブスクリプションサーバーの 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

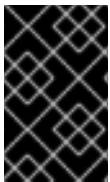
登録のオプションを強制するには **true** に設定します (例: ノードの再登録時など)。

rhel_reg_sat_repo

katello-agent などの Red Hat Satellite 6 の管理ツールが含まれるリポジトリ (例: **rhel-7-server-satellite-tools-6.1-rpms**)。

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-10-rpms,rhel-7-server-rhceph-2-osd-rpms,rhel-7-server-rhceph-2-
mon-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: ""
```

第6章 コンポーザブルサービスとカスタムロール

オーバークラウドは通常、コントローラーノード、コンピューターノード、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に分類されたノードで構成されます。これらのデフォルトの各ロールには、director ノード上にあるコアの Heat テンプレートコレクションで定義されているサービスセットが含まれます。ただし、コアの 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

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。

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。

- 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は、任意の **systemd** の管理対象サービスを割り当てることができます。
- Pacemaker が管理するサービスを分割することはできません。これは、Pacemaker がオーバークラウドクラスターの各ノードで、同じサービスセットを管理するためです。Pacemaker が管理するサービスを分割すると、クラスターのデプロイメントでエラーが発生する場合があります。これらのサービスは、コントローラーロールに残す必要があります。
- Red Hat OpenStack Platform 9 から 10 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。アップグレードスクリプトはデフォルトのオーバークラウドロールのみに対応可能です。
- 初回のデプロイメント後に追加のカスタムロールを作成してそれらをデプロイし、既存のサービスをスケールアップすることができます。
- オーバクラウドのデプロイ後には、ロールのサービスリストを変更することはできません。オーバークラウドのデプロイ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。

サポートされているカスタムロールアーキテクチャー

カスタムロールとコンポーザブルサービスは、Red Hat OpenStack Platform 10 の新機能で、現在のよう初期段階では、試験/検証済みのコンポーザブルサービスの組み合わせは数が限定されています。Red Hat は、カスタムロールとコンポーザブルサービスを使用する際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。

6.1. カスタムロールアーキテクチャーの考察

オーバークラウドの作成プロセスは、ロールのデータを記載したテンプレートを使用してロールを定義します。デフォルトのテンプレートは `/usr/share/openstack-tripleo-heat-templates/roles_data.yaml` にあり、すべてのデフォルトロールタイプ (**Controller**、**Compute**、**BlockStorage**、**ObjectStorage**、**CephStorage**) を定義します。



重要

カスタムの `roles_data.yaml` ファイルを作成する場合には、**Controller** ロールを必ず最初に定義する必要があります。このロールはプライマリーロールとして処理します。

各ロールには、以下のパラメーターが含まれます。

name

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

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
...
```

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。

これらのオプションは、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.yaml** ファイルのパラメーターを **overcloud.j2.yaml** Heat テンプレートに統合します。特定の時点で **overcloud.j2.yaml** Heat テンプレートは **roles_data.yaml** のロールの一覧を繰り返し適用し、対応する各ロール固有のパラメーターとリソースを作成します。

たとえば、**overcloud.j2.yaml** Heat テンプレートの各ロールのリソースの定義は、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.yaml** のそれぞれの **name** パラメーターを使用して、対応する各 **OS::Heat::ResourceGroup** リソースを命名します。

6.2. コンポーザブルサービスアーキテクチャーの考察

コア Heat テンプレートコレクションには、**puppet/services** サブディレクトリー内のコンポーザブルサービステンプレートのコレクションが含まれます。これらのサービスは、以下のコマンドで表示することができます。

```
$ ls /usr/share/openstack-tripleo-heat-templates/puppet/services
```

各サービス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、**keystone.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: >
  OpenStack Identity (`keystone`) service configured with Puppet
```

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。たとえば、**keystone.yaml** サービステンプレートは **OS::TripleO::Services::Keystone** リソース種別に登録されます。

```
grep "OS::TripleO::Services::Keystone" /usr/share/openstack-tripleo-heat-
templates/overcloud-resource-registry-puppet.j2.yaml
OS::TripleO::Services::Keystone: puppet/services/keystone.yaml
```

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::Triple0::Services::CACerts
    - OS::Triple0::Services::CephMon
    - OS::Triple0::Services::CephExternal
    - OS::Triple0::Services::CephRgw
    - OS::Triple0::Services::CinderApi
    - OS::Triple0::Services::CinderBackup
    - OS::Triple0::Services::CinderScheduler
    - OS::Triple0::Services::CinderVolume
    - OS::Triple0::Services::Core
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Keystone
    - OS::Triple0::Services::GlanceApi
    - OS::Triple0::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。

6.3. 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::Triple0::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「[インストール](#)」の項を参照してください。このセクションには、オーバークラウドで OpenStack Data Processing service (**sahara**) を有効にする手順が記載されています。

6.4. 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_data.yaml** ファイルのカスタムコピーを作成します。

```
$ cp /usr/share/openstack-tripleo-heat-templates/roles_data.yaml
~/templates/roles_data-no_heat.yaml
```

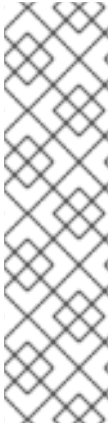
roles_data ファイルを編集して、コントローラーの **ServicesDefault** パラメーターのサービス一覧を変更します。OpenStack Orchestration のサービスまでスクロールしてそれらを削除します。

```
- 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
```

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



注記

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

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

6.5. 新規ロールの作成

以下の例では、OpenStack Networking (**neutron**) エージェントのみをホストする、新しい **Networker** ロールを作成します。この場合には、新しいロールの情報を記載するカスタムの **roles_data** ファイルを作成します。

デフォルトの **roles_data.yaml** ファイルのカスタムコピーを作成します。

```
$ cp /usr/share/openstack-tripleo-heat-templates/roles_data.yaml
~/templates/roles_data-network_node.yaml
```

新しい **roles_data** ファイルを編集して、OpenStack Networking のベースおよびコアのサービスを含む **Networker** ロールを作成します。以下に例を示します。

```
- name: Networker
  CountDefault: 1
  HostnameFormatDefault: '%stackname%-networker-%index%'
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::NeutronDhcpAgent
    - OS::TripleO::Services::NeutronL3Agent
    - OS::TripleO::Services::NeutronMetadataAgent
    - OS::TripleO::Services::NeutronOvsAgent
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::VipHosts
```

また、**CountDefault** を **1** に設定して、デフォルトのオーバークラウドには常に Networking ノードが含まれるようにすることをお勧めします。

既存のオーバークラウド内でサービスをスケーリングする場合には、既存のサービスをコントローラーノード上に保持します。新規オーバークラウドを作成して、OpenStack Networking エージェントのみ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 Networking エージェントを削除します。

```
- name: Controller
```

```

CountDefault: 1
ServicesDefault:
  - OS::Triple0::Services::CACerts
  - OS::Triple0::Services::CephMon
  - OS::Triple0::Services::CephExternal
  - OS::Triple0::Services::CephRgw
  - OS::Triple0::Services::CinderApi
  - OS::Triple0::Services::CinderBackup
  - OS::Triple0::Services::CinderScheduler
  - OS::Triple0::Services::CinderVolume
  - OS::Triple0::Services::Core
  - OS::Triple0::Services::Kernel
  - OS::Triple0::Services::Keystone
  - OS::Triple0::Services::GlanceApi
  - OS::Triple0::Services::GlanceRegistry
  - OS::Triple0::Services::HeatApi
  - OS::Triple0::Services::HeatApiCfn
  - OS::Triple0::Services::HeatApiCloudwatch
  - OS::Triple0::Services::HeatEngine
  - OS::Triple0::Services::MySQL
  - OS::Triple0::Services::NeutronDhcpAgent      # Remove this service
  - OS::Triple0::Services::NeutronL3Agent      # Remove this service
  - OS::Triple0::Services::NeutronMetadataAgent # Remove this service
  - OS::Triple0::Services::NeutronApi
  - OS::Triple0::Services::NeutronCorePlugin
  - OS::Triple0::Services::NeutronOvsAgent      # Remove this service
  - OS::Triple0::Services::RabbitMQ
...

```

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 **networker** フレーバーを作成します。

```

$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4
networker
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="networker" networker

```

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:networker,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、Networker ノードの数とフレーバーを定義します。

```

parameter_defaults:
  OvercloudNetworkerFlavor: networker
  NetworkerCount: 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。

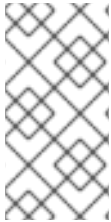

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-
network_node.yaml -e ~/templates/node-count-flavor.yaml
```

デプロイメントが完了すると、コントローラーノードが 1 台、コンピュートノードが 1 台、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。

```
$ nova list
```

6.6. 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 Senu (「[12章 モニタリングツールの設定](#)」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 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。

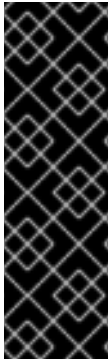
```
parameter_defaults:
  OvercloudGenericFlavor: baremetal
  GenericCount: 1
```

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、コンピュートノードが 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。

6.7. ハイパーコンバージドの **COMPUTE** サービスと **CEPH** サービスの作成



重要

ハイパーコンバージドの Compute サービスと Ceph サービスは、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat サービスレベルアグリーメント (SLA) では完全にサポートされていません。これらは、機能的に完全でない可能性があり、実稼働環境での使用を目的とはしていませんが、近々発表予定のプロダクトイノベーションをリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。テクノロジープレビューとして提供している機能のサポートの対象範囲に関する詳しい情報は、<https://access.redhat.com/support/offerings/techpreview/> を参照してください。

Ceph OSD サービスは、通常はそれら独自の Ceph Storage ノードで実行しますが、コンポーザブルサービスは、Ceph OSD サービスを Ceph Storage ノードの代わりにコンピュートノードで設定する手段を提供します。

たとえば、各ロールのデフォルトのサービス一覧には、以下が含まれます。

コンピュートノード:

```
- name: Compute
  CountDefault: 1
  HostnameFormatDefault: '%stackname%-novacompute-%index%'
  ServicesDefault:
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CephClient
    - OS::Triple0::Services::CephExternal
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::NovaCompute
    - OS::Triple0::Services::NovaLibvirt
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::ComputeNeutronCorePlugin
    - OS::Triple0::Services::ComputeNeutronOvsAgent
    - OS::Triple0::Services::ComputeCeilometerAgent
    - OS::Triple0::Services::ComputeNeutronL3Agent
    - OS::Triple0::Services::ComputeNeutronMetadataAgent
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::NeutronSriovAgent
    - OS::Triple0::Services::OpenDaylightOvs
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::VipHosts
```

Ceph Storage ノード:

```
- name: CephStorage
  ServicesDefault:
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CephOSD
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoPackages
```

- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::SensuClient
- OS::TripleO::Services::FluentdClient
- OS::TripleO::Services::VipHosts

Ceph Storage ロールには、Compute ロールと共通のサービスが含まれているので、それらは無視することができます。1つのサービスが残ります:**OS::TripleO::Services::CephOSD**。

デフォルトの **roles_data** ファイルのカスタムバージョンを作成します。

```
$ cp /usr/share/openstack-tripleo-heat-templates/roles_data.yaml
~/templates/roles_data-ceph_osd_on_compute.yaml
```

ファイルを編集して、コンピュートのサービス一覧に **OS::TripleO::Services::CephOSD** を追加します。

```
- name: Compute
  CountDefault: 1
  HostnameFormatDefault: '%stackname%-novacompute-%index%'
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephOSD
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::NovaCompute
    - OS::TripleO::Services::NovaLibvirt
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::ComputeNeutronCorePlugin
    - OS::TripleO::Services::ComputeNeutronOvsAgent
    - OS::TripleO::Services::ComputeCeilometerAgent
    - OS::TripleO::Services::ComputeNeutronL3Agent
    - OS::TripleO::Services::ComputeNeutronMetadataAgent
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::NeutronSriovAgent
    - OS::TripleO::Services::OpenDaylightOvs
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::VipHosts
```

また、コンピュートのサービス一覧から **OS::TripleO::Services::CephExternal** サービスを完全に削除することができます。これは、オーバークラウドが外部の Ceph Storage クラスターとは統合しないためです。

openstack overcloud deploy コマンドを実行する際には、このロール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-
ceph_osd_on_compute.yaml -e ~/template/storage-environment.yaml
```

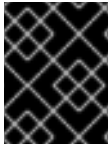
このコマンドには、Ceph Storage 固有のパラメーターを記載したストレージ用のカスタムの環境ファイル (**storage-environment.yaml**) も含まれている点に注意してください。

オーバークラウドのデプロイ後には、コンピュータノード上で Ceph OSD のインストールを検証します。コンピュータノードにログインして以下のコマンドを実行します。

```
[root@overcloud-novacompute-0 ~]# ps ax | grep ceph
17437 ?      Ss      0:00 /bin/bash -c ulimit -n 32768; /usr/bin/ceph-osd -i 0
--pid-file /var/run/ceph/osd.0.pid -c /etc/ceph/ceph.conf --cluster ceph -
f
17438 ?      Sl      0:00 /usr/bin/ceph-osd -i 0 --pid-file
/var/run/ceph/osd.0.pid -c /etc/ceph/ceph.conf --cluster ceph -f
```

6.8. 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
  ServicesDefault:
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::AodhApi
    - OS::TripleO::Services::AodhEvaluator
    - OS::TripleO::Services::AodhListener
    - OS::TripleO::Services::AodhNotifier
    - OS::TripleO::Services::CACerts
    - 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::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
    - OS::TripleO::Services::GnocchiApi
    - 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::Triple0::Services::Horizon
- OS::Triple0::Services::IronicApi
- OS::Triple0::Services::IronicConductor
- OS::Triple0::Services::Kernel
- OS::Triple0::Services::Keepalived
- OS::Triple0::Services::Keystone
- OS::Triple0::Services::ManilaApi
- OS::Triple0::Services::ManilaBackendCephFs
- OS::Triple0::Services::ManilaBackendGeneric
- OS::Triple0::Services::ManilaBackendNetapp
- OS::Triple0::Services::ManilaScheduler
- OS::Triple0::Services::ManilaShare
- OS::Triple0::Services::Memcached
- OS::Triple0::Services::MongoDb
- OS::Triple0::Services::MySQL
- 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::NovaScheduler
- OS::Triple0::Services::NovaVncProxy
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::OpenDaylightApi
- OS::Triple0::Services::OpenDaylightOvs
- OS::Triple0::Services::Pacemaker
- OS::Triple0::Services::RabbitMQ
- OS::Triple0::Services::Redis
- OS::Triple0::Services::SaharaApi
- OS::Triple0::Services::SaharaEngine
- OS::Triple0::Services::SensuClient
- OS::Triple0::Services::Sshd
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::SwiftProxy
- OS::Triple0::Services::SwiftRingBuilder
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::VipHosts

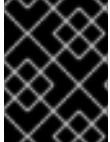
6.9. サービスアーキテクチャー：分割コントローラー

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::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Sshd
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::VipHosts
    - OS::Triple0::Services::CephClient
    - OS::Triple0::Services::CephExternal
    - OS::Triple0::Services::CinderBackup
    - OS::Triple0::Services::CinderVolume
    - OS::Triple0::Services::HAproxy
    - OS::Triple0::Services::Keepalived
    - OS::Triple0::Services::ManilaBackendGeneric
    - OS::Triple0::Services::ManilaBackendNetapp
    - OS::Triple0::Services::ManilaBackendCephFs
    - OS::Triple0::Services::ManilaShare
    - OS::Triple0::Services::Memcached
    - OS::Triple0::Services::MySQL
    - OS::Triple0::Services::Pacemaker
    - OS::Triple0::Services::RabbitMQ
    - OS::Triple0::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::Triple0::Services::Apache
    - OS::Triple0::Services::AodhApi
```

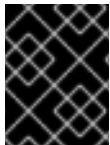
- OS::Triple0::Services::AodhEvaluator
- OS::Triple0::Services::AodhListener
- OS::Triple0::Services::AodhNotifier
- OS::Triple0::Services::CACerts
- OS::Triple0::Services::CeilometerAgentCentral
- OS::Triple0::Services::CeilometerAgentNotification
- OS::Triple0::Services::CeilometerApi
- OS::Triple0::Services::CeilometerCollector
- OS::Triple0::Services::CeilometerExpirer
- OS::Triple0::Services::CephClient
- OS::Triple0::Services::CephExternal
- OS::Triple0::Services::CephMon
- OS::Triple0::Services::CephRgw
- OS::Triple0::Services::CinderApi
- OS::Triple0::Services::CinderScheduler
- OS::Triple0::Services::FluentdClient
- OS::Triple0::Services::GlanceApi
- OS::Triple0::Services::GlanceRegistry
- 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::Kernel
- OS::Triple0::Services::Keystone
- OS::Triple0::Services::ManilaApi
- OS::Triple0::Services::ManilaScheduler
- OS::Triple0::Services::MongoDb
- 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::NovaScheduler
- OS::Triple0::Services::NovaVncProxy
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::OpenDaylightApi
- OS::Triple0::Services::OpenDaylightOvs
- OS::Triple0::Services::SaharaApi
- OS::Triple0::Services::SaharaEngine
- OS::Triple0::Services::SensuClient

- OS::TripleO::Services::Sshd
- OS::TripleO::Services::Snmp
- OS::TripleO::Services::SwiftProxy
- OS::TripleO::Services::SwiftRingBuilder
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::TripleoPackages
- OS::TripleO::Services::VipHosts

6.10. 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::CACerts
- OS::TripleO::Services::FluentdClient
- OS::TripleO::Services::Kernel
- OS::TripleO::Services::Ntp
- OS::TripleO::Services::SensuClient
- OS::TripleO::Services::Sshd
- OS::TripleO::Services::Snmp
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::TripleoPackages
- OS::TripleO::Services::VipHosts

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6.1「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6.1 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
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
Glance	対応
Heat	対応
Horizon	対応
Ironic	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。
Keystone	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。
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
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。
Swift API	対応

ロール	サポートの状態
Swift ストレージ	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:
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Sshd
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::VipHosts
    - 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:
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Sshd
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::VipHosts
    - OS::Triple0::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。

```
- OS::Triple0::Services::CACerts
- OS::Triple0::Services::FluentdClient
- OS::Triple0::Services::Kernel
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::SensuClient
```

- OS::Triple0::Services::Sshd
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::VipHosts
- OS::Triple0::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:
 - OS::Triple0::Services::CACerts
 - OS::Triple0::Services::FluentdClient
 - OS::Triple0::Services::Kernel
 - OS::Triple0::Services::Ntp
 - OS::Triple0::Services::SensuClient
 - OS::Triple0::Services::Sshd
 - OS::Triple0::Services::Snmp
 - OS::Triple0::Services::Timezone
 - OS::Triple0::Services::TripleoFirewall
 - OS::Triple0::Services::TripleoPackages
 - OS::Triple0::Services::VipHosts
 - OS::Triple0::Services::CinderApi
 - OS::Triple0::Services::CinderScheduler

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: ControllerPcmk
 - ServicesDefault:
 - OS::Triple0::Services::CACerts
 - OS::Triple0::Services::FluentdClient
 - OS::Triple0::Services::Kernel
 - OS::Triple0::Services::Ntp
 - OS::Triple0::Services::SensuClient
 - OS::Triple0::Services::Sshd
 - OS::Triple0::Services::Snmp
 - OS::Triple0::Services::Timezone
 - OS::Triple0::Services::TripleoFirewall
 - OS::Triple0::Services::TripleoPackages
 - OS::Triple0::Services::CephClient
 - OS::Triple0::Services::CephExternal
 - OS::Triple0::Services::CinderBackup
 - OS::Triple0::Services::CinderVolume
 - OS::Triple0::Services::HAproxy
 - OS::Triple0::Services::Keepalived
 - OS::Triple0::Services::ManilaBackendGeneric
 - OS::Triple0::Services::ManilaBackendNetapp
 - OS::Triple0::Services::ManilaBackendCephFs
 - OS::Triple0::Services::ManilaShare
 - OS::Triple0::Services::Memcached
 - OS::Triple0::Services::MySQL

- OS::Triple0::Services::Pacemaker
- OS::Triple0::Services::RabbitMQ
- OS::Triple0::Services::Redis
- OS::Triple0::Services::VipHosts

Glance

以下のサービスは、OpenStack Image サービスを構成します。

- name: Glance
 - ServicesDefault:
 - OS::Triple0::Services::CACerts
 - OS::Triple0::Services::FluentdClient
 - OS::Triple0::Services::Kernel
 - OS::Triple0::Services::Ntp
 - OS::Triple0::Services::SensuClient
 - OS::Triple0::Services::Sshd
 - OS::Triple0::Services::Snmp
 - OS::Triple0::Services::Timezone
 - OS::Triple0::Services::TripleoFirewall
 - OS::Triple0::Services::TripleoPackages
 - OS::Triple0::Services::VipHosts
 - OS::Triple0::Services::CephClient
 - OS::Triple0::Services::CephExternal
 - OS::Triple0::Services::GlanceApi
 - OS::Triple0::Services::GlanceRegistry

Heat

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

- name: Heat
 - ServicesDefault:
 - OS::Triple0::Services::CACerts
 - OS::Triple0::Services::FluentdClient
 - OS::Triple0::Services::Kernel
 - OS::Triple0::Services::Ntp
 - OS::Triple0::Services::SensuClient
 - OS::Triple0::Services::Sshd
 - OS::Triple0::Services::Snmp
 - OS::Triple0::Services::Timezone
 - OS::Triple0::Services::TripleoFirewall
 - OS::Triple0::Services::TripleoPackages
 - OS::Triple0::Services::VipHosts
 - OS::Triple0::Services::HeatApi
 - OS::Triple0::Services::HeatApiCfn
 - OS::Triple0::Services::HeatApiCloudwatch
 - OS::Triple0::Services::HeatEngine

Horizon

以下のサービスは、OpenStack Dashboard を構成します。

- name: Horizon
 - ServicesDefault:
 - OS::Triple0::Services::CACerts

```
- OS::Triple0::Services::FluentdClient
- OS::Triple0::Services::Kernel
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::SensuClient
- OS::Triple0::Services::Sshd
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::VipHosts
- OS::Triple0::Services::Apache
- OS::Triple0::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。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。

```
- OS::Triple0::Services::CACerts
- OS::Triple0::Services::FluentdClient
- OS::Triple0::Services::Kernel
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::SensuClient
- OS::Triple0::Services::Sshd
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::VipHosts
- OS::Triple0::Services::IronicApi
- OS::Triple0::Services::IronicConductor
- OS::Triple0::Services::NovaIronic
```

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:
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Sshd
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::VipHosts
    - OS::Triple0::Services::Apache
    - OS::Triple0::Services::Keystone
```

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。

```
- OS::Triple0::Services::CACerts
- OS::Triple0::Services::FluentdClient
- OS::Triple0::Services::Kernel
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::SensuClient
- OS::Triple0::Services::Sshd
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::VipHosts
- OS::Triple0::Services::ManilaApi
- OS::Triple0::Services::ManilaScheduler
```

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:
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Sshd
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::VipHosts
    - 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:
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::Sshd
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::VipHosts
```

- 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

Nova

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

- name: Nova
 - ServicesDefault:
 - OS::Triple0::Services::CACerts
 - OS::Triple0::Services::FluentdClient
 - OS::Triple0::Services::Kernel
 - OS::Triple0::Services::Ntp
 - OS::Triple0::Services::SensuClient
 - OS::Triple0::Services::Sshd
 - OS::Triple0::Services::Snmp
 - OS::Triple0::Services::Timezone
 - OS::Triple0::Services::TripleoFirewall
 - OS::Triple0::Services::TripleoPackages
 - OS::Triple0::Services::VipHosts
 - OS::Triple0::Services::NovaApi
 - OS::Triple0::Services::NovaConductor
 - OS::Triple0::Services::NovaConsoleauth
 - OS::Triple0::Services::NovaScheduler
 - OS::Triple0::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:
 - OS::Triple0::Services::CACerts
 - OS::Triple0::Services::FluentdClient
 - OS::Triple0::Services::Kernel
 - OS::Triple0::Services::Ntp
 - OS::Triple0::Services::SensuClient
 - OS::Triple0::Services::Sshd
 - OS::Triple0::Services::Snmp
 - OS::Triple0::Services::Timezone
 - OS::Triple0::Services::TripleoFirewall
 - OS::Triple0::Services::TripleoPackages
 - OS::Triple0::Services::VipHosts
 - 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 10** 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:
 - OS::Triple0::Services::CACerts
 - OS::Triple0::Services::FluentdClient
 - OS::Triple0::Services::Kernel
 - OS::Triple0::Services::Ntp
 - OS::Triple0::Services::SensuClient
 - OS::Triple0::Services::Sshd
 - OS::Triple0::Services::Snmp
 - OS::Triple0::Services::Timezone
 - OS::Triple0::Services::TripleoFirewall
 - OS::Triple0::Services::TripleoPackages
 - OS::Triple0::Services::VipHosts
 - OS::Triple0::Services::OpenDaylightApi
 - OS::Triple0::Services::OpenDaylightOvs

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。

- OS::Triple0::Services::CACerts
- OS::Triple0::Services::FluentdClient
- OS::Triple0::Services::Kernel
- OS::Triple0::Services::Ntp
- OS::Triple0::Services::SensuClient
- OS::Triple0::Services::Sshd
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::VipHosts
- 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:
 - OS::Triple0::Services::CACerts
 - OS::Triple0::Services::FluentdClient
 - OS::Triple0::Services::Kernel
 - OS::Triple0::Services::Ntp
 - OS::Triple0::Services::SensuClient

- OS::Triple0::Services::Sshd
- OS::Triple0::Services::Snmp
- OS::Triple0::Services::Timezone
- OS::Triple0::Services::TripleoFirewall
- OS::Triple0::Services::TripleoPackages
- OS::Triple0::Services::VipHosts
- OS::Triple0::Services::SwiftProxy
- OS::Triple0::Services::SwiftRingBuilder

Swift ストレージ

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

- name: ObjectStorage
 - ServicesDefault:
 - OS::Triple0::Services::CACerts
 - OS::Triple0::Services::FluentdClient
 - OS::Triple0::Services::Kernel
 - OS::Triple0::Services::Ntp
 - OS::Triple0::Services::SensuClient
 - OS::Triple0::Services::Sshd
 - OS::Triple0::Services::Snmp
 - OS::Triple0::Services::Timezone
 - OS::Triple0::Services::TripleoFirewall
 - OS::Triple0::Services::TripleoPackages
 - OS::Triple0::Services::VipHosts
 - OS::Triple0::Services::SwiftRingBuilder
 - OS::Triple0::Services::SwiftStorage

Telemetry

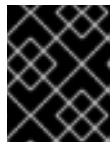
以下のサービスは、OpenStack Telemetry サービスを構成します。

- name: Telemetry
 - ServicesDefault:
 - OS::Triple0::Services::CACerts
 - OS::Triple0::Services::FluentdClient
 - OS::Triple0::Services::Kernel
 - OS::Triple0::Services::Ntp
 - OS::Triple0::Services::SensuClient
 - OS::Triple0::Services::Sshd
 - OS::Triple0::Services::Snmp
 - OS::Triple0::Services::Timezone
 - OS::Triple0::Services::TripleoFirewall
 - OS::Triple0::Services::TripleoPackages
 - OS::Triple0::Services::VipHosts
 - OS::Triple0::Services::Apache
 - OS::Triple0::Services::AodhApi
 - OS::Triple0::Services::AodhEvaluator
 - OS::Triple0::Services::AodhListener
 - OS::Triple0::Services::AodhNotifier
 - OS::Triple0::Services::CeilometerAgentCentral
 - OS::Triple0::Services::CeilometerAgentNotification
 - OS::Triple0::Services::CeilometerApi
 - OS::Triple0::Services::CeilometerCollector
 - OS::Triple0::Services::CeilometerExpirer

- OS::Triple0::Services::GnocchiApi
- OS::Triple0::Services::GnocchiMetricd
- OS::Triple0::Services::GnocchiStatsd
- OS::Triple0::Services::MongoDb

6.11. 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 で利用可能なすべてのコンポーザブルサービスをまとめています。



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。

サービス	説明
OS::Triple0::Services::AodhApi	Puppet で設定される OpenStack Telemetry Alarming (aodh) API サービス
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

サービス	説明
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
OS::Triple0::Services::CinderScheduler	Puppet で設定される OpenStack Block Storage (cinder) Scheduler サービス
OS::Triple0::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::Triple0::Services::ComputeCeilometerAgent	Puppet で設定される OpenStack Telemetry (ceilometer) Compute Agent サービス
OS::Triple0::Services::ComputeNeutronCorePlugin	Puppet で設定される OpenStack Networking (neutron) ML2 プラグイン
OS::Triple0::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::Triple0::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::Triple0::Services::ComputeNeutronOvsAgent	Puppet で設定される OpenStack Networking (neutron) OVS エージェント
OS::Triple0::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::Triple0::Services::GlanceApi	Puppet で設定される OpenStack Image (glance) API サービス

サービス	説明
OS::TripleO::Services::GlanceRegistry	Puppet で設定される OpenStack Image (glance) Registry 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
OS::TripleO::Services::HeatApiCloudwatch	Puppet で設定される Openstack Orchestration (heat) CloudWatch API サービス
OS::TripleO::Services::HeatEngine	Puppet で設定される Openstack Orchestration (heat) Engine サービス
OS::TripleO::Services::Horizon	Puppet で設定される Openstack Dashboard (horizon) サービス
OS::TripleO::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::TripleO::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::TripleO::Services::Keepalived	Puppet で設定される Keepalived サービス
OS::TripleO::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::TripleO::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

サービス	説明
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::Keystone	Puppet で設定される Openstack Identity (keystone) 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
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

サービス	説明
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
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::OpenDaylight	(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)

サービス	説明
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::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::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
OS::Triple0::Services::SwiftRingBuilder	OpenStack Object Storage (swift) Ringbuilder
OS::Triple0::Services::SwiftStorage	Puppet で設定される OpenStack Object Storage (swift) サービス
OS::Triple0::Services::Timezone	コンポーザブルな Timezone サービス
OS::Triple0::Services::TripleoFirewall	ファイアウォールの設定
OS::Triple0::Services::TripleoPackages	パッケージのインストールの設定

第7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。

- ネットワーク 1: プロビジョニング
- ネットワーク 2: 内部 API
- ネットワーク 3: テナントネットワーク
- ネットワーク 4: ストレージ
- ネットワーク 5: ストレージ管理
- ネットワーク 6: 管理
- ネットワーク 7: 外部および Floating IP (オーバークラウドの作成後にマッピング)

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7.1 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
内部 API	172.16.0.0/24	201
テナント	172.17.0.0/24	202
ストレージ	172.18.0.0/24	203
ストレージ管理	172.19.0.0/24	204
管理	172.20.0.0/24	205
外部 / Floating IP	10.1.1.0/24	100

7.1. 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、デフォルトのボンディング 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/bond-with-vlans** 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/bond-with-vlans ~/templates/nic-configs
```

このコマンドにより、各ロールのボンディング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** のセクションが含まれます。今回のシナリオでは、**resources** セクションのみを編集します。各 **resources** セクションは、以下のように開始されます。

```
resources:
OsNetConfigImpl:
  type: OS::Heat::StructuredConfig
  properties:
    group: os-apply-config
    config:
      os_net_config:
        network_config:
```

このコマンドでは、**os-apply-config** コマンドと **os-net-config** サブコマンドがノードのネットワークプロパティを設定するように要求が作成されます。**network_config** セクションには、種別順に並べられたカスタムのインターフェース設定が含まれます。これらの種別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。

```
- type: ovs_bridge
  name: {get_input: bridge_name}
  members:
    - type: ovs_bond
      name: bond1
      members:
        - type: interface
          name: nic2
          primary: true
        - type: interface
          name: nic3
    - type: vlan
      device: bond1
      vlan_id: {get_param: ExternalNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: ExternalIpSubnet}
```

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](#)」の項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**、および **vlan** オブジェクトを接続する Linux ブリッジを定義します。

```

- type: linux_bridge
  name: bridge1
  addresses:
    - ip_netmask:
        list_join:
          - '/'
          - - {get_param: ControlPlaneIp}
            - {get_param: ControlPlaneSubnetCidr}
  members:
    - type: interface
      name: nic1
      primary: true
- type: vlan
  vlan_id: {get_param: ExternalNetworkVlanID}
  device: bridge1
  addresses:
    - ip_netmask: {get_param: ExternalIpSubnet}
  routes:
    - ip_netmask: 0.0.0.0/0
      default: true
      next_hop: {get_param: ExternalInterfaceDefaultRoute}

```

各アイテムの完全なパラメーター一覧については「[15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。

この例では、デフォルトのボンディングインターフェース設定を使用します。たとえば `/home/stack/templates/nic-configs/controller.yaml` テンプレートは以下の **network_config** を使用します。

```

resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            - 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}
- type: ovs_bridge
  name: {get_input: bridge_name}
  dns_servers: {get_param: DnsServers}
  members:
    - type: ovs_bond
      name: bond1
      ovs_options: {get_param: BondInterfaceOvsOptions}
      members:
        - type: interface
          name: nic2
          primary: true
        - type: interface
          name: nic3
    - type: vlan
      device: bond1
      vlan_id: {get_param: ExternalNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: ExternalIpSubnet}
      routes:
        - default: true
          next_hop: {get_param:
ExternalInterfaceDefaultRoute}
    - type: vlan
      device: bond1
      vlan_id: {get_param: InternalApiNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: InternalApiIpSubnet}
    - type: vlan
      device: bond1
      vlan_id: {get_param: StorageNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: StorageIpSubnet}
    - type: vlan
      device: bond1
      vlan_id: {get_param: StorageMgmtNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: StorageMgmtIpSubnet}
    - type: vlan
      device: bond1
      vlan_id: {get_param: TenantNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: TenantIpSubnet}
    - type: vlan
      device: bond1
      vlan_id: {get_param: ManagementNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: ManagementIpSubnet}

```



注記

管理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、管理ネットワークを有効化します。

このテンプレートは、ブリッジ (通常 **br-ex** という名前の外部ブリッジ) を定義し、**nic2** と **nic3** の 2 つの番号付きインターフェースから、**bond1** と呼ばれるボンディングインターフェースを作成します。ブリッジにはタグ付けされた VLAN デバイスの番号が含まれており、**bond1** を親デバイスとして使用します。またこのテンプレートには、director に接続するインターフェースも含まれます。

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。

これらのパラメーターの多くは **get_param** 関数を使用する点に注意してください。これらのパラメーターは、使用するネットワーク専用に作成した環境ファイルで定義します。



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、使用済みの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
```

7.2. 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。外部ネットワークの無効化 (**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。外部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-bond-with-vlans.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-bond-with-vlans.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: ""
# Customize bonding options if required
BondInterfaceOvsOptions:
    "bond_mode=balance-slb"

```

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、内部 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。

外部ネットワークは、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 の両方に外部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 アドレスのみを外部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。

BondInterfaceOvsOptions オプションは、**nic2** および **nic3** を使用するボンディングインターフェースのオプションを提供します。ボンディングオプションについての詳しい情報は、「[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。

7.3. 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。

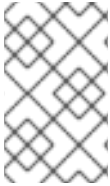
たとえば、ハイライトしたセクションを変更して、ストレージ管理ネットワークサービスをストレージネットワークに再割り当てすることができます。

```
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 # 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 # Changed from storage_mgmt
    CephPublicNetwork: storage
    ControllerHostnameResolveNetwork: internal_api
    ComputeHostnameResolveNetwork: internal_api
    BlockStorageHostnameResolveNetwork: internal_api
    ObjectStorageHostnameResolveNetwork: internal_api
    CephStorageHostnameResolveNetwork: storage
```

これらのパラメーターを **storage** に変更すると、対象のサービスはストレージ管理ネットワークではなく、ストレージネットワークに割り当てられます。つまり、**parameter_defaults** セットをストレージ管理ネットワークではなくストレージ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。

7.4. 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。

分離された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。分離されたネットワーク上の 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
  # Triple0 overcloud networks
  OS::Triple0::Network::External: /usr/share/openstack-tripleo-heat-
templates/network/external.yaml
  OS::Triple0::Network::InternalApi: /usr/share/openstack-tripleo-heat-
templates/network/internal_api.yaml
  OS::Triple0::Network::StorageMgmt: /usr/share/openstack-tripleo-heat-
templates/network/storage_mgmt.yaml
  OS::Triple0::Network::Storage: /usr/share/openstack-tripleo-heat-
templates/network/storage.yaml
  OS::Triple0::Network::Tenant: /usr/share/openstack-tripleo-heat-
templates/network/tenant.yaml
  OS::Triple0::Network::Management: /usr/share/openstack-tripleo-heat-
templates/network/management.yaml

  # Port assignments for the VIPs
  OS::Triple0::Network::Ports::ExternalVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::Triple0::Network::Ports::InternalApiVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::Triple0::Network::Ports::StorageVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::Triple0::Network::Ports::StorageMgmtVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
  OS::Triple0::Network::Ports::TenantVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
  OS::Triple0::Network::Ports::ManagementVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml
  OS::Triple0::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/vip.yaml

  # Port assignments for the controller role
  OS::Triple0::Controller::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::Triple0::Controller::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::Triple0::Controller::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::Triple0::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::TripleO::CephStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::CephStorage::Ports::StorageMgmtPort: OS::Heat::None

# 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: OS::Heat::None

# 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: OS::Heat::None

parameter_defaults:
  ServiceNetMap:
    ApacheNetwork: internal_api
    NeutronTenantNetwork: tenant
    CeilometerApiNetwork: internal_api
    AodhApiNetwork: internal_api
    GnocchiApiNetwork: internal_api
    MongoDBNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    GlanceApiNetwork: internal_api
    GlanceRegistryNetwork: internal_api
    IronicApiNetwork: ctlplane
    IronicNetwork: ctlplane
    KeystoneAdminApiNetwork: ctlplane # allows undercloud to config
endpoints
  KeystonePublicApiNetwork: internal_api
  ManilaApiNetwork: internal_api
  NeutronApiNetwork: internal_api
  HeatApiNetwork: internal_api
  HeatApiCfnNetwork: internal_api
  HeatApiCloudwatchNetwork: internal_api

```

```
NovaApiNetwork: internal_api
NovaColdMigrationNetwork: ctlplane
NovaMetadataNetwork: internal_api
NovaVncProxyNetwork: internal_api
NovaLibvirtNetwork: internal_api
SwiftStorageNetwork: storage # 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 # Changed from storage_mgmt
CephMonNetwork: storage
CephRgwNetwork: storage
PublicNetwork: external
OpendaylightApiNetwork: 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、ストレージ管理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。ストレージ管理サービスをストレージ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8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。

8.1. 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。

```
ironic node-update <id> replace properties/capabilities='node:controller-0,boot_option:local'
```

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 ...
```

8.2. 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ができます。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します。

8.3. 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。コアの 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:
```



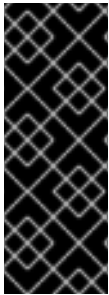
```

- 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。

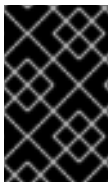
選択した IP アドレスは、ネットワーク環境ファイルで定義されている各ネットワークの割り当てプールの範囲に入らないようにしてください (「[ネットワーク環境ファイルの作成](#)」参照)。たとえば、**internal_api** の割り当ては **InternalApiAllocationPools** の範囲外となるようにします。これにより、自動的に選択される IP アドレスと競合が発生しないようになります。また同様に、IP アドレスの割り当てが標準の予測可能な仮想 IP 配置 (「[予測可能な仮想 IP の割り当て](#)」を参照) または外部のロードバランシング (「[外部の負荷分散機能の設定](#)」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 (「[7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]

```

8.4. 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。

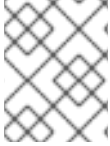
```
parameter_defaults:
    ...
    # Predictable VIPs
    ControlFixedIPs: [{'ip_address': '192.168.201.101'}]
    InternalApiVirtualFixedIPs: [{'ip_address': '172.16.0.9'}]
    PublicVirtualFixedIPs: [{'ip_address': '10.1.1.9'}]
    StorageVirtualFixedIPs: [{'ip_address': '172.18.0.9'}]
    StorageMgmtVirtualFixedIPs: [{'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プンクラウドのみが対象です。外部のロードバランサーで VIP を割り当てる場合には、そのための専用の『[オープンクラウド向けの外部のロードバランシング](#)』ガイドに記載の手順に従ってください。

第9章 オーバークラウドの 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。内部 API や管理 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、「[7章 ネットワークの分離](#)」を参照してください。

9.1. 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
```

9.2. 認証局の作成

通常、SSL/TLS 証明書の署名には、外部の認証局を使用します。場合によっては、独自の認証局を使用する場合もあります。たとえば、内部のみの認証局を使用するように設定する場合などです。

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。

9.3. 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が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。

9.4. 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
```

9.5. 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 エントリーおよび DNS エントリーとして、同じパブリック API の IP アドレスを追加します。DNS も使用する場合は、同じセクションに DNS エントリーとしてそのサーバーのホスト名を追加します。`openssl.cnf` 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。

9.6. 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`」オプションで設定します。
- 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。

9.7. 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。

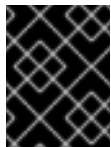
```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/enable-
tls.yaml ~/templates/.
```

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
```

9.8. ルート証明書の注入

証明書の署名者がオーバークラウドのイメージにあるデフォルトのトラストストアに含まれない場合には、オーバークラウドのイメージに認証局を注入する必要があります。Heat テンプレートコレクションから **inject-trust-anchor.yaml** 環境ファイルをコピーします。

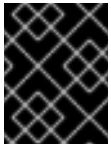
```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/inject-trust-anchor.yaml ~/templates/.
```

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
```

9.9. DNS エンドポイントの設定

DNS ホスト名を使用して SSL/TLS でオーバークラウドにアクセスする場合は、新しい環境ファイル (**~/templates/cloudname.yaml**) を作成して、オーバー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

オーバー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"]
```

9.10. 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
```

9.11. 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:

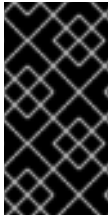
- **enable-tls.yaml** ファイルを編集し
て、**SSLCertificate**、**SSLKey**、**SSLIntermediateCertificate** のパラメーターを更新してください。
- 認証局が変更された場合には、**inject-trust-anchor.yaml** ファイルを編集し
て、**SSLRootCertificate** パラメーターを更新してください。

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10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。

10.1. 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、コアとなる Heat テンプレートコレクション内に既に存在する環境ファイルの変更がベースとなります。

コアの 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るはずです。

```
parameter_defaults:
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: false
  CinderEnableNfsBackend: true
  NovaEnableRbdBackend: false
  GlanceBackend: 'file'

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

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

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。

10.2. 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。

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す。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。

オーバークラウドの Ceph Storage に関する詳しい情報は、両シナリオに沿った全手順を記載している専用の『[オーバークラウド向けの Red Hat Ceph Storage](#)』ガイドを参照してください。

10.3. サードパーティー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。

設定に関する完全な情報は、『[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

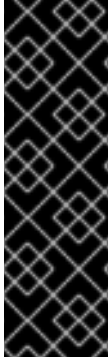
Block Storage (cinder) サービス用に NetApp ストレージアプライアンスをバックエンドとしてデプロイします。

環境ファイルは `/usr/share/openstack-tripleo-heat-templates/environments/cinder-netapp-config.yaml` にあります。

設定に関する完全な情報は、『[NetApp Block Storage Back End Guide](#)』を参照してください。

第11章 コンテナ化されたコンピュートノードの設定

director には、OpenStack のコンテナ化プロジェクト (Kolla) のサービスとオープンクラウドのコンピュートノードを統合するオプションがあります。たとえば、Red Hat Enterprise Linux Atomic Host をベースのオペレーティングシステムや個別のコンテナとして使用して、異なる OpenStack サービスを実行するコンピュートノードを作成します。



重要

コンテナ化されたコンピュートノードは、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat サービスレベルアグリーメント (SLA) では完全にサポートされていません。これらは、機能的に完全でない可能性があり、実稼働環境での使用を目的とはしていませんが、近々発表予定のプロダクトイノベーションをリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。テクノロジープレビューとして提供している機能のサポートの対象範囲に関する詳しい情報は、<https://access.redhat.com/support/offerings/techpreview/> を参照してください。

director のコアとなる Heat テンプレートコレクションには、コンテナ化されているコンピュートノードの設定をサポートする環境ファイルが含まれます。これらのファイルには以下が含まれます。

- **docker.yaml**: コンテナ化されているコンピュートノードを設定する主要な環境ファイル
- **docker-network.yaml**: コンテナ化されたコンピュートノードのネットワークの環境ファイル (ネットワークの分離なし)
- **docker-network-isolation.yaml**: コンテナ化されたコンピュートノードのネットワークの環境ファイル (ネットワークの分離あり)

11.1. スタックの深度を高くする方法

コンテナ化されたコンピュートの Heat テンプレート内のリソーススタック数に対応するには、アンダークラウドで OpenStack Orchestration (heat) のスタックの深度を高くする必要があります。スタックの深度を増やすには、以下の手順を使用します。

1. **/etc/heat/heat.conf** を編集して、**max_nested_stack_depth** パラメーターを特定します。このパラメーターの値を **10** に増やします。

```
max_nested_stack_depth = 10
```

このファイルを保存します。

2. OpenStack Orchestration (heat) サービスを再起動します。

```
sudo systemctl restart openstack-heat-engine.service
```



重要

アンダークラウドのマイナーおよびメジャーバージョンの更新を実行すると、`/etc/heat/heat.conf` ファイルへの変更が元に戻されてしまう可能性があります。必要な場合には、`heat::engine::max_nested_stack_depth` hieradata を設定して、スタックの深度の設定を永続的にします。アンダークラウドの hieradata を設定するには、`undercloud.conf` ファイル内の `hieradata_override` パラメーターにカスタムの hieradata を含むファイルを指定してください。

11.2. コンテナ化されたコンピュートの環境ファイル (DOCKER.YAML) の検証

`docker.yaml` ファイルは、コンテナ化されたコンピュートノードの設定用の主な環境ファイルです。このファイルには、`resource_registry` のエントリーが含まれます。

```
resource_registry:
  OS::TripleO::ComputePostDeployment: ../docker/compute-post.yaml
  OS::TripleO::NodeUserData:
    ../docker/firstboot/install_docker_agents.yaml
```

OS::TripleO::NodeUserData

初回起動時にカスタムの設定を使用する Heat テンプレートを提供します。今回の場合は、初回起動時に、`openstack-heat-docker-agents` コンテナをコンピュートノードにインストールします。このコンテナは、初期化スクリプトのセットを提供して、コンテナ化されたコンピュートノードと Heat フックを設定して director と通信します。

OS::TripleO::ComputePostDeployment

コンピュートノードに対するデプロイ後の設定リソースが含まれる Heat テンプレートを提供します。これには、Puppet に `tags` セットを提供するソフトウェア設定リソースが含まれます。

```
ComputePuppetConfig:
  type: OS::Heat::SoftwareConfig
  properties:
    group: puppet
    options:
      enable_hiera: True
      enable_facter: False
      tags:
package, file, concat, file_line, nova_config, neutron_config, neutron_agent_ovs, neutron_plugin_ml2
  inputs:
    - name: tripleo::packages::enable_install
      type: Boolean
      default: True
  outputs:
    - name: result
  config:
    get_file: ../puppet/manifests/overcloud_compute.pp
```

これらのタグは、Puppet モジュールを `openstack-heat-docker-agents` コンテナに渡すように定義します。

`docker.yaml` ファイルには、`NovaImage` と呼ばれる `parameter` が含まれており、コンピュートノード

ドをプロビジョニングする際に **overcloud-full** イメージを異なるイメージ (**atomic-image**) に置き換えます。このような新規イメージをアップロードする方法は、「[Atomic Host のイメージのアップロード](#)」を参照してください。

docker.yaml ファイルには、Docker レジストリーとイメージがコンピュートノードサービスを使用するように定義する **parameter_defaults** セクションも含まれます。このセクションを変更して、デフォルトの `registry.access.redhat.com` の代わりにローカルのレジストリーを使用するように指定することもできます。ローカルのレジストリーの設定方法は、「[ローカルのレジストリーの使用](#)」を参照してください。

11.3. ATOMIC HOST のイメージのアップロード

director では、**atomic-image** としてイメージストアにインポートする Red Hat Enterprise Linux 7 Atomic Host のクラウドイメージのコピーが必要です。これは、コンピュートノードにはオーバークラウド作成のプロビジョニングの際に、ベースの OS イメージが必要なためです。

Red Hat Enterprise Linux 7 Atomic Host の製品ページ (https://access.redhat.com/downloads/content/271/ver=/rhel---7/7.2.2-2/x86_64/product-software) からクラウドのイメージのコピーをダウンロードし、**stack** ユーザーのホームディレクトリーの **images** サブディレクトリーに保存します。

イメージのダウンロードが完了したら、**stack** ユーザーとして director にイメージをインポートします。

```
$ glance image-create --name atomic-image --file ~/images/rhel-atomic-cloud-7.2-12.x86_64.qcow2 --disk-format qcow2 --container-format bare
```

以下のコマンドでは、その他のオーバークラウドのイメージとこのイメージをインポートします。

```
$ glance image-list
+-----+-----+
| ID                                         | Name                               |
+-----+-----+
| 27b5bad7-f8b2-4dd8-9f69-32dfe84644cf     | atomic-image                      |
| 08c116c6-8913-427b-b5b0-b55c18a01888     | bm-deploy-kernel                  |
| aec4c104-0146-437b-a10b-8ebc351067b9     | bm-deploy-ramdisk                 |
| 9012ce83-4c63-4cd7-a976-0c972be747cd     | overcloud-full                    |
| 376e95df-c1c1-4f2a-b5f3-93f639eb9972     | overcloud-full-initrd             |
| 0b5773eb-4c64-4086-9298-7f28606b68af     | overcloud-full-vmlinuz            |
+-----+-----+
```

11.4. ローカルのレジストリーの使用

デフォルトの設定は、Red Hat のコンテナレジストリーをイメージのダウンロードに使用しますが、オプションの手順として、ローカルレジストリーを使用して、オーバークラウドの作成プロセス中の帯域幅を確保することができます。

既存のローカルレジストリーを使用するか、新たにインストールします。新しいレジストリーをインストールするには、『[Getting Started with Containers](#)』の「[Get Started with Docker Formatted Container Images](#)」に記載の手順を参照してください。

必要なイメージをレジストリーにプルします。

```
$ sudo docker pull
```

```
registry.access.redhat.com/rhosp10_tech_preview/openstack-nova-
compute:latest
$ sudo docker pull
registry.access.redhat.com/rhosp10_tech_preview/openstack-data:latest
$ sudo docker pull
registry.access.redhat.com/rhosp10_tech_preview/openstack-nova-
libvirt:latest
$ sudo docker pull
registry.access.redhat.com/rhosp10_tech_preview/openstack-neutron-
openvswitch-agent:latest
$ sudo docker pull
registry.access.redhat.com/rhosp10_tech_preview/openstack-openvswitch-
vswitchd:latest
$ sudo docker pull
registry.access.redhat.com/rhosp10_tech_preview/openstack-openvswitch-db-
server:latest
$ sudo docker pull
registry.access.redhat.com/rhosp10_tech_preview/openstack-heat-docker-
agents:latest
```

イメージをプルした後は、正しいレジストリーホストにタグ付けします。

```
$ sudo docker tag
registry.access.redhat.com/rhosp10_tech_preview/openstack-nova-
compute:latest localhost:8787/registry.access.redhat.com/openstack-nova-
compute:latest
$ sudo docker tag
registry.access.redhat.com/rhosp10_tech_preview/openstack-data:latest
localhost:8787/registry.access.redhat.com/openstack-data:latest
$ sudo docker tag
registry.access.redhat.com/rhosp10_tech_preview/openstack-nova-
libvirt:latest localhost:8787/registry.access.redhat.com/openstack-nova-
libvirt:latest
$ sudo docker tag
registry.access.redhat.com/rhosp10_tech_preview/openstack-neutron-
openvswitch-agent:latest
localhost:8787/registry.access.redhat.com/openstack-neutron-openvswitch-
agent:latest
$ sudo docker tag
registry.access.redhat.com/rhosp10_tech_preview/openstack-openvswitch-
vswitchd:latest localhost:8787/registry.access.redhat.com/openstack-
openvswitch-vswitchd:latest
$ sudo docker tag
registry.access.redhat.com/rhosp10_tech_preview/openstack-openvswitch-db-
server:latest localhost:8787/registry.access.redhat.com/openstack-
openvswitch-db-server:latest
$ sudo docker tag
registry.access.redhat.com/rhosp10_tech_preview/openstack-heat-docker-
agents:latest localhost:8787/registry.access.redhat.com/openstack-heat-
docker-agents:latest
```

レジストリーにプッシュします。

```
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
nova-compute:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
```

```
data:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
nova-libvirt:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
neutron-openvswitch-agent:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
openvswitch-vswitchd:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
openvswitch-db-server:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
heat-docker-agents:latest
```

メインの **docker.yaml** 環境ファイルのコピーを **templates** サブディレクトリーに作成します。

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

ファイルを編集して **resource_registry** が絶対パスを使用するように変更します。

```
resource_registry:
  OS::TripleO::ComputePostDeployment: /usr/share/openstack-tripleo-heat-
templates/docker/compute-post.yaml
  OS::TripleO::NodeUserData: /usr/share/openstack-tripleo-heat-
templates/docker/firstboot/install_docker_agents.yaml
```

parameter_defaults の **DockerNamespace** をレジストリーの URL に変更します。ま
た、**DockerNamespaceIsRegistry** を **true** に設定します。以下に例を示します。

```
parameter_defaults:
  DockerNamespace: registry.example.com:8787/registry.access.redhat.com
  DockerNamespaceIsRegistry: true
```

ローカルレジストリーに必要な Docker イメージが追加され、コンテナ化されたコンピュートの設定
はこのレジストリーを使用するように変更されました。

11.5. オーバークラウドのデプロイメントへの環境ファイルの追加

オーバークラウドの作成時には、**openstack overcloud deploy** のコマンドで、コンテナ化され
たコンピュートノード用のメインの環境ファイル (**docker.yaml**) とネットワーク環境ファイル
(**docker-network.yaml**) を指定します。以下に例を示します。

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-
heat-templates/environments/docker.yaml -e /usr/share/openstack-tripleo-
heat-templates/environments/docker-network.yaml [OTHER OPTIONS] ...
```

コンテナ化されたコンピュートノードは、ネットワークが分離されたオーバークラウドでも機能しま
す。これには、主要な環境ファイルに加え、ネットワーク分離ファイル (**docker-network-
isolation.yaml**) も必要です。「[7章 ネットワークの分離](#)」のネットワーク分離ファイルの前に、こ
れらの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/docker.yaml -e /usr/share/openstack-tripleo-
heat-templates/environments/docker-network-isolation.yaml -e
```



```
/usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-  
with-vlans.yaml -e /usr/share/openstack-tripleo-heat-  
templates/environments/network-isolation.yaml [OTHER OPTIONS] ...
```

director により、コンテナ化されたコンピュートノードによりオーバークラウドが作成されました。

第12章 モニタリングツールの設定

モニタリングツールは、オペレーターが OpenStack 環境を維持管理するのに役立つオプションのツールセットです。これらのツールは、以下の機能を果たします。

- 集中ロギング: OpenStack 環境の全コンポーネントからのログを 1 つの場所に収集することができます。すべてのノードとサービスにわたって問題を特定することができます。また、オプションでログデータをエクスポートして、問題を診断するサポートを受けることもできます。
- 可用性監視: OpenStack 環境内の全コンポーネントをモニタリングして、いずれかのコンポーネントが現在停止中または機能していない状態かどうかを判断します。応答時間を最適化するために、問題の発生時に通知を受信することもできます。

12.1. アーキテクチャー

モニタリングツールは、クライアントが Red Hat OpenStack Platform オーバークラウドノードにデプロイされる、クライアント/サーバーモデルを使用します。Fluentd サービスがクライアント側の集中ロギング (CL) を提供し、Sensu クライアントサービスが可用性監視 (AM) を提供します。

12.1.1. 集中ロギング

集中ロギングにより、OpenStack 環境全体にわたるログを一箇所で確認することができます。これらのログは、syslog や audit ログファイルなどのオペレーティングシステム、RabbitMQ や MariaDB などのインフラストラクチャーコンポーネント、Identity や Compute などの OpenStack サービスから収集されます。

集中ロギングのツールチェーンは、以下のような複数のコンポーネントで構成されます。

- ログ収集エージェント (Fluentd)
- ログリレー/トランスフォーマー (Fluentd)
- データストア (Elasticsearch)
- API/プレゼンテーション層 (Kibana)



注記

director は、集中ロギング向けのサーバー側のコンポーネントはデプロイしません。Red Hat では、ログアグリゲーターとして実行するプラグインを使用する Elasticsearch データベース、Kibana、Fluentd などのサーバー側のコンポーネントはサポートしていません。

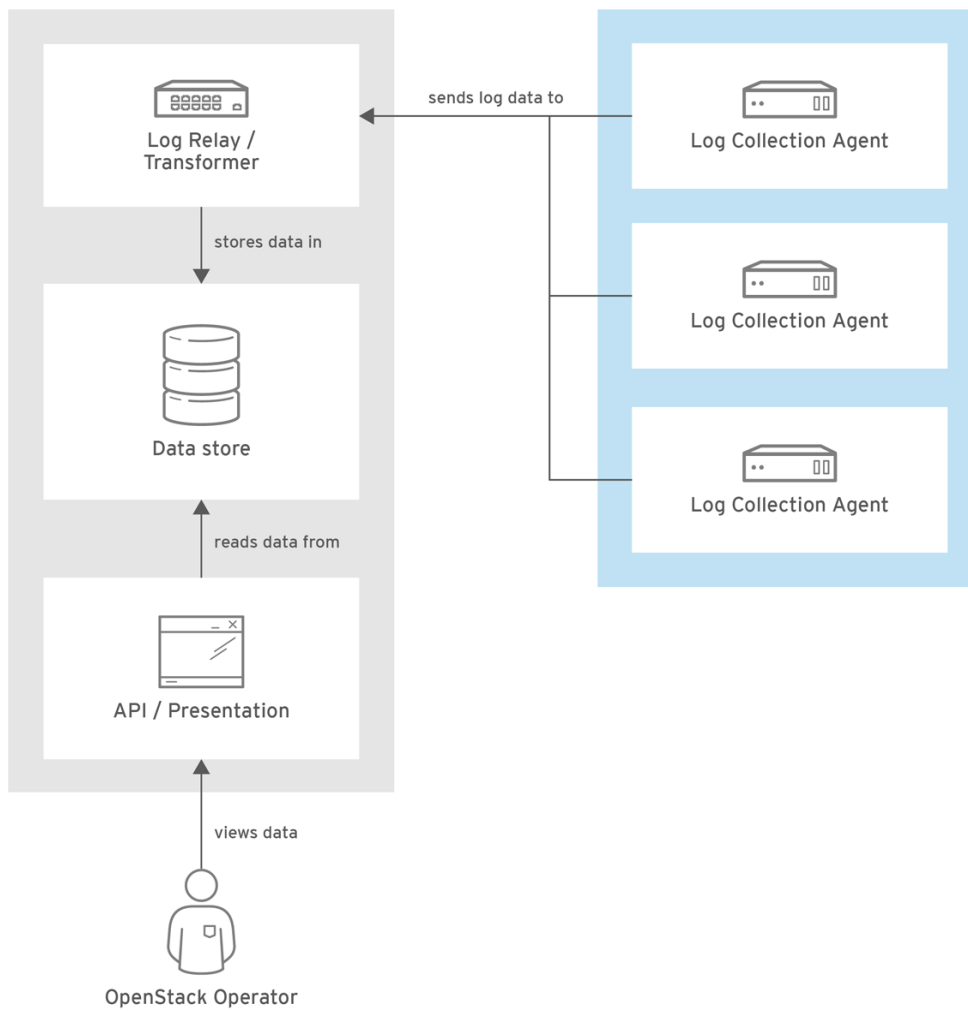
以下の図は、集中ロギングのコンポーネントとそれらの対話を示しています。



注記

青で示した項目は Red Hat のサポート対象コンポーネントです。

図12.1 集中ロギングのハイレベルアーキテクチャー



OPENSTACK_435795_0117

図12.2 Red Hat OpenStack Platform の単一ノードデプロイメント

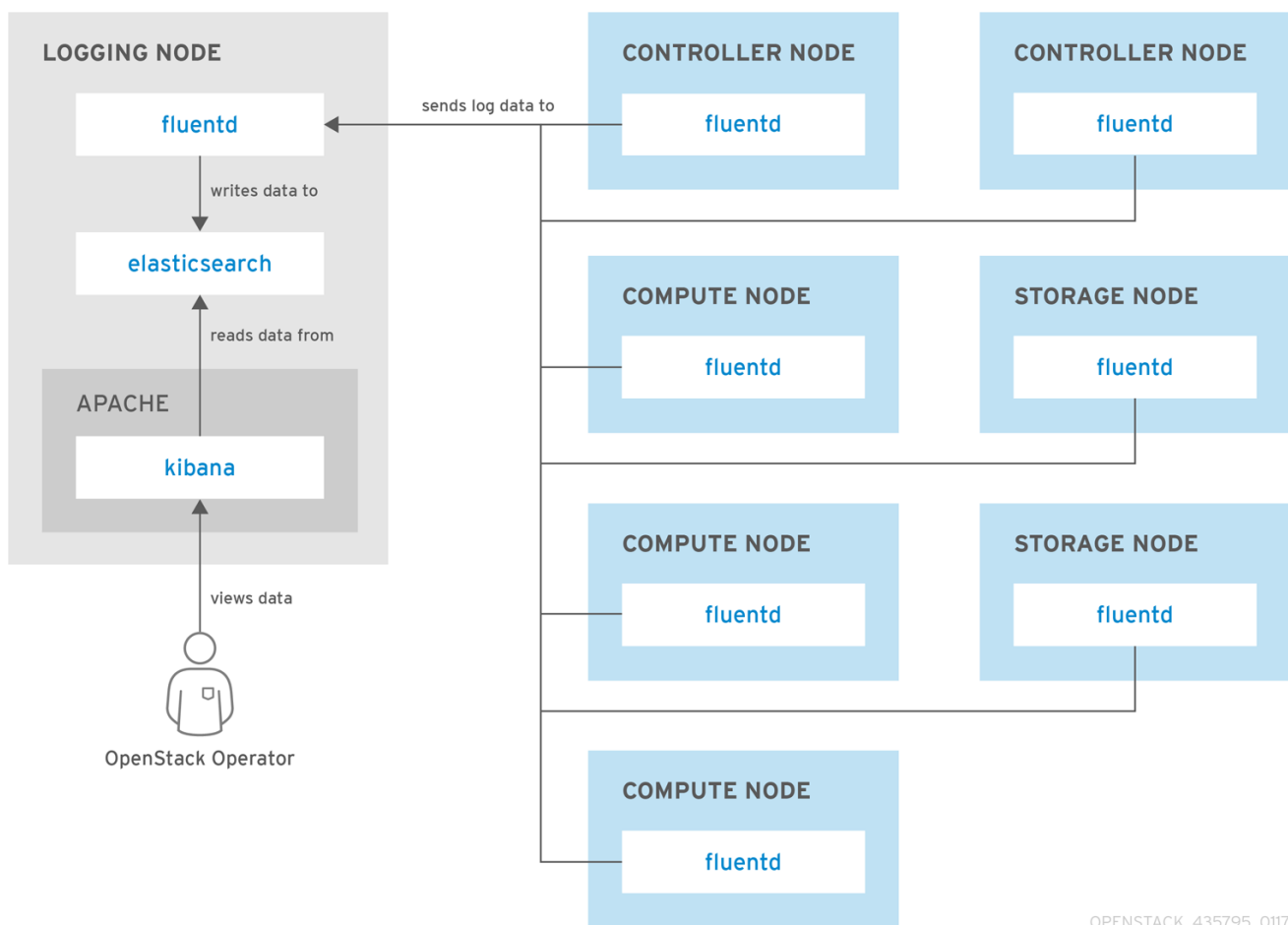
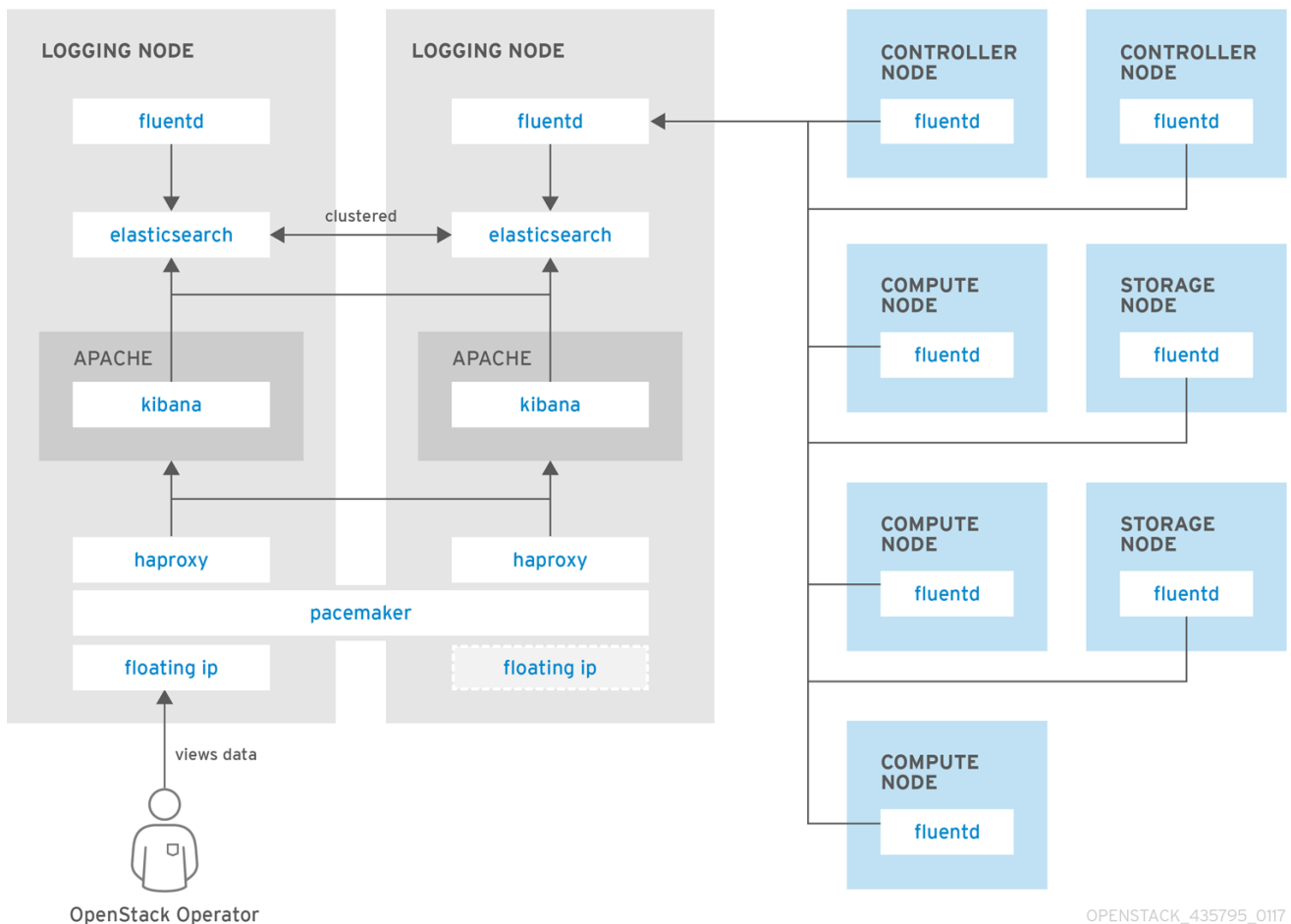


図12.3 Red Hat OpenStack Platform の HA デプロイメント

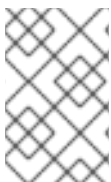


12.1.2. 可用性監視

可用性監視により、OpenStack 環境全体にわたる全コンポーネントのハイレベルな機能性を一元的に監視することができます。

可用性監視のツールチェーンは、以下を含む複数のコンポーネントで構成されます。

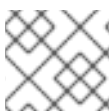
- 監視エージェント (Sensu クライアント)
- 監視リレー/プロキシー (RabbitMQ)
- 監視コントローラー/サーバー (Sensu サーバー)
- API/プレゼンテーション層 (Uchiwa)



注記

`director` はサーバー側の可用性監視のコンポーネントはデプロイしません。Red Hat では、Uchiwa、Sensu Server、Sensu API + RabbitMQ、監視ノードを実行する Redis インスタンスなどのサーバー側のコンポーネントはサポートしていません。

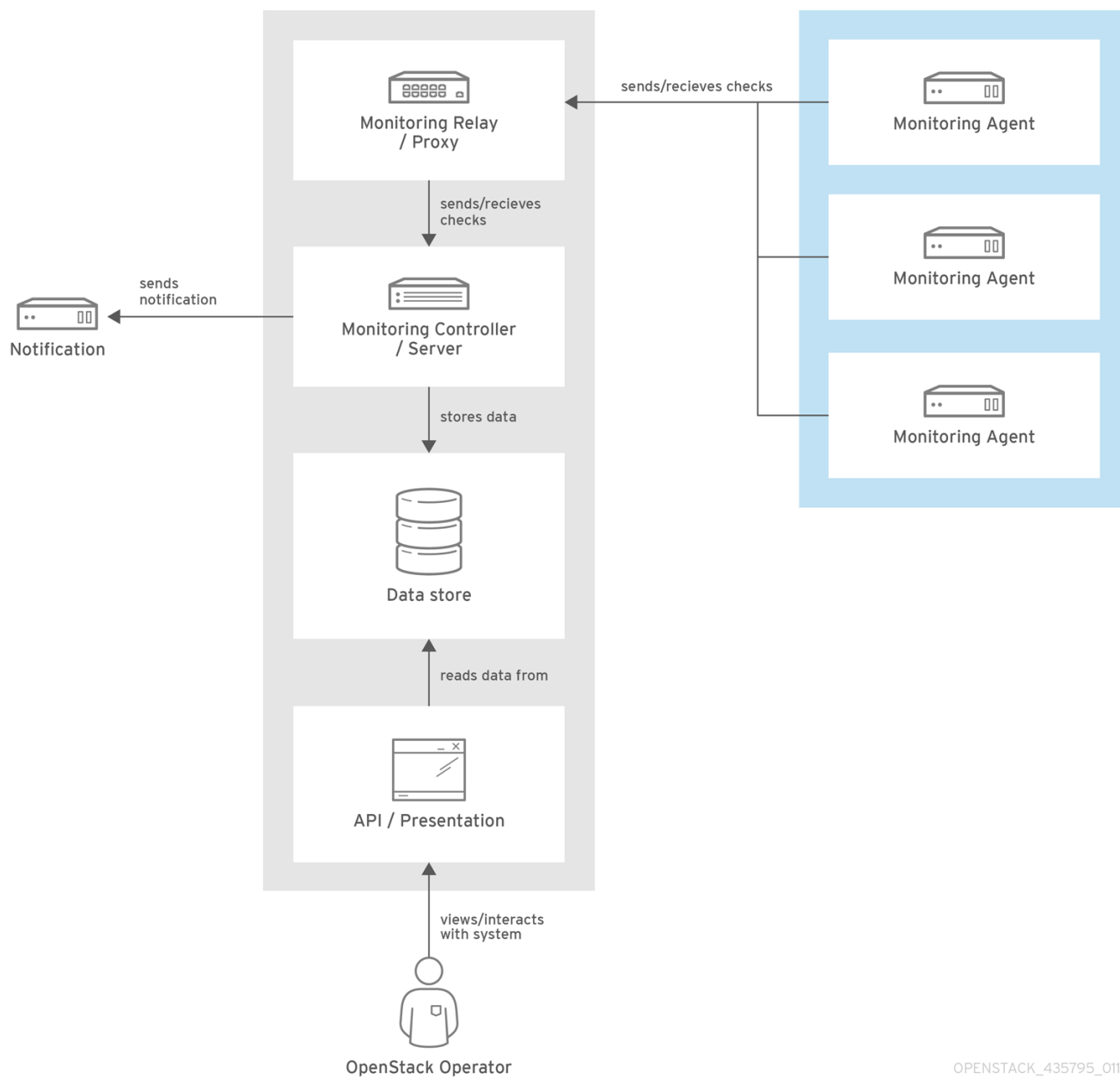
以下の図は、可用性監視のコンポーネントとそれらの対話を示しています。



注記

青で示した項目は Red Hat のサポート対象コンポーネントです。

図12.4 可用性監視のハイレベルアーキテクチャ



OPENSTACK_435795_0117

図12.5 Red Hat OpenStack Platform の単一ノードデプロイメント

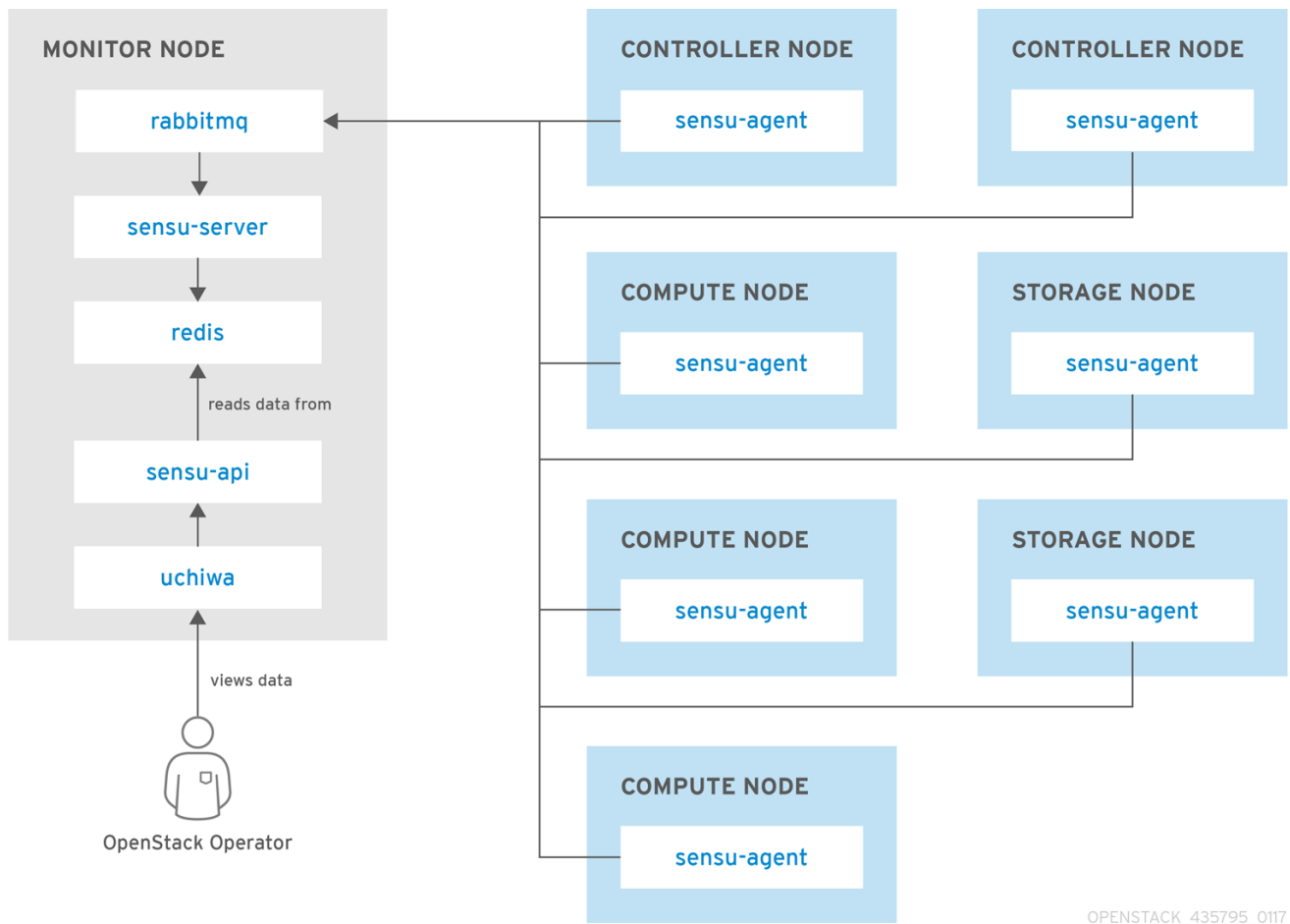
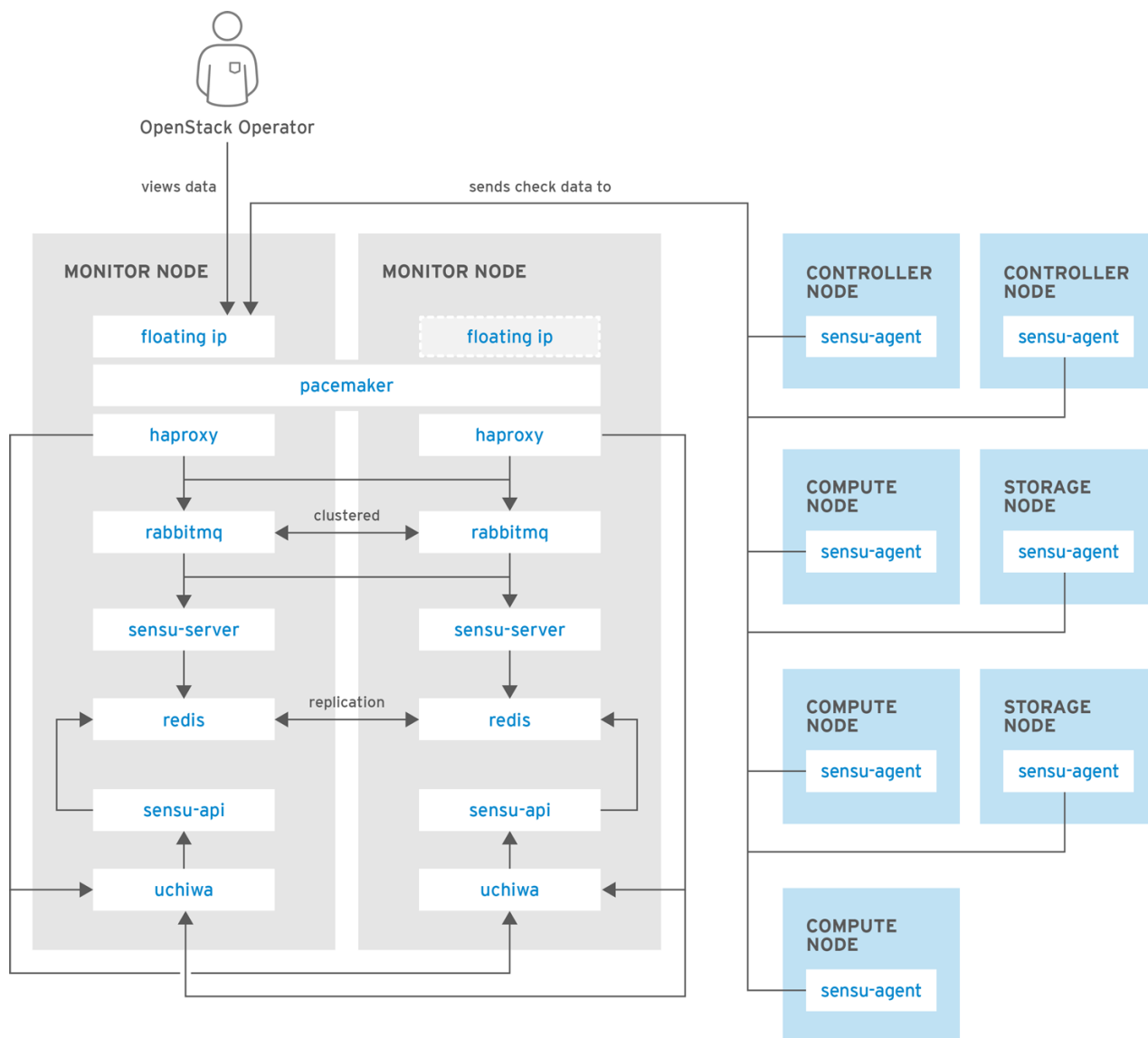


図12.6 Red Hat OpenStack Platform の HA デプロイメント



OPENSTACK_435795_0117

12.2. クライアント側のツールのインストール

オーバークラウドをデプロイする前には、各クライアントに適用する構成の設定を決定する必要があります。director の Heat テンプレートコレクションからサンプルの環境ファイルをコピーし、ご使用の環境に応じて変更します。

12.2.1. 集中ロギングのパラメーターの設定

Fluentd の構成設定には、`/usr/share/openstack-tripleo-heat-templates/environments/logging-environment.yaml` をコピーし、ご使用の環境に応じて変更します。以下に例を示します。

簡易設定

```
resource_registry:
  OS::TripleO::Services::FluentdClient:
    ../puppet/services/logging/fluentd-client.yaml
```



```
parameter_defaults:
  LoggingServers:
    - host: log0.example.com
      port: 24224
    - host: log1.example.com
      port: 24224
```

SSL の設定例

```
## (note the use of port 24284 for ssl connections)

resource_registry:
  OS::TripleO::Services::FluentdClient:
    ../puppet/services/logging/fluentd-client.yaml

parameter_defaults:
  LoggingServers:
    - host: 192.0.2.11
      port: 24284
  LoggingUsesSSL: true
  LoggingSharedKey: secret
  LoggingSSLCertificate: |
    -----BEGIN CERTIFICATE-----
    ...certificate data here...
    -----END CERTIFICATE-----
```

- **LoggingServers:** Fluentd ログメッセージの受信先のシステム
- **LoggingUsesSSL:** ログメッセージの送信時に**secure_forward**を使用するかどうかを決定する設定
- **LoggingSharedKey:** **secure_forward** が使用する共有シークレット
- **LoggingSSLCertificate:** PEM エンコードされた SSL CA 証明書の内容

12.2.2. 可用性監視クライアントのパラメーターの設定

Sensu クライアントの構成設定には、**/usr/share/openstack-tripleo-heat-templates/environments/monitoring-environment.yaml** をコピーし、ご使用の環境に応じて変更します。以下に例を示します。

```
resource_registry:
  OS::TripleO::Services::SensuClient: ../puppet/services/monitoring/sensu-client.yaml

parameter_defaults:
  MonitoringRabbitHost: 10.10.10.10
  MonitoringRabbitPort: 5672
  MonitoringRabbitUserName: sensu
  MonitoringRabbitPassword: sensu
  MonitoringRabbitUseSSL: false
  MonitoringRabbitVhost: "/sensu"
  SensuClientCustomConfig:
    api:
      warning: 10
```

critical: 20

- **MonitoringRabbit***: これらのパラメーターは、監視サーバーで実行する RabbitMQ インスタンスに Sensu クライアントサービスを接続します。
- **MonitoringRabbitUseSSL**: このパラメーターは、現在可用性監視には利用できません。
- **SensuClientCustomConfig**: Sensu クライアントの追加の設定を指定します。ユーザー名/パスワード、auth_url、テナント、リージョンを含む OpenStack の認証情報を定義します。

12.2.3. オーバークラウドノードへの運用ツールのインストール

openstack overcloud deploy コマンドで変更した YAML ファイルを指定して、Sensu クライアントと Fluentd ツールを全オーバークラウドノードにインストールします。以下に例を示します。

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml -e network-environment.yaml -e ~/templates/monitoring-environment.yaml -e ~/templates/logging-environment.yaml --control-scale 3 --compute-scale 1 -ntp-server 192.168.122.10
```

12.3. サーバー側のコンポーネントのインストール



注記

Red Hat では、サーバー側のコンポーネントおよびそれらのデプロイメントプロセスはサポートしていません。

opstools-ansible プレイブックを使用してサーバー側のコンポーネントを Red Hat Enterprise Linux 7 にインストールすることができます。これらのサーバー側のコンポーネントには、Red Hat がサポートするクライアント側のコンポーネントを補完する可用性監視や集中ロギングのサービスが含まれます。最も多く検証される **opstools-ansible** のシナリオは、サーバー側のコンポーネントを CentOS 7 にデプロイするシナリオです。詳しい手順については、<https://github.com/centos-opstools/opstools-ansible> で **README.md** を参照してください。

12.4. OPENSTACK PLATFORM の監視

Sensu のスタックインフラストラクチャーについて

は、<https://sensuapp.org/docs/latest/overview/architecture.html> の Sensu のドキュメントを参照してください。

Red Hat は、**osops-tools-monitoring-oschecks** パッケージで、check スクリプトのセットを提供しています。check スクリプトの大半は、OpenStack コンポーネントの API 接続のみをチェックしますが、特定のスクリプトは、OpenStack Compute (nova)、OpenStack Block Storage (cinder)、OpenStack Image (glance)、OpenStack Networking (neutron) を対象とする追加の OpenStack リソースのテストも実行します。たとえば、OpenStack Identity (keystone) API check は、**keystone** の実行時には以下のような結果を提示します。

```
OK: Got a token, Keystone API is working.
```

12.5. SENSU クライアントインストールの検証

1. オーバークラウドノードで **sensu-client** のステータスを確認します。

```
# systemctl status sensu-client
```

2. エラーログ (**/var/log/sensu/sensu-client.log**) で問題があるかどうかを確認します。
3. 各オーバークラウドノードに、監視サーバーの IP アドレスを設定する **/etc/sensu/conf.d/rabbitmq.json** ファイルがあることを確認します。

12.6. ノードの状態の確認

Uchiwa ダッシュボードがデプロイされている場合には、そのダッシュボードを Sensu サーバーと共に使用して、ノードの状態を確認することができます。

1. Uchiwa ダッシュボードにログインして、**Data Center** タブをクリックし、データセンターが稼働中であることを確認します。

```
http://<SERVER_IP_ADDRESS>:3000
```

2. 全オーバークラウドノードが **Connected** の状態であることを確認します。
3. オーバークラウドノードの 1 台を適時に再起動し、そのノードの状態を Uchiwa ダッシュボードで確認します。再起動の完了後には、ノードが Sense サーバーに正常に再接続されて check の実行を開始するかどうかを確認します。

12.7. OPENSTACK サービスの状態の確認

以下の例では、**openstack-ceilometer-central** サービスの監視をテストします。

1. **openstack-ceilometer-central** サービスが実行中であることを確認します。

```
systemctl status openstack-ceilometer-central.service
```

2. Uchiwa ダッシュボードに接続して、正常な **ceilometer** check があり、**ceilometer** JSON ファイルで定義されているとおりに実行されていることを確認します。
3. **openstack-ceilometer-central** サービスを停止します。



注記

これにより、サービスが中断される場合があるので、このテストは適切な時間に実行してください。

```
systemctl stop openstack-ceilometer-central.service
```

4. Uchiwa ダッシュボードにログインして、**ceilometer** check が失敗したことが報告されていることを確認します。
5. **openstack-ceilometer-central** サービスを起動します。

```
systemctl start openstack-ceilometer-central.service
```

6. Uchiwa ダッシュボードにログインして **ceilometer** check レポートの間隔を確認し、**ceilometer** JSON ファイルで定義されている間隔で check が実行されていることを確認します。

第13章 セキュリティーの強化

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

13.1. オーバークラウドのファイアウォールの管理

OpenStack Platform の各コアサービスには、それぞれのコンポーザブルサービステンプレートにファイアウォールルールが含まれています。これにより、各オーバークラウドノードにファイアウォールルールのデフォルトセットが自動的に作成されます。

オーバークラウドの Heat テンプレートには、追加のファイアウォール管理に役立つパラメーターのセットが含まれています。

ManageFirewall

ファイアウォールルールを自動管理するかどうかを定義します。**true** に設定すると、Puppet は各ノードでファイアウォールを自動的に設定することができます。ファイアウォールを手動で管理する場合には **false** に設定してください。デフォルトは **true** です。

PurgeFirewallRules

ファイアウォールルールを新規設定する前に、デフォルトの Linux ファイアウォールルールを完全削除するかどうかを定義します。デフォルトは **false** です。

ManageFirewall が **true** に設定されている場合には、デプロイメントに追加のファイアウォールルールを作成することができます。オーバークラウドの環境ファイルで、設定フックを使用して (「[Puppet: ロール用の 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。

13.2. 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 コミュニティ文字列が変更されます。

13.3. HAPROXY の SSL/TLS の暗号およびルールの変更

オーバークラウドで SSL/TLS を有効化した場合には (「[9章 オーバークラウドの SSL/TLS の有効化](#)」を参照)、HAProxy 設定を使用する SSL/TLS の暗号とルールを強化することをお勧めします。これにより、[POODLE TLS 脆弱性](#) などの SSL/TLS の脆弱性を回避することができます。

オーバークラウドの環境ファイルで **ExtraConfig** フックを使用して以下の hieradata を設定します。

tripleo::haproxy::ssl_cipher_suite

HAProxy で使用する暗号スイート

tripleo::haproxy::ssl_options

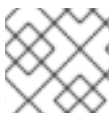
HAProxy で使用する SSL/TLS ルール

たとえば、以下のような暗号およびルールを使用することができます。

- Cipher: **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-ssl3 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-ssl3 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。

オーバークラウドの作成時にこの環境ファイルを追加します。

第14章 その他の設定

14.1. 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『[オーバークラウド向けの外部のロードバランシング](#)』ガイドを参照してください。

14.2. 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『[オーバークラウド向けの IPv6 ネットワーク](#)』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	内部 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	ストレージネットワークのネットワークおよびサブネット	
StorageMgmtNetCidr	ストレージ管理ネットワークのネットワークのおよびサブネット	
TenantNetCidr	テナントネットワークのネットワークおよびサブネット	
ExternalNetCidr	外部ネットワークのネットワークおよびサブネット	
InternalApiAllocationPools	内部 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	ストレージネットワークの割り当てプール (タプル形式)	
StorageMgmtAllocationPools	ストレージ管理ネットワークの割り当てプール (タプル形式)	
TenantAllocationPools	テナントネットワークの割り当てプール (タプル形式)	
ExternalAllocationPools	外部ネットワークの割り当てプール (タプル形式)	
InternalApiNetworkVlanID	内部 API ネットワークの VLAN ID	200
StorageNetworkVlanID	ストレージネットワークの VLAN ID	
StorageMgmtNetworkVlanID	ストレージ管理ネットワークの VLAN ID	
TenantNetworkVlanID	テナントネットワークの VLAN ID	
ExternalNetworkVlanID	外部ネットワークの VLAN ID	

パラメーター	説明	例
ExternalInterfaceDefaultRoute	外部ネットワークのゲートウェイ IP アドレス	10.1.2.1
ControlPlaneDefaultRoute	プロビジョニング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	プロビジョニングネットワーク用の 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。外部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 のテナントネットワーク種別	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。

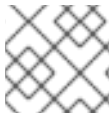
B.2. ルートおよびデフォルトルートの設定

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用されます。

静的な IP が指定されたインターフェースに静的なルートを設定するには、サブネットにルートを指定します。たとえば、内部 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、外部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: {get_input: 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。外部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、ストレージネットワークは全ロールにアタッチされているため、ストレージ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、ストレージ、ストレージ管理、内部 API、テナント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、テナント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15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。

15.1. 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

15.2. 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

15.3. 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

15.4. 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

15.5. 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

15.6. 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 ボンディングで使うことができます。以下に例を示します。

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

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

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。