



# Red Hat OpenStack Platform 9

## **director** のインストールと使用方法

Red Hat OpenStack Platform director を使用した OpenStack クラウド作成のエンド  
ツーエンドシナリオ



# Red Hat OpenStack Platform 9 director のインストールと使用方法

---

Red Hat OpenStack Platform director を使用した OpenStack クラウド作成のエンドツーエンドシナリオ

OpenStack Team  
rhos-docs@redhat.com

## 法律上の通知

Copyright © 2018 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## 概要

本ガイドでは、エンタープライズ環境で Red Hat OpenStack Platform director を使用して Red Hat OpenStack Platform 9 をインストールする方法について説明します。これには、director のインストール、環境のプランニング、director を使用した OpenStack 環境の構築などが含まれます。

## 目次

<b>第1章 はじめに</b>	<b>6</b>
1.1. アンダークラウド	6
1.2. オーバークラウド	7
1.3. 高可用性	8
1.4. CEPH STORAGE	9
<b>第2章 要件</b>	<b>10</b>
2.1. 環境要件	10
2.2. アンダークラウドの要件	10
2.3. ネットワーク要件	11
2.4. オーバークラウドの要件	13
2.4.1. コンピュートノードの要件	13
2.4.2. コントローラーノードの要件	14
2.4.3. Ceph Storage ノードの要件	15
2.4.4. Object Storage ノードの要件	16
2.5. リポジトリの要件	16
<b>第3章 オーバークラウドのプランニング</b>	<b>18</b>
3.1. ノードのデプロイメントロールのプランニング	18
3.2. ネットワークのプランニング	19
3.3. ストレージのプランニング	24
<b>第4章 アンダークラウドのインストール</b>	<b>26</b>
4.1. DIRECTOR のインストールユーザの作成	26
4.2. テンプレートとイメージ用のディレクトリーの作成	26
4.3. システムのホスト名設定	26
4.4. システムの登録	27
4.5. DIRECTOR パッケージのインストール	28
4.6. DIRECTOR の設定	28
4.7. オーバークラウドノードのイメージの取得	31
4.8. アンダークラウドの NEUTRON サブネットでのネームサーバーの設定	32
4.9. アンダークラウドのバックアップ	32
4.10. アンダークラウドの設定完了	32
<b>第5章 基本的なオーバークラウド要件の設定</b>	<b>33</b>
5.1. オーバークラウドへのノードの登録	33
5.2. ノードのハードウェアの検査	35
5.3. プロファイルへのノードのタグ付け	36
5.4. ノードのルートディスクの定義	36
5.5. 基本設定の完了	38
<b>第6章 オーバークラウドの高度なカスタマイズ設定</b>	<b>40</b>
6.1. HEAT テンプレートの理解	40
6.1.1. Heat テンプレート	40
6.1.2. 環境ファイル	41
6.1.3. コアとなるオーバークラウドの Heat テンプレート	42
6.2. ネットワークの分離	42
6.2.1. カスタムのインターフェーステンプレートの作成	43
6.2.2. ネットワーク環境ファイルの作成	48
6.2.3. OpenStack サービスの分離ネットワークへの割り当て	50
6.2.4. デプロイするネットワークの選択	51
6.3. ノード配置の制御	55
6.3.1. 特定のノード ID の割り当て	56

6.3.2. カスタムのホスト名の割り当て	57
6.3.3. 予測可能な IP の割り当て	57
6.3.4. 予測可能な仮想 IP の割り当て	59
6.4. コンテナ化されたコンピュートノードの設定	59
6.4.1. 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	60
6.4.2. Atomic Host のイメージのアップロード	61
6.4.3. ローカルのレジストリーの使用	61
6.4.4. 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	63
6.5. 外部の負荷分散機能の設定	64
6.6. IPV6 ネットワークの設定	64
6.7. NFS ストレージの設定	64
6.8. CEPH STORAGE の設定	66
6.9. サードパーティーのストレージの設定	66
6.10. オーバークラウドの SSL/TLS の有効化	67
6.10.1. SSL/TLS の有効化	67
6.10.2. ルート証明書の注入	69
6.10.3. DNS エンドポイントの設定	69
6.10.4. オーバークラウド作成時の環境ファイルの追加	70
6.11. ベースパラメーターの設定	70
6.12. オーバークラウドの登録	71
6.12.1. 方法 1: コマンドライン	71
6.12.2. 方法 2: 環境ファイル	72
6.13. 初回起動での設定のカスタマイズ	73
6.14. オーバークラウドの設定前のカスタマイズ	74
6.15. オーバークラウドの設定後のカスタマイズ	76
6.16. PUPPET 設定データのカスタマイズ	78
6.17. カスタムの PUPPET 設定の適用	79
6.18. カスタムのコア HEAT テンプレートの使用	80
<b>第7章 オーバークラウドの作成</b>	<b>81</b>
7.1. オーバークラウドのパラメーター設定	81
7.2. オーバークラウド作成時の環境ファイルの追加	86
7.3. オーバークラウドの作成例	87
7.4. オーバークラウド作成の監視	88
7.5. オーバークラウドへのアクセス	88
7.6. オーバークラウド作成の完了	89
<b>第8章 オーバークラウド作成後のタスクの実行</b>	<b>90</b>
8.1. オーバークラウドのテナントネットワークの作成	90
8.2. オーバークラウドの外部ネットワークの作成	90
8.3. 追加の FLOATING IP ネットワークの作成	91
8.4. オーバークラウドのプロバイダーネットワークの作成	92
8.5. オーバークラウドの検証	93
8.6. コントローラーノードのフェンシング	95
8.7. オーバークラウド環境の変更	97
8.8. オーバークラウドへの仮想マシンのインポート	98
8.9. オーバークラウドのコンピュートノードからの仮想マシンの移行	98
8.10. オーバークラウドが削除されてないように保護	100
8.11. オーバークラウドの削除	100
<b>第9章 オーバークラウドのスケーリング</b>	<b>101</b>
9.1. ノードのさらなる追加	101
9.2. コンピュートノードの削除	103
9.3. コンピュートノードの置き換え	104

9.4. コントローラーノードの置き換え	105
9.4.1. 事前のチェック	105
9.4.2. ノードの置き換え	107
9.4.3. 手動での介入	110
9.4.4. オーバークラウドサービスの最終処理	116
9.4.5. L3 エージェントのルーターホスティングの最終処理	116
9.4.6. Compute サービスの最終処理	117
9.4.7. 結果	118
9.5. CEPH STORAGE ノードの置き換え	118
9.6. OBJECT STORAGE ノードの置き換え	118
<b>第10章 オーバークラウドのリブート</b>	<b>122</b>
10.1. DIRECTOR の再起動	122
10.2. コントローラーノードの再起動	122
10.3. CEPH STORAGE ノードの再起動	123
10.4. コンピュートノードの再起動	124
10.5. OBJECT STORAGE ノードの再起動	125
<b>第11章 DIRECTOR の問題のトラブルシューティング</b>	<b>126</b>
11.1. ノード登録のトラブルシューティング	126
11.2. ハードウェアイントロスペクションのトラブルシューティング	126
11.3. オーバークラウドの作成のトラブルシューティング	129
11.3.1. Orchestration	129
11.3.2. Bare Metal Provisioning	129
11.3.3. デプロイメント後の設定	130
11.4. プロビジョニングネットワークでの IP アドレスの競合に対するトラブルシューティング	132
11.5. "NO VALID HOST FOUND" エラーのトラブルシューティング	133
11.6. オーバークラウド作成後のトラブルシューティング	134
11.6.1. オーバークラウドスタックの変更	134
11.6.2. コントローラーサービスのエラー	135
11.6.3. Compute サービスのエラー	135
11.6.4. Ceph Storage サービスのエラー	136
11.7. アンダークラウドの調整	136
11.8. アンダークラウドとオーバークラウドの重要なログ	137
<b>付録A SSL/TLS 証明書の設定</b>	<b>139</b>
A.1. 署名ホストの初期化	139
A.2. 認証局の作成	139
A.3. クライアントへの認証局の追加	139
A.4. SSL/TLS キーの作成	139
A.5. SSL/TLS 証明書署名要求の作成	140
A.6. SSL/TLS 証明書の作成	141
A.7. アンダークラウドで証明書を使用する場合	142
A.8. オーバークラウドで証明書を使用する場合	142
<b>付録B 電源管理ドライバー</b>	<b>143</b>
B.1. DELL REMOTE ACCESS CONTROLLER (DRAC)	143
B.2. INTEGRATED LIGHTS-OUT (ILO)	143
B.3. CISCO UNIFIED COMPUTING SYSTEM (UCS)	143
B.4. FUJITSU INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC)	144
B.5. SSH と VIRSH	145
B.6. フェイク PXE ドライバー	145
<b>付録C プロファイルの自動タグ付け</b>	<b>147</b>

C.1. ポリシーファイルの構文	147
C.2. ポリシーファイルの例	149
C.3. ポリシーファイルのインポート	150
C.4. プロファイルの自動タグ付けのプロパティ	151
<b>付録D 基本パラメーター</b>	<b>152</b>
<b>付録E ネットワークインターフェースのパラメーター</b>	<b>169</b>
E.1. インターフェースのオプション	169
E.2. VLAN のオプション	169
E.3. OVS ボンディングのオプション	170
E.4. OVS ブリッジのオプション	171
E.5. LINUX ボンディングのオプション	172
E.6. LINUX BRIDGE のオプション	173
<b>付録F ネットワークインターフェースのテンプレート例</b>	<b>175</b>
F.1. インターフェースの設定	175
F.2. ルートおよびデフォルトルートの設定	176
F.3. FLOATING IP のためのネイティブ VLAN の使用	176
F.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	177
F.5. ジャンボフレームの設定	177
<b>付録G ネットワーク環境のオプション</b>	<b>179</b>
<b>付録H OPEN VSWITCH ボンディングのオプション</b>	<b>182</b>

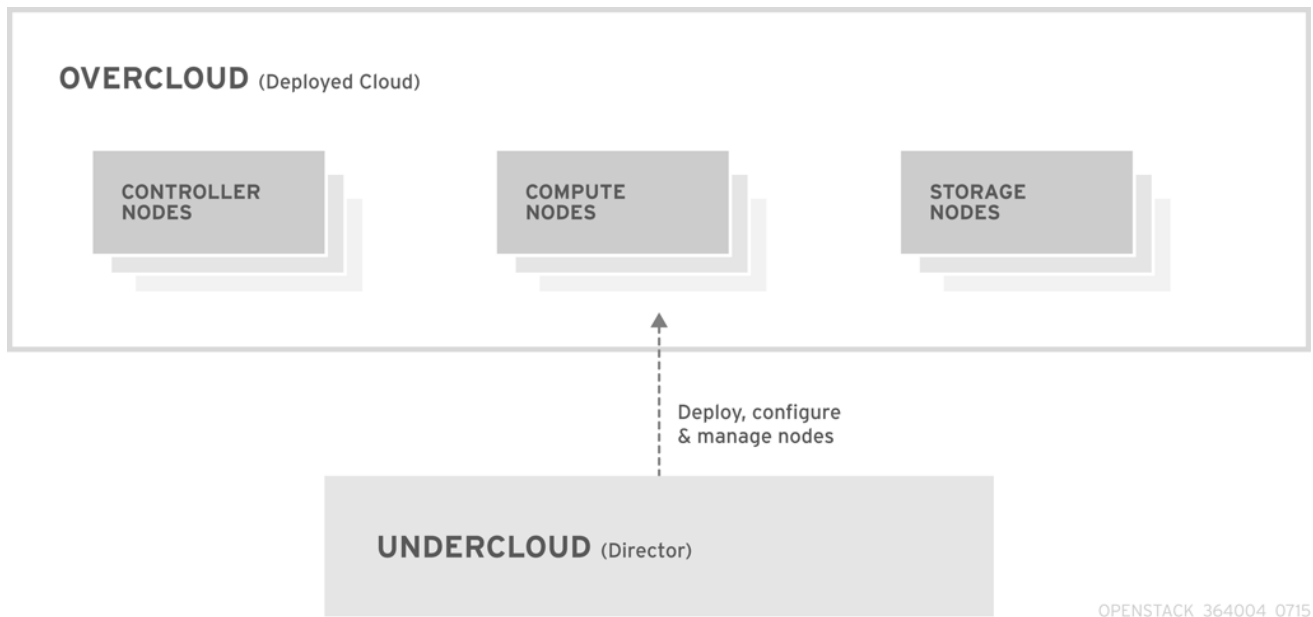




## 第1章 はじめに

Red Hat OpenStack Platform director は、完全な OpenStack 環境をインストールおよび管理するためのツールセットで、OpenStack のプロジェクト TripleO (OpenStack-On-OpenStack の略) をベースとしています。このプロジェクトは、OpenStack のコンポーネントを活用して、完全に機能する OpenStack 環境をインストールします。これには、OpenStack ノードとして使用するベアメタルシステムのプロビジョニングや制御を行う OpenStack のコンポーネントが含まれます。director により、効率的で堅牢性の高い、完全な Red Hat OpenStack Platform 環境を簡単にインストールできます。

Red Hat OpenStack Platform director は、アンダークラウドとオーバークラウドという 2 つの主要な概念を採用しています。以下の数項では、それぞれの概念について説明します。



### 1.1. アンダークラウド

アンダークラウドは、director の主要ノードで、OpenStack をインストールした単一システムです。このノードには、OpenStack 環境 (オーバークラウド) を構成する OpenStack ノードのプロビジョニング/管理のためのコンポーネントが含まれます。アンダークラウドを形成するコンポーネントは、以下の機能を提供します。

- 環境プランニング: アンダークラウドは、コンピュート、コントローラー、各種ストレージロールなどの Red Hat OpenStack Platform ロールを割り当てるプランニング機能を提供します。
- ベアメタルシステムの制御: アンダークラウドは、各ノードの Intelligent Platform Management Interface (IPMI) を使用して電源管理機能を制御し、PXE ベースのサービスを使用してハードウェア属性を検出し、各ノードに OpenStack をインストールします。この機能により、ベアメタルシステムを OpenStack ノードとしてプロビジョニングする方法が提供されます。
- オークストレーション: アンダークラウドは、OpenStack 環境を構築するための YAML テンプレートセットの提供および読み込みを行います。

Red Hat OpenStack Platform director は、ターミナルベースのコマンドラインインターフェースで、これらのアンダークラウド機能を実行します。

アンダークラウドは、以下のコンポーネントで構成されます。

- OpenStack Bare Metal (Ironic) および OpenStack Compute (Nova): ベアメタルノードの管理

- OpenStack Networking (Neutron) および Open vSwitch: ベアメタルノードのネットワークの制御
- OpenStack Image サービス (Glance): ベアメタルマシンへ書き込むイメージの格納
- OpenStack Orchestration (Heat) および Puppet: director がオーバークラウドイメージをディスクに書き込んだ後のノードのオーケストレーションおよび設定
- OpenStack Telemetry (Ceilometer): 監視とデータの収集。これには、以下が含まれます。
  - OpenStack Telemetry Metrics (gnocchi): メトリック向けの時系列データベース
  - OpenStack Telemetry Alarming (aodh): モニタリング向けのアラームコンポーネント
- OpenStack Identity (keystone): director のコンポーネントの認証および承認
- OpenStack Object Storage (swift): 以下のさまざまな OpenStack Platform のコンポーネントに対してオブジェクトストレージを提供します。
  - OpenStack Image サービスのイメージストレージ
  - OpenStack Bare Metal のイントロスペクションデータ
  - OpenStack Workflow サービスのデプロイメントプラン

## 1.2. オーバークラウド

オーバークラウドは、アンダークラウドを使用して構築した Red Hat OpenStack Platform 環境で、以下のノード種別の 1 つまたは複数で構成されます。

### コントローラー

OpenStack 環境に管理、ネットワーク、高可用性の機能を提供するノード。理想的な OpenStack 環境には、このノード 3 台で高可用性クラスターを構成することを推奨します。デフォルトのコントローラーノードには、以下のコンポーネントが含まれます。

- OpenStack Dashboard (horizon)
- OpenStack Identity (keystone)
- OpenStack Compute (nova) API
- OpenStack Networking (neutron)
- OpenStack Image サービス (glance)
- OpenStack Block Storage (cinder)
- OpenStack Object Storage (swift)
- OpenStack Orchestration (heat)
- OpenStack Telemetry (ceilometer)
- OpenStack Telemetry Metrics (gnocchi)
- OpenStack Telemetry Alarming (aodh)

- OpenStack Clustering (sahara)
- MariaDB
- Open vSwitch
- 高可用性サービス向けの Pacemaker および Galera

## Compute

これらのノードは OpenStack 環境にコンピュートリソースを提供します。コンピュートノードをさらに追加して、環境を徐々にスケールアウトすることができます。デフォルトのコンピュートノードには、以下のコンポーネントが含まれます。

- OpenStack Compute (nova)
- KVM/QEMU
- OpenStack Telemetry (ceilometer) エージェント
- Open vSwitch

## ストレージ

OpenStack 環境にストレージを提供するノード。これには、以下のストレージ用のノードが含まれます。

- Ceph Storage ノード: ストレージクラスターを構成するために使用します。各ノードには、Ceph Object Storage Daemon (OSD) が含まれており、Ceph Storage ノードをデプロイする場合には、director により Ceph Monitor がコンピュートノードにインストールされます。
- Block storage (Cinder): HA コントローラーノードの外部ブロックストレージとして使用します。このノードには、以下のコンポーネントが含まれます。
  - OpenStack Block Storage (cinder) ボリューム
  - OpenStack Telemetry (ceilometer) エージェント
  - Open vSwitch.
- Object storage (swift): これらのノードは、OpenStack Swift の外部ストレージ層を提供します。コントローラーノードは、Swift プロキシを介してこれらのノードにアクセスします。このノードには、以下のコンポーネントが含まれます。
  - OpenStack Object Storage (swift) のストレージ
  - OpenStack Telemetry (ceilometer) エージェント
  - Open vSwitch.

## 1.3. 高可用性

Red Hat OpenStack Platform director は、OpenStack Platform 環境に高可用性サービスを提供するためにコントローラーノードクラスターを使用します。director は、各コントローラーノードにコンポーネントの複製セットをインストールし、それらをまとめて単一のサービスとして管理します。このタイプのクラスター構成では、1つのコントローラーノードが機能しなくなった場合にフォールバックするので、OpenStack のユーザーには一定の運用継続性が提供されます。

OpenStack Platform director は、複数の主要なソフトウェアを使用して、コントローラーノード上のコンポーネントを管理します。

- Pacemaker: Pacemaker はクラスターリソースマネージャーで、クラスター内の全ノードにおける OpenStack コンポーネントの可用性を管理/監視します。
- HA Proxy: クラスターに負荷分散およびプロキシサービスを提供します。
- Galera: クラスター全体の OpenStack Platform データベースを複製します。
- Memcached: データベースのキャッシュを提供します。



#### 注記

Red Hat OpenStack Platform director は複数のコントローラーノードの高可用性を一括に自動設定します。ただし、フェンシングや電源管理制御を有効化するには、ノードを手動で設定する必要があります。本ガイドでは、これらの手順を記載しています。

## 1.4. CEPH STORAGE

一般的に、OpenStack を使用する大規模な組織では、数千単位のクライアントにサービスを提供します。OpenStack クライアントは、ブロックストレージリソースを消費する際には、それぞれに固有のニーズがある可能性が高く、Glance (イメージ)、Cinder (ボリューム)、Nova (コンピュート) を単一ノードにデプロイすると、数千単位のクライアントがある大規模なデプロイメントで管理することができなくなります。このような課題は、OpenStack をスケールアウトすることによって解決できます。

ただし、実際には、Red Hat Ceph Storage などのソリューションを活用して、ストレージ層を仮想化する必要もでてきます。ストレージ層の仮想化により、Red Hat OpenStack Platform のストレージ層を数十テラバイト規模からペタバイトさらにはエクサバイトのストレージにスケールアップすることが可能です。Red Hat Ceph Storage は、市販のハードウェアを使用しながらも、高可用性/高パフォーマンスのストレージ仮想化層を提供します。仮想化によってパフォーマンスが低下するというイメージがありますが、Ceph はブロックデバイスイメージをクラスター全体でオブジェクトとしてストライプ化するため、大きい Ceph のブロックデバイスイメージはスタンドアロンのディスクよりもパフォーマンスが優れているということになります。Ceph ブロックデバイスでは、パフォーマンスを強化するために、キャッシュ、Copy On Write クローン、Copy On Read クローンもサポートされています。

Red Hat Ceph Storage に関する情報は、[Red Hat Ceph Storage](#) を参照してください。

## 第2章 要件

本章では、director を使用して Red Hat OpenStack Platform をプロビジョニングする環境をセットアップするための主要な要件を記載します。これには、director のセットアップ/アクセス要件や OpenStack サービス用に director がプロビジョニングするホストのハードウェア要件が含まれます。



### 注記

Red Hat OpenStack Platform をデプロイする前には、利用可能なデプロイメソッドの特性を考慮することが重要です。詳しくは、「[Red Hat OpenStack Platform のインストールにおける推奨されるベストプラクティス](#)」を参照してください。

### 2.1. 環境要件

#### 最小要件

- Red Hat OpenStack Platform director 用のホストマシン 1 台
- Red Hat OpenStack Platform コンピュートノード用のホストマシン 1 台
- Red Hat OpenStack Platform コントローラーノード用のホストマシン 1 台

#### 推奨要件

- Red Hat OpenStack Platform director 用のホストマシン 1 台
- Red Hat OpenStack Platform コンピュートノード用のホストマシン 3 台
- Red Hat OpenStack Platform コントローラーノード用のホストマシン 3 台
- クラスター内に Red Hat Ceph Storage ノード用のホストマシン 3 台

以下の点に注意してください。

- 全ノードにはベアメタルシステムを使用することを推奨します。最低でも、コンピュートノードにはベアメタルシステムが必要です。
- director は電源管理制御を行うため、オーバークラウドのベアメタルシステムにはすべて、Intelligent Platform Management Interface (IPMI) が必要です。

### 2.2. アンダークラウドの要件

director をホストするアンダークラウドシステムは、オーバークラウド内の全ノードのプロビジョニングおよび管理を行います。

- Intel 64 または AMD64 CPU 拡張機能をサポートする、8 コア 64 ビット x86 プロセッサー
- 最小 16 GB の RAM
- ルートディスク上に最小 40 GB のディスク領域。オーバークラウドのデプロイまたは更新を試みる前には、空き領域が少なくとも 10 GB あることを確認してください。この空き領域は、イメージの変換やノードのプロビジョニングプロセスのキャッシュに使用されます。

- 最小 2 枚の 1 Gbps ネットワークインターフェースカード。ただし、特にオーバークラウド環境で多数のノードをプロビジョニングする場合には、ネットワークトラフィックのプロビジョニング用に 10 Gbps インターフェースを使用することを推奨します。
- ホストのオペレーティングシステムに Red Hat Enterprise Linux 7.2 (以降) がインストール済みであること



### 重要

論理ボリューム管理 (LVM) を使用する場合には、アンダークラウドのファイルシステムに root パーティションと swap パーティションのみが含まれるようにしてください。詳しくは、Red Hat カスタマーポータル[の「Director node fails to boot after undercloud installation」](#)を参照してください。

## 2.3. ネットワーク要件

アンダークラウドのホストには、最低でも 2 つのネットワークが必要です。

- プロビジョニングネットワーク: オーバークラウドで使用するベアメタルシステムの検出がしやすくなるように、DHCP および PXE ブート機能を提供します。このネットワークは通常、director が PXE ブートおよび DHCP の要求に対応できるように、トランキングされたインターフェースでネイティブ VLAN を使用する必要があります。一部のサーバーのハードウェアの BIOS は、VLAN からの PXE ブートをサポートしていますが、その BIOS が、ブート後に VLAN をネイティブ VLAN に変換する機能もサポートする必要があります。この機能がサポートされていない場合には、アンダークラウドに到達できません。現在この機能を完全にサポートしているサーバーハードウェアはごく一部です。プロビジョニングネットワークは、オーバークラウドノード上で Intelligent Platform Management Interface (IPMI) により電源管理を制御するのに使用するネットワークでもあります。
- 外部ネットワーク: 全ノードへのリモート接続に使用する別個のネットワーク。このネットワークに接続するこのインターフェースには、静的または外部の DHCP サービス経由で動的に定義された、ルーティング可能な IP アドレスが必要です。

これは、必要なネットワークの最小数を示します。ただし、director は他の Red Hat OpenStack Platform ネットワークトラフィックをその他のネットワーク内に分離することができます。Red Hat OpenStack Platform は、ネットワークの分離に物理インターフェースとタグ付けされた VLAN の両方をサポートしています。ネットワークの分離に関する詳しい情報は、[「ネットワークの分離」](#)を参照してください。

以下の点に注意してください。

- 標準的な最小限のオーバークラウドのネットワーク構成には、以下が含まれます。
  - シングル NIC 構成: ネイティブの VLAN および異なる種別のオーバークラウドネットワークのサブネットを使用するタグ付けされた VLAN 上にプロビジョニングネットワーク用の NIC を 1 つ。
  - デュアル NIC 構成: プロビジョニングネットワーク用の NIC を 1 つと、外部ネットワーク用の NIC を 1 つ。
  - デュアル NIC 構成: ネイティブの VLAN 上にプロビジョニングネットワーク用の NIC を 1 つと、異なる種別のオーバークラウドネットワークのサブネットを使用するタグ付けされた VLAN 用の NIC を 1 つ。
  - 複数 NIC 構成: 各 NIC は、異なる種別のオーバークラウドネットワークのサブセットを使用します。

- 追加の物理 NIC は、個別のネットワークの分離、ボンディングインターフェースの作成、タグ付けされた VLAN トラフィックの委譲に使用することができます。
- ネットワークトラフィックの種別を分離するのに VLAN を使用している場合には、802.1Q 標準をサポートするスイッチを使用してタグ付けされた VLAN を提供します。
- オーバークラウドの作成時には、全オーバークラウドマシンで 1 つの名前を使用して NIC を参照します。理想としては、混乱を避けるため、対象のネットワークごとに、各オーバークラウドノードで同じ NIC を使用してください。たとえば、プロビジョニングネットワークにはプライマリー NIC を使用して、OpenStack サービスにはセカンダリー NIC を使用します。
- プロビジョニングネットワークの NIC は director マシン上でリモート接続に使用する NIC とは異なります。director のインストールでは、プロビジョニング NIC を使用してブリッジが作成され、リモート接続はドロップされます。director システムへリモート接続する場合には、外部 NIC を使用します。
- プロビジョニングネットワークには、環境のサイズに適した IP 範囲が必要です。以下のガイドラインを使用して、この範囲に含めるべき IP アドレスの総数を決定してください。
  - プロビジョニングネットワークに接続されているノード 1 台につき最小で 1 IP アドレスを含めます。
  - 高可用性を設定する予定がある場合には、クラスターの仮想 IP 用に追加の IP アドレスを含めます。
  - 環境のスケーリング用の追加の IP アドレスを範囲に追加します。



#### 注記

プロビジョニングネットワーク上で IP アドレスが重複するのを避ける必要があります。詳しい説明は、[「ネットワークのプランニング」](#)を参照してください。



#### 注記

ストレージ、プロバイダー、テナントネットワークの IP アドレスの使用範囲をプランニングすることに関する情報は、[『ネットワークガイド』](#)を参照してください。

- すべてのオーバークラウドシステムをプロビジョニング NIC から PXE ブートするように設定して、同システム上の外部 NIC およびその他の NIC の PXE ブートを無効にします。また、プロビジョニング NIC の PXE ブートは、ハードディスクや CD/DVD ドライブよりも優先されるように、起動順序の最上位に指定します。
- オーバークラウドのベアメタルシステムにはすべて、Intelligent Platform Management Interface (IPMI) などのサポート対象の電源管理インターフェースが必要です。このインターフェースにより、director は各ノードの電源管理を制御することが可能となります。
- 各オーバークラウドシステムの詳細 (プロビジョニング NIC の MAC アドレス、IPMI NIC の IP アドレス、IPMI ユーザー名、IPMI パスワード) をメモしてください。この情報は、後ほどオーバークラウドノードの設定時に役立ちます。
- インスタンスが外部のインターネットからアクセス可能である必要がある場合には、パブリックネットワークから Floating IP アドレスを割り当てて、そのアドレスをインスタンスに関連付けます。インスタンスは、引き続きプライベートの IP アドレスを確保しますが、ネットワークトラフィックは NAT を使用して、Floating IP アドレスに到達します。Floating IP アドレスは、



複数のプライベート IP アドレスではなく、単一のインスタンスにのみ割り当て可能である点に注意してください。ただし、Floating IP アドレスは、単一のテナントで使用するよう確保され、そのテナントは必要に応じて特定のインスタンスに関連付け/関連付け解除することができます。この構成を使用すると、インフラストラクチャーが外部のインターネットに公開されるので、適切なセキュリティープラクティスを順守しているかどうかを確認する必要があります。

- 1 つのブリッジには単一のインターフェースまたは単一のボンディングのみをメンバーにすると、Open vSwitch でネットワークループが発生するリスクを緩和することができます。複数のボンディングまたはインターフェースが必要な場合には、複数のブリッジを設定することが可能です。
- オーバークラウドノードが Red Hat Content Delivery Network やネットワークタイムサーバーなどの外部のサービスに接続できるようにするには、DNS によるホスト名解決を使用することを推奨します。

## 重要

OpenStack Platform の実装のセキュリティーレベルは、その環境のセキュリティーレベルと同等です。ネットワーク環境内の適切なセキュリティー原則に従って、ネットワークアクセスが正しく制御されるようにします。以下に例を示します。

- ネットワークのセグメント化を使用して、ネットワークトラフィックを軽減し、機密データを分離します。フラットなネットワークはセキュリティーレベルがはるかに低くなります。
- サービスアクセスとポートを最小限に制限します。
- 適切なファイアウォールルールとパスワードが使用されるようにします。
- SELinux が有効化されていることを確認します。

システムのセキュリティー保護については、以下のドキュメントを参照してください。

- [『Red Hat Enterprise Linux 7 セキュリティーガイド』](#)
- [『Red Hat Enterprise Linux 7 SELinux ユーザーおよび管理者のガイド』](#)

## 2.4. オーバークラウドの要件

以下の項では、オーバークラウドのインストール内の個別システムおよびノードの要件について詳しく説明します。

### 注記

SAN (FC-AL、FCoE、iSCSI) からのオーバークラウドノードのブートはサポートされていません。

### 2.4.1. コンピュートノードの要件

コンピュートノードは、仮想マシンインスタンスが起動した後にそれらを稼働させる役割を果たします。コンピュートノードは、ハードウェアの仮想化をサポートしている必要があります。また、ホストする仮想マシンインスタンスの要件をサポートするのに十分なメモリーとディスク容量も必要です。

#### プロセッサー

Intel 64 または AMD64 CPU 拡張機能をサポートする 64 ビット x86 プロセッサで Intel VT または AMD-V のハードウェア仮想化拡張機能が有効化されていること。このプロセッサには最小でも 4 つのコアが搭載されていることを推奨しています。

#### メモリー

最小で 6 GB のメモリー。この要件には、仮想マシンインスタンスに割り当てるメモリー容量に基づいて、追加の RAM を加算します。

#### ディスク領域

最小 40 GB の空きディスク領域

#### ネットワークインターフェースカード

最小 1 枚の 1 Gbps ネットワークインターフェースカード (実稼働環境では最低でも NIC を 2 枚使用することを推奨)。タグ付けされた VLAN トラフィックを委譲する場合や、ボンディングインターフェース向けの場合には追加のネットワークインターフェースを使用します。

#### 電源管理

各コントローラーノードには、Intelligent Platform Management Interface (IPMI) 機能などのサポート対象の電源管理インターフェースがサーバーのマザーボードに搭載されている必要があります。

### 2.4.2. コントローラーノードの要件

コントローラーノードは、Red Hat OpenStack Platform 環境の中核となるサービス (例: Horizon Dashboard、バックエンドのデータベースサーバー、Keystone 認証、高可用性サービスなど) をホストする役割を果たします。

#### プロセッサ

Intel 64 または AMD64 CPU 拡張機能のサポートがある 64 ビットの x86 プロセッサ

#### メモリー

最小のメモリー容量は 16 GB です。推奨のメモリー容量は、CPU のコア数により異なります。以下の計算を参考にしてください。

- **コントローラーの最小メモリー容量の算出:**

- コア毎に 1.5 GB のメモリーを使用します。たとえば、コアが 48 個あるマシンにはメモリーは 72 GB 必要です。

- **コントローラーの推奨メモリー容量の算出:**

- コア毎に 3 GB のメモリーを使用します。たとえば、コアが 48 個あるマシンにはメモリーは 144 GB 必要です。

メモリーの要件に関する詳しい情報は、Red Hat カスタマーポータルで「[Red Hat OpenStack Platform で、クラスター化されたコントローラーに必要なハードウェア \(CPU、メモリー\) 要件](#)」の記事を参照してください。

#### ディスク領域

最小 40 GB の空きディスク領域

#### ネットワークインターフェースカード

最小 2 枚の 1 Gbps ネットワークインターフェースカード。タグ付けされた VLAN トラフィックを委譲する場合や、ボンディングインターフェース向けの場合には追加のネットワークインターフェースを使用します。

#### 電源管理

各コントローラーノードには、Intelligent Platform Management Interface (IPMI) 機能などのサポート対象の電源管理インターフェースがサーバーのマザーボードに搭載されている必要があります。

### 2.4.3. Ceph Storage ノードの要件

Ceph Storage ノードは、Red Hat OpenStack Platform 環境でオブジェクトストレージを提供する役割を果たします。

#### プロセッサ

Intel 64 または AMD64 CPU 拡張機能のサポートがある 64 ビットの x86 プロセッサ

#### メモリー

メモリー要件はストレージ容量によって異なります。ハードディスク容量 1 TB あたり最小で 1 GB のメモリーを使用するのが理想的です。

#### ディスク領域

ストレージ要件はメモリーの容量によって異なります。ハードディスク容量 1 TB あたり最小で 1 GB のメモリーを使用するのが理想的です。

#### ディスクのレイアウト

推奨される Red Hat Ceph Storage ノードの設定には、以下のようなディスクレイアウトが必要です。

- **/dev/sda**: ルートディスク。director は、主なオーバークラウドイメージをディスクにコピーします。
- **/dev/sdb**: ジャーナルディスク。このディスクは、**/dev/sdb1**、**/dev/sdb2**、**/dev/sdb3** などのように、Ceph OSD ジャーナル向けにパーティションを分割します。ジャーナルディスクは通常、システムパフォーマンスの向上に役立つ Solid State Drive (SSD) です。
- **/dev/sdc** 以降: OSD ディスク。ストレージ要件に必要な数のディスクを使用します。本ガイドには、Ceph Storage ディスクを director にマッピングするために必要な手順を記載しています。

#### ネットワークインターフェースカード

最小で 1 x 1 Gbps ネットワークインターフェースカード (実稼働環境では、最低でも NIC を 2 つ以上使用することを推奨します)。ボンディングインターフェース向けの場合や、タグ付けされた VLAN トラフィックを委譲する場合には、追加のネットワークインターフェースを使用します。特に大量のトラフィックにサービスを提供する OpenStack Platform 環境を構築する場合には、ストレージノードには 10 Gbps インターフェースを使用することを推奨します。

#### 電源管理

各コントローラーノードには、Intelligent Platform Management Interface (IPMI) 機能などのサポート対象の電源管理インターフェースがサーバーのマザーボードに搭載されている必要があります。

#### 重要

director は、ジャーナルディスク上にはパーティションを作成しません。これらのジャーナルパーティションは、director が Ceph Storage ノードをデプロイする前に手動で作成しておく必要があります。

Ceph Storage OSD およびジャーナルのパーティションには、GPT ディスクラベルが必要です。このラベルも、カスタマイズの前に設定します。たとえば、Ceph Storage ホストとなるマシンで以下のコマンドを実行して、ディスクまたはパーティションの GPT ディスクラベルを作成します。

```
# parted [device] mklabel gpt
```

## 2.4.4. Object Storage ノードの要件

Object Storage ノードは、オーバークラウドのオブジェクトストレージ層を提供します。Object Storage プロキシは、コントローラーノードにインストールされます。ストレージ層には、ノード毎に複数のディスクを持つベアメタルノードが必要です。

### プロセッサ

Intel 64 または AMD64 CPU 拡張機能のサポートがある 64 ビットの x86 プロセッサ

### メモリー

メモリー要件はストレージ容量によって異なります。ハードディスク容量 1 TB あたり最小で 1 GB のメモリーを使用するのが理想的です。最適なパフォーマンスを得るには、特にワークロードが小さいファイル (100 GB 未満) の場合にはハードディスク容量 1 TB あたり 2 GB のメモリーを使用することを推奨します。

### ディスク領域

ストレージ要件は、ワークロードに必要とされる容量により異なります。アカウントとコンテナのデータを保存するには SSD ドライブを使用することを推奨します。アカウントおよびコンテナデータとオブジェクトの容量比率は、約 1 % です。たとえば、ハードドライブの容量 100 TB 毎に、アカウントおよびコンテナデータの SSD 容量は 1 TB 用意するようにします。

ただし、これは保存したデータの種類により異なります。保存するオブジェクトサイズの大半が小さい場合には、SSD の容量がさらに必要です。オブジェクトが大きい場合には (ビデオ、バックアップなど)、SSD の容量を減らします。

### ディスクのレイアウト

推奨のノード設定には、以下のようなディスクレイアウトが必要です。

- **/dev/sda**: ルートディスク。director は、主なオーバークラウドイメージをディスクにコピーします。
- **/dev/sdb**: アカウントデータに使用します。
- **/dev/sdc**: コンテナデータに使用します。
- **/dev/sdc** 以降: オブジェクトサーバーディスク。ストレージ要件で必要な数のディスクを使用します。

### ネットワークインターフェースカード

最小 2 枚の 1 Gbps ネットワークインターフェースカード。タグ付けされた VLAN トラフィックを委譲する場合や、ボンディングインターフェース向けの場合には追加のネットワークインターフェースを使用します。

### 電源管理

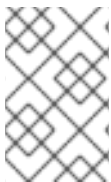
各コントローラーノードには、Intelligent Platform Management Interface (IPMI) 機能などのサポート対象の電源管理インターフェースがサーバーのマザーボードに搭載されている必要があります。

## 2.5. リポジトリの要件

アンダークラウドもオーバークラウドも、Red Hat コンテンツ配信ネットワーク (CDN) または Red Hat Satellite 5 または 6 を利用した Red Hat リポジトリへのアクセスが必要です。Red Hat Satellite サーバーを使用する場合は、必要なりポジトリをお使いの OpenStack Platform 環境に同期してください。以下の CDN チャンネル名一覧をガイドとして使用してください。

表2.1 OpenStack Platform リポジトリ

Name	リポジトリ	要件の説明
Red Hat Enterprise Linux 7 Server (RPMs)	<b>rhel-7-server-rpms</b>	ベースオペレーティングシステムのリポジトリ
Red Hat Enterprise Linux 7 Server - Extras (RPMs)	<b>rhel-7-server-extras-rpms</b>	Red Hat OpenStack Platform の依存関係が含まれます。
Red Hat Enterprise Linux 7 Server - RH Common (RPMs)	<b>rhel-7-server-rh-common-rpms</b>	Red Hat OpenStack Platform のデプロイと設定ツールが含まれます。
Red Hat Satellite Tools for RHEL 7 Server RPMs x86_64	<b>rhel-7-server-satellite-tools-6.2-rpms</b>	Red Hat Satellite 6 でのホスト管理ツール
Red Hat Enterprise Linux High Availability (for RHEL 7 Server) (RPMs)	<b>rhel-ha-for-rhel-7-server-rpms</b>	Red Hat Enterprise Linux の高可用性ツール。コントローラーノードの高可用性に使用します。
Red Hat Enterprise Linux OpenStack Platform 9 for RHEL 7 (RPMs)	<b>rhel-7-server-openstack-9-rpms</b>	Red Hat OpenStack Platform のコアリポジトリ。Red Hat OpenStack Platform director のパッケージも含まれます。
Red Hat Ceph Storage OSD 1.3 for Red Hat Enterprise Linux 7 Server (RPMs)	<b>rhel-7-server-rhceph-1.3-osd-rpms</b>	(Ceph Storage ノード向け) Ceph Storage Object Storage デーモンのリポジトリ。Ceph Storage ノードにインストールします。
Red Hat Ceph Storage MON 1.3 for Red Hat Enterprise Linux 7 Server (RPMs)	<b>rhel-7-server-rhceph-1.3-mon-rpms</b>	(Ceph Storage ノード向け) Ceph Storage Monitor デーモンのリポジトリ。Ceph Storage ノードを使用して OpenStack 環境にあるコントローラーノードにインストールします。
Red Hat Ceph Storage Tools 1.3 for Red Hat Enterprise Linux 7 Workstation (RPMs)	<b>rhel-7-server-rhceph-1.3-tools-rpms</b>	Ceph Storage クラスターと通信するためのノード用のツールを提供します。このリポジトリは、Ceph Storage クラスターを使用するオーバークラウドをデプロイする際に、全ノードに有効化する必要があります。



## 注記

ネットワークがオフラインの Red Hat OpenStack Platform 環境用リポジトリを設定するには、「[Configuring Red Hat OpenStack Platform Director in an Offline Environment](#)」の記事を参照してください。

## 第3章 オーバークラウドのプランニング

以下の項には、ノードロールの定義、ネットワークボロジのプランニング、ストレージなど、Red Hat OpenStack Platform 環境のさまざまな面のプランニングに関するガイドラインを記載します。

### 3.1. ノードのデプロイメントロールのプランニング

director はオーバークラウドの構築に、デフォルトで複数のノード種別を提供します。これらのノード種別は以下のとおりです。

#### コントローラー

環境を制御するための主要なサービスを提供します。これには、Dashboard (Horizon)、認証 (Keystone)、イメージストレージ (Glance)、ネットワーク (Neutron)、オーケストレーション (Heat)、高可用性サービスが含まれます。高可用性の場合は、Red Hat OpenStack Platform 環境にコントローラーノードが3台必要です。



#### 注記

1 台のノードで構成される環境はテスト目的で 사용할 ことができます。2 台のノードまたは 3 台以上のノードで構成される環境はサポートされません。

#### Compute

ハイパーバイザーとして機能し、環境内で仮想マシンを実行するのに必要な処理能力を提供する物理サーバー。基本的な Red Hat OpenStack Platform 環境には少なくとも 1 つのコンピュートノードが必要です。

#### Ceph-Storage

Red Hat Ceph Storage を提供するホスト。Ceph Storage ホストはクラスターに追加され、クラスターをスケールリングします。このデプロイメントロールはオプションです。

#### Cinder-Storage

OpenStack の Cinder サービスに外部ブロックストレージを提供するホスト。このデプロイメントロールはオプションです。

#### Swift-Storage

OpenStack の Swift サービスに外部オブジェクトストレージを提供するホスト。このデプロイメントロールはオプションです。

以下の表には、オーバークラウドの構成例と各シナリオで使用するノードタイプの定義をまとめています。

表3.1 各種シナリオに使用するノードデプロイメントロール

	コントローラー	Compute	Ceph-Storage	Swift-Storage	Cinder-Storage	合計
小規模のオーバークラウド	1	1	-	-	-	2
中規模のオーバークラウド	1	3	-	-	-	4

追加のオブジェクトおよびブロックストレージのある中規模のオーバークラウド	1	3	-	1	1	6
高可用性の中規模オーバークラウド	3	3	-	-	-	6
高可用性で Ceph Storage のある中規模オーバークラウド	3	3	3	-	-	9

## 3.2. ネットワークのプランニング

ロールとサービスを適切にマッピングして相互に正しく通信できるように、環境のネットワークポロジおよびサブネットのプランニングを行うことが重要です。Red Hat OpenStack Platform では、自律的に動作してソフトウェアベースのネットワーク、静的/Floating IP アドレス、DHCP を管理する Neutron ネットワークサービスを使用します。director は、オーバークラウド環境の各コントローラーノードに、このサービスをデプロイします。

Red Hat OpenStack Platform は、さまざまなサービスをマッピングして、お使いの環境の各種サブネットに割り当てられたネットワークトラフィックの種別を分類します。これらのネットワークトラフィック種別は以下のとおりです。

表3.2 ネットワーク種別の割り当て

ネットワーク種別	説明	そのネットワーク種別を使用するノード
IPMI	ノードの電源管理に使用するネットワーク。このネットワークは、アンダークラウドのインストール前に事前定義されます。	全ノード
プロビジョニング	director は、このネットワークトラフィック種別を使用して、PXE ブートで新規ノードをデプロイし、オーバークラウドベアメタルサーバーに OpenStack Platform のインストールをオーケストレーションします。このネットワークは、アンダークラウドのインストール前に事前定義されます。	全ノード

内部 API	内部 API ネットワークは、API 通信、RPC メッセージ、データベース通信経由で OpenStack のサービス間の通信を行う際に使用します。	コントローラー、コンピュート、Cinder Storage、Swift Storage
テナント	Neutron は、VLAN 分離 (各テナントネットワークがネットワーク VLAN) または VXLAN か GRE 経由のトンネリングを使用した独自のネットワークを各テナントに提供します。ネットワークトラフィックは、テナントのネットワークごとに分割されます。テナントネットワークにはそれぞれ IP サブネットが割り当てられています。また、ネットワーク名前空間が複数あると、複数のテナントネットワークが同じアドレスを使用できるので、競合は発生しません。	コントローラー、コンピュート
ストレージ	Block Storage、NFS、iSCSI など。理想的には、これはパフォーマンスの関係上、全く別のスイッチファブリックに分離します。	全ノード
ストレージ管理	OpenStack Object Storage (swift) は、このネットワークを使用して、参加するレプリカノード間でデータオブジェクトを同期します。プロキシサービスは、ユーザー要求と背後にあるストレージ層の間の仲介インターフェースとして機能します。プロキシは、受信要求を受け取り、必要なレプリカの位置を特定して要求データを取得します。Ceph バックエンドを使用するサービスは、Ceph と直接対話せずにフロントエンドのサービスを使用するため、ストレージ管理ネットワーク経由で接続を確立します。RBD ドライバーは例外で、このトラフィックは直接 Ceph に接続する点に注意してください。	コントローラー、Ceph Storage、Cinder Storage、Swift Storage



外部	グラフィカルシステム管理用の OpenStack Dashboard (Horizon)、OpenStack サービス用のパブリック API をホストして、インスタンスへの受信トラフィック向けに SNAT を実行します。外部ネットワークがプライベート IP アドレスを使用する場合には (RFC-1918 に準拠)、インターネットからのトラフィックに対して、さらに NAT を実行する必要があります。	コントローラー
Floating IP	受信トラフィックが Floating IP アドレスとテナントネットワーク内のインスタンスに実際に割り当てられた IP アドレスとの間の 1 対 1 の IP アドレスマッピングを使用してインスタンスに到達できるようにします。外部ネットワークからは分離した VLAN 上で Floating IP をホストする場合には、Floating IP VLAN をコントローラーノードにトランキングして、オーバークラウドの作成後に Neutron を介して VLAN を追加します。これにより、複数のブリッジに接続された複数の Floating IP ネットワークを作成する手段が提供されます。VLAN は、トランキングされますが、インターフェースとしては設定されません。その代わりに、Neutron は各 Floating IP ネットワークに選択したブリッジ上の VLAN セグメンテーション ID を使用して、OVS ポートを作成します。	コントローラー
管理	SSH アクセス、DNS トラフィック、NTP トラフィックなどのシステム管理機能を提供します。このネットワークは、コントローラー以外のノード用のゲートウェイとしても機能します。	全ノード

一般的な Red Hat OpenStack Platform のシステム環境では通常、ネットワーク種別の数は物理ネットワークのリンク数を超えます。全ネットワークを正しいホストに接続するには、オーバークラウドは VLAN タグ付けを使用して、1 つのインターフェースに複数のネットワークを提供します。ネットワークの多くは、サブネットが分離されていますが、インターネットアクセスまたはインフラストラクチャーにネットワーク接続ができるようにルーティングを提供するレイヤー 3 のゲートウェイが必要です。



## 注記

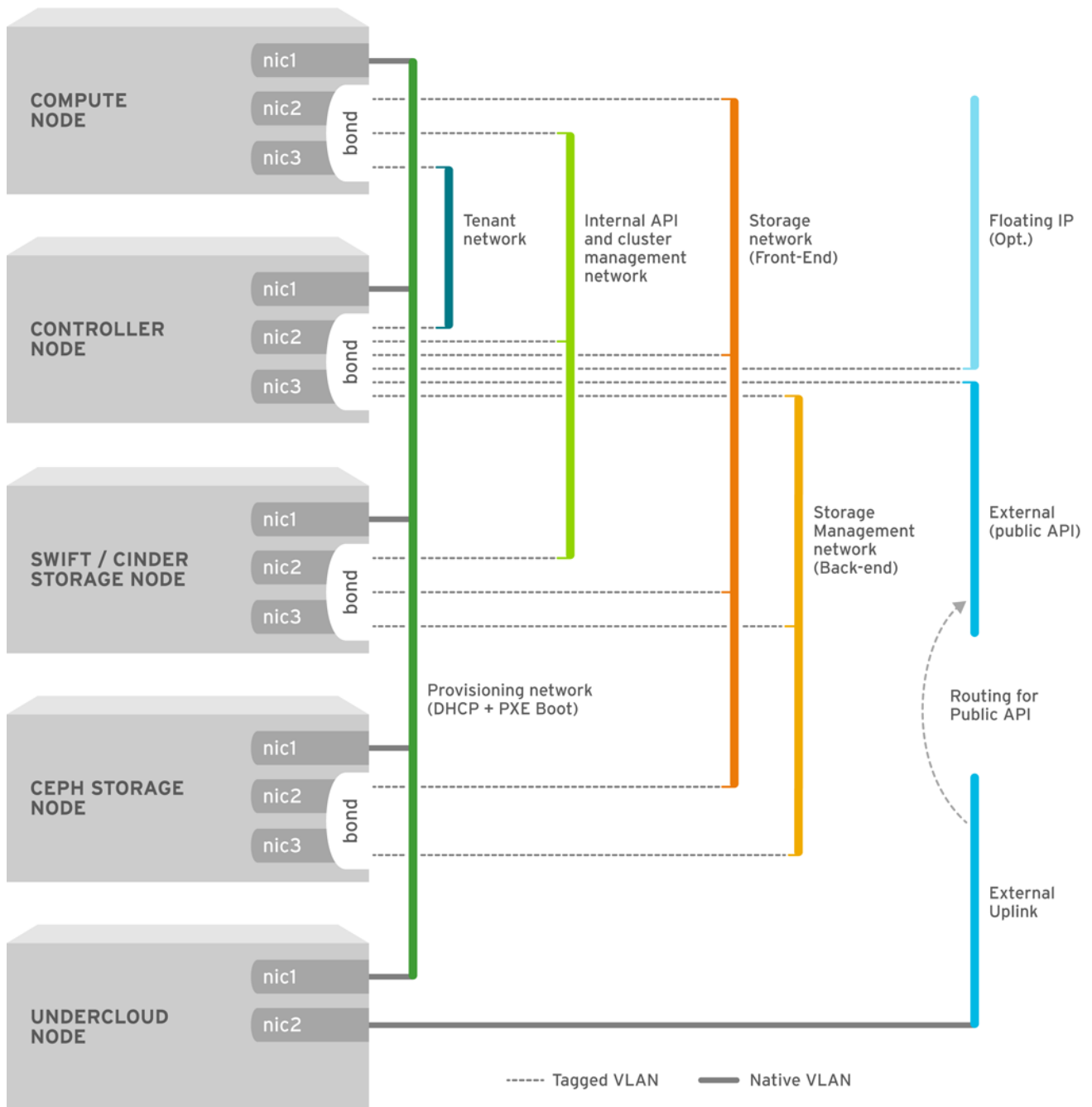
デプロイ時に neutron VLAN モード (トンネリングは無効) を使用する場合でも、プロジェクトネットワーク (GRE または VXLAN でトンネリング) をデプロイすることを推奨します。これには、デプロイ時にマイナーなカスタマイズを行う必要があり、将来ユーティリティーネットワークまたは仮想化ネットワークとしてトンネルネットワークを使用するためのオプションが利用可能な状態になります。VLAN を使用してテナントネットワークを作成することは変わりませんが、テナントの VLAN を消費せずに特別な用途のネットワーク用に VXLAN トンネルを作成することも可能です。また、テナント VLAN を使用するデプロイメントに VXLAN 機能を追加することは可能ですが、サービスを中断せずにテナント VLAN を既存のオーバークラウドに追加することはできません。

director は、トラフィック種別の中から 6 つを特定のサブネットまたは VLAN にマッピングする方法を提供します。このようなトラフィック種別には、以下が含まれます。

- 内部 API
- ストレージ
- ストレージ管理
- テナントネットワーク
- 外部
- 管理

未割り当てのネットワークは、プロビジョニングネットワークと同じサブネットに自動的に割り当てられます。

下図では、ネットワークが個別の VLAN に分離されたネットワークトポロジーの例を紹介しています。各オーバークラウドノードは、ボンディングで 2 つ (**nic2** および **nic3**) のインターフェースを使用して、対象の VLAN 経由でこれらのネットワークを提供します。また、各オーバークラウドのノードは、ネイティブの VLAN (**nic1**) を使用するプロビジョニングネットワークでアンダークラウドと通信します。



OPENSTACK\_364029\_0715

以下の表は、異なるネットワークのレイアウトをマッピングするネットワークトラフィック例が記載されています。

表3.3 ネットワークマッピング

	マッピング	インターフェースの総数	VLAN の総数
--	-------	-------------	----------

外部アクセスのあるフラットネットワーク	<p>ネットワーク 1: プロビジョニング、内部 API、ストレージ、ストレージ管理、テナントネットワーク</p> <p>ネットワーク 2: 外部、Floating IP (オーバークラウドの作成後にマッピング)</p>	2	2
分離ネットワーク	<p>ネットワーク 1: プロビジョニング</p> <p>ネットワーク 2: 内部 API</p> <p>ネットワーク 3: テナントネットワーク</p> <p>ネットワーク 4: ストレージ</p> <p>ネットワーク 5: ストレージ管理</p> <p>ネットワーク 6: ストレージ管理</p> <p>ネットワーク 7: 外部、Floating IP (オーバークラウドの作成後にマッピング)</p>	3 (ボンディングインターフェース 2 つを含む)	7

### 3.3. ストレージのプランニング

director は、オーバークラウド環境にさまざまなストレージオプションを提供します。オプションは以下のとおりです。

#### Ceph Storage ノード

director は、Red Hat Ceph Storage を使用して拡張可能なストレージノードセットを作成します。オーバークラウドは、各種ノードを以下の目的で使用します。

- **イメージ:** Glance は仮想マシンのイメージを管理します。イメージは変更できないため、OpenStack はイメージバイナリーブロブとして処理し、それに応じてイメージをダウンロードします。Ceph ブロックデバイスでイメージを格納するには、Glance を使用することができます。
- **ボリューム:** Cinder ボリュームはブロックデバイスです。OpenStack は、仮想マシンの起動や、実行中の仮想マシンへのボリュームのアタッチにボリュームを使用し、Cinder サービスを使用してボリュームを管理します。さらに、イメージの CoW (Copy-on-Write) のクローンを使用して仮想マシンを起動する際には Cinder を使用します。
- **ゲストディスク:** ゲストディスクは、ゲストオペレーティングシステムのディスクです。デ

フォルトでは、Nova で仮想マシンを起動すると、ディスクは、ハイパーバイザーのファイルシステム上のファイルとして表示されます (通常 `/var/lib/nova/instances/<uuid>/` の配下)。Cinder を使用せずに直接 Ceph 内にある全仮想マシンを起動することができます。これは、ライブマイグレーションのプロセスで簡単にメンテナンス操作を実行できるため好都合です。また、ハイパーバイザーが停止した場合には、**nova evacuate** をトリガーして仮想マシンをほぼシームレスに別の場所で行うこともできるので便利です。



### 重要

Ceph で仮想マシンを起動するには (一時バックエンドまたはボリュームからの起動)、Glance のイメージ形式は **RAW** でなければなりません。Ceph は、仮想マシンディスクのホスティングで QCOW2 や VMDK などのその他の形式はサポートしていません。

その他の情報については、[『Red Hat Ceph Storage Architecture Guide』](#) を参照してください。

## Swift Storage ノード

director は、外部オブジェクトストレージノードを作成します。これは、オーバークラウド環境でコントローラーノードをスケーリングまたは置き換える必要があるが、高可用性クラスター外にオブジェクトストレージを保つ必要がある場合に便利です。

## 第4章 アンダークラウドのインストール

Red Hat OpenStack Platform 環境の構築では、最初にアンダークラウドシステムに director をインストールします。これには、必要なサブスクリプションやリポジトリを有効化するために複数の手順を実行する必要があります。

### 4.1. DIRECTOR のインストールユーザーの作成

director のインストールプロセスでは、root 以外のユーザーがコマンドを実行する必要があります。以下のコマンドを使用して、**stack** という名前のユーザーを作成して、パスワードを設定します。

```
[root@director ~]# useradd stack
[root@director ~]# passwd stack # specify a password
```

**sudo** を使用する際に、このユーザーがパスワードを要求されないようにします。

```
[root@director ~]# echo "stack ALL=(root) NOPASSWD:ALL" | tee -a
/etc/sudoers.d/stack
[root@director ~]# chmod 0440 /etc/sudoers.d/stack
```

新規作成した **stack** ユーザーに切り替えます。

```
[root@director ~]# su - stack
[stack@director ~]$
```

**stack** ユーザーで director のインストールを続行します。

### 4.2. テンプレートとイメージ用のディレクトリーの作成

director はシステムのイメージと Heat テンプレートを使用して、オーバークラウド環境を構築します。これらのファイルを整理するには、イメージとテンプレート用にディレクトリーを作成するように推奨します。

```
$ mkdir ~/images
$ mkdir ~/templates
```

本書の他の項では、2 つのディレクトリーを使用して特定のファイルを保存します。

### 4.3. システムのホスト名設定

director では、インストールと設定プロセスにおいて完全修飾ドメイン名が必要です。つまり、director のホストのホスト名を設定する必要がある場合があります。以下のコマンドで、ホストのホスト名をチェックします。

```
$ hostname # Checks the base hostname
$ hostname -f # Checks the long hostname (FQDN)
```

いずれのコマンドでも正しいホスト名が返されない場合やエラーが報告される場合には、**hostnamectl** でホスト名を設定します。

```
$ sudo hostnamectl set-hostname manager.example.com
$ sudo hostnamectl set-hostname --transient manager.example.com
```

director では、**/etc/hosts** にシステムのホスト名とベース名も入力する必要があります。たとえば、システムの名前が **manager.example.com** の場合には、**/etc/hosts** に以下のように入力する必要があります。

```
127.0.0.1    manager.example.com manager localhost localhost.localdomain
localhost4  localhost4.localdomain4
```

## 4.4. システムの登録

Red Hat OpenStack Platform director をインストールするには、まず Red Hat サブスクリプションマネージャーを使用してホストシステムを登録し、必要なチャンネルをサブスクライブします。

コンテンツ配信ネットワークにシステムを登録します。プロンプトが表示されたら、カスタマーポータルของผู้ーザー名とパスワードを入力します。

```
$ sudo subscription-manager register
```

Red Hat OpenStack Platform director のエンタイトルメントプールを検索します。

```
$ sudo subscription-manager list --available --all
```

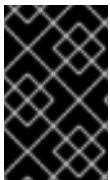
上記のステップで特定したプール ID を使用して、Red Hat OpenStack Platform 9 のエンタイトルメントをアタッチします。

```
$ sudo subscription-manager attach --pool=pool_id
```

デフォルトのリポジトリをすべて無効にしてから、必要な Red Hat Enterprise Linux リポジトリを有効にします。

```
$ sudo subscription-manager repos --disable=*
$ sudo subscription-manager repos --enable=rhel-7-server-rpms --
enable=rhel-7-server-extras-rpms --enable=rhel-7-server-openstack-9-rpms --
enable=rhel-7-server-openstack-9-director-rpms --enable=rhel-7-server-rh-
common-rpms
```

これらのリポジトリには、director のインストールに必要なパッケージが含まれます。



### 重要

上記にリストしたリポジトリのみを有効にします。追加のリポジトリを使用すると、パッケージとソフトウェアの競合が発生する場合があります。他のリポジトリは有効にしないでください。

システムで更新を実行して、ベースシステムパッケージを最新の状態にします。

```
$ sudo yum update -y
$ sudo reboot
```

システムは、director をインストールできる状態になりました。

## 4.5. DIRECTOR パッケージのインストール

以下のコマンドを使用して、director のインストールおよび設定に必要なコマンドラインツールをインストールします。

```
$ sudo yum install -y python-tripleoclient
```

これにより、director のインストールに必要なパッケージがすべてインストールされます。

## 4.6. DIRECTOR の設定

director のインストールプロセスには、ネットワーク設定を判断する特定の設定が必要です。この設定は、**stack** ユーザーのホームディレクトリーに **undercloud.conf** として配置されているテンプレートに保存されています。

Red Hat は、インストールに必要な設定を判断しやすいように、基本テンプレートを提供しています。このテンプレートは、**stack** ユーザーのホームディレクトリーにコピーします。

```
$ cp /usr/share/instack-undercloud/undercloud.conf.sample  
~/undercloud.conf
```

基本テンプレートには、以下のパラメーターが含まれています。

### local\_ip

director のプロビジョニング NIC 用に定義する IP アドレス。これは、director が DHCP および PXE ブートサービスに使用する IP アドレスでもあります。環境内の既存の IP アドレスまたはサブネットと競合するなど、プロビジョニングネットワークに別のサブネットを使用する場合以外は、この値はデフォルトの **192.0.2.1/24** のままにしてください。

### network\_gateway

オーバークラウドインスタンスのゲートウェイ。外部ネットワークにトラフィックを転送するアンダークラウドのホストです。director に別の IP アドレスを使用する場合または外部ゲートウェイを直接使用する場合以外は、この値はデフォルト (**192.0.2.1**) のままにします。



### 注記

director の設定スクリプトは、適切な **sysctl** カーネルパラメーターを使用して IP フォワーディングを自動的に有効にする操作も行います。

### undercloud\_public\_vip

director のパブリック API 用に定義する IP アドレス。他の IP アドレスまたはアドレス範囲と競合しないプロビジョニングネットワークの IP アドレスを使用します。たとえば、**192.0.2.2** で、director の設定により、この IP アドレスは **/32** ネットマスクを使用するルーティングされた IP アドレスとしてソフトウェアブリッジに接続されます。

### undercloud\_admin\_vip

director の管理 API 用に定義する IP アドレス。他の IP アドレスまたはアドレス範囲と競合しないプロビジョニングネットワークの IP アドレスを使用します。たとえば、**192.0.2.3** で、director の設定により、この IP アドレスは **/32** ネットマスクを使用するルーティングされた IP アドレスとしてソフトウェアブリッジに接続されます。



## undercloud\_service\_certificate

OpenStack SSL 通信の証明書の場合とファイル名。理想的には、信頼できる認証局から、この証明書を取得します。それ以外の場合は、「付録A SSL/TLS 証明書の設定」のガイドラインを使用して独自の自己署名の証明書を作成します。これらのガイドラインには、自己署名の証明書が認証局からの証明書に拘らず、証明書の SELinux コンテキストを設定する方法が含まれています。

## local\_interface

director のプロビジョニング NIC 用に選択するインターフェース。これは、director が DHCP および PXE ブートサービスに使用するデバイスでもあります。どのデバイスが接続されているかを確認するには、**ip addr** コマンドを使用します。以下に **ip addr** コマンドの出力結果の例を示します。

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
    link/ether 52:54:00:75:24:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.178/24 brd 192.168.122.255 scope global dynamic
eth0
    valid_lft 3462sec preferred_lft 3462sec
    inet6 fe80::5054:ff:fe75:2409/64 scope link
    valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state
DOWN
    link/ether 42:0b:c2:a5:c1:26 brd ff:ff:ff:ff:ff:ff
```

この例では、外部 NIC は **eth0** を、プロビジョニング NIC は未設定の **eth1** を使用します。今回は、**local\_interface** を **eth1** に設定します。この設定スクリプトにより、このインターフェースが **inspection\_interface** パラメーターで定義したカスタムのブリッジにアタッチされます。

## network\_cidr

オーバークラウドインスタンスの管理に director が使用するネットワーク。これはプロビジョニングネットワークです。プロビジョニングネットワークに別のサブネットを使用しない限り、この値はデフォルト (**192.0.2.0/24**) のままにします。

## masquerade\_network

外部にアクセスするためにマスカレードするネットワークを定義します。これにより、プロビジョニングネットワークにネットワークアドレス変換 (NAT) の範囲が提供され、director 経由で外部アクセスが可能になります。プロビジョニングネットワークに別のサブネットを使用しない限り、この値はデフォルト (**192.0.2.0/24**) のままにします。

## dhcp\_start; dhcp\_end

オーバークラウドノードの DHCP 割り当て範囲 (開始アドレスと終了アドレス)。ノードを割り当てるのに十分な IP アドレスがこの範囲に含まれるようにします。

## inspection\_interface

ノードのイントロスペクションに director が使用するブリッジ。これは、director の設定により作成されるカスタムのブリッジです。**LOCAL\_INTERFACE** でこのブリッジをアタッチします。これは、デフォルトの **br-ctlplane** のままにします。

## inspection\_iprange

director のイントロスペクションサービスが PXE ブートとプロビジョニングプロセスの際に使用する IP アドレス範囲。この範囲の開始アドレスと終了アドレスの定義には、**192.0.2.100, 192.0.2.120** などのように、コンマ区切りの値を使用します。この範囲には、お使いのノードの IP アドレスが含まれており、**dhcp\_start** と **dhcp\_end** の範囲と競合がないようにします。

## inspection\_extras

イントロスペクション時に追加のハードウェアコレクションを有効化するかどうかを定義します。イントロスペクションイメージでは **python-hardware** または **python-hardware-detect** パッケージが必要です。

### inspection\_runbench

ノードイントロスペクション時に一連のベンチマークを実行します。有効にするには、**true** に設定します。このオプションは、登録ノードのハードウェアを検査する際にベンチマーク分析を実行する場合に必要です。詳細は、「[ノードのハードウェアの検査](#)」を参照してください。

### undercloud\_debug

アンダークラウドサービスのログレベルを **DEBUG** に設定します。この値は **true** に設定して有効化します。

### enable\_tempest

検証ツールをインストールするかどうかを定義します。デフォルトは、**false** に設定されていますが、**true** で有効化することができます。

### ipxe\_deploy

iPXE か標準の PXE のいずれを使用するか定義します。デフォルトは **true** で iPXE を有効化します。**false** に指定すると、標準の PXE に設定されます。詳しい情報は、Red Hat カスタマーポータル「[Changing from iPXE to PXE in Red Hat OpenStack Platform director](#)」を参照してください。

### store\_events

アンダークラウドの Ceilometer にイベントを保存するかどうかを定義します。

### undercloud\_db\_password、undercloud\_admin\_token、undercloud\_admin\_password、undercloud\_glance\_passwrm など

残りのパラメーターは、全 director サービスのアクセス詳細を指定します。値を変更する必要はありません。**undercloud.conf** で空欄になっている場合には、これらの値は director の設定スクリプトによって自動的に生成されます。設定スクリプトの完了後には、すべての値を取得することができます。



### 重要

これらのパラメーターの設定ファイルの例では、プレースホルダーの値に **<None>** を使用しています。これらの値を **<None>** に設定すると、デプロイメントでエラーが発生します。

これらのパラメーターの値は、お使いのネットワークに応じて変更してください。完了したら、ファイルを保存して以下のコマンドを実行します。

```
$ openstack undercloud install
```

このコマンドで、director の設定スクリプトを起動します。director により、追加のパッケージがインストールされ、**undercloud.conf** の設定に合わせてサービスを設定します。このスクリプトは、完了までに数分かかります。

設定スクリプトにより、完了時には 2 つのファイルが生成されます。

- **undercloud-passwords.conf**: director サービスの全パスワード一覧
- **stackrc**: director のコマンドラインツールへアクセスできるようにする初期化変数セット

**stack** ユーザーを初期化してコマンドラインツールを使用するには、以下のコマンドを実行します。

```
$ source ~/stackrc
```

director のコマンドラインツールが使用できるようになりました。

## 4.7. オーバークラウドノードのイメージの取得

director では、オーバークラウドのノードをプロビジョニングする際に、複数のディスクが必要です。必要なディスクは以下のとおりです。

- イントロスペクションのカーネルおよび ramdisk: PXE ブートでベアメタルシステムのイントロスペクションに使用
- デプロイメントカーネルおよび ramdisk: システムのプロビジョニングおよびデプロイメントに使用
- オーバークラウドカーネル、ramdisk、完全なイメージ: ノードのハードディスクに書き込まれるベースのオーバークラウドシステム

**rhosp-director-images** および **rhosp-director-images-ipa** パッケージからこれらのイメージを取得します。

```
$ sudo yum install rhosp-director-images rhosp-director-images-ipa
```

**stack** ユーザーのホーム (**/home/stack/images**) の **images** ディレクトリーにアーカイブを展開します。

```
$ cd ~/images
$ for i in /usr/share/rhosp-director-images/overcloud-full-latest-9.0.tar
/usr/share/rhosp-director-images/ironic-python-agent-latest-9.0.tar; do
tar -xvf $i; done
```

これらのイメージを director にインポートします。

```
$ openstack overcloud image upload --image-path /home/stack/images/
```

このコマンドにより、**bm-deploy-kernel**、**bm-deploy-ramdisk**、**overcloud-full**、**overcloud-full-initrd**、**overcloud-full-vmlinuz** のイメージが director にアップロードされます。これらは、デプロイメントおよびオーバークラウド用のイメージです。スクリプトにより、director の PXE サーバーにイントロスペクションイメージもインストールされます。

CLI でイメージの一覧を表示します。

```
$ openstack image list
+-----+-----+
| ID                                         | Name                               |
+-----+-----+
| 765a46af-4417-4592-91e5-a300ead3faf6     | bm-deploy-ramdisk                 |
| 09b40e3d-0382-4925-a356-3a4b4f36b514     | bm-deploy-kernel                 |
| ef793cd0-e65c-456a-a675-63cd57610bd5     | overcloud-full                   |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152     | overcloud-full-initrd            |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d     | overcloud-full-vmlinuz           |
+-----+-----+
```

この一覧には、イントロスペクションの PXE イメージは表示されません。director は、これらのファイルを `/httpboot` にコピーします。

```
[stack@host1 ~]$ ls -l /httpboot
total 341460
-rwxr-xr-x. 1 root root 5153184 Mar 31 06:58 agent.kernel
-rw-r--r--. 1 root root 344491465 Mar 31 06:59 agent.ramdisk
-rw-r--r--. 1 root root 337 Mar 31 06:23 inspector.ipxe
```

## 4.8. アンダークラウドの **NEUTRON** サブネットでのネームサーバーの設定

オーバークラウドのノードには、DNS でホスト名が解決できるようにネームサーバーが必要です。ネットワークを分離していない標準のオーバークラウドでは、ネームサーバーはアンダークラウドの **neutron** サブネットを使用して定義されます。環境でネームサーバーを定義するには、以下のコマンドを使用してください。

```
$ neutron subnet-list
$ neutron subnet-update [subnet-uuid] --dns-nameserver [nameserver-ip]
```

サブネットを表示してネームサーバーを確認します。

```
$ neutron subnet-show [subnet-uuid]
+-----+-----+
| Field          | Value                               |
+-----+-----+
| ...            |                                     |
| dns_nameservers | 8.8.8.8                             |
| ...            |                                     |
+-----+-----+
```



### 重要

サービストラフィックを別のネットワークに分離する場合は、オーバークラウドのノードはネットワーク環境テンプレートの **DnsServer** パラメーターを使用します。これは、「[ネットワークの分離](#)」の高度な設定シナリオで説明します。

## 4.9. アンダークラウドのバックアップ

Red Hat では、アンダークラウドホストと Red Hat OpenStack Platform director から重要なデータをバックアップする手順を記載したガイドを提供しています。『[director のアンダークラウドのバックアップと復元](#)』を参照してください。

## 4.10. アンダークラウドの設定完了

これでアンダークラウドの設定が完了しました。次の章では、ノードの登録、検査、さまざまなノードロールのタグ付けなど、オーバークラウドの基本的な設定について説明します。

## 第5章 基本的なオーバークラウド要件の設定

本章では、エンタープライズレベルの OpenStack Platform 環境の基本的な設定手順を説明します。基本設定のオーバークラウドには、カスタムの機能が含まれていませんが、基本的なオーバークラウドに高度な設定オプションを追加して、「[6章 オーバークラウドの高度なカスタマイズ設定](#)」の説明に従い、仕様に合わせてカスタマイズすることができます。

本章の例では、すべてのノードが電源管理に IPMI を使用したベアメタルシステムとなっています。電源管理の種別およびそのオプションに関する詳細は、「[付録B 電源管理ドライバ](#)」を参照してください。

### ワークフロー

1. ノード定義のテンプレートを作成して director で空のノードを登録します。
2. 全ノードのハードウェアを検査します。
3. ロールにノードをタグ付けします。
4. 追加のノードプロパティを定義します。

### 要件

- 「[4章 アンダークラウドのインストール](#)」で作成した director ノード
- ノードに使用するベアメタルマシンのセット。必要なノード数は、作成予定のオーバークラウドのタイプにより異なります (オーバークラウドロールに関する情報は「[ノードのデプロイメントロールのプランニング](#)」を参照してください)。これらのマシンは、各ノード種別の要件セットに従う必要があります。これらの要件については、「[オーバークラウドの要件](#)」を参照してください。これらのノードにはオペレーティングシステムは必要ありません。director が Red Hat Enterprise Linux 7 のイメージを各ノードにコピーします。
- ネイティブ VLAN として設定したプロビジョニングネットワーク用のネットワーク接続 1 つ。全ノードは、このネイティブに接続して、「[ネットワーク要件](#)」で設定した要件に準拠する必要があります。この章の例では、以下の IP アドレスの割り当てで、プロビジョニングサブネットとして 192.0.2.0/24 を使用します。

表5.1 プロビジョニングネットワークの IP 割り当て

ノード名	IP アドレス	MAC アドレス	IPMI IP アドレス
director	192.0.2.1	aa:aa:aa:aa:aa:aa	不要
コントローラー	定義済みの DHCP	bb:bb:bb:bb:bb:bb	192.0.2.205
Compute	定義済みの DHCP	cc:cc:cc:cc:cc:cc	192.0.2.206

- その他のネットワーク種別はすべて、OpenStack サービスにプロビジョニングネットワークを使用しますが、ネットワークトラフィックの他のタイプに追加でネットワークを作成することができます。詳しい情報は、「[ネットワークの分離](#)」を参照してください。

### 5.1. オーバークラウドへのノードの登録

director では、手動で作成したノード定義のテンプレートが必要です。このファイル (**instackenv.json**) は、JSON 形式のファイルを使用して、ノードのハードウェアおよび電源管理の情報が含まれます。たとえば、2 つのノードを登録するテンプレートは、以下のようになります。

```
{
  "nodes": [
    {
      "mac": [
        "bb:bb:bb:bb:bb:bb"
      ],
      "cpu": "4",
      "memory": "6144",
      "disk": "40",
      "arch": "x86_64",
      "pm_type": "pxe_ipmitool",
      "pm_user": "admin",
      "pm_password": "p@55w0rd!",
      "pm_addr": "192.0.2.205"
    },
    {
      "mac": [
        "cc:cc:cc:cc:cc:cc"
      ],
      "cpu": "4",
      "memory": "6144",
      "disk": "40",
      "arch": "x86_64",
      "pm_type": "pxe_ipmitool",
      "pm_user": "admin",
      "pm_password": "p@55w0rd!",
      "pm_addr": "192.0.2.206"
    }
  ]
}
```

このテンプレートでは、以下の属性を使用します。

#### **pm\_type**

使用する電源管理ドライバー。この例では IPMI ドライバーを使用します (**pxe\_ipmitool**)。

#### **pm\_user; pm\_password**

IPMI のユーザー名およびパスワード

#### **pm\_addr**

IPMI デバイスの IP アドレス

#### **mac**

(オプション) ノード上のネットワークインターフェースの MAC アドレス一覧。各システムのプロビジョニング NIC の MAC アドレスのみを使用します。

#### **cpu**

(オプション) ノード上の CPU 数

#### **memory**

(オプション) メモリーサイズ (MB)

#### **disk**

(オプション) ハードディスクのサイズ (GB)

**arch**

(オプション) システムアーキテクチャー



#### 注記

電源管理の種別およびそのオプションに関する詳細は、「[付録B 電源管理ドライバー](#)」を参照してください。

テンプレートの作成後に、**stack** ユーザーのホームディレクトリー (`/home/stack/instackenv.json`) にファイルを保存して、以下のコマンドを使用して director にインポートします。

```
$ openstack baremetal import --json ~/instackenv.json
```

このコマンドでテンプレートをインポートして、テンプレートから director に各ノードを登録します。

カーネルと ramdisk イメージを全ノードに割り当てます。

```
$ openstack baremetal configure boot
```

director でのノードの登録および設定が完了しました。CLI でこれらのノードの一覧を表示します。

```
$ ironic node-list
```

## 5.2. ノードのハードウェアの検査

director は各ノードでイントロスペクションプロセスを実行することができます。このプロセスを実行すると、各ノードが PXE を介してイントロスペクションエージェントを起動します。このエージェントは、ノードからハードウェアのデータを収集して、director に送り返します。次に director は、director 上で実行中の OpenStack Object Storage (swift) サービスにこのイントロスペクションデータを保管します。director は、プロファイルのタグ付け、ベンチマーキング、ルートディスクの手動割り当てなど、さまざまな目的でハードウェア情報を使用します。



#### 注記

ポリシーファイルを作成して、イントロスペクションの直後にノードをプロファイルに自動でタグ付けすることも可能です。ポリシーファイルを作成してイントロスペクションプロセスに組み入れる方法に関する詳しい情報は、「[付録C プロファイルの自動タグ付け](#)」を参照してください。または、「[プロファイルへのノードのタグ付け](#)」に記載の手順に従って、ノードをプロファイルに手動でタグ付けすることもできます。

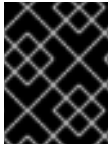
以下のコマンドを実行して、各ノードのハードウェア属性を検証します。

```
$ openstack baremetal introspection bulk start
```

別のターミナルウィンドウで以下のコマンドを使用してイントロスペクションの進捗状況をモニタリングします。

```
$ sudo journalctl -l -u openstack-ironic-inspector -u openstack-ironic-inspector-dnsmasq -u openstack-ironic-conductor -f
```



**重要**

このプロセスが最後まで実行されて正常に終了したことを確認してください。ベアメタルの場合には、通常 15 分ほどかかります。

または、各ノードに 1 回ずつ個別にイントロスペクションを実行します。ノードを管理モードに切り替えて、イントロスペクションを実行してから、管理モードから元に戻します。

```
$ ironic node-set-provision-state [NODE UUID] manage
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-provision-state [NODE UUID] provide
```

### 5.3. プロファイルへのノードのタグ付け

各ノードのハードウェアを登録、検査した後は、特定のプロファイルにノードをタグ付けします。このプロファイルタグにより、ノードがフレーバーに照合され、次にそのフレーバーがデプロイメントロールに割り当てられます。アンダークラウドのインストール時に、デフォルトプロファイルのフレーバー **compute**、**control**、**swift-storage**、**ceph-storage**、**block-storage** が作成され、大半の環境で変更なしに使用することができます。

**注記**

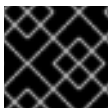
多くのノードでは、プロファイルの自動タグ付けを使用します。詳しい情報は、「[付録C プロファイルの自動タグ付け](#)」を参照してください。

特定のプロファイルにノードをタグ付けする場合には、各ノードの **properties/capabilities** パラメーターに **profile** オプションを追加します。たとえば、2 つのノードをタグ付けしてコントローラープロファイルとコンピュータープロファイルをそれぞれ使用するには、以下のコマンドを実行します。

```
$ ironic node-update 58c3d07e-24f2-48a7-bbb6-6843f0e8ee13 add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 add
properties/capabilities='profile:control,boot_option:local'
```

**profile:compute** と **profile:control** オプションを追加することで、この 2 つのノードがそれぞれのプロファイルにタグ付けされます。

またこのコマンドは、各ノードのブートモードを定義する **boot\_option:local** パラメーターを設定します。

**重要**

director は現在、UEFI ブートモードはサポートしていません。

ノードのタグ付けが完了した後は、割り当てたプロファイルまたはプロファイルの候補を確認します。

```
$ openstack overcloud profiles list
```

### 5.4. ノードのルートディスクの定義



ノードによっては、複数のディスクを使用するものもあります。つまり、プロビジョニングの際に、director は、ルートディスクに使用するディスクを特定する必要があるという意味です。以下のように、director がルートディスクを容易に特定できるように使用可能なプロパティが複数あります。

- **model** (文字列): デバイスの ID
- **vendor** (文字列): デバイスのベンダー
- **serial** (文字列): ディスクのシリアル番号
- **wwn** (文字列): 一意のストレージ ID
- **hctl** (文字列): SCSI のホスト:チャネル:ターゲット:Lun
- **size** (整数): デバイスのサイズ (GB)

以下の例では、root デバイスを特定するディスクのシリアル番号を使用して、オーバークラウドイメージをデプロイするドライブを指定します。

最初に、director がイントロスペクションで取得した各ノードのハードウェア情報のコピーを収集します。この情報は、OpenStack Object Storage (swift) サーバーに保管されています。この情報を新規ディレクトリーにダウンロードします。

```
$ mkdir swift-data
$ cd swift-data
$ export IRONIC_DISCOVERD_PASSWORD=`sudo grep admin_password /etc/ironic-inspector/inspector.conf | awk '! /^#/ {print $NF}' | awk -F=' ' '{print $2}'`
$ for node in $(ironic node-list | awk '!/UUID/ {print $2}'); do swift -U service:ironic -K $IRONIC_DISCOVERD_PASSWORD download ironic-inspector inspector_data-$node; done
```

この操作により、イントロスペクションで取得した各 **inspector\_data** オブジェクトからデータがダウンロードされます。全オブジェクトのオブジェクト名の一部には、ノードの UUID が使用されます。

```
$ ls -l
inspector_data-15fc0edc-eb8d-4c7f-8dc0-a2a25d5e09e3
inspector_data-46b90a4d-769b-4b26-bb93-50eaefcdb3f4
inspector_data-662376ed-faa8-409c-b8ef-212f9754c9c7
inspector_data-6fc70fe4-92ea-457b-9713-eed499eda206
inspector_data-9238a73a-ec8b-4976-9409-3fcff9a8dca3
inspector_data-9cbfe693-8d55-47c2-a9d5-10e059a14e07
inspector_data-ad31b32d-e607-4495-815c-2b55ee04cdb1
inspector_data-d376f613-bc3e-4c4b-ad21-847c4ec850f8
```

各ノードのディスク情報をチェックします。以下のコマンドを実行すると、各ノードの ID とディスク情報が表示されます。

```
$ for node in $(ironic node-list | awk '!/UUID/ {print $2}'); do echo "NODE: $node" ; cat inspector_data-$node | jq '.inventory.disks' ; echo "-----" ; done
```

たとえば、1 つのノードのデータで 3 つのディスクが表示される場合があります。

```
NODE: 46b90a4d-769b-4b26-bb93-50eaefcdb3f4
```

```
[
  {
    "size": 1000215724032,
    "vendor": "ATA",
    "name": "/dev/sda",
    "model": "WDC WD1002F9YZ",
    "wwn": "0x0000000000000001",
    "serial": "WD-000000000001"
  },
  {
    "size": 1000215724032,
    "vendor": "ATA",
    "name": "/dev/sdb",
    "model": "WDC WD1002F9YZ",
    "wwn": "0x0000000000000002",
    "serial": "WD-000000000002"
  },
  {
    "size": 1000215724032,
    "vendor": "ATA",
    "name": "/dev/sdc",
    "model": "WDC WD1002F9YZ",
    "wwn": "0x0000000000000003",
    "serial": "WD-000000000003"
  }
]
```

以下の例では、ルートデバイスを、シリアル番号 **WD-000000000002** の disk 2 に設定します。そのためには、ノードの定義に **root\_device** パラメーターを追加する必要があります。

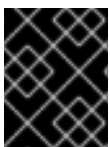
```
$ ironic node-update 97e3f7b3-5629-473e-a187-2193ebe0b5c7 add
properties/root_device='{"serial": "WD-000000000002"}'
```

これにより、director がルートディスクとして使用する特定のディスクを識別しやすくなります。オーバークラウドの作成の開始時には、director はこのノードをプロビジョニングして、オーバークラウドのイメージをこのディスクに書き込みます。



#### 注記

各ノードの BIOS を設定して、選択したルートディスクからの起動が含まれるようにします。推奨のブート順は最初がネットワークブートで、次にルートディスクブートです。



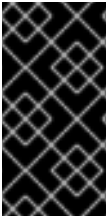
#### 重要

**name** でルートディスクを設定しないでください。この値は、ノードブート時に変更される可能性があります。

## 5.5. 基本設定の完了

これで、オーバークラウドの基本設定で必要とされる手順が終了しました。次に、以下のいずれかを実行することができます。

- 高度な設定の手順を使用して環境をカスタマイズします。詳しい情報は、「[6章 オーバークラウドの高度なカスタマイズ設定](#)」を参照してください。
- または、基本のオーバークラウドをデプロイします。詳しい情報は、「[7章 オーバークラウドの作成](#)」を参照してください。



### 重要

基本的なオーバークラウドでは、ブロックストレージにローカルの LVM ストレージを使用しますが、この設定はサポートされません。ブロックストレージには、外部ストレージソリューションを使用することを推奨します。たとえば、ブロックストレージとして NFS 共有を設定するには「[NFS ストレージの設定](#)」を参照してください。

## 第6章 オーバークラウドの高度なカスタマイズ設定

本章は「[5章 基本的なオーバークラウド要件の設定](#)」の続きとなります。この時点では、director によりノードが登録され、オーバークラウドの作成に必要なサービスが設定されました。次に、本章の手法を使用して、オーバークラウドをカスタマイズすることができます。



### 注記

本章に記載する例は、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。

### 6.1. 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、ネットワークの分離やネットワークインターフェースの設定など、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。

#### 6.1.1. 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、コンピュートリソース、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 のオブジェクトのことです。

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。

- **パラメーター**: Heat に渡す設定。値を渡さずにスタックやパラメーターのデフォルト値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。
- **リソース**: スタックの一部として作成/設定する特定のオブジェクト。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。これらは、テンプレートの **resources** セクションで定義されます。
- **出力**: スタックの作成後に Heat から渡される値。これらの値は、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、キーを使用する **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
```

### 6.1.2. 環境ファイル

環境ファイルとは、Heat テンプレートをカスタマイズする特別な種類のテンプレートです。このファイルは、3 つの主要な部分で構成されます。

- **リソースレジストリー**: このセクションは、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。基本的には、この設定により、コアリソースコレクションに存在しないカスタムのリソースを作成する方法が提供されます。これらは、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。
- **パラメーターのデフォルト値**: 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からです。

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。

### 6.1.3. 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.yaml**: これはオーバークラウド環境を作成するために使用する主要なテンプレートファイルです。
- **overcloud-resource-registry-puppet.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。
- **environments**: オーバークラウドのデプロイメントに適用する環境ファイルのサンプルが含まれるディレクトリーです。

## 6.2. 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6.1 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

ネットワーク設定の他のサンプルについては、「[ネットワークのプランニング](#)」を参照してください。

### 6.2.1. 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。

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](#)」のセクションを参照してください。

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

■

## linux\_bridge

複数の **interface**、**linux\_bond**、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「[付録E 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F 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、使用していないインターフェースを **ovs\_bridge** デバイスから削除し、DHCP とデフォルトのルート設定を無効にします。

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

## 6.2.2. 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の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。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。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
```

お使いの環境に関連したパラメーターが含まれるように環境ファイルを変更します。特に以下のパラメーターを確認します。

- **resource\_registry** 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を記載する必要があります。「[環境ファイル](#)」を参照してください。
- **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る必要があります。これらのオプションについての詳しい参考情報は「[付録G ネットワーク環境のオプション](#)」を参照してください。

例

```
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
# Set to "br-ex" if using floating IPs on native VLAN on bridge br-ex
NeutronExternalNetworkBridge: ""
# Customize bonding options if required
BondInterfaceOvsOptions:
    "bond_mode=balance-slb"
```

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H 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。

## 6.2.3. 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、**ServiceNetMap** を変更して、ストレージ管理ネットワークの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
        KeystonePublicApiNetwork: internal_api
        NeutronApiNetwork: internal_api
        HeatApiNetwork: internal_api
        NovaApiNetwork: internal_api
        NovaMetadataNetwork: internal_api
        NovaVncProxyNetwork: internal_api
        NovaLibvirtNetwork: internal_api
        SwiftMgmtNetwork: storage          # Change 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       # Change from 'storage_mgmt'
        CephPublicNetwork: storage
        ControllerHostnameResolveNetwork: internal_api
        ComputeHostnameResolveNetwork: internal_api
        BlockStorageHostnameResolveNetwork: internal_api
        ObjectStorageHostnameResolveNetwork: internal_api
        CephStorageHostnameResolveNetwork: storage
        NovaColdMigrationNetwork: ctlplane
        NovaLibvirtNetwork: internal_api
    ...
```

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。

#### 6.2.4. 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、これはデプロイ可能なネットワークとポートの包括的なセットです。

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

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

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



```
tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Controller::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml

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

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

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

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

このファイルの最初のセクションには、**OS::TripleO::Network::\*** リソースのリソースレジストリーの宣言が含まれます。デフォルトでは、これらのリソースは **noop.yaml** ファイルを参照しており、このファイルではネットワークは作成されません。ネットワークごとの YAML ファイルにあるリソースを参照すると、ネットワークの作成が有効化されます。

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。

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 **noop.yaml** 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: /usr/share/openstack-tripleo-heat-
templates/network/noop.yaml
OS::TripleO::Network::Storage: /usr/share/openstack-tripleo-heat-
templates/network/storage.yaml
OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-
templates/network/tenant.yaml

# Port assignments for the VIPs
OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
OS::TripleO::Network::Ports::StorageMgmtVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml
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: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml
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: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.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/noop.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/noop.yaml

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
```

**noop.yaml** 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。

### 6.3. 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ル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。

#### 6.3.1. 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** の機能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**



### 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 ...
```

## 6.3.2. 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、director は特定のカスタムホスト名を各ノードに割り当てることもできます。システムの場所 (例: **rack2-row12**) を定義する必要がある場合や、インベントリー ID を照合する必要がある場合、またはカスタムのホスト名が必要となるその他の状況において、カスタムのホスト名は便利です。

ノードのホスト名をカスタマイズするには、「特定のノード ID の割り当て」で作成した **scheduler\_hints\_env.yaml** ファイルなどの環境ファイルで **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 目目の値は、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。

## 6.3.3. 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**

各パラメーターは、アドレスの一覧へのネットワーク名のマッピングです。各ネットワーク種別には、そのネットワークにあるノード数と同じ数のアドレスが最低でも必要です。director はアドレスを順番に割り当てます。各種別の最初のノードは、適切な一覧にある最初のアドレスが割り当てられ、2 番目のノードは 2 番目のアドレスというように割り当てられていきます。

たとえば、オーバークラウドに 3 つの Ceph Storage ノードが含まれる場合には、CephStorageIPs パラメーターは以下のようになります。

```
CephStorageIPs:
  storage:
    - 172.16.1.100
    - 172.16.1.101
    - 172.16.1.102
  storage_mgmt:
```

- 172.16.3.100
- 172.16.3.101
- 172.16.3.102

最初の Ceph Storage ノードは 172.16.1.100 と 172.16.3.100 の 2 つのアドレスを取得し、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。

デプロイメント中にこの設定を適用するには、**openstack overcloud deploy** コマンドで環境ファイルを指定してください。ネットワーク分離の機能を使用する場合には(「[ネットワークの分離](#)」を参照)、このファイルを **network-isolation.yaml** ファイルの後に追加します。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 ~/templates/ips-from-pool-all.yaml [OTHER OPTIONS]
```

#### 6.3.4. 予測可能な仮想 IP の割り当て

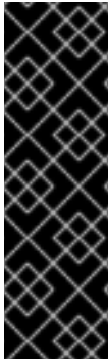
director は、各ノードの予測可能な IP アドレスの定義に加えて、クラスター化されたサービス向けに予測可能な仮想 IP (VIP) を定義する同様の機能も提供します。この定義を行うには、「[ネットワーク環境ファイルの作成](#)」で作成したネットワークの環境ファイルを編集して、**parameter\_defaults** セクションに仮想 IP のパラメーターを追加します。

```
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。

## 6.4. 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) のサービスとオーバー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タノードは、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タ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タ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タ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タ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)

### 6.4.1. コンテナ化されたコンピュータの環境ファイル (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タ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タ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タノードと Heat フックを設定して director と通信します。

#### OS::TripleO::ComputePostDeployment

コンピュータ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_0
vs, 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。

### 6.4.2. 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](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

```

このコマンドでは、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             |
+-----+-----+-----+

```

### 6.4.3. 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 のコンテナー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「[Chapter 2. Get Started with Docker Formatted Container Images](#)」の説明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/rhosp9_tech_preview/openstack-nova-
compute:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-data:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-nova-
libvirt:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-neutron-
openvswitch-agent:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-openvswitch-
vswitchd:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-openvswitch-db-
server:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_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/rhosp9_tech_preview/openstack-nova-
compute:latest localhost:8787/registry.access.redhat.com/openstack-nova-
compute:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-data:latest
localhost:8787/registry.access.redhat.com/openstack-data:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-nova-
libvirt:latest localhost:8787/registry.access.redhat.com/openstack-nova-
libvirt:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_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/rhosp9_tech_preview/openstack-openvswitch-
vswitchd:latest localhost:8787/registry.access.redhat.com/openstack-
openvswitch-vswitchd:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_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/rhosp9_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** をお使いのレジストリーの 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は、必要な Docker イメージと、コンテナ化されたコンピュートの設定が含まれ、このレジストリーを使用する準備が整いました。

#### 6.4.4. 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。「[ネットワークの分離](#)」から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。

## 6.5. 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。

## 6.6. 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。

## 6.7. 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** に設定してください。

#### CinderEnableRbdBackend

Ceph Storage バックエンドを有効にするパラメーター。 **false** に設定してください。

#### CinderEnableNfsBackend

NFS バックエンドを有効にするパラメーター。 **true** に設定してください。

#### 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** に設定してください。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)

#### GlanceFilePcmkManage

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。

#### GlanceFilePcmkFstype

Pacemaker がイメージストレージ用に使用するファイルシステムの種別を定義するパラメーター。 **nfs** に設定します。

#### GlanceFilePcmkDevice

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)

#### GlanceFilePcmkOptions

イメージストレージ用の NFS マウントオプション

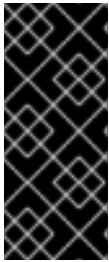
環境ファイルのオプションは、以下の例のようになります。

```
parameter_defaults:
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: false
  CinderEnableNfsBackend: true
  NovaEnableRbdBackend: false
```

```
GlanceBackend: 'file'

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

GlanceFilePcmkManage: true
GlanceFilePcmkFstype: 'nfs'
GlanceFilePcmkDevice: '192.0.2.230:/glance'
GlanceFilePcmkOptions:
'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。

## 6.8. 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。

### 専用の Ceph Storage Cluster を使用する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。

## 6.9. サードパーティーのストレージの設定

director には、環境ファイルが 2 つ含まれており、以下のサードパーティー製のストレージプロバイダーの設定に役立ちます。

### Dell Storage Center

Block Storage (cinder) サービス用に単一の Dell 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 EqualLogic

Block Storage (cinder) サービス用に単一の Dell EqualLogic 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-eqlx-config.yaml` にあります。

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

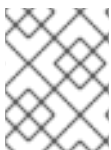
### NetApp ブロックストレージ

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-dellsc-config.yaml/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。

## 6.10. 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対して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は、パブリック API エンドポイントの SSL/TLS を有効化するために追加の環境ファイルが必要という意味です。



### 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「[ネットワークの分離](#)」を参照してください。

秘密鍵と認証局からの証明書が作成されていることを確認します。有効な SSL/TLS 鍵および認証局ファイルの作成に関する情報は、「[付録A SSL/TLS 証明書の設定](#)」を参照してください。

### 6.10.1. 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。

## SSLKey

秘密鍵の内容を **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-----
MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP01V09hkJZnGP6qb6wtYUoy1bVP7
...
ctlKn3rAAadyumi4JDjESAXHIKFjJN0LrBmpQyES4XpZUC7yhqPaU
-----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。

## EndpointMap

**EndpointMap** には、HTTPS および HTTP 通信を使用したサービスのマッピングが含まれます。SSL 通信に DNS を使用する場合は、このセクションをデフォルト設定のままにしておいてください。ただし、SSL 証明書の共通名に IP アドレスを使用する場合は (「[付録A SSL/TLS 証明書の設定](#)」参照)、**CLOUDNAME** のインスタンスをすべて **IP\_ADDRESS** に置き換えてください。これには以下のコマンドを使用してください。

```
$ sed -i 's/CLOUDNAME/IP_ADDRESS/' ~/templates/enable-tls.yaml
```



### 重要

**IP\_ADDRESS** または **CLOUDNAME** は、実際の値に置き換えしないでください。Heat により、オーバークラウドの作成時にこれらの変数が適切な値に置き換えられます。

## 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

```



### 6.10.2. 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::NodeTLSCADData

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

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

### 6.10.3. 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: ["10.0.0.1"]
```

#### 6.10.4. オーバークラウド作成時の環境ファイルの追加

「[7章 オーバークラウドの作成](#)」に記載のデプロイメントのコマンド (**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。以下の順番にこのセクションから環境ファイルを追加します。

- 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**)

例

```
$ openstack overcloud deploy --templates [...] -e
/home/stack/templates/enable-tls.yaml -e ~/templates/cloudname.yaml -e
~/templates/inject-trust-anchor.yaml
```

### 6.11. ベースパラメーターの設定

オーバークラウドの Heat テンプレートには、**openstack overcloud deploy** コマンドを実行する前に設定する基本パラメーターのセットが記載されています。これらの基本パラメーターは、環境ファイルの **parameter\_defaults** セクションに追加してから、**openstack overcloud deploy** コマンドでその環境ファイルを指定します。

オーバークラウド用の基本パラメーターの全一覧は、「[付録D 基本パラメーター](#)」を参照してください。

#### 例 1: タイムゾーンの設定

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'
```

#### 例 2: Layer 3 High Availability (L3HA) の無効化

OpenStack Networking には、Layer 3 High Availability (L3HA) を無効化する必要がある場合には、以下の設定を環境ファイルに追加します。

```
parameter_defaults:
  NeutronL3HA: False
```

#### 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新コンポーネント (**gnocchi**) が含まれています。Red Hat OpenStack Platform では、デフォルトの Ceilometer dispatcher を切り替えて、標準のデータベースの代わりにこの新コンポーネントを使用することができます。これは、**CeilometerMeterDispatcher** で設定します。値は、以下のいずれかを指定します。

- **database**: Ceilometer dispatcher に 標準の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。
- **gnocchi**: Ceilometer dispatcher に新しい時系列データベースを使用します。

時系列データベースを切り替えるには、環境ファイルに以下の行を追加します。

```
parameter_defaults:
    CeilometerMeterDispatcher: gnocchi
```

### 重要

**CeilometerMeterDispatcher** パラメーターを **gnocchi** に設定して Red Hat OpenStack Platform 9 をインストールする場合は、インストールが完了した後に **openstack-ceilometer-collector** サービスを再起動して、コレクターが **gnocchi** と正常に通信できるようにします。このサービスを再起動するには、以下のコマンドを使用してください。

```
$ sudo pcs resource restart openstack-ceilometer-collector-clone
```

gnocchi ユーザーを keystone に登録する **keystone init** は、インストールプロセスの一番最後に実行されるので、このステップが必要となります。そのため、コレクターの起動時に gnocchi ユーザーは keystone のカタログには入っていません。インストール後にコレクターを再起動することにより、**ceilometer** が gnocchi と通信して、適切にデータをディスパッチできるようになります。この問題は、Red Hat OpenStack Platform の次のリリースで解決されます。

## 例 4: RabbitMQ ファイル記述子の上限の設定

設定によっては、RabbitMQ サーバーのファイル記述子の上限を増やす必要があります。**RabbitFDLimit** パラメーターを必要な上限値に設定します。

```
parameter_defaults:
    RabbitFDLimit: 65536
```

## 6.12. オーバークラウドの登録

オーバークラウドは、Red Hat コンテンツ配信ネットワーク、Red Hat Satellite 5 または 6 サーバーにノードを登録する方法を提供します。これは、環境ファイルまたはコマンドラインのいずれかを使用して実行することができます。

### 6.12.1. 方法 1: コマンドライン

デプロイメントのコマンド (**openstack overcloud deploy**) は、一連のオプションを使用して登録情報を定義します。「[オーバークラウドのパラメーター設定](#)」の表には、これらのオプションと説明についてまとめています。これらのオプションは、「[7章 オーバークラウドの作成](#)」でデプロイメントのコマンドを実行する時に追加してください。以下に例を示します。

```
# openstack overcloud deploy --templates --rhel-reg --reg-method satellite
--reg-sat-url http://example.satellite.com --reg-org MyOrg --reg-activation-key MyKey --reg-force [...]
```

## 6.12.2. 方法 2: 環境ファイル

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。**true** に設定して有効にするか、**false** に設定して無効にします。

### 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](https://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

### 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。有効化するリポジトリについては、[「リポジトリの要件」](#)を参照してください。

#### **rhel\_reg\_activation\_key**

登録に使用するアクティベーションキー

#### **rhel\_reg\_user、rhel\_reg\_password**

登録用のユーザー名およびパスワード。可能な場合には、登録用のアクティベーションキーを使用します。

#### **rhel\_reg\_machine\_name**

マシン名。ノードのホスト名を使用するには、空欄のままにします。

#### **rhel\_reg\_force**

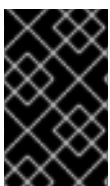
登録のオプションを強制するには **true** に設定します (例: ノードの再登録時など)。

#### **rhel\_reg\_sat\_repo**

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

[「7章 オーバークラウドの作成」](#)に記載の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、[「オーバークラウドの設定前のカスタマイズ」](#)を参照してください。

## **6.13. 初回起動で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。

## 6.14. オーバークラウドの設定前のカスタマイズ

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、カスタム設定を提供するリソースのセットを用意します。これには、以下のリソースが含まれます。

### 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 CephStorage ノードに適用される追加の設定

**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:
  ExtraPreConfig:
    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}

  ExtraPreDeployment:
    type: OS::Heat::SoftwareDeployment
    properties:
      config: {get_resource: ExtraPreConfig}
      server: {get_param: server}
      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: [ExtraPreDeployment, deploy_stdout]}
```

この例では、**resources** セクションに以下が含まれています。

## ExtraPreConfig

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。

## ExtraPreDeployment

これは、**ExtraPreConfig** 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。

- **server** 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。
- **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::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。

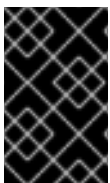
```
resource_registry:
  OS::TripleO::NodeExtraConfig: /home/stack/templates/nameserver.yaml

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

これらのリソースは、それぞれ 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、リソースごとに使用する Heat テンプレートが上書きされます。

## 6.15. オーバークラウドの設定後のカスタマイズ

オーバークラウドの作成が完了してから、オーバークラウドの初回作成時またはその後の更新時に追加の設定が必要となる状況が発生する可能性があります。このような場合は、**OS::TripleO::NodeExtraConfigPost** リソースを使用して、標準の **OS::Heat::SoftwareConfig** 種別を使用した設定を適用します。これにより、メインのオーバークラウド設定が完了してから、追加の設定が適用されます。

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、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:
  ExtraConfig:
    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}

  ExtraDeployments:
    type: OS::Heat::SoftwareDeployments
    properties:
      config: {get_resource: ExtraConfig}
      servers: {get_param: servers}
      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。

### ExtraConfig

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。

### ExtraDeployments

これは、**ExtraConfig** 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。

- **server** 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。
- **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。

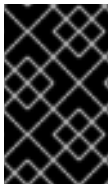
```
resource_registry:
    OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml

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。それ以外に使用すると、使用する Heat テンプレートが上書きされます。

## 6.16. PUPPET 設定データのカスタマイズ

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。

### ExtraConfig

全ノードに追加する設定

### 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

ブロックストレージノードに追加する設定

### ObjectStorageExtraConfig

オブジェクトストレージノードに追加する設定

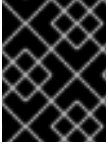
### CephStorageExtraConfig

Ceph ストレージ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。

## 6.17. カスタムの PUPPET 設定の適用

特定の状況では、追加のコンポーネントをオーバークラウドノードにインストールして設定する必要があります。これには、カスタムの Puppet マニフェストを使用して、主要な設定が完了してからノードに適用します。基本的な例として、各ノードに **motd** をインストールするとします。そのためにはまず、Puppet 設定を起動する Heat テンプレート (`/home/stack/templates/custom_puppet_config.yaml`) を作成します。

```
heat_template_version: 2014-10-16

description: >
  Run Puppet extra configuration to set new MOTD

parameters:
  servers:
    type: json

resources:
  ExtraPuppetConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      config: {get_file: motd.pp}
      group: puppet
      options:
        enable_hiera: True
        enable_facter: False

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

これにより、テンプレート内に `/home/stack/templates/motd.pp` が追加され、設定のためにノードに渡されます。**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::TripleO::NodeExtraConfigPost**: リソース種別として Heat テンプレートを登録する環境ファイル (`/home/stack/templates/puppet_post_config.yaml`) を作成します。

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

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。

## 6.18. 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
```

これにより、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** を実行する際には、**--templates** オプションを使用してローカルのテンプレートディレクトリーを指定します。これについては、本ガイドの後半に記載しています ([「7章 オーバークラウドの作成」](#)を参照)。



### 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以下の項に記載する方法を使用することを推奨します。

- [「オーバークラウドの設定前のカスタマイズ」](#)
- [「オーバークラウドの設定後のカスタマイズ」](#)
- [「Puppet 設定データのカスタマイズ」](#)
- [「カスタムの Puppet 設定の適用」](#)

Heat テンプレートコレクションのコピーを作成する場合には、**git** などのバージョン管理システムを使用して、テンプレートに加えられた変更をトラッキングすべきです。

## 第7章 オーバークラウドの作成

OpenStack 環境作成における最後の段階では、**openstack overcloud deploy** コマンドを実行して OpenStack 環境を作成します。このコマンドを実行する前に、キーオプションやカスタムの環境ファイルの追加方法を十分に理解しておく必要があります。本章では、**openstack overcloud deploy** コマンドと、それに関連するオプションについて説明します。



### 警告

バックグラウンドプロセスとして **openstack overcloud deploy** を実行しないでください。バックグラウンドのプロセスとして開始された場合にはオーバークラウドの作成は途中で停止してしまう可能性があります。

### 7.1. オーバークラウドのパラメーター設定

以下の表では、**openstack overcloud deploy** コマンドを使用する際の追加パラメーターを一覧表示します。

表7.1 デプロイメントパラメーター

パラメーター	説明	例
--templates [TEMPLATES]	デプロイする Heat テンプレートが格納されているディレクトリー。空欄にした場合には、コマンドはデフォルトのテンプレートの場所である <b>/usr/share/openstack-tripleo-heat-templates/</b> を使用します。	~/templates/my-overcloud
--stack STACK	作成または更新するスタックの名前	overcloud
-t [TIMEOUT], --timeout [TIMEOUT]	デプロイメントのタイムアウト (分単位)	240
--control-scale [CONTROL_SCALE]	スケールアウトするコントローラーノード数	3
--compute-scale [COMPUTE_SCALE]	スケールアウトするコンピューターノード数	3
--ceph-storage-scale [CEPH_STORAGE_SCALE]	スケールアウトする Ceph Storage ノードの数	3

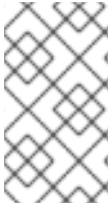
--block-storage-scale [BLOCK_STORAGE_SCALE]	スケールアウトする Cinder ノード数	3
--swift-storage-scale [SWIFT_STORAGE_SCALE]	スケールアウトする Swift ノード数	3
--control-flavor [CONTROL_FLAVOR]	コントローラーノードに使用するフレーバー	control
--compute-flavor [COMPUTE_FLAVOR]	コンピュートノードに使用するフレーバー	compute
--ceph-storage-flavor [CEPH_STORAGE_FLAVOR]	Ceph Storage ノードに使用するフレーバー	ceph-storage
--block-storage-flavor [BLOCK_STORAGE_FLAVOR]	Cinder ノードに使用するフレーバー	cinder-storage
--swift-storage-flavor [SWIFT_STORAGE_FLAVOR]	Swift Storage ノードに使用するフレーバー	swift-storage
--neutron-flat-networks [NEUTRON_FLAT_NETWORKS]	(非推奨) 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。デフォルトは「datacentre」に設定され、外部ネットワークの作成が許可されます。	datacentre
--neutron-physical-bridge [NEUTRON_PHYSICAL_BRIDGE]	(非推奨) 各ハイパーバイザーで作成する Open vSwitch ブリッジ。デフォルト値は「br-ex」で、通常この値は変更する必要はないはずです。	br-ex
--neutron-bridge-mappings [NEUTRON_BRIDGE_MAPPINGS]	(非推奨) 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。	datacentre:br-ex
--neutron-public-interface [NEUTRON_PUBLIC_INTERFACE]	(非推奨) 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。	nic1、eth0
--neutron-network-type [NEUTRON_NETWORK_TYPE]	(非推奨) Neutron のテナントネットワーク種別	gre または vxlan

--neutron-tunnel-types [NEUTRON_TUNNEL_TYPES]	(非推奨) 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。	vxlan gre,vxlan
--neutron-tunnel-id-ranges [NEUTRON_TUNNEL_ID_RANGES]	(非推奨) 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	1:1000
--neutron-vni-ranges [NEUTRON_VNI_RANGES]	(非推奨) 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	1:1000
--neutron-disable-tunneling	(非推奨) 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無効化します。	
--neutron-network-vlan-ranges [NEUTRON_NETWORK_VLAN_RANGES]	(非推奨) 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 「 <b>datacentre</b> 」上の VLAN を許可するように設定されています。	datacentre:1:1000
--neutron-mechanism-drivers [NEUTRON_MECHANISM_DRIVERS]	(非推奨) 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。	openvswitch,l2population
--libvirt-type [LIBVIRT_TYPE]	ハイパーバイザーに使用する仮想化タイプ	kvm、qemu
--ntp-server [NTP_SERVER]	時刻の同期に使用する Network Time Protocol (NTP) サーバー。コンマ区切りリストで複数の NTP サーバーを指定することも可能です (例: <b>--ntp-server 0.centos.pool.org,1.centos.pool.org</b> )。高可用性クラスターのデプロイメントの場合には、コントローラーが一貫して同じタイムソースを参照することが必須となります。標準的な環境には、確立された慣行によって、NTP タイムソースがすでに指定されている可能性がある点に注意してください。	pool.ntp.org

<code>--no-proxy [NO_PROXY]</code>	環境変数 <code>no_proxy</code> のカスタム値を定義します。これにより、プロキシ通信からの特定のドメイン拡張は除外されます。	
<code>--overcloud-ssh-user</code> <code>OVERCLOUD_SSH_USER</code>	オーバークラウドノードにアクセスする SSH ユーザーを定義します。通常、SSH アクセスは <b>heat-admin</b> ユーザーで実行されます。	<code>ocuser</code>
<code>-e [EXTRA HEAT TEMPLATE], --extra-template [EXTRA HEAT TEMPLATE]</code>	オーバークラウドデプロイメントに渡す追加の環境ファイル。複数回指定することが可能です。 <b>openstack overcloud deploy</b> コマンドに渡す環境ファイルの順序が重要である点に注意してください。たとえば、逐次的に渡される各環境ファイルは、前の環境ファイルのパラメーターを上書きします。	<code>-e ~/templates/my-config.yaml</code>
<code>--environment-directory</code>	デプロイメントに追加する環境ファイルが格納されているディレクトリー。このコマンドは、これらの環境ファイルを番号順で処理した後に、アルファベット順で処理します。	<code>--environment-directory</code> <code>~/templates</code>
<code>--validation-errors-fatal</code>	オーバークラウドの作成プロセスでは、一式のデプロイメントチェックが行われます。このオプションは、事前デプロイメントチェックで何らかのエラーが発生した場合に存在します。どのようなエラーが発生してもデプロイメントが失敗するので、このオプションを使用することを推奨します。	
<code>--validation-warnings-fatal</code>	オーバークラウドの作成プロセスで、デプロイ前に一連のチェックを行います。このオプションは、デプロイ前のチェックでクリティカルではない警告が発生した場合に存在します。	
<code>--dry-run</code>	オーバークラウドに対する検証チェックを実行しますが、オーバークラウドを実際には作成しません。	



<code>--force-postconfig</code>	オーバークラウドのデプロイ後の設定を強制的に行います。	<code>--force-postconfig</code>
<code>--answers-file ANSWERS_FILE</code>	引数とパラメーターが記載されたYAML ファイルへのパス	<code>--answers-file ~/answers.yaml</code>
<code>--rhel-reg</code>	カスタマーポータルまたはSatellite 6 にオーバークラウドノードを登録します。	
<code>--reg-method</code>	オーバークラウドノードに使用する登録メソッド	Red Hat Satellite 6 または Red Hat Satellite 5 は <b>satellite</b> 、カスタマーポータルは <b>portal</b>
<code>--reg-org [REG_ORG]</code>	登録に使用する組織	
<code>--reg-force</code>	すでに登録済みの場合でもシステムを登録します。	
<code>--reg-sat-url [REG_SAT_URL]</code>	<p>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、<a href="https://satellite.example.com">https://satellite.example.com</a> ではなく <a href="http://satellite.example.com">http://satellite.example.com</a> 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 <b>katello-ca-consumer-latest.noarch.rpm</b> ファイルを取得して <b>subscription-manager</b> に登録し、<b>katello-agent</b> 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、<b>RHN-ORG-TRUSTED-SSL-CERT</b> ファイルを取得して <b>rhnreg_ks</b> に登録します。</p>	
<code>--reg-activation-key [REG_ACTIVATION_KEY]</code>	登録に使用するアクティベーションキー	



## 注記

オプションの完全一覧については、以下のコマンドを実行します。

```
$ openstack help overcloud deploy
```

## 7.2. オーバークラウド作成時の環境ファイルの追加

オーバークラウドをカスタマイズするには、**-e** を指定して、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、環境ファイルの順番は重要です。以下の一覧は、環境ファイルの順序の例です。

- Heat テンプレートコレクションの初期化ファイル (**environments/network-isolation.yaml**) を含むネットワーク分離ファイルと、次にカスタムの NIC 設定ファイル。ネットワークの分離についての詳しい情報は、「[ネットワークの分離](#)」を参照してください。
- 外部のロードバランシングの環境ファイル
- Ceph Storage、NFS、iSCSI などのストレージ環境ファイル
- Red Hat CDN または Satellite 登録用の環境ファイル
- その他のカスタム環境ファイル

**-e** オプションを使用してオーバークラウドに追加した環境ファイルはいずれも、オーバークラウドのスタック定義の一部となります。

また同様に、**--environment-directory** オプションを使用して、環境ファイルを格納しているディレクトリー全体を追加することも可能です。デプロイメントコマンドにより、このディレクトリー内の環境ファイルは、最初に番号順、その後にアルファベット順で処理されます。この方法を使用する場合には、ファイル名に数字のプレフィックスを使用することを推奨します。以下に例を示します。

```
$ ls -1 ~/templates
10-network-isolation.yaml
20-network-environment.yaml
30-storage-environment.yaml
40-rhel-registration.yaml
```

director は、「[8章 オーバークラウド作成後のタスクの実行](#)」に記載の再デプロイおよびデプロイ後の機能にこれらの環境ファイルを必要とします。これらのファイルが含まれていない場合には、オーバークラウドが破損する可能性があります。

オーバークラウド設定を後で変更する予定の場合には、以下の作業を行う必要があります。

1. カスタムの環境ファイルおよび Heat テンプレートのパラメーターを変更します。
2. 同じ環境ファイルを指定して **openstack overcloud deploy** コマンドを再度実行します。

オーバークラウドを手動で編集しても、director を使用してオーバークラウドスタックの更新を行う際に director の設定で上書きされてしまうので、設定は直接編集しないでください。

## 重要

後で使用および変更するために、最初のデプロイメントコマンドを保存しておきます。たとえば、**deploy-overcloud.sh** という名前のスクリプトファイルでデプロイメントコマンドを保存するには、以下のように編集します。

```
#!/bin/bash
openstack overcloud deploy --templates \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/network-isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  -t 150 \
  --control-scale 3 \
  --compute-scale 3 \
  --ceph-storage-scale 3 \
  --swift-storage-scale 0 \
  --block-storage-scale 0 \
  --compute-flavor compute \
  --control-flavor control \
  --ceph-storage-flavor ceph-storage \
  --swift-storage-flavor swift-storage \
  --block-storage-flavor block-storage \
  --ntp-server pool.ntp.org \
  --libvirt-type qemu
```

これにより、将来オーバークラウドに変更を加えたり、スケーリングしたりする際に使用するオーバークラウドのデプロイメントコマンドのパラメーターと環境ファイルが保持されるので、今後オーバークラウドをカスタマイズする際にこのスクリプトを編集して再度実行することができます。

## 7.3. オーバークラウドの作成例

以下のコマンドは、カスタムの環境ファイルを追加で指定して、どのようにオーバークラウドの作成を開始するかに関する例です。

```
$ openstack overcloud deploy --templates \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-
isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  --control-scale 3 \
  --compute-scale 3 \
  --ceph-storage-scale 3 \
  --control-flavor control \
  --compute-flavor compute \
  --ceph-storage-flavor ceph-storage \
  --ntp-server pool.ntp.org \
```

このコマンドでは、以下の追加オプションも使用できます。

- **--templates: /usr/share/openstack-tripleo-heat-templates** の Heat テンプレートコレクションを使用してオーバークラウドを作成します。

- **-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml: -e** オプションは、オーバークラウドデプロイメントに別の環境ファイルを追加します。この場合は、ネットワーク分離の設定を初期化する環境ファイルです。
- **-e ~/templates/network-environment.yaml: -e** オプションは、オーバークラウドデプロイメントに別の環境ファイルを追加します。この場合は、[「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環境ファイルです。
- **-e ~/templates/storage-environment.yaml: -e** オプションは、オーバークラウドデプロイメントに別の環境ファイルを追加します。この場合は、ストレージの設定を初期化する環境ファイルです。
- **--control-scale 3:** コントローラーノードを 3 台にスケーリングします。
- **--compute-scale 3:** コンピュートノードを 3 台にスケーリングします。
- **--ceph-storage-scale 3:** Ceph Storage ノードを 3 台にスケーリングします。
- **--control-flavor control:** 対象のコントローラーノードに特定のフレーバーを使用します。
- **--compute-flavor compute:** コンピュートノードに特定のフレーバーを使用します。
- **--ceph-storage-flavor ceph-storage:** Ceph Storage ノードに特定のフレーバーを使用します。
- **--ntp-server pool.ntp.org:** 時刻の同期に NTP サーバーを使用します。これは、コントローラーノードクラスターの同期を保つ際に便利です。

## 7.4. オーバークラウド作成の監視

オーバークラウドの作成プロセスが開始され、director によりノードがプロビジョニングされます。このプロセスは完了するまで多少時間がかかります。オーバークラウドの作成のステータスを確認するには、**stack** ユーザーとして別のターミナルを開き、以下を実行します。

```
$ source ~/stackrc                # Initializes the stack user to use the
CLI commands
$ heat stack-list --show-nested
```

**heat stack-list --show-nested** コマンドは、オーバークラウド作成の現在のステージを表示します。

## 7.5. オーバークラウドへのアクセス

director は、director ホストからオーバークラウドに対話するための設定を行い、認証をサポートするスクリプトを作成して、**stack** ユーザーのホームディレクトリーにこのファイル (**overcloudrc**) を保存します。このファイルを使用するには、以下のコマンドを実行します。

```
$ source ~/overcloudrc
```

これで、director のホストの CLI からオーバークラウドと対話するために必要な環境変数が読み込まれます。director のホストとの対話に戻るには、以下のコマンドを実行します。

```
$ source ~/stackrc
```

■

オーバークラウドの各ノードには、**heat-admin** と呼ばれるユーザーが含まれます。**stack** ユーザーには、各ノードに存在するこのユーザーに SSH 経由でアクセスすることができます。SSH でノードにアクセスするには、希望のノードの IP アドレスを特定します。

```
$ nova list
```

次に、**heat-admin** ユーザーとノードの IP アドレスを使用して、ノードに接続します。

```
$ ssh heat-admin@192.0.2.23
```

## 7.6. オーバークラウド作成の完了

これでオーバークラウドの作成が完了しました。作成後の機能については、「[8章 オーバークラウド作成後のタスクの実行](#)」を参照してください。

## 第8章 オーバークラウド作成後のタスクの実行

本章では、任意のオーバークラウドを作成後に実行するタスクについて考察します。

### 8.1. オーバークラウドのテナントネットワークの作成

オーバークラウドには、インスタンス用のテナントネットワークが必要です。source コマンドで **overcloud** を読み込んで、Neutron で初期テナントネットワークを作成します。以下に例を示します。

```
$ source ~/overcloudrc
$ neutron net-create default
$ neutron subnet-create --name default --gateway 172.20.1.1 default
172.20.0.0/16
```

上記のステップにより、**default** という名前の基本的な Neutron ネットワークが作成されます。オーバークラウドは、内部 DHCP メカニズムを使用したこのネットワークから、IP アドレスを自動的に割り当てます。

**neutron net-list** で作成したネットワークを確認します。

```
$ neutron net-list
+-----+-----+-----+
| id                  | name          | subnets |
+-----+-----+-----+
| 95fadaa1-5dda-4777... | default       | 7e060813-35c5-462c-a56a-1c6f8f4f332f 172.20.0.0/16 |
+-----+-----+-----+
```

### 8.2. オーバークラウドの外部ネットワークの作成

先ほど「[ネットワークの分離](#)」で外部ネットワークに使用するノードインターフェースを設定しましたが、インスタンスに Floating IP を割り当てることができるように、オーバークラウドでこのネットワークを作成する必要があります。

#### ネイティブ VLAN の使用

以下の手順では、外部ネットワーク向けの専用インターフェースまたはネイティブの VLAN が設定されていることが前提です。

source コマンドで **overcloud** を読み込み、Neutron で外部ネットワークを作成します。以下に例を示します。

```
$ source ~/overcloudrc
$ neutron net-create public --router:external --provider:network_type flat
--provider:physical_network datacentre
$ neutron subnet-create --name public --enable_dhcp=False --allocation-
pool=start=10.1.1.51,end=10.1.1.250 --gateway=10.1.1.1 public 10.1.1.0/24
```

以下の例では、**public** という名前のネットワークを作成します。オーバークラウドには、デフォルトの Floating IP プールにこの特定の名前が必要です。このネットワークは、「[オーバークラウドの検証](#)」の検証テストでも重要となります。

このコマンドにより、ネットワークと **datacentre** の物理ネットワークのマッピングも行われます。デフォルトでは、**datacentre** は **br-ex** ブリッジにマッピングされます。オーバークラウドの作成時にカスタムの Neutron の設定を使用していない限りは、このオプションはデフォルトのままにしてください。

### 非ネイティブ VLAN の使用

ネイティブ VLAN を使用しない場合には、以下のコマンドでネットワークを VLAN に割り当てます。

```
$ source ~/overcloudrc
$ neutron net-create public --router:external --provider:network_type vlan
--provider:physical_network datacentre --provider:segmentation_id 104
$ neutron subnet-create --name public --enable_dhcp=False --allocation-
pool=start=10.1.1.51,end=10.1.1.250 --gateway=10.1.1.1 public 10.1.1.0/24
```

**provider:segmentation\_id** の値は、使用する VLAN を定義します。この場合は、104 を使用します。

**neutron net-list** で作成したネットワークを確認します。

```
$ neutron net-list
+-----+-----+-----+
+-----+
| id                  | name          | subnets
|
+-----+-----+-----+
+-----+
| d474fe1f-222d-4e32... | public        | 01c5f621-1e0f-4b9d-9c30-
7dc59592a52f 10.1.1.0/24 |
+-----+-----+-----+
+-----+
```

## 8.3. 追加の FLOATING IP ネットワークの作成

Floating IP ネットワークは、以下の条件を満たす限りは、**br-ex** だけでなく、どのブリッジにも使用することができます。

- ネットワーク環境ファイルで、**NeutronExternalNetworkBridge** が **''** に設定されていること。
- デプロイ中に追加のブリッジをマッピングしていること。たとえば、**br-floating** という新規ブリッジを **floating** という物理ネットワークにマッピングするには、以下のコマンドを実行します。

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-
tripleo-heat-templates/environments/network-isolation.yaml -e
~/templates/network-environment.yaml --neutron-bridge-mappings
datacentre:br-ex,floating:br-floating
```

オーバークラウドの作成後に Floating IP ネットワークを作成します。

```
$ neutron net-create ext-net --router:external --provider:physical_network
floating --provider:network_type vlan --provider:segmentation_id 105
$ neutron subnet-create --name ext-subnet --enable_dhcp=False --
allocation-pool start=10.1.2.51,end=10.1.2.250 --gateway 10.1.2.1 ext-net
10.1.2.0/24
```

## 8.4. オーバークラウドのプロバイダーネットワークの作成

プロバイダーネットワークは、デプロイしたオーバークラウドの外部に存在するネットワークに物理的に接続されたネットワークです。これは、既存のインフラストラクチャーネットワークや、Floating IP の代わりにルーティングによって直接インスタンスに外部アクセスを提供するネットワークを使用することができます。

プロバイダーネットワークを作成する際には、ブリッジマッピングを使用する物理ネットワークに関連付けます。これは、Floating IP ネットワークの作成と同様です。コンピュータードは、仮想マシンの仮想ネットワークインターフェースをアタッチされているネットワークインターフェースに直接接続するため、プロバイダーネットワークはコントローラーとコンピュータの両ノードに追加します。

たとえば、使用するプロバイダーネットワークが br-ex ブリッジ上の VLAN の場合には、以下のコマンドを使用してプロバイダーネットワークを VLAN 201 上に追加します。

```
$ neutron net-create --provider:physical_network datacentre --
provider:network_type vlan --provider:segmentation_id 201 --shared
provider_network
```

このコマンドにより、共有ネットワークが作成されます。また、「--shared」と指定する代わりにテナントを指定することも可能です。そのネットワークは、指定されたテナントに対してのみ提供されます。プロバイダーネットワークを外部としてマークした場合には、そのネットワークでポートを作成できるのはオペレーターのみとなります。

Neutron が DHCP サービスをテナントのインスタンスに提供するように設定するには、プロバイダーネットワークにサブネットを追加します。

```
$ neutron subnet-create --name provider-subnet --enable_dhcp=True --
allocation-pool start=10.9.101.50,end=10.9.101.100 --gateway 10.9.101.254
provider_network 10.9.101.0/24
```

他のネットワークがプロバイダーネットワークを介して外部にアクセスする必要がある場合があります。このような場合には、新規ルーターを作成して、他のネットワークがプロバイダーネットワークを介してトラフィックをルーティングできるようにします。

```
$ neutron router-create external
$ neutron router-gateway-set external provider_network
```

このルーターに他のネットワークを接続します。たとえば、**subnet1** という名前のサブネットがある場合には、以下のコマンドを実行してルーターに接続することができます。

```
$ neutron router-interface-add external subnet1
```

これにより、**subnet1** がルーティングテーブルに追加され、**subnet1** を使用するトラフィックをプロバイダーネットワークにルーティングできるようになります。



## 8.5. オーバークラウドの検証

オーバークラウドは、Tempest を使用して一連の統合テストを実行します。以下の手順では、Tempest を使用してオーバークラウドを検証する方法を説明します。アンダークラウドからこのテストを実行する場合は、アンダークラウドのホストがオーバークラウドの内部 API ネットワークにアクセスできるようにします。たとえば、172.16.0.201/24 のアドレスを使用して内部 API ネットワーク (ID: 201) にアクセスするにはアンダークラウドホストに一時 VLAN を追加します。

```
$ source ~/stackrc
$ sudo ovs-vsctl add-port br-ctlplane vlan201 tag=201 -- set interface
vlan201 type=internal
$ sudo ip l set dev vlan201 up; sudo ip addr add 172.16.0.201/24 dev
vlan201
```

Tempest を実行する前に、**heat\_stack\_owner** ロールがオーバークラウドに存在することを確認してください。

```
$ source ~/overcloudrc
$ openstack role list
+-----+-----+
| ID                                           | Name               |
+-----+-----+
| 6226a517204846d1a26d15aae1af208f          | swiftoperator      |
| 7c7eb03955e545dd86bbfeb73692738b          | heat_stack_owner   |
+-----+-----+
```

このロールが存在しない場合は、作成します。

```
$ keystone role-create --name heat_stack_owner
```

Tempest ツールセットのインストール

```
$ sudo yum install openstack-tempest
```

**stack** ユーザーのホームディレクトリーに **tempest** ディレクトリーを設定して、Tempest スイートをローカルにコピーします。

```
$ mkdir ~/tempest
$ cd ~/tempest
$ /usr/share/openstack-tempest-10.0.0/tools/configure-tempest-directory
```

上記のコマンドにより、Tempest ツールセットのローカルバージョンが作成されます。

オーバークラウドの作成プロセスが完了した後は、director により **~/tempest-deployer-input.conf** というファイルが作成されます。このファイルは、オーバークラウドに関連する Tempest の設定オプションを提供します。このファイルを使用して Tempest を設定するには、以下のコマンドを実行します。

```
$ tools/config_tempest.py --deployer-input ~/tempest-deployer-input.conf -
-debug --create identity.uri $OS_AUTH_URL identity.admin_password
$OS_PASSWORD --network-id d474fe1f-222d-4e32-9242-cd1fefe9c14b
```

`$OS_AUTH_URL` および `$OS_PASSWORD` の環境変数は、以前にソースとして使用した `overcloudrc` ファイルの値セットを使用します。 `--network-id` は「[オーバークラウドの外部ネットワークの作成](#)」で作成した外部ネットワークの UUID です。



### 重要

設定スクリプトにより、Tempest テスト用に Cirros イメージをダウンロードします。director にはインターネットアクセスがあり、インターネットへのアクセスのあるプロキシが使用されるようにしてください。`http_proxy` の環境変数を設定して、コマンドラインの操作にプロキシを使用します。

以下のコマンドを使用して、Tempest テストの全スイートを実行します。

```
$ tools/run-tests.sh
```



### 注記

完全な Tempest テストスイートには、数時間かかる場合があります。代わりに、`.*smoke` オプションを使用してテストの一部を実行します。

```
$ tools/run-tests.sh '.*smoke'
```

各テストはオーバークラウドに対して実行され、次の出力で各テストとその結果が表示されます。同じディレクトリで生成された `tempest.log` ファイルで、各テストの詳しい情報を確認することができます。たとえば、出力で以下のように失敗したテストについて表示される場合があります。

```
{2}
tempest.api.compute.servers.test_servers.ServersTestJSON.test_create_specify_keypair [18.305114s] ... FAILED
```

これは、詳細情報が含まれるログのエントリーと一致します。コロンで区切られたテストの名前空間の最後の 2 つに関するログを検索します。この例では、ログで `ServersTestJSON:test_create_specify_keypair` を検索します。

```
$ grep "ServersTestJSON:test_create_specify_keypair" tempest.log -A 4
2016-03-17 14:49:31.123 10999 INFO tempest_lib.common.rest_client [req-a7a29a52-0a52-4232-9b57-c4f953280e2c ] Request
(ServersTestJSON:test_create_specify_keypair): 500 POST
http://192.168.201.69:8774/v2/2f8bef15b284456ba58d7b149935cbc8/os-keypairs
4.331s
2016-03-17 14:49:31.123 10999 DEBUG tempest_lib.common.rest_client [req-a7a29a52-0a52-4232-9b57-c4f953280e2c ] Request - Headers: {'Content-Type': 'application/json', 'Accept': 'application/json', 'X-Auth-Token': '<omitted>'}
Body: {"keypair": {"name": "tempest-key-722237471"}}
Response - Headers: {'status': '500', 'content-length': '128', 'x-compute-request-id': 'req-a7a29a52-0a52-4232-9b57-c4f953280e2c', 'connection': 'close', 'date': 'Thu, 17 Mar 2016 04:49:31 GMT', 'content-type': 'application/json; charset=UTF-8'}
Body: {"computeFault": {"message": "The server has either erred or
```

```
is incapable of performing the requested operation.", "code": 500}}
_log_request_full /usr/lib/python2.7/site-
packages/tempest_lib/common/rest_client.py:414
```



#### 注記

-A 4 オプションを指定すると次の 4 行が表示されます。この 4 行には、通常要求ヘッダーとボディー、応答ヘッダーとボディーが含まれます。

検証が完了したら、オーバークラウドの内部 API への一時接続を削除します。この例では、以下のコマンドを使用して、以前にアンダークラウドで作成した VLAN を削除します。

```
$ source ~/stackrc
$ sudo ovs-vsctl del-port vlan201
```

## 8.6. コントローラーノードのフェンシング

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

director は、Pacemaker を使用して、高可用性のコントローラーノードクラスターを提供します。Pacemaker は、問題のあるノードをフェンシングするのに役立つ STONITH (Shoot-The-Other-Node-In-The-Head) というプロセスを使用します。デフォルトでは、STONITH はお使いのクラスター上では無効化されているため、Pacemaker がクラスター内の各ノードの電源管理を制御できるように手動で設定する必要があります。



#### 注記

director 上の **stack** ユーザーから、**heat-admin** ユーザーとして各ノードにログインします。オーバークラウドを作成すると自動的に **stack** ユーザーの SSH キーが各ノードの **heat-admin** にコピーされます。

**pcs status** を使用して、クラスターが稼働していることを確認します。

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

**pcs property show** で、STONITH が無効化されていることを確認します。

```
$ sudo pcs property show
Cluster Properties:
cluster-infrastructure: corosync
cluster-name: openstackHA
```

```
dc-version: 1.1.12-a14efad
have-watchdog: false
stonith-enabled: false
```

コントローラーノードには、director がサポートするさまざまな電源管理デバイス用のフェンシングエージェントのセットが実装されています。これには、以下が含まれます。

表8.1 フェンスエージェント

デバイス	タイプ
<b>fence_ipmilan</b>	Intelligent Platform Management Interface (IPMI)
<b>fence_idrac</b> 、 <b>fence_drac5</b>	Dell Remote Access Controller (DRAC)
<b>fence_ilo</b>	Integrated Lights-Out (iLO)
<b>fence_ucs</b>	Cisco UCS: 詳しい情報は「 <a href="#">Configuring Cisco Unified Computing System (UCS) Fencing on an OpenStack High Availability Environment</a> 」の記事を参照してください。
<b>fence_xvm</b> 、 <b>fence_virt</b>	Libvirt と SSH

本項ではこれ以降、IPMI エージェント (**fence\_ipmilan**) を例として使用します。

Pacemaker がサポートする IPMI オプションの完全一覧を表示します。

```
$ sudo pcs stonith describe fence_ipmilan
```

各ノードには、電源管理を制御する IPMI デバイスの設定が必要です。これには、各ノードの Pacemaker に **stonith** デバイスを追加する必要があります。クラスターに以下のコマンドを実行します。



#### 注記

各例の 2 番目のコマンドは、ノードが自らをフェンシングするかどうかを尋ねないようにします。

コントローラーノード 0 の場合:

```
$ sudo pcs stonith create my-ipmilan-for-controller-0 fence_ipmilan
pcmk_host_list=overcloud-controller-0 ipaddr=192.0.2.205 login=admin
passwd=p@55w0rd! lanplus=1 cipher=1 op monitor interval=60s
$ sudo pcs constraint location my-ipmilan-for-controller-0 avoids
overcloud-controller-0
```

コントローラーノード 1 の場合:

```
$ sudo pcs stonith create my-ipmilan-for-controller-1 fence_ipmilan
pcmk_host_list=overcloud-controller-1 ipaddr=192.0.2.206 login=admin
```

```
passwd=p@55w0rd! lanplus=1 cipher=1 op monitor interval=60s
$ sudo pcs constraint location my-ipmilan-for-controller-1 avoids
overcloud-controller-1
```

コントローラーノード 2 の場合:

```
$ sudo pcs stonith create my-ipmilan-for-controller-2 fence_ipmilan
pcmk_host_list=overcloud-controller-2 ipaddr=192.0.2.207 login=admin
passwd=p@55w0rd! lanplus=1 cipher=1 op monitor interval=60s
$ sudo pcs constraint location my-ipmilan-for-controller-2 avoids
overcloud-controller-2
```

以下のコマンドを実行して、すべての STONITH リソースを表示します。

```
$ sudo pcs stonith show
```

以下のコマンドを実行して、特定の STONITH リソースを表示します。

```
$ sudo pcs stonith show [stonith-name]
```

最後に、**stonith** プロパティを **true** に設定して、フェンシングを有効にします。

```
$ sudo pcs property set stonith-enabled=true
```

プロパティを確認します。

```
$ sudo pcs property show
```

## 8.7. オーバークラウド環境の変更

オーバークラウドを変更して、別機能を追加したり、操作の方法を変更したりする場合があります。オーバークラウドを変更するには、カスタムの環境ファイルと Heat テンプレートに変更を加えて、最初に作成したオーバークラウドから **openstack overcloud deploy** コマンドをもう 1 度実行します。たとえば、「[7章 オーバークラウドの作成](#)」の方法を使用してオーバークラウドを作成した場合には、以下のコマンドを再度実行します。

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

director は Heat 内の **overcloud** スタックを確認してから、環境ファイルと Heat テンプレートのあるスタックで各アイテムを更新します。オーバークラウドは再度作成されずに、既存のオーバークラウドに変更が加えられます。

新規環境ファイルを追加する場合には、**openstack overcloud deploy** コマンドで **-e** オプションを使用して そのファイルを追加します。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 ~/templates/network-
```

```
environment.yaml -e ~/templates/storage-environment.yaml -e
~/templates/new-environment.yaml --control-scale 3 --compute-scale 3 --
ceph-storage-scale 3 --control-flavor control --compute-flavor compute --
ceph-storage-flavor ceph-storage --ntp-server pool.ntp.org
```

これにより、環境ファイルからの新規パラメーターやリソースがスタックに追加されます。



### 重要

director により後で上書きされてしまう可能性があるため、オーバークラウドの設定には手動で変更を加えないことを推奨します。

## 8.8. オーバークラウドへの仮想マシンのインポート

既存の OpenStack 環境があり、仮想マシンを Red Hat OpenStack Platform 環境に移行する予定がある場合には、以下の手順を使用します。

実行中のサーバーのスナップショットを作成して新規イメージを作成し、そのイメージをダウンロードします。

```
$ nova image-create instance_name image_name
$ glance image-download image_name --file exported_vm.qcow2
```

エクスポートしたイメージをオーバークラウドにアップロードして、新規インスタンスを起動します。

```
$ glance image-create --name imported_image --file exported_vm.qcow2 --
disk-format qcow2 --container-format bare
$ nova boot --poll --key-name default --flavor m1.demo --image
imported_image --nic net-id=net_id imported
```



### 重要

各仮想マシンのディスクは、既存の OpenStack 環境から新規の Red Hat OpenStack Platform にコピーする必要があります。QCOW を使用したスナップショットでは、元の階層化システムが失われます。

## 8.9. オーバークラウドのコンピュートノードからの仮想マシンの移行

オーバークラウドのコンピュートノードでメンテナンスを行う場合があります。ダウンタイムを防ぐには、以下の手順に従ってそのコンピュートノード上の仮想マシンを同じオーバークラウド内の別のコンピュートノードに移行します。

director は全コンピュートノードがセキュアな移行を提供するように設定します。移行プロセス中に、各ホストの **nova** ユーザーが他のコンピュートノードにアクセスできるようにするための共有 SSH キーも全コンピュートノードに必要です。director はこのキーを自動的に作成します。

## 重要

Red Hat OpenStack Platform 9 の最新の更新には、ライブマイグレーションの機能に必要なパッチが含まれています。初期リリースでは、director のコアテンプレートコレクションにこの機能は含まれていませんでしたが、**openstack-tripleo-heat-templates-2.0.0-57.el7ost** パッケージ以降のバージョンで含まれるようになりました。

環境を更新して、**openstack-tripleo-heat-templates-2.0.0-57.el7ost** 以降のパッケージの Heat テンプレートを使用するようにします。

詳しくは、「[Red Hat OpenStack Platform director \(TripleO\) CVE-2017-2637 bug and Red Hat OpenStack Platform](#)」を参照してください。

インスタンスを移行するには、**overcloudrc** を読み込んで、現在の nova サービスの一覧を取得します。

```
$ source ~/stack/overcloudrc
$ nova service-list
```

移行予定のノードで **nova-compute** サービスを無効にします。

```
$ nova service-disable [hostname] nova-compute
```

これにより、新規インスタンスはそのノード上でスケジュールされなくなります。

ノードからのインスタンスの移行プロセスを開始します。以下のコマンドは、単一のインスタンスを移行します。

```
$ openstack server migrate [server-name]
```

無効になったコンピュートノードから移行する必要のあるインスタンスごとにこのコマンドを実行します。

移行プロセスの現況は、以下のコマンドで取得できます。

```
$ nova migration-list
```

各インスタンスの移行が完了したら、nova の状態は **VERIFY\_RESIZE** に変わります。ここで、移行を正常に完了するか、環境をロールバックするかを確定する機会が提供されます。移行を確定するには、以下のコマンドを使用してください。

```
$ nova resize-confirm [server-name]
```

ステータスが **VERIFY\_RESIZE** のインスタンスごとに、このコマンドを実行します。

これにより、ホストからすべてのインスタンスが移行されます。インスタンスのダウンタイムなしにホスト上のメンテナンスを実行できるようになります。ホストを有効な状態に戻すには、以下のコマンドを実行します。

```
$ nova service-enable [hostname] nova-compute
```

## 8.10. オーバークラウドが削除されていないように保護

**heat stack-delete overcloud** コマンドで誤って削除されないように、Heat には特定のアクションを制限するポリシーセットが含まれます。**/etc/heat/policy.json** を開いて、以下のパラメーターを検索します。

```
"stacks:delete": "rule:deny_stack_user"
```

このパラメーターの設定を以下のように変更します。

```
"stacks:delete": "rule:deny_everybody"
```

ファイルを保存します。

これにより **heat** クライアントでオーバークラウドが削除されないようにします。オーバークラウドを削除できるように設定するには、ポリシーを元の値に戻します。

## 8.11. オーバークラウドの削除

オーバークラウドはすべて、必要に応じて削除することができます。

既存のオーバークラウドを削除します。

```
$ heat stack-delete overcloud
```

オーバークラウドが削除されていることを確認します。

```
$ heat stack-list
```

削除には、数分かかります。

削除が完了したら、デプロイメントシナリオの標準ステップに従って、オーバークラウドを再度作成します。



## 第9章 オーバークラウドのスケールリング

オーバークラウドの作成後に、ノードを追加または削除する必要がある場合があります。たとえば、オーバークラウドのコンピューターノードを追加する場合などです。このような状況では、オーバークラウドの更新が必要です。



### 警告

コンピューターインスタンスの高可用性 (またはインスタンス HA。『[コンピューターインスタンスの高可用性](#)』で説明) を使用している場合は、アップグレードとスケールアップはできません。操作を試みても失敗します。

HA を有効化しているインスタンスがある場合には、アップグレードまたはスケールアップを実行する前に無効にしてください。そのためには、『[Rollback](#)』に記載の **ロールバック** の操作を実行してください。

以下の表を使用して、各ノード種別のスケールリングに対するサポートを判断してください。

表9.1 各ノード種別のスケールリングサポート

ノード種別	スケールアップ	スケールダウン	備考
コントローラー	x	x	
Compute	Y	Y	
Ceph Storage ノード	Y	x	オーバークラウドを最初に作成する際に Ceph Storage ノードを 1 つ以上設定する必要があります。
Block Storage ノード	x	x	
Object Storage ノード	Y	Y	リングを手動で管理する必要があります (『 <a href="#">Object Storage ノードの置き換え</a> 』に説明を記載)。



### 重要

オーバークラウドをスケールリングする前には、空き領域が少なくとも 10 GB のあることを確認してください。この空き領域は、イメージの変換やノードのプロビジョニングプロセスのキャッシュに使用されます。

### 9.1. ノードのさらなる追加

director のノードプールにさらにノードを追加するには、登録する新規ノードの詳細を記載した新しい JSON ファイル (例: **newnodes.json**) を作成します。

```
{
  "nodes": [
    {
      "mac": [
        "dd:dd:dd:dd:dd:dd"
      ],
      "cpu": "4",
      "memory": "6144",
      "disk": "40",
      "arch": "x86_64",
      "pm_type": "pxe_ipmitool",
      "pm_user": "admin",
      "pm_password": "p@55w0rd!",
      "pm_addr": "192.0.2.207"
    },
    {
      "mac": [
        "ee:ee:ee:ee:ee:ee"
      ],
      "cpu": "4",
      "memory": "6144",
      "disk": "40",
      "arch": "x86_64",
      "pm_type": "pxe_ipmitool",
      "pm_user": "admin",
      "pm_password": "p@55w0rd!",
      "pm_addr": "192.0.2.208"
    }
  ]
}
```

これらのパラメーターについての説明は、[「オーバークラウドへのノードの登録」](#)を参照してください。

以下のコマンドを実行して、これらのノードを登録します。

```
$ openstack baremetal import --json newnodes.json
```

新規ノードを追加した後は、それらのイントロスペクションプロセスを起動します。各新規ノードに以下のコマンドを使用します。

```
$ ironic node-set-provision-state [NODE UUID] manage
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-provision-state [NODE UUID] provide
```

このコマンドは、ノードのハードウェアプロパティの検出とベンチマークを実行します。

イントロスペクションプロセスの完了後には、各新規ノードを任意のロールにタグ付けしてスケーリングします。たとえば、コンピュートノードの場合には、以下のコマンドを使用します。

```
$ ironic node-update [NODE UUID] add
properties/capabilities='profile:compute,boot_option:local'
```

デプロイメント中に使用するブートイメージを設定します。**bm-deploy-kernel** および **bm-deploy-ramdisk** イメージの UUID を確認します。

```
$ glance image-list
+-----+-----+
| ID                                     | Name                               |
+-----+-----+
| 09b40e3d-0382-4925-a356-3a4b4f36b514 | bm-deploy-kernel                 |
| 765a46af-4417-4592-91e5-a300ead3faf6 | bm-deploy-ramdisk                 |
| ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full                    |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd             |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz            |
+-----+-----+
```

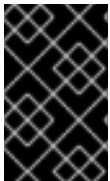
新規ノードの **deploy\_kernel** および **deploy\_ramdisk** 設定にこれらの UUID を設定します。

```
$ ironic node-update [NODE UUID] add driver_info/deploy_kernel='09b40e3d-0382-4925-a356-3a4b4f36b514'
$ ironic node-update [NODE UUID] add driver_info/deploy_ramdisk='765a46af-4417-4592-91e5-a300ead3faf6'
```

オーバークラウドをスケーリングするには、ロールに必要なノード数を指定して **openstack overcloud deploy** を再実行する必要があります。たとえば、コンピュートノード 5 台にスケーリングするには、以下のコマンドを実行します。

```
$ openstack overcloud deploy --templates --compute-scale 5 [OTHER_OPTIONS]
```

上記のコマンドにより、オーバークラウドのスタック全体が更新されます。このコマンドが更新するのは、スタックのみである点に注意してください。オーバークラウドの削除や、スタックの置き換えは行われません。



### 重要

コンピュート以外のノードに対する同様のスケジューリングパラメーターなど、最初に作成したオーバークラウドからの環境ファイルおよびオプションをすべて追加するようにしてください。

## 9.2. コンピュートノードの削除

オーバークラウドからコンピュートノードを削除する必要がある状況が出てくる可能性があります。たとえば、問題のあるコンピュートノードを置き換える必要がある場合などです。



### 重要

オーバークラウドからコンピュートノードを削除する前に、ワークロードをそのノードから別のコンピュートノードに移行してください。詳しくは、「[オーバークラウドのコンピュートノードからの仮想マシンの移行](#)」を参照してください。

次に、オーバークラウド上でノードの Compute サービスを無効化します。これにより、ノードで新規インスタンスがスケジューリングされないようになります。

```
$ source ~/stack/overcloudrc
```

```
$ nova service-list
$ nova service-disable [hostname] nova-compute
$ source ~/stack/stackrc
```

オーバークラウドノードを削除するには、ローカルのテンプレートファイルを使用して **overcloud** スタックへの更新が必要です。最初に、オーバークラウドスタックの UUID を特定します。

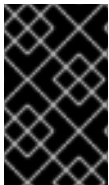
```
$ heat stack-list
```

削除するノードの UUID を特定します。

```
$ nova list
```

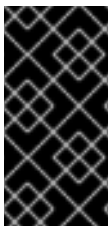
以下のコマンドを実行してスタックからノードを削除し、それに応じてプランを更新します。

```
$ openstack overcloud node delete --stack [STACK_UUID] --templates -e
[ENVIRONMENT_FILE] [NODE1_UUID] [NODE2_UUID] [NODE3_UUID]
```



### 重要

オーバークラウドの作成時に追加の環境ファイルを渡した場合には、オーバークラウドに、不要な変更が手動で加えられないように、ここで **-e** または **--environment-file** オプションを使用して環境ファイルを再度指定します。



### 重要

操作を続行する前に、**openstack overcloud node delete** コマンドが完全に終了したことを確認します。**openstack stack list** コマンドを使用して、**overcloud** スタックが **UPDATE\_COMPLETE** のステータスに切り替わっているかどうかをチェックしてください。

最後に、ノードの Compute サービスを削除します。

```
$ source ~/stack/overcloudrc
$ nova service-list
$ nova service-delete [service-id]
$ source ~/stack/stackrc
```

ノードの Open vSwitch エージェントも削除します。

```
$ source ~/stack/overcloudrc
$ neutron agent-list
$ neutron agent-delete [openvswitch-agent-id]
$ source ~/stack/stackrc
```

オーバークラウドから自由にノードを削除して、別の目的でそのノードを再プロビジョニングすることができます。

## 9.3. コンピュートノードの置き換え

コンピュートノードに障害が発生した場合に、機能しているノードに置き換えることができます。コンピュートノードを置き換えるには、以下の手順を使用します。

- 既存のコンピューットノードからワークロードを移行して、ノードをシャットダウンします。この手順は「[オーバークラウドのコンピューットノードからの仮想マシンの移行](#)」を参照してください。
- オーバークラウドからコンピューットノードを削除します。この手順は「[コンピューットノードの削除](#)」を参照してください。
- 新しいコンピューットノードでオーバークラウドをスケーリングアウトします。その手順は、「[ノードのさらなる追加](#)」を参照してください。

このプロセスでは、インスタンスの可用性に影響を与えることなく、ノードを置き換えることができますようにします。

## 9.4. コントローラーノードの置き換え

特定の状況では、高可用性クラスター内のコントローラーノードに障害が発生することがあり、その場合は、そのコントローラーノードをクラスターから削除して新しいコントローラーノードに置き換える必要があります。このステップには、クラスター内の他のノードとの接続を確認する作業も含まれます。

本項では、コントローラーノードの置き換えの手順について説明します。このプロセスでは **openstack overcloud deploy** コマンドを実行してコントローラーノードの置き換えを要求し、オーバークラウドを更新します。このプロセスは、自動的に完了しない点に注意してください。オーバークラウドスタックの更新プロセスの途中で、**openstack overcloud deploy** コマンドによりエラーが報告されて、オーバークラウドスタックの更新が停止します。この時点で、プロセスに手動での介入が必要となり、その後に **openstack overcloud deploy** のプロセスを続行することができます。



### 重要

以下の手順は、高可用性環境のみに適用します。コントローラーノード 1 台の場合には、この手順は使用しないでください。

### 9.4.1. 事前のチェック

オーバークラウドコントローラーノードの置き換えを試みる前に、Red Hat OpenStack Platform 環境の現在の状態をチェックしておくことが重要です。このチェックしておくこと、コントローラーの置き換えプロセス中に複雑な事態が発生するのを防ぐことができます。以下の事前チェックリストを使用して、コントローラーノードの置き換えを実行しても安全かどうかを確認してください。チェックのためのコマンドはすべてアンダークラウドで実行します。

1. アンダークラウドで、**overcloud** スタックの現在の状態をチェックします。

```
$ source stackrc
$ heat stack-list --show-nested
```

**overcloud** スタックと後続の子スタックは、**CREATE\_COMPLETE** または **UPDATE\_COMPLETE** のステータスである必要があります。

2. アンダークラウドデータベースのバックアップを実行します。

```
$ mkdir /home/stack/backup
$ sudo mysqldump --all-databases --quick --single-transaction | gzip
> /home/stack/backup/dump_db_undercloud.sql.gz
$ sudo systemctl stop openstack-ironic-api.service openstack-ironic-
```

```
conductor.service openstack-ironic-inspector.service openstack-
ironic-inspector-dnsmasq.service
$ sudo cp /var/lib/ironic-inspector/inspector.sqlite
/home/stack/backup
$ sudo systemctl start openstack-ironic-api.service openstack-
ironic-conductor.service openstack-ironic-inspector.service
openstack-ironic-inspector-dnsmasq.service
```

3. アンダークラウドで、新規ノードのプロビジョニング時にイメージのキャッシュと変換に対応できる 10 GB の空きストレージ領域があるかどうかをチェックします。
4. コントローラーノードで実行中の Pacemaker の状態をチェックします。たとえば、実行中のコントローラーノードの IP アドレスが 192.168.0.47 の場合には、以下のコマンドで Pacemaker のステータス情報を取得します。

```
$ ssh heat-admin@192.168.0.47 'sudo pcs status'
```

出力には、既存のノードで実行中のサービスと、障害が発生しているノードで停止中のサービスがすべて表示されるはずです。

5. オーバークラウドの MariaDB クラスターの各ノードで以下のパラメーターをチェックします。

- **wsrep\_local\_state\_comment: Synced**

- **wsrep\_cluster\_size: 2**

実行中のコントローラーノードで以下のコマンドを使用して、パラメーターをチェックします (IP アドレスにはそれぞれ 192.168.0.47 と 192.168.0.46 を使用します)。

```
$ for i in 192.168.0.47 192.168.0.46 ; do echo "**** $i ****" ; ssh
heat-admin@$i "sudo mysql --exec=\"SHOW STATUS LIKE
'wsrep_local_state_comment'\" ; sudo mysql --exec=\"SHOW STATUS
LIKE 'wsrep_cluster_size'\""; done
```

6. RabbitMQ のステータスをチェックします。たとえば、実行中のコントローラーノードの IP アドレスが 192.168.0.47 の場合には、以下のコマンドを実行してステータスを取得します。

```
$ ssh heat-admin@192.168.0.47 "sudo rabbitmqctl cluster_status"
```

**running\_nodes** キーには、障害が発生しているノードは表示されず、稼働中のノード 2 台のみが表示されるはずです。

7. フェンシングが有効化されている場合には無効にします。たとえば、実行中のコントローラーノードの IP アドレスが 192.168.0.47 の場合には、以下のコマンドを実行してフェンシングを無効にします。

```
$ ssh heat-admin@192.168.0.47 "sudo pcs property set stonith-
enabled=false"
```

以下のコマンドを実行してフェンシングのステータスを確認します。

```
$ ssh heat-admin@192.168.0.47 "sudo pcs property show stonith-
enabled"
```

8. director ノードで **nova-compute** サービスをチェックします。

```
$ sudo systemctl status openstack-nova-compute
$ nova hypervisor-list
```

出力では、メンテナンスモードに入っていないすべてのノードが **up** のステータスで表示されるはずです。

9. アンダークラウドサービスがすべて実行中であることを確認します。

```
$ sudo systemctl -t service
```

### 9.4.2. ノードの置き換え

削除するノードのインデックスを特定します。ノードのインデックスは、**nova list** の出力に表示されるインスタンス名のサフィックスです。

```
[stack@director ~]$ nova list
```

ID	Name
861408be-4027-4f53-87a6-cd3cf206ba7a	overcloud-compute-0
0966e9ae-f553-447a-9929-c4232432f718	overcloud-compute-1
9c08fa65-b38c-4b2e-bd47-33870bfff06c7	overcloud-compute-2
a7f0f5e1-e7ce-4513-ad2b-81146bc8c5af	overcloud-controller-0
cfefaf60-8311-4bc3-9416-6a824a40a9ae	overcloud-controller-1
97a055d4-aefd-481c-82b7-4a5f384036d2	overcloud-controller-2

この例では、**overcloud-controller-1** ノードを削除して、**overcloud-controller-3** に置き換えます。初めにノードをメンテナンスモードに切り替えて、director が障害の発生したノードを再プロビジョニングしないようにします。**nova list** で表示されるインスタンスの ID を、**ironic node-list** で表示されるノード ID と関連させます。

```
[stack@director ~]$ ironic node-list
```

UUID	Name	Instance UUID
36404147-7c8a-41e6-8c72-a6e90afc7584	None	7bee57cf-4a58-4eaf-b851-2a8bf6620e48
91eb9ac5-7d52-453c-a017-c0e3d823efd0	None	None
75b25e9a-948d-424a-9b3b-f0ef70a6eacf	None	None
038727da-6a5c-425f-bd45-fda2f4bd145b	None	763bfec2-9354-466a-ae65-2401c13e07e5
dc2292e6-4056-46e0-8848-d6e96df1f55d	None	2017b481-706f-44e1-852a-2ee857c303c4
c7eadcea-e377-4392-9fc3-cf2b02b7ec29	None	5f73c7d7-4826-49a5-b6be-8bfd558f3b41
da3a8d19-8a59-4e9d-923a-6a336fe10284	None	cfefaf60-8311-4bc3-9416-6a824a40a9ae
807cb6ce-6b94-4cd1-9969-5c47560c2eee	None	c07c13e6-a845-4791-9628-

```
260110829c3a |
+-----+-----+-----+
-----+
```

ノードをメンテナンスモードに切り替えます。

```
[stack@director ~]$ ironic node-set-maintenance da3a8d19-8a59-4e9d-923a-
6a336fe10284 true
```

新規ノードを **control** プロファイルでタグ付けします。

```
[stack@director ~]$ ironic node-update 75b25e9a-948d-424a-9b3b-
f0ef70a6eacf add
properties/capabilities='profile:control,boot_option:local'
```

削除するノードインデックスを定義する YAML ファイルを作成します (**~/templates/remove-controller.yaml**)。

```
parameters:
  ControllerRemovalPolicies:
    [{'resource_list': ['1']}]
```



## 重要

インデックス 0 のノードを置き換える場合には、ノードの置き換えを開始する前に、Heat テンプレートを編集してブートストラップのノードインデックスとノード検証インデックスを変更します。director の Heat テンプレートコレクションのコピーを作成して ([「カスタムのコア Heat テンプレートの使用」](#) を参照)、以下のコマンドを **overcloud.yaml** ファイルに対して実行します。

```
$ sudo sed -i "s/resource\.0/resource.1/g" ~/templates/my-overcloud/overcloud.yaml
```

これにより、以下のリソースのノードインデックスが変更されます。

```
ControllerBootstrapNodeConfig:
  type: OS::TripleO::BootstrapNode::SoftwareConfig
  properties:
    bootstrap_nodeid: {get_attr: [Controller,
resource.0.hostname]}
    bootstrap_nodeid_ip: {get_attr: [Controller,
resource.0.ip_address]}
```

および

```
AllNodesValidationConfig:
  type: OS::TripleO::AllNodes::Validation
  properties:
    PingTestIps:
      list_join:
        - ' '
        - - {get_attr: [Controller,
resource.0.external_ip_address]}
          - {get_attr: [Controller,
resource.0.internal_api_ip_address]}
          - {get_attr: [Controller,
resource.0.storage_ip_address]}
          - {get_attr: [Controller,
resource.0.storage_mgmt_ip_address]}
          - {get_attr: [Controller,
resource.0.tenant_ip_address]}
```

## 注記

Corosync での settle の試行回数を減らすことによって、置き換えのプロセスをスピードアップすることができます。環境ファイルの **ExtraConfig** パラメーターに以下の hieradata を追加します。

```
parameter_defaults:
  ExtraConfig:
    pacemaker::corosync::settle_tries: 5
```

ノードインデックスを特定した後は、オーバークラウドを再デプロイして、**remove-controller.yaml** 環境ファイルを追加します。

```
[stack@director ~]$ openstack overcloud deploy --templates --control-scale
3 -e ~/templates/remove-controller.yaml [OTHER OPTIONS]
```

オーバークラウドの作成時に追加の環境ファイルまたはオプションを渡した場合には、予定外の変更がオーバークラウドに加えられないように、その環境ファイルまたはオプションをここで再度渡してください。

ただし、**-e ~/templates/remove-controller.yaml** が必要なのは、この場合には 1 回のみである点に注意してください。

director は古いノードを削除して、新しいノードを作成してから、オーバークラウドスタックを更新します。以下のコマンドを使用すると、オーバークラウドスタックのステータスをチェックすることができます。

```
[stack@director ~]$ heat stack-list --show-nested
```

### 9.4.3. 手動での介入

**ControllerNodesPostDeployment** の段階中には、オーバークラウドスタックの更新が **ControllerLoadBalancerDeployment\_Step1** で **UPDATE\_FAILED** エラーにより停止します。これは、一部の Puppet モジュールがノードの置き換えをサポートしてないためです。処理のこの時点で手動による介入が必要です。以下に記載する設定ステップに従ってください。

1. コントローラーノードの IP アドレスの一覧を取得します。以下に例を示します。

```
[stack@director ~]$ nova list
... +-----+ ... +-----+
... | Name                | ... | Networks                |
... +-----+ ... +-----+
... | overcloud-compute-0  | ... | ctlplane=192.168.0.44    |
... | overcloud-controller-0 | ... | ctlplane=192.168.0.47    |
... | overcloud-controller-2 | ... | ctlplane=192.168.0.46    |
... | overcloud-controller-3 | ... | ctlplane=192.168.0.48    |
... +-----+ ... +-----+
```

2. 既存のノードの **/etc/corosync/corosync.conf** ファイルで、削除されたノードの **nodeid** の値を確認します。たとえば、既存のノードが 192.168.0.47 の **overcloud-controller-0** の場合には、以下のコマンドを実行します。

```
[stack@director ~]$ ssh heat-admin@192.168.0.47 "sudo cat
/etc/corosync/corosync.conf"
```

このコマンドにより、削除されたノードの ID が含まれる **nodelist** が表示されます (**overcloud-controller-1**)。

```
nodelist {
  node {
    ring0_addr: overcloud-controller-0
    nodeid: 1
  }
  node {
    ring0_addr: overcloud-controller-1
    nodeid: 2
  }
}
```

```
node {
  ring0_addr: overcloud-controller-2
  nodeid: 3
}
```

削除された **nodeid** の値は、後で使用するののために書き留めておいてください。上記の例では、2 がその値です。

3. 各ノードの Corosync 設定から障害の発生したノードを削除して、Corosync を再起動します。この例では、**overcloud-controller-0** と **overcloud-controller-2** にログインして以下のコマンドを実行します。

```
[stack@director] ssh heat-admin@192.168.0.47 "sudo pcs cluster
localnode remove overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.0.47 "sudo pcs cluster
reload corosync"
[stack@director] ssh heat-admin@192.168.0.46 "sudo pcs cluster
localnode remove overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.0.46 "sudo pcs cluster
reload corosync"
```

4. 残りのノードの中の 1 台にログインして、**crm\_node** コマンドで対象のノードをクラスターから削除します。

```
[stack@director] ssh heat-admin@192.168.0.47
[heat-admin@overcloud-controller-0 ~]$ sudo crm_node -R overcloud-
controller-1 --force
```

このノードにログインした状態を維持します。

5. 障害が発生したノードを RabbitMQ クラスターから削除します。

```
[heat-admin@overcloud-controller-0 ~]$ sudo rabbitmqctl
forget_cluster_node rabbit@overcloud-controller-1
```

6. 障害が発生したノードを MongoDB から削除します。ノードの内部 API 接続のための IP アドレスを特定します。

```
[heat-admin@overcloud-controller-0 ~]$ sudo netstat -tulnp | grep
27017
tcp          0      0 192.168.0.47:27017    0.0.0.0:*
LISTEN      13415/mongod
```

ノードが **primary** レプリカセットであることを確認します。

```
[root@overcloud-controller-0 ~]# echo "db.isMaster()" | mongo --host
192.168.0.47:27017
MongoDB shell version: 2.6.11
connecting to: 192.168.0.47:27017/echo
{
  "setName" : "tripleo",
  "setVersion" : 1,
  "ismaster" : true,
```

```

    "secondary" : false,
    "hosts" : [
      "192.168.0.47:27017",
      "192.168.0.46:27017",
      "192.168.0.45:27017"
    ],
    "primary" : "192.168.0.47:27017",
    "me" : "192.168.0.47:27017",
    "electionId" : ObjectId("575919933ea8637676159d28"),
    "maxBsonObjectSize" : 16777216,
    "maxMessageSizeBytes" : 48000000,
    "maxWriteBatchSize" : 1000,
    "localTime" : ISODate("2016-06-09T09:02:43.340Z"),
    "maxWireVersion" : 2,
    "minWireVersion" : 0,
    "ok" : 1
  }
bye

```

これで、現在のノードがプライマリーかどうかが表示されるはずです。そうでない場合には、**primary** キーに示されているノードの IP アドレスを使用します。

プライマリーノードで MongoDB に接続します。

```

[heat-admin@overcloud-controller-0 ~]$ mongo --host 192.168.0.47
MongoDB shell version: 2.6.9
connecting to: 192.168.0.47:27017/test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
http://docs.mongodb.org/
Questions? Try the support group
http://groups.google.com/group/mongodb-user
tripleo:PRIMARY>

```

MongoDB クラスターのステータスを確認します。

```
tripleo:PRIMARY> rs.status()
```

**\_id** キーを使用してノードを特定し、**name** キーを使用して障害の発生したノードを削除します。この場合には **name** に **192.168.0.45:27017** を指定して Node 1 を削除します。

```
tripleo:PRIMARY> rs.remove('192.168.0.45:27017')
```

## 重要

**PRIMARY** レプリカセットに対してコマンドを実行する必要があります。

```
"replSetReconfig command must be sent to the current
replica set primary."
```

上記のメッセージが表示された場合には、**PRIMARY** に指定されているノード上の MongoDB に再ログインします。



## 注記

障害の発生したノードのレプリカセットを削除する際には、通常以下のような出力が表示されます。

```
2016-05-07T03:57:19.541+0000 DBClientCursor::init call()
failed
2016-05-07T03:57:19.543+0000 Error: error doing query:
failed at src/mongo/shell/query.js:81
2016-05-07T03:57:19.545+0000 trying reconnect to
192.168.0.47:27017 (192.168.0.47) failed
2016-05-07T03:57:19.547+0000 reconnect 192.168.0.47:27017
(192.168.0.47) ok
```

MongoDB を終了します。

```
tripleo:PRIMARY> exit
```

- Galera クラスター内のノードの一覧を更新します。

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource update
galera wsrep_cluster_address=gcomm://overcloud-controller-
0,overcloud-controller-3,overcloud-controller-2
```

- 新規ノードに対して Galera クラスターのチェックが行われるように設定します。既存のノードから新規ノードの同じ場所に **/etc/sysconfig/clustercheck** をコピーします。
- root** ユーザーが新規ノードの Galera にアクセスできるように設定します。既存のノードから新規ノードの同じ場所に **/root/.my.cnf** をコピーします。
- 新規ノードをクラスターに追加します。

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster node add
overcloud-controller-3
```

- 各ノードで **/etc/corosync/corosync.conf** ファイルをチェックします。新規ノードの **nodeid** が削除したノードと同じ場合には、その値を **nodeid** 値に更新します。たとえば、**/etc/corosync/corosync.conf** ファイルに新規ノード (**overcloud-controller-3**) のエントリーが記載されています。

```
nodelist {
  node {
    ring0_addr: overcloud-controller-0
    nodeid: 1
  }
  node {
    ring0_addr: overcloud-controller-2
    nodeid: 3
  }
  node {
    ring0_addr: overcloud-controller-3
```

```

    nodeid: 2
  }
}
```

上記の例では、新規ノードが削除されたノードと同じ **nodeid** を使用している点に注意してください。この値を、使用していないノードの ID 値に更新します。以下に例を示します。

```

node {
  ring0_addr: overcloud-controller-3
  nodeid: 4
}
```

新規ノードを含む各コントローラーノードの **/etc/corosync/corosync.conf** ファイルで **nodeid** の値を更新します。

12. 既存のノードのみで Corosync サービスを再起動します。たとえば、**overcloud-controller-0** で以下のコマンドを実行します。

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster reload
corosync
```

**overcloud-controller-2** で以下のコマンドを実行します。

```
[heat-admin@overcloud-controller-2 ~]$ sudo pcs cluster reload
corosync
```

このコマンドは、新規ノードでは実行しないでください。

13. 新規コントローラーノードを起動します。

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster start
overcloud-controller-3
```

14. 新規ノード上で Keystone サービスを有効にします。残りのノードから director ホストに **/etc/keystone** ディレクトリーをコピーします。

```
[heat-admin@overcloud-controller-0 ~]$ sudo -i
[root@overcloud-controller-0 ~]$ scp -r /etc/keystone
stack@192.168.0.1:~/.
```

新規コントローラーノードにログインします。新規コントローラーノードから **/etc/keystone** ディレクトリーを削除して、director ホストから **keystone** ファイルをコピーします。

```
[heat-admin@overcloud-controller-3 ~]$ sudo -i
[root@overcloud-controller-3 ~]$ rm -rf /etc/keystone
[root@overcloud-controller-3 ~]$ scp -r stack@192.168.0.1:~/keystone
/etc/.
[root@overcloud-controller-3 ~]$ chown -R keystone: /etc/keystone
[root@overcloud-controller-3 ~]$ chown root
/etc/keystone/logging.conf /etc/keystone/default_catalog.templates
```

**/etc/keystone/keystone.conf** を編集して **admin\_bind\_host** および

**public\_bind\_host** のパラメーターを新規コントローラーノードの IP アドレスに設定します。これらの IP アドレスを確認するには、**ip addr** コマンドを使用して、以下のネットワーク内の IP アドレスを見つけます。

- **admin\_bind\_host**: プロビジョニングネットワーク
- **public\_bind\_host**: 内部 API ネットワーク



#### 注記

カスタムの **ServiceNetMap** パラメーターを使用してオーバークラウドをデプロイした場合には、これらのネットワークは異なる場合があります。

たとえば、プロビジョニングネットワークが **192.168.0.0/24** サブネットを使用して、内部 API が **172.17.0.0/24** サブネットを使用している場合には、以下のコマンドを使用して、これらのネットワーク上のノードの IP アドレスを特定します。

```
[root@overcloud-controller-3 ~]$ ip addr | grep "192\.168\.0\..*/24"
[root@overcloud-controller-3 ~]$ ip addr | grep "172\.17\.0\..*/24"
```

15. Pacemaker を使用して、いくつかのサービスを有効化および再起動します。クラスターは現在メンテナンスモードに設定されていますが、サービスを有効化するには、このモードを一時的に無効にする必要があります。以下に例を示します。

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs property set
maintenance-mode=false --wait
```

16. 全ノードで Galera サービスが起動するのを待ちます。

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs status | grep galera
-A1
Master/Slave Set: galera-master [galera]
Masters: [ overcloud-controller-0 overcloud-controller-2 overcloud-
controller-3 ]
```

必要な場合には、新規ノードで **cleanup** を実行します。

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs resource cleanup
galera --node overcloud-controller-3
```

17. 全ノードで **httpd** サービスが起動するまで待ちます。

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs status | grep httpd
-A1
Clone Set: httpd-clone [httpd]
Started: [ overcloud-controller-0 overcloud-controller-2 overcloud-
controller-3 ]
```

必要な場合には、新規ノードで **cleanup** を実行します。

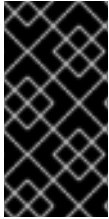
```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs resource cleanup
httpd --node overcloud-controller-3
```

18. クラスターをメンテナンスモードに再度切り替えます。

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs property set
maintenance-mode=true --wait
```

手動の設定が完了しました。オーバークラウドのコマンドを再度実行して、スタックの更新を続けます。

```
[stack@director ~]$ openstack overcloud deploy --templates --control-scale
3 [OTHER OPTIONS]
```



### 重要

オーバークラウドの作成時に追加の環境ファイルまたはオプションを渡した場合には、予定外の変更がオーバークラウドに加えられないように、その環境ファイルまたはオプションをここで再度渡してください。ただし、**remove-controller.yaml** ファイルは必要なくなった点に注意してください。

#### 9.4.4. オーバークラウドサービスの最終処理

オーバークラウドのスタックの更新が完了したら、最終の設定が必要です。コントローラーノードの 1 つにログインして、Pacemaker で停止されているサービスを更新します。

```
[heat-admin@overcloud-controller-0 ~]$ for i in `sudo pcs status|grep -B2
Stop |grep -v "Stop\|Start"|awk -F"[" ' /\[/ {print
substr($NF,0,length($NF)-1)}'`; do echo $i; sudo pcs resource cleanup $i;
done
```

最終のステータスチェックを実行して、サービスが正しく実行されていることを確認します。

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs status
```



### 注記

エラーが発生したサービスがある場合には、**pcs resource cleanup** コマンドを使用して、問題の解決後にそのサービスを再起動します。

director を終了します。

```
[heat-admin@overcloud-controller-0 ~]$ exit
```

#### 9.4.5. L3 エージェントのルーターホスティングの最終処理

オーバークラウドと対話できるようにするために、source コマンドで **overcloudrc** ファイルを読み込みます。ルーターをチェックして、L3 エージェントがオーバークラウド環境内のルーターを適切にホストしていることを確認します。以下の例では、**r1** という名前のルーターを使用します。

```
[stack@director ~]$ source ~/overcloudrc
[stack@director ~]$ neutron l3-agent-list-hosting-router r1
```



このリストには、新しいノードの代わりに、依然として古いノードが表示される場合があります。これを置き換えるには、環境内の L3 ネットワークエージェントを一覧表示します。

```
[stack@director ~]$ neutron agent-list | grep "neutron-l3-agent"
```

新しいノードと古いノード上でエージェントの UUID を特定します。新しいノードのエージェントにルーターを追加し、古いノードからそのルーターを削除します。以下に例を示します。

```
[stack@director ~]$ neutron l3-agent-router-add fd6b3d6e-7d8c-4e1a-831a-4ec1c9ebb965 r1
[stack@director ~]$ neutron l3-agent-router-remove b40020af-c6dd-4f7a-b426-eba7bac9dbc2 r1
```

ルーターに対して最終チェックを実行し、すべてがアクティブであることを確認します。

```
[stack@director ~]$ neutron l3-agent-list-hosting-router r1
```

古いコントローラーノードをポイントしている既存の Neutron エージェントを削除します。以下に例を示します。

```
[stack@director ~]$ neutron agent-list -F id -F host | grep overcloud-controller-1
| ddae8e46-3e8e-4a1b-a8b3-c87f13c294eb | overcloud-controller-1.localdomain |
[stack@director ~]$ neutron agent-delete ddae8e46-3e8e-4a1b-a8b3-c87f13c294eb
```

#### 9.4.6. Compute サービスの最終処理

削除されたノードの Compute サービスはオーバークラウドにまだ存在しているので、削除する必要があります。source コマンドで **overcloudrc** ファイルを読み込み、オーバークラウドと対話できるようにします。削除したノードの Compute サービスをチェックします。

```
[stack@director ~]$ source ~/overcloudrc
[stack@director ~]$ nova service-list | grep "overcloud-controller-1.localdomain"
```

ノードの Compute サービスを削除します。たとえば、**overcloud-controller-1.localdomain** の **nova-scheduler** サービスの ID が 5 の場合には、以下のコマンドを実行します。

```
[stack@director ~]$ nova service-delete 5
```

削除したノードの各サービスでこのタスクを実行します。

新しいノードで **openstack-nova-consoleauth** サービスをチェックします。

```
[stack@director ~]$ nova service-list | grep consoleauth
```

サービスが実行していない場合には、コントローラーノードにログインしてサービスを再起動します。

```
[stack@director] ssh heat-admin@192.168.0.47
[heat-admin@overcloud-controller-0 ~]$ pcs resource restart openstack-
nova-consoleauth
```

### 9.4.7. 結果

障害が発生したコントローラーノードと、関連サービスが新しいノードに置き換えられました。



#### 重要

「[Object Storage ノードの置き換え](#)」のように Object Storage でリングファイルの自動構築を無効にした場合には、新規ノード用に Object Storage リングファイルを手動で構築する必要があります。リングファイルの手動構築についての詳しい情報は、「[Object Storage ノードの置き換え](#)」を参照してください。

## 9.5. CEPH STORAGE ノードの置き換え

director では、director で作成したクラスター内の Ceph Storage ノードを置き換えることができます。手順については、『[オーバークラウド向けの Red Hat Ceph Storage](#)』を参照してください。

## 9.6. OBJECT STORAGE ノードの置き換え

Object Storage クラスターのノードを置き換えるには、以下を行う必要があります。

- 新規 Object Storage ノードでオーバークラウドを更新して、director によりリングファイルが作成されないようにします。
- **swift-ring-builder** を使用して手動でクラスターにノードを追加するか、クラスターからノードを削除します。

以下の手順では、クラスターの整合性を保ちながらノードを置き換える方法を説明します。この例では、ノードが 2 つある Object Storage クラスターを使用します。目的は、別のノードを追加して、問題のあるノードを置き換えることです。

まず、以下の内容が含まれる `~/templates/swift-ring-prevent.yaml` という名前の環境ファイルを作成します。

```
parameter_defaults:
  SwiftRingBuild: false
  RingBuild: false
  ObjectStorageCount: 3
```

**SwiftRingBuild** および **RingBuild** パラメーターにより、オーバークラウドが自動的に Object Storage とコントローラーノードそれぞれのリングファイルを構築するかどうか定義されます。**ObjectStorageCount** は、環境内で Object Storage ノードをいくつ指定するかを定義します。今回の例では、ノードを 2 つから 3 つにスケーリングします。

**swift-ring-prevent.yaml** の一部として、オーバークラウドの残りの環境ファイルと合わせて、**openstack overcloud deploy** ファイルも追加します。

```
$ openstack overcloud deploy --templates [ENVIRONMENT_FILES] -e swift-
ring-prevent.yaml
```



### 注記

このファイルのパラメーターが以前の環境ファイルのパラメーターよりも優先されるように、このファイルを環境ファイルの一覧の最後に追加します。

再デプロイメントが完了したら、オーバークラウドには追加の Object Storage が含まれるようになりますが、ノードのストレージディレクトリーは作成されておらず、ノードのオブジェクトストアのリングファイルもまだ構築されていません。そのため、手動でストレージのディレクトリーを作成して、リングファイルを構築する必要があります。



### 注記

以下の手順を使用して、コントローラーノードでのリングファイル構築も行なってください。

新規ノードにログインしてストレージのディレクトリーを作成します。

```
$ sudo mkdir -p /srv/node/d1
$ sudo chown -R swift:swift /srv/node/d1
```



### 注記

このディレクトリーで、外部のストレージデバイスをマウントすることもできます。

ノードに既存のリングファイルをコピーします。**heat-admin** ユーザーとして、コントローラーノードにログインしてから、スーパーユーザーに切り替えます。たとえば、IP アドレスが 192.168.201.24 のコントローラーノードの場合は以下ようになります。

```
$ ssh heat-admin@192.168.201.24
$ sudo -i
```

コントローラーノードからの **/etc/swift/\*.builder** ファイルを新しい Object Storage ノードの **/etc/swift/** ディレクトリーにコピーします。必要に応じて、ファイルを director ホストに移動します。

```
[root@overcloud-controller-0 ~]# scp /etc/swift/*.builder
stack@192.1.2.1:~/.
```

次に、新規ノードにファイルを転送します。

```
[stack@director ~]$ scp ~/.builder heat-admin@192.1.2.24:~/.
```

新しい Object Storage ノードに **heat-admin** ユーザーとしてログインして、スーパーユーザーに切り替えます。たとえば、IP アドレスが 192.168.201.29 の Object Storage ノードの場合は以下ようになります。

```
$ ssh heat-admin@192.168.201.29
$ sudo -i
```

これらのファイルを **/etc/swift** ディレクトリーにコピーします。

```
# cp /home/heat-admin/*.builder /etc/swift/.
```

新規の Object Storage ノードをアカウント、コンテナ、オブジェクトリングに追加します。新規ノードに以下のコマンドを実行します。

```
# swift-ring-builder /etc/swift/account.builder add zX-IP:6002/d1 weight
# swift-ring-builder /etc/swift/container.builder add zX-IP:6001/d1 weight
# swift-ring-builder /etc/swift/object.builder add zX-IP:6000/d1 weight
```

これらのコマンドでは、以下の値を置き換えてください。

## zX

X は、指定のゾーンに適した整数に置き換えます (例: ゾーン 1 には z1)。

## IP

アカウント、コンテナ、オブジェクトサービスがリッスンするのに使用する IP。これは、特に、`/etc/swift/object-server.conf` と `/etc/swift/account-server.conf`、`/etc/swift/container-server.conf` の **DEFAULT** セクションの **bind\_ip** など、各ストレージノードの IP アドレスと同じでなければなりません。

## 重み

他のデバイスと比較したデバイスの相対的な重み付けを入力します。通常、これは **100** となっています。



## 注記

リングファイル上で **swift-ring-builder** を使用して、リングファイルにある現在のノードの既存値を確認します。

```
# swift-ring-builder /etc/swift/account.builder
```

アカウント、コンテナ、オブジェクトリングから置き換えるノードを削除します。各ノードに対して、以下のコマンドを実行します。

```
# swift-ring-builder /etc/swift/account.builder remove IP
# swift-ring-builder /etc/swift/container.builder remove IP
# swift-ring-builder /etc/swift/object.builder remove IP
```

**IP** はノードの IP アドレスに置き換えます。

すべてのノードをまたいでパーティションの再配分を行います。

```
# swift-ring-builder /etc/swift/account.builder rebalance
# swift-ring-builder /etc/swift/container.builder rebalance
# swift-ring-builder /etc/swift/object.builder rebalance
```

`/etc/swift/` の全コンテンツの所有者を **root** ユーザーと **swift** グループに変更します。

```
# chown -R root:swift /etc/swift
```

**openstack-swift-proxy** サービスを再起動します。

```
# systemctl restart openstack-swift-proxy.service
```

この時点で、リングファイル (\*.ring.gz および \*.builder) は、新しいノード上で更新されているはずです。

```
/etc/swift/account.builder
/etc/swift/account.ring.gz
/etc/swift/container.builder
/etc/swift/container.ring.gz
/etc/swift/object.builder
/etc/swift/object.ring.gz
```

これらのファイルは、コントローラーノードと既存の Object Storage ノード上の **/etc/swift/** にコピーします (削除するノードは除く)。必要に応じて、ファイルを director ホストに移動します。

```
[root@overcloud-objectstorage-2 swift]# scp *.builder stack@192.1.2.1:~/
[root@overcloud-objectstorage-2 swift]# scp *.ring.gz stack@192.1.2.1:~/
```

次に各ノードで **/etc/swift/** にこのファイルをコピーします。

各ノードで、**/etc/swift/** の内容すべての所有者を **root** ユーザーと **swift** グループに変更します。

```
# chown -R root:swift /etc/swift
```

新規ノードが追加され、リングファイルの一部になります。リングから以前のノードを削除する前に、他のノードから新規ノードの初期のデータを複製する必要があります。

リングから以前のノードを削除するには、**ObjectStorageCount** の数を減らして以前のリングを省略します。今回は 3 から 2 に減らします。

```
parameter_defaults:
  SwiftRingBuild: false
  RingBuild: false
  ObjectStorageCount: 2
```

新規環境ファイル (**remove-object-node.yaml**) を作成して、以前の Object Storage ノードを特定し、削除します。今回の場合は **overcloud-objectstorage-1** を削除します。

```
parameter_defaults:
  ObjectStorageRemovalPolicies:
    [{'resource_list': ['1']}]
```

デプロイメントのコマンドで両環境ファイルを指定します。

```
$ openstack overcloud deploy --templates -e swift-ring-prevent.yaml -e
remove-object-node.yaml ...
```

director は、オーバークラウドから Object Storage ノードを削除して、オーバークラウド上の残りのノードを更新し、ノードの削除に対応します。

## 第10章 オーバークラウドのリブート

アンダークラウドおよびオーバークラウドでノードを再起動する必要がある場合があります。以下の手順では、異なるノード種別を再起動する方法を説明します。以下の点に注意してください。

- 1つのロールで全ノードを再起動する場合には、各ノードを個別に再起動することを推奨しています。この方法は、再起動中にそのロールのサービスを保持するのに役立ちます。
- OpenStack Platform 環境の全ノードを再起動する場合、再起動の順序は以下のリストを参考にしてください。

### 推奨されるノード再起動順

1. director の再起動
2. コントローラーノードの再起動
3. Ceph Storage ノードの再起動
4. コンピュートノードの再起動
5. Object Storage ノードの再起動

### 10.1. DIRECTOR の再起動

director ノードを再起動するには、以下のプロセスに従います。

1. ノードを再起動します。

```
$ sudo reboot
```

2. ノードが起動するまで待ちます。

ノードが起動したら、全サービスのステータスを確認します。

```
$ sudo systemctl list-units "openstack*" "neutron*" "openvswitch"
```

オーバークラウドとそのノードが存在しているかどうかを確認します。

```
$ source ~/stackrc
$ openstack server list
$ ironic node-list
$ openstack stack list
```

### 10.2. コントローラーノードの再起動

コントローラーノードを再起動するには、以下のプロセスに従います。

1. 再起動するノードを選択します。そのノードにログインして再起動します。

```
$ sudo reboot
```

クラスター内の残りのコントローラーノードは、再起動中も高可用性サービスが保持されます。

2. ノードが起動するまで待ちます。
3. ノードにログインして、クラスターのステータスを確認します。

```
$ sudo pcs status
```

このノードは、クラスターに再度参加します。



#### 注記

再起動後に失敗するサービスがあった場合には、`sudo pcs resource cleanup` を実行し、エラーを消去して各リソースの状態を **Started** に設定します。エラーが引き続き発生する場合には、Red Hat にアドバイス/サポートをリクエストしてください。

4. コントローラーノード上の全 **systemd** サービスがアクティブであることを確認します。

```
$ sudo systemctl list-units "openstack*" "neutron*" "openvswitch"
```

5. ノードからログアウトして、次に再起動するコントローラーノードを選択し、すべてのコントローラーノードが再起動されるまでこの手順を繰り返します。

## 10.3. CEPH STORAGE ノードの再起動

Ceph Storage のノードを再起動するには、以下のプロセスに従います。

1. 再起動する最初の Ceph Storage ノードを選択して、ログインします。
2. Ceph Storage クラスターのリバランシングを一時的に無効にします。

```
$ sudo ceph osd set noout
$ sudo ceph osd set norebalance
```

3. ノードを再起動します。

```
$ sudo reboot
```

4. ノードが起動するまで待ちます。
5. ノードにログインして、クラスターのステータスを確認します。

```
$ sudo ceph -s
```

**pgmap** により、すべての **pgs** が正常な状態 (**active+clean**) として報告されることを確認します。

6. ノードからログアウトして、次のノードを再起動し、ステータスを確認します。全 Ceph Storage ノードが再起動されるまで、このプロセスを繰り返します。
7. 操作が完了したら、クラスターのリバランシングを再度有効にします。

```
$ sudo ceph osd unset noout
$ sudo ceph osd unset norebalance
```

8. 最終のステータスチェックを実行して、クラスターが **HEALTH\_OK** と報告することを確認します。

```
$ sudo ceph status
```

## 10.4. コンピュートノードの再起動

コンピュートノードを個別に再起動して、OpenStack Platform 環境のインスタンスのダウンタイムがゼロになるようにします。この操作は、以下のワークフローに従って実行します。

1. 再起動するコンピュートノードを選択します。
2. インスタンスを別のコンピュートノードに移行します。
3. 空のコンピュートノードを再起動します。

全コンピュートノードとその UUID を一覧表示します。

```
$ nova list | grep "compute"
```

再起動するコンピュートノードを選択してから、まず最初に以下のプロセスに従ってそのノードのインスタンスを移行します。

1. アンダークラウドから、再起動するコンピュートノードを選択し、そのノードを無効にします。

```
$ source ~/overcloudrc
$ openstack compute service list
$ openstack compute service set [hostname] nova-compute --disable
```

2. コンピュートノード上の全インスタンスを一覧表示します。

```
$ openstack server list --host [hostname]
```

3. インスタンスの移行のターゲットホストとして機能する 2 番目のコンピュートノードを選択し、このホストには、移行されるインスタンスをホストするのに十分なリソースが必要です。無効化されたホストからターゲットホストに各インスタンスを移行する操作をアンダークラウドから実行します。

```
$ nova live-migration [instance-name] [target-hostname]
$ nova migration-list
$ nova resize-confirm [instance-name]
```

4. コンピュートノードからすべてのインスタンスが移行されるまで、このステップを繰り返します。



### 重要

インスタンスの設定および移行に関する詳しい説明については、「[オーバークラウドのコンピュートノードからの仮想マシンの移行](#)」を参照してください。



以下の手順に従ってコンピュートノードを再起動します。

1. コンピュートノードのログインしてリブートします。

```
$ sudo reboot
```

2. ノードが起動するまで待ちます。
3. コンピュートノードを再度有効化します。

```
$ source ~/overcloudrc  
$ openstack compute service set [hostname] nova-compute --enable
```

4. リブートする次のノードを選択します。

## 10.5. OBJECT STORAGE ノードの再起動

Object Storage ノードを再起動するには、以下のプロセスに従います。

1. 再起動する Object Storage ノードを選択します。そのノードにログインして再起動します。

```
$ sudo reboot
```

2. ノードが起動するまで待ちます。
3. ノードにログインして、ステータスを確認します。

```
$ sudo systemctl list-units "openstack-swift*"
```

4. ノードからログアウトして、次の Object Storage ノードでこのプロセスを繰り返します。

## 第11章 DIRECTOR の問題のトラブルシューティング

director プロセスの特定の段階で、エラーが発生する可能性があります。本項では、一般的な問題の診断に関する情報を提供します。

director のコンポーネントの共通ログを確認してください。

- **/var/log** ディレクトリーには、多数の OpenStack Platform の共通コンポーネントのログや、標準の Red Hat Enterprise Linux アプリケーションのログが含まれています。
- **journal** サービスは、さまざまなコンポーネントのログを提供します。Ironic は **openstack-ironic-api** と **openstack-ironic-conductor** の 2 つのユニットを使用する点に注意してください。同様に、**ironic-inspector** は **openstack-ironic-inspector** と **openstack-ironic-inspector-dnsmasq** の 2 つのユニットを使用します。該当するコンポーネントごとに両ユニットを使用します。以下に例を示します。

```
$ sudo journalctl -u openstack-ironic-inspector -u openstack-ironic-inspector-dnsmasq
```

- **ironic-inspector** は、**/var/log/ironic-inspector/ramdisk/** に ramdisk ログを gz 圧縮の tar ファイルとして保存します。ファイル名には、日付、時間、ノードの IPMI アドレスが含まれます。イントロスペクションの問題を診断するには、これらのログを使用します。

### 11.1. ノード登録のトラブルシューティング

ノード登録における問題は通常、ノードの情報が間違っていることが原因で発生します。このような場合には、**ironic** を使用して、登録したノードデータの問題を修正します。以下にいくつか例を示します。

割り当てられたポートの UUID を確認します。

```
$ ironic node-port-list [NODE UUID]
```

MAC アドレスを更新します。

```
$ ironic port-update [PORT UUID] replace address=[NEW MAC]
```

以下のコマンドを実行します。

```
$ ironic node-update [NODE UUID] replace driver_info/ipmi_address=[NEW IPMI ADDRESS]
```

### 11.2. ハードウェアイントロスペクションのトラブルシューティング

イントロスペクションのプロセスは最後まで実行する必要があります。ただし、Ironic の Discovery Daemon (**ironic-inspector**) は、検出する ramdisk が応答しない場合にはデフォルトの 1 時間が経過するとタイムアウトします。検出 ramdisk のバグが原因とされる場合もありますが、通常は特に BIOS の起動設定などの環境の誤設定が原因で発生します。

以下には、環境設定が間違っている場合の一般的なシナリオと、診断/解決方法に関するアドバイスを示します。

#### ノードのイントロスペクション開始におけるエラー

通常、イントロスペクションプロセスは、Ironic サービス全体に対するコマンドとして機能する **baremetal introspection** を使用します。ただし、**ironic-inspector** で直接イントロスペクションを実行している場合には、「AVAILABLE」の状態のノードの検出に失敗する可能性があります。このコマンドは、デプロイメント用であり、検出用ではないためです。検出前に、ノードの状態を「MANAGEABLE」に変更します。

```
$ ironic node-set-provision-state [NODE UUID] manage
```

検出が完了したら、状態を「AVAILABLE」に戻してからプロビジョニングを行います。

```
$ ironic node-set-provision-state [NODE UUID] provide
```

### イントロスペクション済みのノードが PXE でブートできない問題

ノードがリブートする前に、**ironic-inspector** はアンダークラウドのファイアウォールの **ironic-inspector** チェーンに MAC アドレスを追加します。これにより、ノードは PXE でブートします。設定が正しいことを確認するには、以下のコマンドを実行します。

```
$ `sudo iptables -L`
```

出力には、以下のチェーンテーブルで MAC アドレスが表示されるはずです。

```
Chain ironic-inspector (1 references)
target      prot opt source                destination           MAC
DROP        all  --  anywhere              anywhere              MAC
XX:XX:XX:XX:XX:XX
ACCEPT      all  --  anywhere              anywhere
```

MAC アドレスが表示されない場合に最も多く見られる原因は、SQLite データベース内にある **ironic-inspector** キャッシュの破損です。この問題を修正するには、以下のコマンドで SQLite ファイルを削除します。

```
$ sudo rm /var/lib/ironic-inspector/inspector.sqlite
```

次に以下のコマンドで、このファイルを再度作成します。

```
$ sudo ironic-inspector-dbsync --config-file /etc/ironic-
inspector/inspector.conf upgrade
$ sudo systemctl restart openstack-ironic-inspector
```

### 検出プロセスの停止

現在 **ironic-inspector** では、直接検出プロセスを停止する方法はありません。回避策として、プロセスがタイムアウトするまで待つことを推奨します。必要であれば、**/etc/ironic-inspector/inspector.conf** の **timeout** 設定を別のタイムアウト時間 (分単位) に変更します。

最悪の場合には以下のプロセスを使用して全ノードの検出を停止することができます。

各ノードの電源状態を OFF に変更します。

```
$ ironic node-set-power-state [NODE UUID] off
```

**ironic-inspector** キャッシュを削除して、再起動します。

```
$ rm /var/lib/ironic-inspector/inspector.sqlite
```

**ironic-inspector** キャッシュを再同期します。

```
$ sudo ironic-inspector-dbsync --config-file /etc/ironic-inspector/inspector.conf upgrade
$ sudo systemctl restart openstack-ironic-inspector
```

## イントロスペクション ramdisk へのアクセス

イントロスペクションの ramdisk は、動的なログイン要素を使用します。これは、イントロスペクションのデバッグ中にノードにアクセスするための一時パスワードまたは SSH キーのいずれかを提供できることを意味します。以下のプロセスを使用して、ramdisk アクセスを設定します。

1. 以下のように **openssl passwd -1** コマンドに一時パスワードを指定して MD5 ハッシュを生成します。

```
$ openssl passwd -1 mytestpassword
$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/
```

2. **/httpboot/inspector.ipxe** ファイルを編集して、**kernel** で開始する行を特定し、**rootpwd** パラメーターと MD5 ハッシュを追記します。以下に例を示します。

```
kernel http://192.2.0.1:8088/agent.kernel ipa-inspection-callback-url=http://192.168.0.1:5050/v1/continue ipa-inspection-collectors=default,extra-hardware,logs
systemd.journald.forward_to_console=yes BOOTIF=${mac} ipa-debug=1
ipa-inspection-benchmarks=cpu,mem,disk
rootpwd="$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/" selinux=0
```

または、**sshkey** パラメーターに SSH 公開キーを追記します。



### 注記

**rootpwd** および **sshkey** のパラメーターにはいずれも引用符が必要です。

3. イントロスペクションを開始し、**arp** コマンドまたは DHCP のログから IP アドレスを特定します。

```
$ arp
$ sudo journalctl -u openstack-ironic-inspector-dnsmasq
```

4. 一時パスワードまたは SSH キーを使用して、root ユーザーとして SSH 接続します。

```
$ ssh root@192.0.2.105
```

## イントロスペクションストレージのチェック

director は OpenStack Object Storage (swift) を使用して、イントロスペクションプロセス中に取得したハードウェアデータを保存します。このサービスが稼働していない場合には、イントロスペクションは失敗する場合があります。以下のコマンドを実行して、OpenStack Object Storage に関連したサービスをすべてチェックし、このサービスが稼働中であることを確認します。

```
$ sudo systemctl list-units openstack-swift*
```

## 11.3. オーバークラウドの作成のトラブルシューティング

デプロイメントが失敗する可能性のあるレイヤーは 3 つあります。

- Orchestration (Heat および Nova サービス)
- Bare Metal Provisioning (Ironic サービス)
- デプロイメント後の設定 (Puppet)

オーバークラウドのデプロイメントがこれらのレベルで失敗した場合には、OpenStack クライアントおよびサービスログファイルを使用して、失敗したデプロイメントの診断を行います。

### 11.3.1. Orchestration

多くの場合は、オーバークラウドの作成に失敗した後に、Heat により失敗したオーバークラウドスタックが表示されます。

```
$ heat stack-list
+-----+-----+-----+-----+
+-----+
| id                | stack_name | stack_status      | creation_time
|
+-----+-----+-----+-----+
+-----+
| 7e88af95-535c-4a55... | overcloud  | CREATE_FAILED     | 2015-04-
06T17:57:16Z |
+-----+-----+-----+-----+
+-----+
```

スタック一覧が空の場合には、初期の Heat 設定に問題があることが分かります。Heat テンプレートと設定オプションをチェックし、さらに **openstack overcloud deploy** を実行後のエラーメッセージを確認してください。

### 11.3.2. Bare Metal Provisioning

**ironic** をチェックして、全登録ノードと現在の状態を表示します。

```
$ ironic node-list

+-----+-----+-----+-----+-----+-----+
+-----+
| UUID      | Name | Instance UUID | Power State | Provision State |
Maintenance |
+-----+-----+-----+-----+-----+-----+
+-----+
| f1e261... | None | None          | power off   | available        | False
|
| f0b8c1... | None | None          | power off   | available        | False
|
+-----+-----+-----+-----+-----+-----+
+-----+
```

■

プロビジョニングプロセスでよく発生する問題を以下に示します。

- 結果の表の「Provision State」および「Maintenance」の列を確認します。以下をチェックしてください。
  - 空の表または、必要なノード数よりも少ない
  - 「Maintenance」が True に設定されている
  - プロビジョニングの状態が **manageable** に設定されている。これにより、登録または検出プロセスに問題があることが分かります。たとえば、「Maintenance」が True に自動的に設定された場合は通常、ノードの電源管理の認証情報が間違っています。
- 「Provision State」が **available** の場合には、ベアメタルのデプロイメントが開始される前に問題が発生します。
- 「Provision State」が **active** で、「Power Stat」が **power on** の場合には、ベアメタルのデプロイメントは正常に完了しますが、問題は、デプロイメント後の設定ステップで発生することになります。
- ノードの「Provision State」が **wait call-back** の場合には、このノードではまだ Bare Metal Provisioning プロセスが完了していません。このステータスが変更されるまで待機してください。または、問題のあるノードの仮想コンソールに接続して、出力を確認します。
- 「Provision State」が **error** または **deploy failed** の場合には、このノードの Bare Metal Provisioning は失敗しています。ベアメタルノードの詳細を確認してください。

```
$ ironic node-show [NODE UUID]
```

エラーの説明が含まれる **last\_error** フィールドがないか確認します。エラーメッセージは曖昧なため、ログを使用して解明します。

```
$ sudo journalctl -u openstack-ironic-conductor -u openstack-ironic-api
```

- **wait timeout error** が表示されており、「Power State」が **power on** の場合には、問題のあるノードの仮想コンソールに接続して、出力を確認します。

### 11.3.3. デプロイメント後の設定

設定ステージでは多くのことが発生する可能性があります。たとえば、設定に問題があるために、特定の Puppet モジュールの完了に失敗する可能性があります。本項では、これらの問題を診断するプロセスを説明します。

オーバークラウドスタックからのリソースをすべて表示して、どのスタックに問題があるのかを確認します。

```
$ heat resource-list overcloud
```

このコマンドでは、全リソースとその状態の表が表示されるため、**CREATE\_FAILED** の状態のリソースを探します。

問題のあるリソースを表示します。

■

```
$ heat resource-show overcloud [FAILED RESOURCE]
```

**resource\_status\_reason** のフィールドの情報で診断に役立つ可能性のあるものを確認します。

**nova** コマンドを使用して、オーバークラウドノードの IP アドレスを表示します。

```
$ nova list
```

デプロイされたノードの 1 つに **heat-admin** ユーザーとしてログインします。たとえば、スタックのリソース一覧から、コントローラーノード上にエラーが発生していることが判明した場合には、コントローラーノードにログインします。**heat-admin** ユーザーには、**sudo** アクセスが設定されています。

```
$ ssh heat-admin@192.0.2.14
```

**os-collect-config** ログを確認して、考えられる失敗の原因をチェックします。

```
$ sudo journalctl -u os-collect-config
```

場合によっては、Nova によるノードへのデプロイメントが完全に失敗する可能性があります。このような場合にはオーバークラウドのロール種別の 1 つの **OS::Heat::ResourceGroup** が失敗していることが示されるため、**nova** を使用して問題を確認します。

```
$ nova list
$ nova show [SERVER ID]
```

最もよく表示されるエラーは、**No valid host was found** のエラーメッセージです。このエラーのトラブルシューティングについては、[「No Valid Host Found」エラーのトラブルシューティング](#)を参照してください。その他の場合は、以下のログファイルを参照してトラブルシューティングを実施してください。

- **/var/log/nova/\***
- **/var/log/heat/\***
- **/var/log/ironic/\***

システムのハードウェアおよび設定に関する情報を収集する SOS ツールセットを使用します。この情報は、診断目的とデバッグに使用します。SOS は通常、技術者や開発者のサポートに使用され、アンダークラウドでもオーバークラウドでも便利です。以下のコマンドで **sos** パッケージをインストールします。

```
$ sudo yum install sos
```

レポートを生成します。

```
$ sudo sosreport --all-logs
```

コントローラーノードのデプロイ後のプロセスは、6 つの主なステップで構成されます。以下のステップが含まれます。

表11.1 コントローラーノードの設定ステップ

手順	説明
<b>ControllerLoadBalancerDeployment_Step1</b>	Pacemaker、RabbitMQ、Memcached、Redis、および Galera を含むロードバランシング用のソフトウェアの初期設定。
<b>ControllerServicesBaseDeployment_Step2</b>	Pacemaker の設定、HAProxy、MongoDB、Galera、Ceph Monitor、OpenStack Platform の各種サービス用のデータベースの初期化を含む、クラスターの初期設定
<b>ControllerRingbuilderDeployment_Step3</b>	OpenStack Object Storage ( <b>swift</b> ) 用のリングファイルの初期構築
<b>ControllerOvercloudServicesDeployment_Step4</b>	OpenStack Platform の全サービス ( <b>nova</b> 、 <b>neutron</b> 、 <b>cinder</b> 、 <b>sahara</b> 、 <b>ceilometer</b> 、 <b>heat</b> 、 <b>horizon</b> 、 <b>aodh</b> 、 <b>gnocchi</b> ) の設定
<b>ControllerOvercloudServicesDeployment_Step5</b>	サービス起動順序やサービス起動パラメーターを決定するための制約事項を含む、Pacemaker でのサービスの起動設定値の設定
<b>ControllerOvercloudServicesDeployment_Step6</b>	オーバークラウドの設定の最終段階

## 11.4. プロビジョニングネットワークでの IP アドレスの競合に対するトラブルシューティング

宛先のホストに、すでに使用中の IP アドレスが割り当てられている場合には、検出およびデプロイメントのタスクは失敗します。この問題を回避するには、プロビジョニングネットワークのポートスキャンを実行して、検出の IP アドレス範囲とホストの IP アドレス範囲が解放されているかどうかを確認することができます。

アンダークラウドホストで以下のステップを実行します。

**nmap** をインストールします。

```
# yum install nmap
```

**nmap** を使用して、アクティブなアドレスの IP アドレス範囲をスキャンします。この例では、192.0.2.0/24 の範囲をスキャンします。この値は、プロビジョニングネットワークの IP サブネットに置き換えてください (CIDR 表記のビットマスク)。

```
# nmap -sn 192.0.2.0/24
```

**nmap** スキャンの出力を確認します。

たとえば、アンダークラウドおよびサブネット上に存在するその他のホストの IP アドレスを確認する必要があります。アクティブな IP アドレスが undercloud.conf の IP アドレス範囲と競合している場合



には、オーバークラウドノードのイントロスペクションまたはデプロイを実行する前に、IP アドレスの範囲を変更するか、IP アドレスを解放するかのいずれかを行う必要があります。

```
# nmap -sn 192.0.2.0/24
```

```
Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-02 15:14 EDT
Nmap scan report for 192.0.2.1
Host is up (0.00057s latency).
Nmap scan report for 192.0.2.2
Host is up (0.00048s latency).
Nmap scan report for 192.0.2.3
Host is up (0.00045s latency).
Nmap scan report for 192.0.2.5
Host is up (0.00040s latency).
Nmap scan report for 192.0.2.9
Host is up (0.00019s latency).
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.45 seconds
```

## 11.5. "NO VALID HOST FOUND" エラーのトラブルシューティング

`/var/log/nova/nova-conductor.log` に、以下のエラーが含まれる場合があります。

```
NoValidHost: No valid host was found. There are not enough hosts
available.
```

これは、Nova Scheduler が新規インスタンスを起動するのに適したベアメタルノードを検出できなかったことを意味し、そうすると通常は、Nova が検出を想定しているリソースと、Ironic が Nova に通知するリソースが一致しなくなります。その場合には、以下の点をチェックしてください。

1. イントロスペクションが正常に完了することを確認してください。または、各ノードに必要な Ironic ノードのプロパティが含まれていることをチェックしてください。各ノードに以下のコマンドを実行します。

```
$ ironic node-show [NODE UUID]
```

**properties** JSON フィールドの **cpus**、**cpu\_arch**、**memory\_mb**、**local\_gb** キーに有効な値が指定されていることを確認してください。

2. 使用する Nova フレーバーが、必要なノード数において、上記の Ironic ノードプロパティを超えていないかどうかを確認します。

```
$ nova flavor-show [FLAVOR NAME]
```

3. **ironic node-list** の通りに **available** の状態のノードが十分に存在することを確認します。ノードの状態が **manageable** の場合は通常、イントロスペクションに失敗しています。
4. また、ノードがメンテナンスモードではないことを確認します。**ironic node-list** を使用してチェックしてください。通常、自動でメンテナンスモードに切り替わるノードは、電源の認証情報が間違っています。認証情報を確認して、メンテナンスモードをオフにします。

```
$ ironic node-set-maintenance [NODE UUID] off
```

5. Automated Health Check (AHC) ツールを使用して、自動でノードのタグ付けを行う場合には、

各フレーバー/フレーバーに対応するノードが十分に存在することを確認します。**properties** フィールドの **capabilities** キーに **ironic node-show** がないか確認します。たとえば、コンピュートロールのタグを付けられたノードには、**profile:compute** が含まれているはずです。

- イントロスペクションの後に Ironic から Nova にノードの情報が反映されるには若干時間がかかります。これは通常、director のツールが原因です。ただし、一部のステップを手動で実行した場合には、短時間ですが、Nova でノードが利用できなくなる場合があります。以下のコマンドを使用して、システム内の合計リソースをチェックします。

```
$ nova hypervisor-stats
```

## 11.6. オーバークラウド作成後のトラブルシューティング

オーバークラウドを作成した後は、将来そのオーバークラウドで特定の操作を行うようにすることができます。たとえば、利用可能なノードをスケーリングしたり、障害の発生したノードを置き換えたりすることができます。このような操作を行うと、特定の問題が発生する場合があります。本項には、オーバークラウド作成後の操作が失敗した場合の診断とトラブルシューティングに関するアドバイスを記載します。

### 11.6.1. オーバークラウドスタックの変更

director を使用して **overcloud** スタックを変更する際に問題が発生する場合があります。スタックの変更例には、以下のような操作が含まれます。

- ノードのスケーリング
- ノードの削除
- ノードの置き換え

スタックの変更は、スタックの作成と似ており、director は要求されたノード数が利用可能かどうかをチェックして追加のノードをプロビジョニングしたり、既存のノードを削除してから、Puppet の設定を適用します。**overcloud** スタックを変更する場合に従うべきガイドラインを以下に記載します。

第一段階として、「[デプロイメント後の設定](#)」に記載したアドバイスに従います。この手順と同じステップを、**overcloud** Heat スタック更新の問題の診断に役立てることができます。特に、以下のコマンドを使用して問題のあるリソースを特定します。

#### heat stack-list --show-nested

全スタックを一覧表示します。**--show-nested** はすべての子スタックとそれぞれの親スタックを表示します。このコマンドは、スタックでエラーが発生した時点を特定するのに役立ちます。

#### heat resource-list overcloud

**overcloud** スタック内の全リソースとそれらの状態を一覧表示します。このコマンドは、スタック内でどのリソースが原因でエラーが発生しているかを特定するのに役立ちます。このリソースのエラーの原因となっている Heat テンプレートコレクションと Puppet モジュール内のパラメーターと設定を特定することができます。

#### heat event-list overcloud

**overcloud** スタックに関連するすべてのイベントを時系列で一覧表示します。これには、スタック内の全リソースのイベント開始/完了時間とエラーが含まれます。この情報は、リソースの障害点を特定するのに役立ちます。

以下のセクションには、特定の種別のノード上の問題診断に関するアドバイスを記載します。

### 11.6.2. コントローラーサービスのエラー

オーバークラウドコントローラーノードには、Red Hat OpenStack Platform のサービスの大部分が含まれます。同様に、高可用性のクラスターで複数のコントローラーノードを使用することができます。ノード上の特定のサービスに障害が発生すると、高可用性のクラスターは一定レベルのフェイルオーバーを提供します。ただし、オーバークラウドをフル稼働させるには、障害のあるサービスの診断が必要となります。

コントローラーノードは、Pacemaker を使用して高可用性クラスター内のリソースとサービスを管理します。Pacemaker Configuration System (**pcs**) コマンドは、Pacemaker クラスターを管理するツールです。クラスター内のコントローラーノードでこのコマンドを実行して、設定およびモニタリングの機能を実行します。高可用性クラスター上のオーバークラウドサービスのトラブルシューティングに役立つコマンドを以下にいくつか記載します。

#### **pcs status**

有効なリソース、エラーが発生したリソース、オンラインのノードなどを含む、クラスター全体のステータス概要を提供します。

#### **pcs resource show**

リソースの一覧をそれぞれのノードで表示します。

#### **pcs resource disable [resource]**

特定のリソースを停止します。

#### **pcs resource enable [resource]**

特定のリソースを起動します。

#### **pcs cluster standby [node]**

ノードをスタンバイモードに切り替えます。そのノードはクラスターで利用できなくなります。このコマンドは、クラスターに影響を及ぼさずに特定のノードメンテナンスを実行するのに役立ちます。

#### **pcs cluster unstandby [node]**

ノードをスタンバイモードから解除します。ノードはクラスター内で再度利用可能となります。

これらの Pacemaker コマンドを使用して障害のあるコンポーネントおよびノードを特定します。コンポーネントを特定した後は、**/var/log/** でそれぞれのコンポーネントのログファイルを確認します。

### 11.6.3. Compute サービスのエラー

コンピュートノードは、Compute サービスを使用して、ハイパーバイザーベースの操作を実行します。これは、このサービスを中心にコンピュートノードのメインの診断が行われていることを意味します。以下に例を示します。

- 以下の **systemd** 機能を使用してサービスのステータスを確認します。

```
$ sudo systemctl status openstack-nova-compute.service
```

同様に、以下のコマンドを使用して、サービスの **systemd** ジャーナルを確認します。

```
$ sudo journalctl -u openstack-nova-compute.service
```

- コンピュートノードの主なログファイルは **/var/log/nova/nova-compute.log** です。コンピュートノードの通信で問題が発生した場合には、このログファイルは診断を開始するのに適した場所です。

- コンピュートノードでメンテナンスを実行する場合には、既存のインスタンスをホストから稼働中のコンピュートノードに移行し、ノードを無効にします。ノードの移行についての詳しい情報は、「[オーバークラウドのコンピュートノードからの仮想マシンの移行](#)」を参照してください。

#### 11.6.4. Ceph Storage サービスのエラー

Red Hat Ceph Storage クラスタで発生した問題については、『Red Hat Ceph Storage Configuration Guide』の「[Part 10. Logging and Debugging](#)」を参照してください。本項では、全 Ceph Storage サービスのログ診断についての情報を記載します。

### 11.7. アンダークラウドの調整

このセクションでのアドバイスは、アンダークラウドのパフォーマンスを向上に役立たせることが目的です。必要に応じて、推奨事項を実行してください。

- Identity Service (keystone) は、他の OpenStack サービスに対するアクセス制御にトークンベースのシステムを使用します。ある程度の期間が過ぎた後には、データベースに未使用のトークンが多数たまります。デフォルトの cronjob は毎日トークンをフラッシュします。環境をモニタリングして、トークンのフラッシュ間隔を調節することを推奨します。アンダークラウドの場合は、**crontab -u keystone -e** のコマンドで間隔を調整することができます。この変更は一時的で、**openstack undercloud update** を実行すると、この cronjob の設定はデフォルトに戻ってしまう点に注意してください。
- Heat は、**openstack overcloud deploy** を実行するたびにデータベースの **raw\_template** テーブルにある全一時ファイルのコピーを保存します。**raw\_template** テーブルは、過去のテンプレートをすべて保持し、サイズが増加します。**raw\_templates** テーブルにある未使用のテンプレートを削除するには、以下のように、日次の cron ジョブを作成して、未使用のまま 1 日以上データベースに存在するテンプレートを消去してください。

```
0 04 * * * /bin/heat-manage purge_deleted -g days 1
```

- **openstack-heat-engine** および **openstack-heat-api** サービスは、一度に過剰なリソースを消費する可能性があります。そのような場合は **/etc/heat/heat.conf** で **max\_resources\_per\_stack=-1** を設定して、Heat サービスを再起動します。

```
$ sudo systemctl restart openstack-heat-engine openstack-heat-api
```

- **director** には、同時にノードをプロビジョニングするリソースが十分でない場合があります。同時に提供できるノード数はデフォルトで 10 個となっています。同時にプロビジョニングするノード数を減らすには、**/etc/nova/nova.conf** の **max\_concurrent\_builds** パラメーターを 10 未満に設定して Nova サービスを再起動します。

```
$ sudo systemctl restart openstack-nova-api openstack-nova-scheduler
```

- **/etc/my.cnf.d/server.cnf** ファイルを編集します。調整が推奨される値は、以下のとおりです。

**max\_connections**

データベースに同時接続できる数。推奨の値は 4096 です。

**innodb\_additional\_mem\_pool\_size**

データベースがデータのディクショナリーの情報や他の内部データ構造を保存するのに使用するメモリープールサイズ (バイト単位)。デフォルトは通常 8 M ですが、アンダークラウドの理想の値は 20 M です。

#### **innodb\_buffer\_pool\_size**

データベースがテーブルやインデックスデータをキャッシュするメモリー領域つまり、バッファプールのサイズ (バイト単位)。通常デフォルトは 128 M で、アンダークラウドの理想の値は 1000 M です。

#### **innodb\_flush\_log\_at\_trx\_commit**

コミット操作の厳密な ACID 準拠と、コミット関連の I/O 操作を再編成してバッチで実行することによって実現可能なパフォーマンス向上の間のバランスを制御します。1 に設定します。

#### **innodb\_lock\_wait\_timeout**

行のロックがされるまで、データベースのトランザクションが待機するのを中断するまでの期間 (秒単位)。

#### **innodb\_max\_purge\_lag**

この変数は、解析操作が遅れている場合に INSERT、UPDATE、DELETE 操作を遅延させる方法を制御します。10000 に設定します。

#### **innodb\_thread\_concurrency**

同時に実行するオペレーティングシステムのスレッド数の上限。理想的には、各 CPU およびディスクリソースに対して少なくとも 2 つのスレッドを提供します。たとえば、クワッドコア CPU と単一のディスクを使用する場合は、スレッドを 10 個使用します。

- オーバークラウドを作成する際には、Heat に十分なワーカーが配置されているようにします。通常、アンダークラウドに CPU がいくつあるかにより左右されます。ワーカーの数を手動で設定するには、`/etc/heat/heat.conf` ファイルを編集して **num\_engine\_workers** パラメーターを必要なワーカー数 (理想は 4) に設定し、Heat エンジン再起動します。

```
$ sudo systemctl restart openstack-heat-engine
```

## 11.8. アンダークラウドとオーバークラウドの重要なログ

以下のログを使用して、トラブルシューティングの際にアンダークラウドとオーバークラウドの情報を割り出します。

表11.2 アンダークラウドとオーバークラウドの重要なログ

情報	アンダークラウドまたはオーバークラウド	ログの場所
一般的な director サービス	アンダークラウド	<code>/var/log/nova/*</code> <code>/var/log/heat/*</code> <code>/var/log/ironic/*</code>
イントロスペクション	アンダークラウド	<code>/var/log/ironic/*</code> <code>/var/log/ironic-inspector/*</code>

プロビジョニング	アンダークラウド	<code>/var/log/ironic/*</code>
cloud-init ログ	オーバークラウド	<code>/var/log/cloud-init.log</code>
オーバークラウドの設定 (最後に実行した Puppet のサマリー)	オーバークラウド	<code>/var/lib/puppet/state/latest_run_summary.yaml</code>
オーバークラウドの設定 (最後に実行した Puppet からのレポート)	オーバークラウド	<code>/var/lib/puppet/state/latest_run_report.yaml</code>
オーバークラウドの設定 (全 Puppet レポート)	オーバークラウド	<code>/var/lib/puppet/reports/overcloud-*/*</code>
一般のオーバークラウドサービス	オーバークラウド	<code>/var/log/ceilometer/*</code> <code>/var/log/ceph/*</code> <code>/var/log/cinder/*</code> <code>/var/log/glance/*</code> <code>/var/log/heat/*</code> <code>/var/log/horizon/*</code> <code>/var/log/httpd/*</code> <code>/var/log/keystone/*</code> <code>/var/log/libvirt/*</code> <code>/var/log/neutron/*</code> <code>/var/log/nova/*</code> <code>/var/log/openvswitch/*</code> <code>/var/log/rabbitmq/*</code> <code>/var/log/redis/*</code> <code>/var/log/swift/*</code>
高可用性ログ	オーバークラウド	<code>/var/log/pacemaker.log</code>

## 付録A SSL/TLS 証明書の設定

「[director の設定](#)」または「[オーバークラウドの SSL/TLS の有効化](#)」で説明したプロセスのオプションとして、アンダークラウドまたはオーバークラウドのいずれかでの通信に SSL/TLS を使用するように設定できます。ただし、独自の認証局で発行した SSL 証明書を使用する場合には、その証明書には以下の項に記載する設定のステップが必要です。

### A.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
```

### A.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。

### A.3. クライアントへの認証局の追加

SSL/TLS を使用して通信することを目的としている外部のクライアントの場合は、Red Hat OpenStack Platform 環境にアクセスする必要のある各クライアントに認証局ファイルをコピーします。クライアントへのコピーが完了したら、そのクライアントで以下のコマンドを実行して、認証局のトラストバンドルに追加します。

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

### A.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作成するのに使用します。

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

## A.5. SSL/TLS 証明書署名要求の作成

次の手順では、アンダークラウドおよびオーバークラウドのいずれかの証明書署名要求を作成します。

カスタマイズするデフォルトの OpenSSL 設定ファイルをコピーします。

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

カスタムの **openssl.cnf** ファイルを編集して、director 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 = 192.168.0.1
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 = 192.168.0.1
DNS.1 = instack.localdomain
DNS.2 = vip.localdomain
DNS.3 = 192.168.0.1
```

**commonName\_default** をパブリック API の IP アドレスに設定するか、完全修飾ドメイン名を使用している場合は完全修飾ドメイン名に設定します。

- アンダークラウドでは、**undercloud.conf** で **undercloud\_public\_vip** パラメーターを使用します。この IP アドレスの完全修飾ドメイン名を使用する場合は、そのドメイン名を使用してください。



- オーバークラウドでは、パブリック API の IP アドレスを使用します。これは、ネットワーク分離環境ファイルにある **ExternalAllocationPools** パラメーターの最初のアドレスです。この IP アドレスに完全修飾ドメイン名を使用する場合には、そのドメイン名を使用します。

**alt\_names** セクションを編集して、以下のエントリーを追加します。

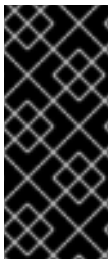
- **IP:** SSL 経由で director にアクセスするためのクライアントの IP アドレス一覧
- **DNS:** SSL 経由で director にアクセスするためのクライアントのドメイン名一覧。**alt\_names** セクションの最後に DNS エントリーとしてパブリック API の IP アドレスも追加します。

**openssl.cnf** に関する詳しい情報については、**man openssl.cnf** を実行します。

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

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out
server.csr.pem
```

「[SSL/TLS 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。



### 重要

**openssl req** コマンドは、Common Name を含む、証明書に関するいくつかの情報を尋ねます。Common Name は、(作成する証明書セットに応じて) アンダークラウドまたはオーバークラウドのパブリック API の IP アドレスに設定するようにしてください。**openssl.cnf** ファイルは、この IP アドレスをデフォルト値として使用する必要があります。

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。

## A.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。

## A.7. アンダークラウドで証明書を使用する場合

以下のコマンドを実行して、証明書とキーを統合します。

```
$ cat server.crt.pem server.key.pem > undercloud.pem
```

このコマンドにより、**undercloud.pem** が作成されます。**undercloud.conf** ファイルの **undercloud\_service\_certificate** オプションにこのファイルの場所を指定します。このファイルには、HAProxy ツールが読み取ることができるように、特別な SELinux コンテキストも必要です。以下の例を目安にしてください。

```
$ sudo mkdir /etc/pki/instack-certs
$ sudo cp ~/undercloud.pem /etc/pki/instack-certs/.
$ sudo semanage fcontext -a -t etc_t "/etc/pki/instack-certs(/.*)?"
$ sudo restorecon -R /etc/pki/instack-certs
```

**undercloud.conf** ファイルの **undercloud\_service\_certificate** オプションに **undercloud.pem** の場所を追記します。以下に例を示します。

```
undercloud_service_certificate = /etc/pki/instack-certs/undercloud.pem
```

また、「[認証局の作成](#)」で作成した認証局をアンダークラウドの信頼済み認証局の一覧に認証局を追加して、アンダークラウド内の異なるサービスが認証局にアクセスできるようにします。

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

「[director の設定](#)」に記載の手順に従ってアンダークラウドのインストールを続行します。

## A.8. オーバークラウドで証明書を使用する場合

「[オーバークラウドの SSL/TLS の有効化](#)」で設定した **enable-tls.yaml** 環境ファイルに証明書ファイルの内容を記載します。オーバークラウドのデプロイメントプロセスでは、**enable-tls.yaml** からパラメーターを取り、オーバークラウド内の各ノードでそれらのパラメーターを自動的に統合します。

## 付録B 電源管理ドライバー

IPMI は、director が電源管理制御に使用する主要な手法ですが、director は他の電源管理タイプもサポートします。この付録では、サポートされる電源管理機能の一覧を提供します。「[オーバークラウドへのノードの登録](#)」には、以下の電源管理設定を使用します。

### B.1. DELL REMOTE ACCESS CONTROLLER (DRAC)

DRAC は、電源管理やサーバー監視などの帯域外 (OOB) リモート管理機能を提供するインターフェースです。

#### pm\_type

このオプションは **pxe\_drac** に設定します。

#### pm\_user; pm\_password

DRAC のユーザー名およびパスワード

#### pm\_addr

DRAC ホストの IP アドレス

### B.2. INTEGRATED LIGHTS-OUT (ILO)

Hewlett-Packard の iLO は、電源管理やサーバー監視などの帯域外 (OOB) リモート管理機能を提供するインターフェースです。

#### pm\_type

このオプションは **pxe\_ilo** に設定します。

#### pm\_user; pm\_password

iLO のユーザー名およびパスワード

#### pm\_addr

iLO インターフェースの IP アドレス

- **/etc/ironic/ironic.conf** ファイルを編集して、**enabled\_drivers** オプションに **pxe\_ilo** を追加し、このドライバーを有効化します。
- また director では、iLO 向けに追加のユーティリティーセットが必要です。**python-proliantutils** パッケージをインストールして **openstack-ironic-conductor** サービスを再起動します。

```
$ sudo yum install python-proliantutils
$ sudo systemctl restart openstack-ironic-conductor.service
```

- HP ノードは、正常にイントロスペクションするには 2015 年のファームウェアバージョンが必要です。ファームウェアバージョン 1.85 (2015 年 5 月 13 日) を使用したノードで、director は正常にテストされています。
- 共有 iLO ポートの使用はサポートされません。

### B.3. CISCO UNIFIED COMPUTING SYSTEM (UCS)

Cisco の UCS は、コンピュータ、ネットワーク、ストレージのアクセス、仮想化リソースを統合するデータセンタープラットフォームです。このドライバーは、UCS に接続されたベアメタルシステムの電源管理を重視しています。

#### pm\_type

このオプションは **pxe\_ucs** に設定します。

#### pm\_user; pm\_password

UCS のユーザー名およびパスワード

#### pm\_addr

UCS インターフェースの IP アドレス

#### pm\_service\_profile

使用する UCS サービスプロファイル。通常 **org-root/ls-[service\_profile\_name]** の形式を取ります。たとえば、以下のとおりです。

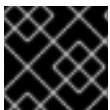
```
"pm_service_profile": "org-root/ls-Nova-1"
```

- **/etc/ironic/ironic.conf** ファイルを編集して、**enabled\_drivers** オプションに **pxe\_ucs** を追加し、このドライバーを有効化します。
- また director では、iLO 向けに追加のユーティリティーセットが必要です。 **python-UcsSdk** パッケージをインストールして **openstack-ironic-conductor** サービスを再起動します。

```
$ sudo yum install python-UcsSdk
$ sudo systemctl restart openstack-ironic-conductor.service
```

## B.4. FUJITSU INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC)

Fujitsu の iRMC は、LAN 接続と拡張された機能を統合した Baseboard Management Controller (BMC) です。このドライバーは、iRMC に接続されたベアメタルシステムの電源管理にフォーカスしています。



### 重要

iRMC S4 以降のバージョンが必要です。

#### pm\_type

このオプションを **pxe\_irmc** に設定します。

#### pm\_user; pm\_password

iRMC インターフェースのユーザー名とパスワード

#### pm\_addr

iRMC インターフェースの IP アドレス

#### pm\_port (オプション)

iRMC の操作に使用するポート。デフォルトは 443 です。

#### pm\_auth\_method (オプション)

iRMC 操作の認証方法。 **basic** または **digest** を使用します。デフォルトは **basic** です。

### pm\_client\_timeout (オプション)

iRMC 操作のタイムアウト (秒単位)。デフォルトは 60 秒です。

### pm\_sensor\_method (オプション)

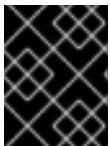
センサーデータの取得方法。**ipmitool** または **scciclient** です。デフォルトは **ipmitool** です。

- **/etc/ironic/ironic.conf** ファイルを編集して、**enabled\_drivers** オプションに **pxe\_irmc** を追加し、このドライバーを有効化します。
- センサーの方法として SCCI を有効にした場合には、director には、追加のユーティリティセットも必要です。**python-scciclient** パッケージをインストールして、**openstack-ironic-conductor** サービスを再起動します。

```
$ yum install python-scciclient
$ sudo systemctl restart openstack-ironic-conductor.service
```

## B.5. SSH と VIRSH

director は、libvirt を実行中のホストに SSH 経由でアクセスして、仮想マシンをノードとして使用することができます。director は、virsh を使用してこれらのノードの電源管理の制御を行います。



### 重要

このオプションは、テストおよび評価の目的でのみ利用いただけます。Red Hat OpenStack Platform のエンタープライズ環境には推奨していません。

### pm\_type

このオプションは **pxe\_ssh** に設定します。

### pm\_user; pm\_password

SSH ユーザー名および秘密鍵の内容。秘密鍵は一行に記載する必要があり、改行はエスケープ文字 (`\n`) に置き換えます。以下に例を示します。

```
-----BEGIN RSA PRIVATE KEY-----\nMIIEogIBAAKCAQEA .... kk+WXt9Y=\n-----
END RSA PRIVATE KEY-----
```

SSH 公開鍵を libvirt サーバーの **authorized\_keys** コレクションに追加します。

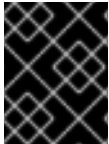
### pm\_addr

virsh ホストの IP アドレス

- libvirt をホストするサーバーでは、公開鍵と SSH のキーペアを **pm\_password** 属性に設定する必要があります。
- 選択した **pm\_user** には libvirt 環境への完全なアクセス権が指定されているようにします。

## B.6. フェイク PXE ドライバー

このドライバーは、電源管理なしにベアメタルデバイスを使用する方法を提供します。これは、director が登録されたベアメタルデバイスを制御しないので、イントロセクションとデプロイの特定の時点に手動で電源をコントロールする必要があることを意味します。



## 重要

このオプションは、テストおよび評価の目的でのみ利用いただけます。Red Hat OpenStack Platform のエンタープライズ環境には推奨していません。

### pm\_type

このオプションは **fake\_pxe** に設定します。

- このドライバーは、電源管理を制御しないので、認証情報は使用しません。
- **/etc/ironic/ironic.conf** ファイルを編集して、**enabled\_drivers** オプションに **fake\_pxe** を追加し、このドライバーを有効化します。ファイルを編集した後は、baremetal サービスを再起動します。

```
$ sudo systemctl restart openstack-ironic-api openstack-ironic-conductor
```

- ノードでイントロスペクションを実行する際には、**openstack baremetal introspection bulk start** コマンドの実行後に手動で電源をオンにします。
- オーバークラウドのデプロイ実行時には、**ironic node-list** コマンドでノードのステータスを確認します。ノードのステータスが **deploying** から **deploy wait-callback** に変わるまで待ってから、手動でノードの電源をオンにします。
- オーバークラウドのプロビジョニングプロセスが完了したら、ノードを再起動します。プロビジョニングが完了したかどうかをチェックするには、**ironic node-list** コマンドでノードのステータスをチェックし、**active** に変わるのを待ってから、すべてのオーバークラウドノードを手動で再起動します。

## 付録C プロファイルの自動タグ付け

イントロスペクションプロセスでは、一連のベンチマークテストを実行します。director は、これらのテストからデータを保存します。このデータをさまざまな方法で使用するポリシーセットを作成することができます。以下に例を示します。

- ポリシーにより、パフォーマンスの低いノードまたは不安定なノードを特定して、オーバークラウドで使用されないように隔離することができます。
- ポリシーにより、ノードを自動的に特定のプロファイルにタグ付けするかどうかを定義することができます。

### C.1. ポリシーファイルの構文

ポリシーファイルは、JSON 形式で、ルールセットが記載されます。各ルールは、**description**、**condition**、**action** を定義します。

#### 説明

これは、プレーンテキストで記述されたルールの説明です。

#### 例:

```
"description": "A new rule for my node tagging policy"
```

#### conditions

condition は、以下のキー/値のパターンを使用して評価を定義します。

#### field

評価するフィールドを定義します。フィールドの種別については、「[プロファイルの自動タグ付けのプロパティ](#)」を参照してください。

#### op

評価に使用する演算を定義します。これには、以下が含まれます。

- **eq**: 等しい
- **ne**: 等しくない
- **lt**: より小さい
- **gt**: より大きい
- **le**: より小さいか等しい
- **ge**: より大きい等しい
- **in-net**: IP アドレスが指定のネットワーク内にあることをチェックします。
- **matches**: 指定の正規表現と完全に一致する必要があります。
- **contains**: 値には、指定の正規表現が含まれる必要があります。
- **is-empty**: フィールドが空欄であることをチェックします。

## invert

評価の結果をインバージョン (反転) するかどうかを定義するブール値

## multiple

複数の結果が存在する場合に、使用する評価を定義します。これには、以下が含まれます。

- **any**: いずれかの結果が一致する必要があります。
- **all**: すべての結果が一致する必要があります。
- **first**: 最初の結果が一致する必要があります。

## value

評価内の値を定義します。フィールドと演算の結果でその値となった場合には、条件は **true** の結果を返します。そうでない場合には、条件は **false** の結果を返します。

例:

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

## actions

action は、condition が **true** として返された場合に実行されます。これには、**action** キーと、**action** の値に応じて追加のキーが使用されます。

- **fail**: イントロスペクションが失敗します。失敗のメッセージには、**message** パラメーターが必要です。
- **set-attribute**: Ironic ノードで属性を設定します。Ironic の属性へのパス (例: **/driver\_info/ipmi\_address**) である **path** フィールドと、設定する **value** が必要です。
- **set-capability**: Ironic ノードでケイパビリティを設定します。新しいケイパビリティに応じた名前と値を指定する **name** および **value** のフィールドが必要です。同じケイパビリティの既存の値は置き換えられます。たとえば、これを使用してノードのプロファイルを定義します。
- **extend-attribute**: **set-attribute** と同じですが、既存の値を一覧として扱い、その一覧に値を追記します。オプションの **unique** パラメーターが **True** に設定すると、対象の値がすでに一覧に含まれている場合には何も追加されません。

例:

```
"actions": [
  {
    "action": "set-capability",
    "name": "profile",
    "value": "swift-storage"
  }
]
```



## C.2. ポリシーファイルの例

以下は、適用するイントロスペクションルールを記載した JSON ファイル (**rules.json**) の一例です。

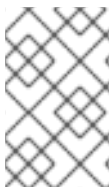
```
[
  {
    "description": "Fail introspection for unexpected nodes",
    "conditions": [
      {
        "op": "lt",
        "field": "memory_mb",
        "value": 4096
      }
    ],
    "actions": [
      {
        "action": "fail",
        "message": "Memory too low, expected at least 4 GiB"
      }
    ]
  },
  {
    "description": "Assign profile for object storage",
    "conditions": [
      {
        "op": "ge",
        "field": "local_gb",
        "value": 1024
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "profile",
        "value": "swift-storage"
      }
    ]
  },
  {
    "description": "Assign possible profiles for compute and controller",
    "conditions": [
      {
        "op": "lt",
        "field": "local_gb",
        "value": 1024
      },
      {
        "op": "ge",
        "field": "local_gb",
        "value": 40
      }
    ],
    "actions": [
      {
        "action": "set-capability",
```

```
[
  {
    "name": "compute_profile",
    "value": "1"
  },
  {
    "action": "set-capability",
    "name": "control_profile",
    "value": "1"
  },
  {
    "action": "set-capability",
    "name": "profile",
    "value": null
  }
]
```

この例には 3 つのルールが含まれます。

- メモリーが 4096 MiB 未満の場合にイントロスペクションが失敗する。このようなルールを適用して、クラウドに含まれないようにノードを除外することができます。
- ハードドライブのサイズが 1 TiB 上のノードの場合は swift-storage プロファイルが無条件で割り当てられる。
- ノードのハードドライブが 1 TiB 未満であるが 40 GiB よりも大きい場合はコンピュートノードかコントロールノードかのいずれかです。**openstack overcloud profiles match** が後ほど最終的に判断を下せるように 2 つのケーパビリティ (**compute\_profile** および **control\_profile**) を割り当てます。これを機能させるには、既存のプロファイルのケーパビリティを削除してください。削除しない場合は優先順位が設定されています。

他のノードは変更されません。



#### 注記

イントロスペクションルールを使用して **profile** 機能を割り当てる場合は常に、既存の値よりこの割り当てた値が優先されます。ただし、既存のプロファイル機能があるノードについては、**[PROFILE]\_profile** 機能は無視されます。

## C.3. ポリシーファイルのインポート

以下のコマンドで、ポリシーファイルを director にインポートします。

```
$ openstack baremetal introspection rule import rules.json
```

次にイントロスペクションプロセスを実行します。

```
$ openstack baremetal introspection bulk start
```

イントロスペクションが完了したら、ノードとノードに割り当てられたプロファイルを確認します。

```
$ openstack overcloud profiles list
```

イントロスペクションルールで間違いがあった場合には、すべてを削除できます。

```
$ openstack baremetal introspection rule purge
```

## C.4. プロファイルの自動タグ付けのプロパティ

プロファイルの自動タグ付けは、各条件の **field** の属性に対する以下のノードプロパティを評価します。

プロパティ	説明
memory_mb	ノードのメモリーサイズ (MB)
cpus	ノードの CPU の合計コア数
cpu_arch	ノードの CPU のアーキテクチャー
local_gb	ノードのルートディスクのストレージの合計容量。 ノードのルートディスクの設定についての詳しい説明は、「 <a href="#">ノードのルートディスクの定義</a> 」を参照してください。

## 付録D 基本パラメーター

以下は、オーバークラウドの設定に使用する基本パラメーターの一覧です。これらのパラメーターは、director のコア Heat テンプレートコレクション内の **overcloud.yaml** ファイルの **parameters** セクションで定義されます。

### AdminPassword

**タイプ:** 文字列

OpenStack Identity の管理アカウントのパスワード

### AdminToken

**タイプ:** 文字列

OpenStack Identity の認証シークレット

### AodhPassword

**タイプ:** 文字列

OpenStack Telemetry Alarming サービスのパスワード

### BlockStorageCount

**タイプ:** 数値

オーバークラウド内の Block Storage ノードの数

### BlockStorageExtraConfig

**タイプ:** json

クラスターに挿入する Block Storage 固有の設定

### BlockStorageHostnameFormat

**タイプ:** 文字列

Block Storage ノードのホスト名の形式

### BlockStorageImage

**タイプ:** 文字列

Block Storage ノードのプロビジョニングに使用するイメージ

### BlockStorageRemovalPolicies

**タイプ:** json

特定のリソースの削除が必要な更新の実行時に **BlockStorageResourceGroup** から削除されるリソースの一覧

### BlockStorageSchedulerHints

**タイプ:** json

オプションのスケジューラーのヒント

### CeilometerBackend

**タイプ:** 文字列

OpenStack Telemetry バックエンドの種別。**mongodb** または **mysql** のいずれかを選択します。

### CeilometerComputeAgent

**タイプ:** 文字列

コンピュートエージェントの有無を示し、それに応じて **nova-compute** が設定されることを想定します。

### CeilometerMeterDispatcher

**タイプ:** 文字列

OpenStack Telemetry (**ceilometer**) サービスには、時系列データストレージ向けの新コンポーネント (**gnocchi**) が含まれています。Red Hat OpenStack Platform では、デフォルトの Ceilometer dispatcher を切り替えて、標準のデータベースの代わりにこの新コンポーネントを使用することができます。これは、**CeilometerMeterDispatcher** で設定します。値は、以下のいずれかを指定します。

- **database**: Ceilometer dispatcher に 標準の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。
- **gnocchi**: Ceilometer dispatcher に新しい時系列データベースを使用します。詳しい情報は、「[ベースパラメーターの設定](#)」の「**例 3**」を参照してください。

### CeilometerMeteringSecret

**タイプ:** 文字列

OpenStack Telemetry サービスによって共有されるシークレット

### CeilometerPassword

**タイプ:** 文字列

OpenStack Telemetry サービスアカウントのパスワード

### CephAdminKey

**タイプ:** 文字列

Ceph 管理クライアントのキー。**ceph-authtool --gen-print-key** で作成することができます。

### CephClientKey

**タイプ:** 文字列

Ceph クライアントのキー。**ceph-authtool --gen-print-key** で作成することができます。現在は、外部の Ceph デプロイメントでの OpenStack ユーザーキーリング作成のみに使用されています。

### CephClusterFSID

**タイプ:** 文字列

Ceph クラスター FSID。UUID である必要があります。

### CephExternalMonHost

**タイプ:** 文字列

外部で管理されている Ceph Monitor ホストの IP。外部の Ceph デプロイメントのみで使用します。

### CephMonKey

**タイプ:** 文字列

Ceph Monitor のキー。**ceph-authtool --gen-print-key** で作成することができます。

### CephStorageCount

**タイプ:** 数値

オーバークラウド内の Ceph Storage ノードの数

### **CephStorageExtraConfig**

**タイプ:** json

クラスターに挿入する Ceph Storage 固有の設定

### **CephStorageHostnameFormat**

**タイプ:** 文字列

Ceph Storage ノードのホスト名の形式

### **CephStorageImage**

**タイプ:** 文字列

Ceph Storage ノードのプロビジョニングに使用するイメージ

### **CephStorageRemovalPolicies**

**タイプ:** json

特定のリソースの削除が必要な更新の実行時に **CephStorageResourceGroup** から削除されるリソースの一覧

### **CephStorageSchedulerHints**

**タイプ:** json

オプションのスケジューラーのヒント

### **CinderEnableiscsiBackend**

**タイプ:** ブール値

Block Storage 用に iSCSI バックエンドを有効化するかどうか

### **CinderEnableNfsBackend**

**タイプ:** ブール値

Block Storage 用に NFS バックエンドを有効化するかどうか

### **CinderEnableRbdBackend**

**タイプ:** ブール値

Block Storage 用に Ceph Storage バックエンドを有効化するかどうか

### **CinderISCSHelper**

**タイプ:** 文字列

Block Storage と共に使用する iSCSI ヘルパー

### **CinderLVMLoopDeviceSize**

**タイプ:** 数値

Block Storage LVM ドライバーによって使用されるループバックファイルのサイズ

### **CinderNfsMountOptions**

**タイプ:** 文字列

Block Storage NFS バックエンドに使用する、NFS マウントのマウントオプション。 **CinderEnableNfsBackend** が true の場合に有効となります。

### CinderNfsServers

**タイプ:** コンマ区切りリスト

Block Storage NFS バックエンドに使用する、NFS サーバー。**CinderEnableNfsBackend** が true の場合に有効となります。

### CinderPassword

**タイプ:** 文字列

Block Storage サービスアカウントのパスワード

### CloudDomain

**タイプ:** 文字列

ホストに使用する DNS ドメイン。これは、アンダークラウドのネットワークで設定されている **dhcp\_domain** と一致する必要があります。デフォルトは **localdomain** です。

### CloudName

**タイプ:** 文字列

このクラウドの DNS 名 (例: **ci-overcloud.tripleo.org**)

### ComputeCount

**タイプ:** 数値

オーバークラウド内のコンピュートノード数

### ComputeHostnameFormat

**タイプ:** 文字列

コンピュートノードのホスト名の形式

### ComputeRemovalPolicies

**タイプ:** json

特定のリソースの削除が必要な更新の実行時に **ComputeResourceGroup** から削除されるリソースの一覧

### ControlFixedIPs

**タイプ:** json

コントローラーノードの Fixed IP の一覧

### ControlVirtualInterface

**タイプ:** 文字列

仮想 IP が割り当てるインターフェース

### ControllerCount

**タイプ:** 数値

オーバークラウド内のコントローラーノード数

### ControllerEnableCephStorage

**タイプ:** ブール値

コントローラーに Ceph Storage (OSD) をデプロイするかどうか

### ControllerEnableSwiftStorage

**タイプ:** ブール値

コントローラーで Object Storage を有効にするかどうか

### **controllerExtraConfig**

**タイプ:** json

クラスターに挿入するコントローラー特有の設定

### **ControllerHostnameFormat**

**タイプ:** 文字列

コントローラーノードのホスト名の形式

### **controllerImage**

**タイプ:** 文字列

Block Storage ノードのプロビジョニングに使用するイメージ

### **ControllerRemovalPolicies**

**タイプ:** json

特定のリソースの削除が必要な更新の実行時に **ControllerResourceGroup** から削除されるリソースの一覧

### **ControllerSchedulerHints**

**タイプ:** json

オプションのスケジューラーのヒント

### **CorosyncIPv6**

**タイプ:** ブール値

Corosync で IPv6 を有効化します。

### **Debug**

**タイプ:** 文字列

全サービスに対するデバッグを有効化するには、**true** に設定します。

### **DeployIdentifier**

**タイプ:** 文字列

オーバークラウドの **stack-update** の設定を実行する任意のデプロイメントタスクを再実行するための一意の値に設定します。

### **EnableFencing**

**タイプ:** ブール値

Pacemaker でフェンシングを有効化するかどうかを定義します。

### **EnableGalera**

**タイプ:** ブール値

通常の MariaDB の代わりに Galera を使用するかどうかを定義します。

### **ExtraConfig**

**タイプ:** json



クラスターに挿入する追加の設定。**controllerExtraConfig**などのロール固有の**ExtraConfig**は、標準の**ExtraConfig**に優先します。

## FencingConfig

**タイプ:** json

Pacemaker のフェンシング設定。JSON は、以下のような構造にする必要があります。

```
{
  "devices": [
    {
      "agent": "AGENT_NAME",
      "host_mac": "HOST_MAC_ADDRESS",
      "params": {"PARAM_NAME": "PARAM_VALUE"}
    }
  ]
}
```

以下に例を示します。

```
{
  "devices": [
    {
      "agent": "fence_xvm",
      "host_mac": "52:54:00:aa:bb:cc",
      "params": {
        "multicast_address": "225.0.0.12",
        "port": "baremetal_0",
        "manage_fw": true,
        "manage_key_file": true,
        "key_file": "/etc/fence_xvm.key",
        "key_file_password": "abcdef"
      }
    }
  ]
}
```

## GlanceBackend

**タイプ:** 文字列

使用する OpenStack Image バックエンドの省略名。**swift**、**rbd**、**file** のいずれかにする必要があります。

## GlanceLogFile

**タイプ:** 文字列

OpenStack Image からのメッセージのログ記録に使用するファイルのパス

## GlanceNotifierStrategy

**タイプ:** 文字列

OpenStack Image の通知のキューに使用するストラテジー。デフォルトでは **noop** です。

## GlancePassword

**タイプ:** 文字列

OpenStack Image サービスアカウントのパスワード。OpenStack Image サービスによって使用されます。

### GnocchiBackend

**タイプ:** 文字列

使用する Gnocchi バックエンドの省略名。**swift**、**rbd**、**file** のいずれかにする必要があります。

### GnocchiIndexerBackend

**タイプ:** 文字列

Gnocchi インデクサーバックエンドの省略名

### GnocchiPassword

**タイプ:** 文字列

Gnocchi サービスアカウントのパスワード

### HAProxySyslogAddress

**タイプ:** 文字列

HAProxy のログの送信先となる Syslog アドレス

### HeatPassword

**タイプ:** 文字列

Heat サービスアカウントのパスワード

### HeatStackDomainAdminPassword

**タイプ:** 文字列

**heat\_stack\_domain\_admin** ユーザーのパスワード

### HorizonAllowedHosts

**タイプ:** コンマ区切りリスト

OpenStack Dashboard への接続が許可されている IP/ホスト名の一覧

### HypervisorNeutronPhysicalBridge

**タイプ:** 文字列

各ハイパーバイザーで作成する OVS ブリッジ。デフォルトでは **br-ex** に設定されます。Open vSwitch エージェントには一貫した設定を使用するので、これは、コントロールプレーンノードと同じです。通常、この値は変更する必要はありません。

### HypervisorNeutronPublicInterface

**タイプ:** 文字列

**HypervisorNeutronPhysicalBridge** に追加するインターフェース

### ImageUpdatePolicy

**タイプ:** 文字列

インスタンスを再構築する際のポリシー。再ビルドには **REBUILD**、**/mnt** を保持するには **REBUILD\_PRESERVE\_EPHEMERAL** です。

### InstanceNameTemplate

**タイプ:** 文字列

テンプレートの文字列は、インスタンス名の生成に使用されます。

### InternalApiVirtualFixedIPs

タイプ: json

InternalApiVirtualInterface ポートの IP 割り当てを制御します。以下に例を示します。

```
[{'ip_address': '1.2.3.4'}]
```

### KeyName

タイプ: 文字列

インスタンスへの SSH アクセスを有効化するための OpenStack Compute のキーペアの名前

### KeystoneCACertificate

タイプ: 文字列

OpenStack Identity の自己署名 CA 証明書

### KeystoneNotificationDriver

タイプ: コンマ区切りリスト

OpenStack Identity が使用する通知ドライバーのコンマ区切りリスト

### KeystoneNotificationFormat

タイプ: 文字列

OpenStack Identity の通知の形式

### KeystoneSSLCertificate

タイプ: 文字列

トークンの有効性を検証するための OpenStack Identity 証明書

### KeystoneSSLCertificateKey

タイプ: 文字列

トークンに署名するための OpenStack Identity キー

### KeystoneSigningCertificate

タイプ: 文字列

トークンの有効性を検証するための OpenStack Identity 証明書

### KeystoneSigningKey

タイプ: 文字列

トークンに署名するための OpenStack Identity キー

### ManageFirewall

タイプ: ブール値

ファイアウォールルールを管理するかどうかを定義します。

### MemcachedIPv6

タイプ: ブール値

Memcached で IPv6 機能を有効化します。

**MongoDbIPv6**

タイプ: ブール値

MongoDB VIP が IPv6 の場合に IPv6 を有効化します。

**MongoDbNoJournal**

タイプ: ブール値

MongoDb ジャーナリングを無効化するかどうかを定義します。

**MysqlInnodbBufferPoolSize**

タイプ: 数値

バッファプールのサイズを MB 単位で指定します。ゼロに設定すると、「値なし」と解釈され、下位のデフォルトに従います。

**MysqlMaxConnections**

タイプ: 数値

MySQL の `max_connections` の設定値を設定します。

**NeutronAgentExtensions**

タイプ: コンマ区切りリスト

OpenStack Networking エージェント用に有効化されている拡張機能のコンマ区切りリスト

**NeutronAgentMode**

タイプ: 文字列

コントローラーホスト上の **neutron-l3-agent** 用のエージェントモード

**NeutronAllowL3AgentFailover**

タイプ: 文字列

L3 エージェントの自動フェイルオーバーを許可します。

**NeutronBridgeMappings**

タイプ: コンマ区切りリスト

使用する OVS 論理/物理間のブリッジマッピング。デフォルトでは、ホスト上の外部ブリッジ (**br-ex**) を物理名 (**datacentre**) にマッピングし、プロバイダーネットワークの作成に使用することができます (また、これをデフォルトのフローティングネットワークに使用します)。この値を変更する場合には、異なるポストインストールネットワークスクリプトを使用するか、**datacentre** をマッピングネットワーク名として維持するようにします。

**NeutronComputeAgentMode**

タイプ: 文字列

コンピュータード上の **neutron-l3-agent** のエージェントモード

**NeutronControlPlaneID**

タイプ: 文字列

**ctlplane** ネットワーク用の Neutron ID または 名前

**NeutronCorePlugin**

タイプ: 文字列

OpenStack Networking のコアプラグイン。値は、`neutron.core_plugins` の名前空間から読み込まれるエントリーポイントにする必要があります。

### NeutronDVR

タイプ: 文字列

OpenStack Networking の分散仮想ルーターを設定するかどうか

### NeutronDhcpAgentsPerNetwork

タイプ: 数値

1 ネットワークにつきスケジュールする OpenStack Networking の dhcp エージェント数

### NeutronDnsmasqOptions

タイプ: 文字列

`neutron-dhcp-agent` の Dnsmasq オプション。このデフォルト値により、MTU が `NeutronTenantMtu` の値に強制的に設定されます。これは、トンネルのオーバーヘッド向けのアカウントに設定する必要があります。

### NeutronEnableIsolatedMetadata

タイプ: 文字列

true の場合には、DHCP が仮想マシンへのメタデータルートを提供します。

### NeutronEnableL2Pop

タイプ: 文字列

OpenStack Networking エージェントの L2 Population 機能を有効化/無効化します。

### NeutronEnableTunnelling

タイプ: 文字列

OpenStack Networking でトンネリングを有効化するかどうかを定義します。

### NeutronExternalNetworkBridge

タイプ: 文字列

外部ネットワークトラフィックに使用するブリッジの名前

### NeutronFlatNetworks

タイプ: コンマ区切りリスト

OpenStack Networking プラグインで設定するフラットなネットワークの一覧を定義します。デフォルトでは、`datacentre` に設定されて、外部ネットワークの作成が許可されます。

### NeutronL3HA

タイプ: 文字列

OpenStack Networking で Layer 3 High Availability (L3HA) を無効にする必要がある場合には、環境ファイルでこの値を `false` に設定してください。

### NeutronMechanismDrivers

タイプ: コンマ区切りリスト

OpenStack Networking テナントネットワーク用のメカニズムドライバー

### NeutronMetadataProxySharedSecret

**タイプ:** 文字列

スプーフィングを防ぐための共有シークレット

### NeutronNetworkType

**タイプ:** コンマ区切りリスト

OpenStack Networking のテナントネットワーク種別

### NeutronNetworkVLANRanges

**タイプ:** コンマ区切りリスト

OpenStack Networking ML2 および OpenVSwitch の VLAN マッピングのサポート範囲。許可されている値については、OpenStack Networking のマニュアルを参照してください。デフォルトでは、**datacentre** 物理ネットワーク上の任意の VLAN を許可します (**NeutronBridgeMappings** 参照)。

### NeutronPassword

**タイプ:** 文字列

OpenStack Networking エージェントが使用する OpenStack Networking サービスアカウントのパスワード

### NeutronPluginExtensions

**タイプ:** コンマ区切りリスト

OpenStack Networking プラグイン用に有効化されている拡張機能のコンマ区切りリスト

### NeutronPublicInterface

**タイプ:** 文字列

ネットワークノード向けに **br-ex** にブリッジするインターフェース

### NeutronPublicInterfaceDefaultRoute

**タイプ:** 文字列

**NeutronPublicInterface** のカスタムのデフォルトルート

### NeutronPublicInterfaceIP

**タイプ:** 文字列

**NeutronPublicInterface** に割り当てるカスタム IP アドレス

### NeutronPublicInterfaceRawDevice

**タイプ:** 文字列

このパラメーター設定した場合には、パブリックインターフェースはこのデバイスを RAW デバイスとして使用する VLAN となります。

### NeutronPublicInterfaceTag

**タイプ:** 文字列

パブリック VLAN を作成するための VLAN タグ。このタグは、各コントロールプレーンノードの外部ブリッジ上にアクセスポートを作成するのに使用され、OpenStack Networking パブリックネットワークから返される IP アドレスがポートに提供されます。

### NeutronServicePlugins

**タイプ:** コンマ区切りリスト

neutron.service\_plugins 名前空間から読み込まれるサービスプラグインのエントリーポイントのコンマ区切りリスト

### NeutronTenantMtu

**タイプ:** 文字列

テナントネットワーク用のデフォルトの MTU。VXLAN/GRE トンネリングの場合には、この設定値を、物理ネットワークの MTU より 50 バイト以上少なくする必要があります。この値は、仮想イーサネットデバイス上の MTU の設定に使用されます。この値は、DHCP を介して仮想マシンホストに割り当てられる MTU を決定するので、**NeutronDnsmasqOptions** の構築に使用されます。

### NeutronTunnelIdRanges

**タイプ:** コンマ区切りリスト

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 ID の範囲を列挙した **<tun\_min>:<tun\_max>** タプルのコンマ区切りリスト

### NeutronTunnelTypes

**タイプ:** コンマ区切りリスト

OpenStack Networking テナントネットワークのトンネルの種別

### NeutronTypeDrivers

**タイプ:** コンマ区切りリスト

読み込むネットワーク種別のドライバーエントリーポイントのコンマ区切りリスト

### NeutronVniRanges

**タイプ:** コンマ区切りリスト

テナントネットワークの割り当てに使用できる VXLAN VNI ID の範囲を列挙した **<vni\_min>:<vni\_max>** タプルのコンマ区切りリスト

### NovaComputeDriver

**タイプ:** 文字列

インスタンスの管理に使用する OpenStack Compute のドライバー。デフォルトでは **libvirt.LibvirtDriver** ドライバーに設定されます。

### NovaComputeExtraConfig

**タイプ:** json

クラスターに挿入するコンピュートノード固有の設定

### NovaComputeLibvirtType

**タイプ:** 文字列

使用する Libvirt の種別を定義します。デフォルトでは **kvm** に設定されます。

### NovaComputeLibvirtVifDriver

**タイプ:** 文字列

ネットワーク用の Libvirt VIF ドライバーの設定

### NovaComputeSchedulerHints

**タイプ:** json

オプションのスケジューラーのヒント

**NovaEnableRbdBackend**

**タイプ:** ブール値

Nova 用に Ceph バックエンドを有効化するかどうか

**NovalPv6**

**タイプ:** ブール値

Nova で IPv6 機能を有効化します。

**NovalImage**

**タイプ:** 文字列

コンピュートノードのプロビジョニングに使用するイメージ

**NovaOVSBridge**

**タイプ:** 文字列

Open vSwitch によって使用される統合ブリッジ名

**NovaPassword**

**タイプ:** 文字列

OpenStack Compute サービスアカウントのパスワード

**NovaSecurityGroupAPI**

**タイプ:** 文字列

セキュリティ API クラスの完全なクラス名

**NtpServer**

**タイプ:** コンマ区切りリスト

NTP サーバーのコンマ区切りリスト

**ObjectStorageCount**

**タイプ:** 数値

オーバークラウド内の Object Storage ノードの数

**ObjectStorageExtraConfig**

**タイプ:** json

クラスターに挿入する Object Storage 固有の設定

**ObjectStorageHostnameFormat**

**タイプ:** 文字列

Object Storage ノードのホスト名の形式

**ObjectStorageRemovalPolicies**

**タイプ:** json

特定のリソースの削除が必要な更新の実行時に **ObjectStorageResourceGroup** から削除されるリソースの一覧

**ObjectStorageSchedulerHints**

**タイプ:** json

オプションのスケジューラーのヒント



**OvercloudBlockStorageFlavor**

**タイプ:** 文字列

デプロイ時に Block Storage ノードが要求するフレーバー

**OvercloudCephStorageFlavor**

**タイプ:** 文字列

デプロイ時に Ceph Storage ノードが要求するフレーバー

**OvercloudComputeFlavor**

**タイプ:** 文字列

デプロイ時に コンピュートノードが要求するフレーバー

**OvercloudControlFlavor**

**タイプ:** 文字列

デプロイ時に コントローラーノードが要求するフレーバー

**OvercloudSwiftStorageFlavor**

**タイプ:** 文字列

デプロイ時に Object Storage ノードが要求するフレーバー

**PublicVirtualFixedIPs**

**タイプ:** json

**PublicVirtualInterface**ポートの IP 割り当てを制御します。以下に例を示します。

```
[{'ip_address': '1.2.3.4'}]
```

**PublicVirtualInterface**

**タイプ:** 文字列

一般に公開する仮想 IP が割り当てられるインターフェースを指定します。VLAN を使用する場合には、これは **int\_public** となります。

**PurgeFirewallRules**

**タイプ:** ブール値

ファイアウォールルールを新規設定する前に消去するかどうかを定義します。

**RabbitClientPort**

**タイプ:** 数値

RabbitMQ サブスクライバーのポートを設定します。

**RabbitClientUseSSL**

**タイプ:** 文字列

RabbitMQ ホストへの SSL 接続を指定する RabbitMQ クライアントサブスクライバーのパラメーター

**RabbitCookieSalt**

**タイプ:** 文字列

RabbitMQ クッキーのソルト。無作為に生成された RabbitMQ クッキーを変えるには、この値を変更します。

**RabbitFDLimit**

**タイプ:** 文字列

RabbitMQ ファイル記述子の上限を設定します。

**RabbitIPv6**

**タイプ:** ブール値

RabbitMQ で IPv6 を有効化します。

**RabbitPassword**

**タイプ:** 文字列

RabbitMQ のパスワード

**RabbitUserName**

**タイプ:** 文字列

RabbitMQ のユーザー名

**RedisPassword**

**タイプ:** 文字列

Redis のパスワード

**SaharaPassword**

**タイプ:** 文字列

OpenStack Clustering サービスアカウントのパスワード

**ServerMetadata**

**タイプ:** json

オーバークラウドでノードを作成するために OpenStack Compute に渡される追加のプロパティまたはメタデータ

**ServiceNetMap**

**タイプ:** json

サービス名からネットワーク名へのマッピング。通常は、リソースレジストリーの **parameter\_defaults** で設定します。

**SnmpdReadonlyUserName**

**タイプ:** 文字列

すべてのオーバークラウド上で実行され、読み取り専用のアクセス権のある SNMPd のユーザー名

**SnmpdReadonlyUserPassword**

**タイプ:** 文字列

すべてのオーバークラウド上で実行され、読み取り専用のアクセス権のある SNMPd のパスワード

**StorageMgmtVirtualFixedIPs**

**タイプ:** json

StorageMgmtVirtualInterface ポートの IP の割り当てを制御します。以下に例を示します。

```
[{'ip_address': '1.2.3.4'}]
```

**StorageVirtualFixedIPs**

タイプ: json

**StorageVirtualInterface** ポートの IP 割り当てを制御します。以下に例を示します。

```
[{'ip_address': '1.2.3.4'}]
```

**SwiftHashSuffix**

タイプ: 文字列

リング内でマッピングを決定するためのハッシングを行う際にソルトとして使用するランダム文字列

**SwiftMinPartHours**

タイプ: 数値

リバランスの後にリング内のパーティションを移動できるようになるまでの最小時間 (時間単位)

**SwiftMountCheck**

タイプ: ブール値

Object Storage account/container/object-server.conf の **mount\_check** の値

**SwiftPartPower**

タイプ: 数値

Object Storage リングの構築時の Partition Power

**SwiftPassword**

タイプ: 文字列

Object Storage プロキシサービスが使用する Object Storage サービスアカウントのパスワード

**SwiftReplicas**

タイプ: 数値

Object Storage リングに使用するレプリカの数

**SwiftStorageImage**

タイプ: 文字列

Object Storage ノードのプロビジョニングに使用するイメージ

**TimeZone**

タイプ: 文字列

オーバークラウドデプロイメントのタイムゾーンを設定します。**TimeZone** パラメーターが空欄の場合には、オーバークラウドはデフォルトで **UTC** に設定します。director は、タイムゾーンデータベース **/usr/share/zoneinfo/** で定義されている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** に設定する場合には、**/usr/share/zoneinfo** の内容を確認して適切なエントリーを特定してください。

```
$ 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および追加のタイムゾーンを含むディレクトリーが含まれます。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が含まれるディレクトリーとなっています。

```

$ 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

```

## UpdateIdentifier

**タイプ:** 文字列

以前に **stack-update** の実行中に使用されなかった値を設定すると、全ノードでパッケージの更新がトリガーされます。

## 付録E 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。

### E.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 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

### E.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 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

### E.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 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

## E.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 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

## E.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 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。 <b>nmcli</b> 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「 <a href="#">4.5.1. ボンディングモジュールのディレクトリ</a> 」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

## E.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 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F 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。

### F.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、まった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番は、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番と同じです。

- **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。

## F.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
```

## F.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 Floating IP ネットワークを 1 つのみを使用すると、オプションで Neutron の外部ブリッジを設定できるようになります。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。

次のセクションには、ネイティブ VLAN に外部ネットワークを指定する NIC 設定への変更が含まれます。外部ネットワークが **br-ex** にマッピングされている場合には、外部ネットワークを Horizon Dashboard およびパブリック API 以外に Floating IP にも使用することができます。

## F.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、ブリッジから対応の VLAN インターフェースを削除します。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、変更が必要な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。

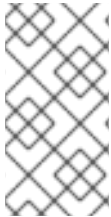
## F.5. ジャンボフレームの設定

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、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}
```

## 付録G ネットワーク環境のオプション

表G.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。デフォルト値は "br-ex" です。ブリッジ上のネイティブ VLAN で Floating IP アドレスを使用する場合には、" <b>br-ex</b> " に設定してください。通常この値の変更は推奨していません。	NeutronExternalNetworkBridge: "br-ex"
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"



パラメーター	説明	例
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"
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: <b>gre,vxlan</b>
NeutronTunnelIdRanges	テナントネットワークを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を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 <b>datacentre</b> 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: <b>openvswitch,l2population</b>

## 付録H 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"
```

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。

### 重要

LACP は OVS ベースのボンディングでは使用しないでください。この構成は問題があるため、サポートされていません。この機能の代わりとして、**bond\_mode=balance-slb** を使用することを検討してください。また、Linux ボンディングでは、お使いのネットワークインターフェーステンプレートで LACP を使用することができます。以下に例を示します。

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

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

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。

この要件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。

表H.1 ボンディングオプション

<b>bond_mode=balance-slb</b>	<p>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。<b>balance-slb</b> 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。</p>
<b>bond_mode=active-backup</b>	<p>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。</p>
<b>lacp=[active passive off]</b>	<p>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 <b>bond_mode=balance-slb</b> または <b>bond_mode=active-backup</b> を使用してください。</p> <p>OVS ベースのボンディングでは LACP を使用しないでください。この構成は問題があるため、サポートされていません。この機能の代わりとして、<b>bond_mode=balance-slb</b> を使用することを検討してください。また、LACP は Linux ボンディングで使用することも可能です。この要件に関する技術的な詳細は、<a href="#">BZ#1267291</a> を参照してください。</p>
<b>other-config: lacp-fallback-ab=true</b>	<p>フォールバックとして <b>bond_mode=active-backup</b> に切り替わるように LACP の動作を設定します。</p>
<b>other_config: lacp-time=[fast slow]</b>	<p>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。</p>
<b>other_config: bond-detect-mode=[miimon carrier]</b>	<p>リンク検出に <b>miimon</b> ハートビート (<b>miimon</b>) またはモニターキャリア (<b>carrier</b>) を設定します。デフォルトは <b>carrier</b> です。</p>

<b>other_config:bond-miimon-interval=100</b>	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。
<b>other_config:bond_updelay=1000</b>	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)
<b>other_config:bond-rebalance-interval=10000</b>	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ので、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。