



Red Hat Enterprise Linux OpenStack Platform 7

director のインストールと使用方法

Red Hat Enterprise Linux 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 Enterprise Linux OpenStack Platform 7 director のインストールと使用方法

Red Hat Enterprise Linux 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 Documentation Team
Red Hat Customer Content Services
rhos-docs@redhat.com

法律上の通知

Copyright © 2015 Red Hat.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

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 Enterprise Linux OpenStack Platform director を使用して Red Hat Enterprise Linux OpenStack Platform 7 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

第1章 はじめに	5
1.1. アンダークラウド	5
1.2. オーバークラウド	6
1.3. 高可用性	7
1.4. CEPH STORAGE	7
第2章 要件	8
2.1. 環境要件	8
2.2. アンダークラウドの要件	8
2.3. ネットワーク要件	9
2.4. オーバークラウドの要件	11
2.5. リポジトリの要件	13
第3章 アンダークラウドのインストール	15
3.1. DIRECTOR のインストールユーザーの作成	15
3.2. テンプレートとイメージ用のディレクトリーの作成	15
3.3. システムのホスト名設定	15
3.4. システムの登録	16
3.5. DIRECTOR パッケージのインストール	17
3.6. DIRECTOR の設定	17
3.7. オーバークラウドノードのイメージの取得	20
3.8. 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	21
3.9. アンダークラウドの設定完了	21
第4章 オーバークラウドのプランニング	22
4.1. ノードのデプロイメントロールのプランニング	22
4.2. ネットワークのプランニング	23
4.3. ストレージのプランニング	27
第5章 HEAT テンプレートについての理解	29
5.1. HEAT テンプレート	29
5.2. 環境ファイル	30
5.3. デフォルトの DIRECTOR プラン	30
5.4. デフォルトの DIRECTOR テンプレート	31
第6章 オーバークラウドのインストール	32
6.1. 基本シナリオ: NFS ストレージを使用する小規模なオーバークラウドの作成	32
6.2. 高度なシナリオ: CEPH STORAGE ノードを使用する大型のオーバークラウドの作成	45
第7章 オーバークラウド作成後のタスクの実行	78
7.1. オーバークラウドのテナントネットワークの作成	78
7.2. オーバークラウドの外部ネットワークの作成	78
7.3. 追加の FLOATING IP ネットワークの作成	79
7.4. オーバークラウドのプロバイダーネットワークの作成	80
7.5. オーバークラウドの検証	80
7.6. オーバークラウド環境の変更	82
7.7. オーバークラウドへの仮想マシンのインポート	83
7.8. 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	84
7.9. オーバークラウドの削除防止	85
7.10. オーバークラウドの削除	85
第8章 オーバークラウドのスケーリング	87
8.1. コンピュートノードまたは CEPH STORAGE ノードの追加	87

8.2. コンピュートノードの削除	89
8.3. コンピュートノードの置き換え	90
8.4. コントローラーノードの置き換え	91
8.5. CEPH STORAGE ノードの置き換え	103
第9章 オーバークラウドのリブート	106
9.1. DIRECTOR の再起動	106
9.2. コントローラーノードの再起動	106
9.3. CEPH STORAGE ノードの再起動	107
9.4. コンピュートノードの再起動	108
9.5. OBJECT STORAGE ノードの再起動	109
第10章 カスタム設定の作成	110
10.1. 初回起動での設定のカスタマイズ	110
10.2. オーバークラウドの設定前のカスタマイズ	111
10.3. オーバークラウドの設定後のカスタマイズ	113
10.4. PUPPET 設定データのカスタマイズ	114
10.5. カスタムの PUPPET 設定の適用	115
10.6. カスタムのオーバークラウド HEAT テンプレートの使用	116
第11章 環境の更新	118
11.1. DIRECTOR パッケージの更新	118
11.2. オーバークラウドと検出イメージの更新	118
11.3. オーバークラウドの更新	119
第12章 DIRECTOR の問題のトラブルシューティング	125
12.1. ノード登録のトラブルシューティング	125
12.2. 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	125
12.3. オーバークラウドの作成のトラブルシューティング	126
12.4. プロビジョニングネットワーク上での IP アドレスの競合の回避	129
12.5. "NO VALID HOST FOUND" エラーのトラブルシューティング	130
12.6. オーバークラウド作成後のトラブルシューティング	131
12.7. アンダークラウドの調整	133
12.8. アンダークラウドとオーバークラウドの重要なログ	135
付録A コンポーネント	137
付録B SSL/TLS 証明書の設定	139
認証局の作成	139
SSL/TLS 証明書の作成	139
アンダークラウドで証明書を使用する場合	140
オーバークラウドで証明書を使用する場合	141
付録C 電源管理ドライバー	142
C.1. DELL REMOTE ACCESS CONTROLLER (DRAC)	142
C.2. INTEGRATED LIGHTS-OUT (ILO)	142
C.3. IBOOT	143
C.4. CISCO UNIFIED COMPUTING SYSTEM (UCS)	143
C.5. FUJITSU INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC)	144
C.6. SSH および VIRSH	145
C.7. フェイク PXE ドライバー	145
付録D AUTOMATED HEALTH CHECK (AHC) ツールのパラメーター	147
D.1. ハードドライブ	147
D.2. システム	148

D.3. ファームウェア	148
D.4. ネットワーク	148
D.5. CPU	149
D.6. メモリー	150
D.7. INFINIBAND	151
付録E ネットワークインターフェースのパラメーター	153
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	156
F.1. インターフェースの設定	156
F.2. ルートおよびデフォルトルートの設定	157
F.3. FLOATING IP のためのネイティブ VLAN の使用	157
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	158
F.5. ジャンボフレームの設定	158
付録G ネットワーク環境のオプション	160
付録H ボンディングオプション	162
付録I デプロイメントパラメーター	164
付録J 改訂履歴	169

第1章 はじめに

Red Hat Enterprise Linux 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 Enterprise Linux OpenStack Platform 環境を簡単にインストールできます。

Red Hat Enterprise Linux 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 つの主要な概念を使用します。以下の数項では、それぞれの概念について説明します。

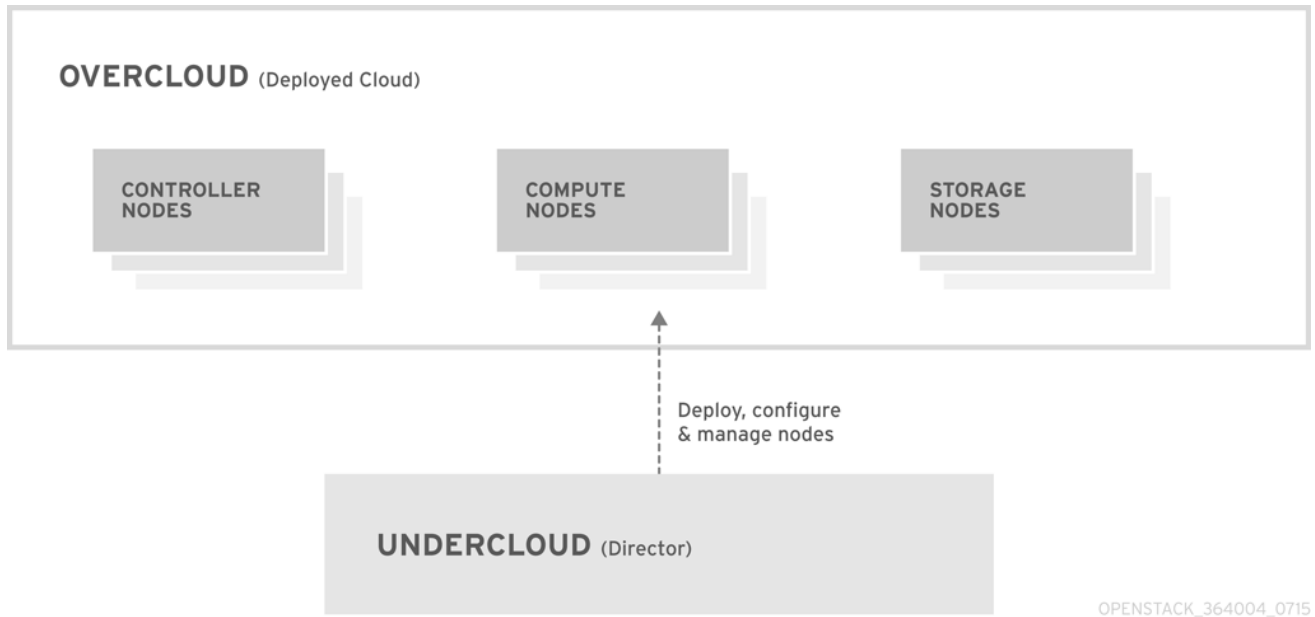


図1.1 アンダークラウドおよびオーバークラウドの基本レイアウト

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、コンピューター、コントローラー、各種ストレージロールなどの Red Hat Enterprise Linux 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、電源管理の制御には各ノードの Intelligent Platform Management Interface (IPMI) を使用し、ハードウェア属性の検出や OpenStack の各ノードへのインストールには PXE ベースのサービスを使用します。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 Enterprise Linux OpenStack Platform director は、Web ベースのグラフィカルユーザーインターフェースおよびターミナルベースのコマンドラインインターフェースの両方で、これらのアンダークラウド機能を活用します。

アンダークラウドは、以下のコンポーネントを使用します。

- OpenStack Dashboard (Horizon): director の Web ベースのダッシュボード
- 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 Server (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
- OpenStack Identity (Keystone): director のコンポーネントの認証
- MariaDB: director のデータベース
- RabbitMQ: director コンポーネントのメッセージキュー

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 Enterprise Linux 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、高可用性機能を提供するノード。理想的な OpenStack 環境には、高可用性のクラスターに 3 つのコントローラーノードが設定されていることが推奨されます。

デフォルトのコントローラーノードには、Horizon、Keystone、Nova API、Neutron Server、Open vSwitch、Glance、Cinder Volume、Cinder API、Swift Storage、Swift Proxy、Heat Engine、Heat API、Ceilometer、MariaDB、RabbitMQ のコンポーネントが含まれています。また、コントローラーは高可用性機能に Pacemaker や Galera も使用します。

- コンピュート: OpenStack 環境にコンピュートリソースを提供するために使用するノード。環境を徐々にスケーリングするにはコンピュートノードをさらに追加します。

デフォルトのコンピュートノードには、Nova、Compute、Nova KVM、Ceilometer Agent、Open vSwitch といったコンポーネントが含まれます。

- 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、Cinder Volume、Ceilometer Agent、Open vSwitch といったコンポーネントが含まれます。
 - 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、swift ストレージ、ceilometer エージェント、Open vSwitch コンポー

ネットが含まれます。

1.3. 高可用性

高可用性機能を利用すると、システムやコンポーネントセットに対して長時間にわたり継続的な操作を実行できるようになります。Red Hat Enterprise Linux OpenStack Platform director は、コントローラーノードクラスターを使用して、OpenStack Platform 環境に高可用性を提供します。director は、各コントローラーノードに同じコンポーネントセットをインストールして、まとめて1つのサービスとして管理します。クラスターを使用すると、単一のコントローラーノードで操作に支障をきたす問題が発生した場合にフォールバックできます。これにより、OpenStack のユーザーは、継続して一定レベルの稼働状況を確認することができます。

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、クラスター内の全マシンにおける 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 データベースの複製を提供します。
- Memcached: データベースのキャッシュを提供します。



注記

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、数千以上のクライアントにサービスを提供します。固有のニーズがある OpenStack クライアントはそれぞれにブロックストレージのリソースを消費します。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、数千以上のクライアントがある大規模なデプロイメントでの管理が不可能になります。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。これにより、Red Hat Enterprise Linux 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 Block Device 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 Enterprise Linux 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、インストーラー自体のセットアップ/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。

2.1. 環境要件

最低要件

- Red Hat Enterprise Linux OpenStack Platform director 用のホストマシン 1 台
- Red Hat Enterprise Linux OpenStack Platform コンピュートノード用のホストマシン 1 台
- Red Hat Enterprise Linux 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 Enterprise Linux OpenStack Platform director 用のホストマシン 1 台
- Red Hat Enterprise Linux OpenStack Platform コンピュートノード用のホストマシン 3 台
- Red Hat Enterprise Linux OpenStack Platform コントローラーノード用のホストマシン 1 台
- クラスタ内に 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
- 最小で 16 GB の RAM
- 最小 40 GB の空きディスク領域。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 がインストール済みであること

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: これは、director がオーバークラウドノードのプロビジョニング/管理に使用するプライベートネットワークです。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。director が PXE ブートおよび DHCP の要求に対応できるように、このネットワークは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。これは、Intelligent Platform Management Interface (IPMI) での全オーバークラウドノードの電源管理制御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 Enterprise Linux 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 Enterprise Linux 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、「[ネットワークのプランニング](#)」を参照してください。

以下の点に注意してください。

- すべてのマシンに少なくとも 2 つの NIC が必要です。標準的な最小設定の場合には、以下のいずれかを使用します。
 - 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。
 - 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、異なる種類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。
- 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 NIC を参照する場合は、全オーバークラウドマシンで 1 つの名前を使用します。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、各システムで同じ 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 アドレスの数を決定してください。
 - 最小で、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 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、[「プロビジョニングネットワーク上での IP アドレスの競合の回避」](#)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。
- director は各ノードの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、プロビジョニングネットワークに接続された Intelligent Platform Management Interface (IPMI) が必要です。
- 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、オーバークラウドノードの設定時に役立ちます。
- 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。



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しっかりと最小限に抑えられるようにします。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とディスク容量も必要です。

プロセッサ

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 RAM

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。

Intelligent Platform Management Interface (IPMI)

各コンピュートノードには、サーバーのマザーボード上に IPMI 機能が必要です。

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、RHEL 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 拡張機能をサポートする 64 ビット x86 プロセッサ

メモリー

最小 6 GB の RAM

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。

Intelligent Platform Management Interface (IPMI)

各コントローラーノードには、サーバーのマザーボード上に IPMI 機能が必要です。

2.4.3. Ceph Storage ノードの要件

Ceph Storage ノードは、RHEL 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 拡張機能をサポートする 64 ビット 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**: root 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 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。ストレージ要件で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 (実稼動環境では、最低でも NIC を 2 つ以上使用することを推奨します)。ボンディングされた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。

Intelligent Platform Management Interface (IPMI)

各 Ceph ノードには、サーバーのマザーボード上に IPMI 機能が必要です。

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する必要があります。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
```

2.5. リポジトリの要件

アンダークラウドおよびオーバークラウドにはいずれも、Red Hat コンテンツ配信ネットワーク (CDN) か Red Hat Satellite 5 または 6 を使用した 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、お使いの OpenStack Platform 環境に必要なリポジトリを同期します。以下の CDN チャンネル名一覧を参考にしてください。

表2.1 OpenStack Platform リポジトリ

名前	リポジトリ	要件の説明
Red Hat Enterprise Linux 7 Server (RPMs)	rhel-7-server-rpms	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)	rhel-7-server-extras-rpms	Red Hat OpenStack Platform の依存関係が含まれます。
Red Hat Enterprise Linux 7 Server - RH Common (RPMs)	rhel-7-server-rh-common-rpms	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	rhel-7-server-satellite-tools-6.1-rpms	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)	rhel-ha-for-rhel-7-server-rpms	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 7.0 director for RHEL 7 (RPMs)	rhel-7-server-openstack-7.0-director-rpms	Red Hat OpenStack Platform director のリポジトリ

名前	リポジトリ	要件の説明
Red Hat Enterprise Linux OpenStack Platform 7.0 for RHEL 7 (RPMs)	rhel-7-server-openstack-7.0-rpms	Red Hat OpenStack Platform のコアリポジトリ
Red Hat Ceph Storage OSD 1.3 for Red Hat Enterprise Linux 7 Server (RPMs)	rhel-7-server-rhceph-1.3-osd-rpms	(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)	rhel-7-server-rhceph-1.3-mon-rpms	(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。

第3章 アンダークラウドのインストール

Red Hat Enterprise Linux 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。

3.1. 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 の使用時にはパスワードなしでログインできるようにします。

```
[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。

3.2. 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。

3.3. 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` ホストのホスト名を設定する必要があります。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)
```

必要に応じて、**hostnamectl** を使用して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` には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
```

3.4. 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。

手順3.1 サブスクリプションマネージャーを使用して必要なチャンネルをサブスクライブする手順

1. 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
```

2. Red Hat Enterprise Linux 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
```

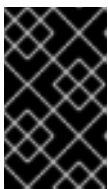
3. 上記のステップで特定したプール ID を使用して、Red Hat Enterprise Linux OpenStack Platform 7 のエンタイトルメントをアタッチします。

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

4. 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-
7.0-rpms --enable=rhel-7-server-openstack-7.0-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。

5. 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。

3.5. 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。

```
[stack@director ~]$ sudo yum install -y python-rdomanager-oscpplugin
```

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。

3.6. 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。使用環境の既存の 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。

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B 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** 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 **discovery_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。

masquerade_network

外部アクセス向けにネットワークをマスカレードに定義します。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

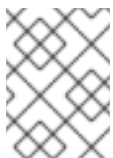
オーバークラウドノードの DHCP 割り当て範囲 (開始アドレスと終了アドレス)。お使いのノードを割り当てするのに十分な IP アドレスがこの範囲に含まれるようにします。

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。

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検出ホストです。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。

discovery_interface

ノードの検出に 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。

discovery_iprange

director の検出サービスが 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。

discovery_runbench

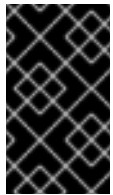
ノード検出時に一連のベンチマークを実行します。有効にするには、**1** に設定します。このオプションは、高度なシナリオで登録ノードのハードウェアを検査する際にベンチマーク分析を実行する場合に必要です。詳細は、「[Automated Health Check \(AHC\) ツールを使用したノードの自動タグ付け](#)」を参照してください。

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。

undercloud_db_password, undercloud_admin_token, undercloud_admin_password, undercloud_glance_password, 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。

お使いのネットワークに適したものに、これらの値を変更してください。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。

3.7. 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。

- 検出カーネルおよび ramdisk: PXE ブートでのベアメタルシステムの検出に使用
- 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こまれるベースのオーバークラウドシステム

Red Hat カスタマーポータルで https://access.redhat.com/downloads/content/191/ver=7/rhel--7/7/x86_64/product-downloads の Red Hat Enterprise Linux OpenStack Platform のダウンロードページからこれらのイメージを取得します。カスタマーポータル上のこの場所では、イメージが TAR アーカイブ形式で提供されています。

これらのイメージアーカイブは、ディレクトリーホスト上の **stack** ユーザーのホーム (`/home/stack/images/`) の **images** ディレクトリーにダウンロードしてください。このアーカイブからイメージを抽出します。

```
$ cd ~/images
$ for tarfile in *.tar; do tar -xf $tarfile; done
```

以下のコマンドを実行して、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/
```

このコマンドでは、**bm-deploy-kernel**、**bm-deploy-ramdisk**、**overcloud-full**、**overcloud-full-initrd**、**overcloud-full-vmlinuz** のイメージを director にアップロードします。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。また、このスクリプトにより、director の PXE サーバーに検出イメージがインストールされます。

以下のコマンドを使用して、CLI でこれらの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            |
+-----+-----+
```

この一覧では、ディスカバリー PXE イメージ (**discovery-ramdisk.***) 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 151636
-rw-r--r--. 1 ironic ironic      269 Sep 19 02:43 boot.ipxe
-rw-r--r--. 1 root  root        252 Sep 10 15:35 discovered.ipxe
-rwxr-xr-x. 1 root  root     5027584 Sep 10 16:32 discovery.kernel
```



```
-rw-r--r--. 1 root    root    150230861 Sep 10 16:32 discovery.ramdisk
drwxr-xr-x. 2 ironic  ironic    4096 Sep 19 02:45 pxelinux.cfg
```



注記

Ironic は、イントロスペクションおよびプロビジョニング中に **/httpboot** 内の **boot.ipxe** ファイルと **pxelinux.cfg** ディレクトリーを管理します。これらのファイルは、このディレクトリーを確認しても見えない可能性があります。

3.8. 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。ネットワークの分離の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** サブネットで定義されます。以下のコマンドを使用して、この環境のネームサーバーを定義します。

```
$ 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、[「高度なオーバークラウドのネットワーク環境ファイルの作成」](#)の高度な設定シナリオで説明します。

3.9. 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。次の数章では、オーバークラウドのプランニングと作成について説明していきます。

第4章 オーバークラウドのプランニング

以下の項では、ノードロールの定義、ネットワークポリシーのプランニング、ストレージなど、オーバークラウド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提供します。

4.1. 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 (複数のコントローラーノードを使用する場合) が含まれます。Red Hat OpenStack Platform 環境に以下のいずれかが必要です。

- 基本的な環境にはノードを 1 台
- 高可用性環境にはノードを 3 台

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。

コンピュート

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ホスト。基本的な Red Hat Enterprise Linux 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 Service 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 Service 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。

本ガイドでは、希望の環境をベースに複数のシナリオを提供しています。以下の表では、各シナリオのノード種別を定義します。

表4.1 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	コンピュート	Ceph-Storage	Swift-Storage	Cinder-Storage	合計
基本環境	1	1	-	-	-	2
Ceph Storage の高度な環境	3	3	3	-	-	9

4.2. 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ト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 Enterprise Linux OpenStack Platform 7 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 Enterprise Linux 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4.2 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 IP サブネットが割り当てられおり、複数のテナントネットワークが同じアドレスを使用する場合があります。	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

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
ストレージ管理	<p>OpenStack Object Storage (swift) は、このネットワークを使用し、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。</p>	コントローラー、Ceph Storage、Cinder Storage、Swift Storage
外部	<p>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。</p>	コントローラー

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
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、インスタン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

一般的な Red Hat Enterprise Linux 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、オーバークラウドにおいて 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。



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。

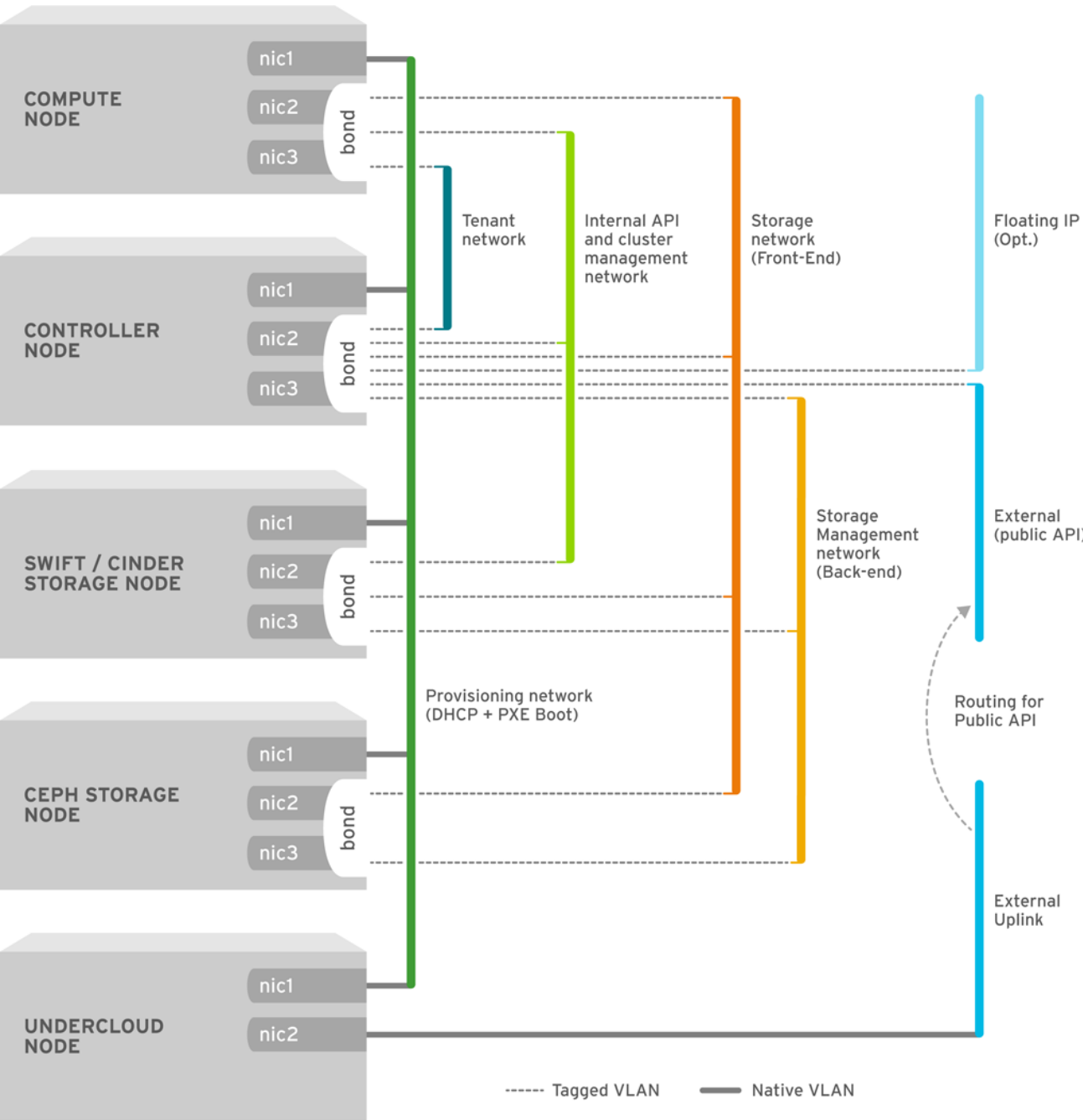
director は、トラフィック種別の中から 5 つを特定のサブネットにや VLAN にマッピングする方法を提供します。このようなトラフィック種別には、以下が含まれます。

- 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

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

図4.1 ボンディングインターフェースを使用する VLAN トポロジーの例

本ガイドでは、希望の環境をベースに複数のシナリオを提供しています。以下の表では、各シナリオのネットワークトラフィックマッピングを定義します。

表4.3 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
基本環境	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 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)	2	2
Ceph Storage の高度な環境	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: 外部、Floating IP (オーバークラウドの作成後にマッピング)	3 (ボンディングインターフェース 2 つを含む)	6

4.3. 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:** OpenStack Glance は仮想マシンのイメージを管理します。イメージは変更しないため、OpenStack はイメージバイナリーブオブとして処理し、それに応じてイメージをダウンロードします。Ceph Block Device でイメージを格納するには、OpenStack Glance を使用することができます。
- **ボリューム:** OpenStack 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 では、仮想マシンディスクのホスティングに対する QCOW2 のサポートはありません。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。

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。

Cinder Storage ノード

director は、外部ブロック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ブロックストレージを保つ必要がある場合に便利です。

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5章 HEAT テンプレートについての理解

本ガイドのシナリオの一部では、カスタムの Heat テンプレートを使用して、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 Enterprise Linux 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。

5.1. 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、リソースごとの設定が含まれる **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 つのセクションから構成されています。

- **Parameters:** これらは Heat に渡す設定で、スタックや値を指定しない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。これらの設定は、テンプレートの **parameters** セクションで定義します。
- **Resources:** これらはスタックの一部として作成/設定する固有のオブジェクトです。OpenStack には全コンポーネントに対応するコアのリソースセットが含まれています。これらの設定は、テンプレートの **resources** セクションで定義されます。
- **Output:** これらは、スタックの作成後に heat から渡される値です。これらの値には、Heat API またはクライアントツールを使用してアクセスすることができます。これらは、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、**type: OS::Nova::Server** のリソース種別を使用して、特定のフレーバー、イメージ、キーを指定した **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** という **instance_name** の値を返します。

5.2. 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。

- **Parameters:** これらは、テンプレートのパラメーターに適用する共通設定で、環境ファイルの **parameters** セクションで定義します。
- **Parameter Defaults:** これらのパラメーターは、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。これらの設定は、環境ファイルの **parameter_defaults** セクションで定義します。
- **Resource Registry:** このセクションでは、カスタムのリソース名、他の Heat テンプレートへのリンクを定義します。これは実質的に、コアリソースコレクションに存在しないカスタムのリソースを作成する方法を提供します。この設定は、環境ファイルの **resource_registry** セクションで定義されます。

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

```

このファイルにより、**OS::Nova::Server::MyServer** と呼ばれる新しいリソース種別が作成されます。**myserver.yaml** ファイルは、このリソース種別を実装する Heat テンプレートファイルで、このファイルでの設定が元の設定よりも優先されます。

5.3. デフォルトの DIRECTOR プラン

director は、そのデータベース内に Heat テンプレートコレクションを格納します。これは、**プラン** として保存されます。director 内のプランの一覧を表示するには、以下のコマンドを実行します。

```
$ openstack management plan list
```

このコマンドから、オーバークラウド設定が含まれる **overcloud** というプランが表示されます。このオーバークラウドプランの詳細を表示するには、以下を実行します。

```
$ openstack management plan show [UUID]
```

また、オーバークラウドプランの Heat テンプレートファイルをダウンロードして表示することも可能です。以下のコマンドを使用して、このプランの中の Heat テンプレートを **stack** ユーザーの **templates** ディレクトリーの中のディレクトリーにダウンロードします。

```
$ mkdir ~/templates/overcloud-plan  
$ openstack management plan download [UUID] -o ~/templates/overcloud-plan/
```

このコレクションには、主要な Heat テンプレート (**plan.yaml**) と環境ファイル (**environment.yaml**) が含まれており、またテンプレートコレクションには、環境ファイルのリソースとして登録されているさまざまなディレクトリーおよびテンプレートファイルが含まれます。

このプランベースのテンプレートは、テストシナリオでオーバークラウドの作成に使用します。

5.4. デフォルトの DIRECTOR テンプレート

director には、オーバークラウドの高度な Heat テンプレートコレクションも含まれます。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は以下の 3 つです。

- **overcloud-without-mergepy.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。
- **overcloud-resource-registry.yaml**: これは、オーバークラウド環境の作成に使用する標準の環境ファイルです。**overcloud-resource-registry-puppet.yaml** は、このファイルをベースにしており、お使いの環境をカスタム設定する際に使用します。

基本的/高度なオーバークラウドシナリオは、このテンプレートコレクションを使用します。いずれも **overcloud-without-mergepy.yaml** テンプレートと **overcloud-resource-registry-puppet.yaml** 環境ファイルを使用して、ノードごとにオーバークラウドのイメージを設定します。また、基本的/高度なシナリオのいずれの場合も、ネットワークの分離設定を行うには、環境ファイルも作成します。

第6章 オーバークラウドのインストール

Red Hat Enterprise Linux OpenStack Platform director を設定して、アンダークラウドのインストールが完了しました。本章では、director を使用してオーバークラウドの環境を作成します。ユーザーのレベルに対応するように、オーバークラウド作成手順は、2 種のインストールシナリオに分けています。シナリオごとに、複雑度と記載内容が異なります。

表6.1 シナリオの概要

シナリオ	レベル	トピック
基本オーバークラウド	中	CLI ツールの使用、ノードの登録、手動でのノードのタグ付け、基本的なネットワークの分離、プランベースのオーバークラウドの作成
高度なオーバークラウド	高	CLI ツールの使用、ノードの登録、ハードウェアをベースにしたノードの自動タグ付け、Ceph Storage の設定、高度なネットの分離、オーバークラウドの作成、高可用性のフェンシング設定

6.1. 基本シナリオ: NFS ストレージを使用する小規模なオーバークラウドの作成

このシナリオでは、小規模なエンタープライズレベルの OpenStack Platform 環境を構築します。この環境は、オーバークラウド内の 2 つのノード (コントローラーノード 1 つとコンピュートノード 1 つ) で構成されます。いずれのマシンも、電源管理に IPMI を使用するベアメタルシステムです。このシナリオは、コマンドラインツールに焦点をあて、コンピュートノードを後でスケーリングすることが可能な小規模な実稼動レベルの Red Hat Enterprise Linux OpenStack Platform 環境を構築する director の機能について実例をあげて説明することを目的としています。

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証します。
3. 手動でロールにノードをタグ付けします。
4. フレーバーを作成してロールにタグ付けします。
5. Heat テンプレートを作成して外部ネットワークを分離します。
6. デフォルトの Heat テンプレートコレクションと追加のネットワーク分離テンプレートを使用してオーバークラウド環境を作成します。

要件

- 「[3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

- ベアメタルマシン 2 台。これらのマシンは、コントローラーノードおよびコンピュートノードに設定された要件に準拠する必要があります。要件については以下を参照してください。
 - 「[コントローラーノードの要件](#)」
 - 「[コンピュートノードの要件](#)」

director により Red Hat Enterprise Linux 7 のイメージが各ノードにコピーされるため、これらのノードではオペレーティングシステムは必要ありません。

- 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6.2 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	
コントローラー	定義済みの DHCP	bb:bb:bb:bb:bb:bb	192.0.2.205
コンピュート	定義済みの DHCP	cc:cc:cc:cc:cc:cc	192.0.2.206

- 外部ネットワークのネットワーク接続 1 つ。コントローラーノードはすべて、このネットワークに接続する必要があります。今回の例では、外部ネットワークには 10.1.1.0/24 を使用します。
- 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。
- このシナリオでは、プロビジョニングネットワーク上の別のサーバーで NFS 共有も使用します。このサーバーの IP アドレスは 192.0.2.230 です。

6.1.1. 基本オーバークラウドへのノードの登録

本項では、ノード定義のテンプレートを作成します。このファイル (`instackenv.json`) は JSON ファイル形式で、2 つのノードのハードウェアおよび電源管理の詳細が含まれます。

このテンプレートでは、以下の属性を使用します。

mac

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。

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

cpu

ノード上の CPU 数

memory

メモリーサイズ (MB)

disk

ハードディスクのサイズ (GB)

arch

システムアーキテクチャー

例:

```
{
  "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"
    }
  ]
}
```

**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C 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から、director にインポートします。これには、以下のコマンドを実行します。

```
$ 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。以下のコマンドを使用して、CLI でこれらのノードの一覧を表示します。

```
$ openstack baremetal list
```

6.1.2. ノードのハードウェアの検証

ノードの登録後には、各ノードのハードウェア属性を検証します。各ノードのハードウェア属性を検証するには、以下のコマンドを実行します。

```
$ 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-discoverd -u openstack-ironic-discoverd-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。ノードをメンテナンス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、メンテナンスモードから元に戻します

```
$ ironic node-set-maintenance [NODE UUID] true
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-maintenance [NODE UUID] false
```

6.1.3. ノードの手動でのタグ付け

各ノードのハードウェアを登録して検証した後には、個別のプロファイルにタグ付けします。これらのプロファイルタグは、ノードからフレーバーに対して照合され、そのフレーバーはデプロイメントロー

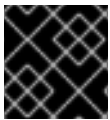
ルに割り当てられます。基本デプロイメントのシナリオでは、ノードが2つしかないため、手動でタグ付けします。ノード数が多い場合は、高度なデプロイメントのシナリオにある Automated Health Check (AHC) ツールを使用します。Automated Health Check (AHC) ツールに関する詳細は、「[Automated Health Check \(AHC\) ツールを使用したノードの自動タグ付け](#)」を参照してください。

特定のプロファイルにノードを手動で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ト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。

6.1.4. 基本シナリオのフレーバーの作成

director には、登録ノードのハードウェアプロファイルまたはフレーバーのセットが必要です。このシナリオでは、コンピュートノードとコントローラーノードごとにプロファイルを作成します。

```
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 control
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 compute
```

これにより、ノードに2つのフレーバー (**control** と **compute**) が作成されました。また、各フレーバーに追加のプロパティを設定します。

```
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="compute" compute
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="control" control
```

capabilities:boot_option はフレーバーのブートモードを設定し、**capabilities:profile** は使用するプロファイルを定義します。これにより、「[ノードの手動でのタグ付け](#)」でタグ付けされた対象の各ノード上にある同じタグにリンクされます。



重要

使用していないロールにも **baremetal** というデフォルトのフレーバーが必要です。このフレーバーがない場合には作成します。

```
$ openstack flavor create --id auto --ram 4096 --disk 40 --
vcpus 1 baremetal
```

6.1.5. 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、Heat テンプレートコレクション内の既存の環境ファイルの変更がベースとなります。

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るはずです。

```
parameters:
  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することになります。

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。

6.1.6. 外部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。

このシナリオでは、2つの分離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。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もこのネットワークを使用します。
- ネットワーク 2: 外部ネットワーク。このネットワークは、オーバークラウドの外部に接続する専用インターフェースを使用します。

以下の項では、Heat テンプレートを作成して残りのサービスから外部ネットワークを分離する方法を説明します。その他のネットワーク設定例は、「[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。

6.1.6.1. 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 (「[5章 Heat テンプレートについて](#)の理解」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。

基本オーバークラウドのシナリオでは、デフォルトの単一 NIC の設定サンプルを使用します。デフォルトの設定ディレクトリーを **nic-configs** として、**stack** ユーザーのホームディレクトリーにコピーします。

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

このコマンドによりローカルの Heat テンプレートが作成され、このテンプレートで外部ネットワークが使用する単一ネットワークインターフェース設定を定義します。各テンプレートには、標準の **parameters**、**resources**、**output** 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 のボンディングを定義します。ボンディングでは、2 つ以上の **interfaces** を結合して、冗長性や帯域幅を向上させます。

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

ovs_bridge

Open vSwitch のブリッジを定義します。ブリッジは、複数の **interface**、**bond**、**vlan** オブジェクトを接続します。

```
- 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}

```

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。

基本シナリオでは、外部ネットワークを **nic2** に移動するように、各インターフェーステンプレートを変更します。これにより、各ノードの2番目のネットワークインターフェースが外部ネットワークに使用されるようになります。たとえば、**templates/nic-configs/controller.yaml** のテンプレートは以下のようになります。

```

network_config:
- type: ovs_bridge
  name: {get_input: bridge_name}
  use_dhcp: true
  members:
    - type: interface
      name: nic1
      # force the MAC address of the bridge to this interface
      primary: true
    - type: vlan
      vlan_id: {get_param: InternalApiNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: InternalApiIpSubnet}
    - type: vlan
      vlan_id: {get_param: StorageNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: StorageIpSubnet}
    - type: vlan
      vlan_id: {get_param: StorageMgmtNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: StorageMgmtIpSubnet}
    - type: vlan
      vlan_id: {get_param: TenantNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: TenantIpSubnet}
- type: interface
  name: nic2
  addresses:
    - ip_netmask: {get_param: ExternalIpSubnet}
  routes:
    - ip_netmask: 0.0.0.0/0
      next_hop: {get_param: ExternalInterfaceDefaultRoute}

```

上記の例においては、新規インターフェース (**nic2**) を作成して外部ネットワークアドレスを再割り当てし、新規インターフェースにルーティングします。

ネットワークインターフェーステンプレートの他のサンプルについては「[付録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、使用済みのインターフェースを **ovs_bridge** デバイスから削除し、DHCP とデフォルトのルート設定を無効にします。

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

6.1.6.2. 基本オーバークラウドのネットワーク環境のテンプレート

ネットワーク環境ファイルでは、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ファイルを参照します。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。

このシナリオでは、**/home/stack/templates/network-environment.yaml** で保存した下記のネットワーク環境ファイルを使用します。

```
resource_registry:
  OS::Triple0::BlockStorage::Net::SoftwareConfig:
    /home/stack/templates/nic-configs/cinder-storage.yaml
  OS::Triple0::Compute::Net::SoftwareConfig: /home/stack/templates/nic-configs/compute.yaml
  OS::Triple0::Controller::Net::SoftwareConfig: /home/stack/templates/nic-configs/controller.yaml
  OS::Triple0::ObjectStorage::Net::SoftwareConfig:
    /home/stack/templates/nic-configs/swift-storage.yaml
  OS::Triple0::CephStorage::Net::SoftwareConfig:
    /home/stack/templates/nic-configs/ceph-storage.yaml

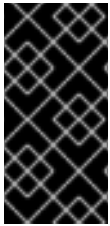
parameter_defaults:
  ExternalNetCidr: 10.1.1.0/24
  ExternalAllocationPools: [{'start': '10.1.1.2', 'end': '10.1.1.50'}]
  ExternalNetworkVlanID: 100
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 10.1.1.1
  # Gateway router for the provisioning network (or Undercloud 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"]
  # Set to "br-ex" if using floating IPs on native VLAN on bridge br-ex
  NeutronExternalNetworkBridge: ""
```

resource_registry セクションには、各ノードロールのネットワークインターフェーステンプレートへのリンクが含まれます。**ExternalAllocationPools** パラメーターは狭い範囲の IP アドレスしか定義しません。これは、別の範囲の IP アドレスを後ほど定義できるようにするためです。

parameter_defaults セクションには、各ネットワーク種別のネットワークオプションを定義するパラメーター一覧が含まれます。これらのオプションについての詳しい参考情報は「[付録G ネットワーク環境のオプション](#)」を参照してください。

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。

このセクションは、外部ネットワークのオプションを定義するだけです。その他のトラフィック種別は、プロビジョニングネットワークに自動的に割り当てられます。

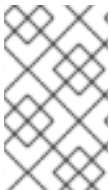


重要

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。

6.1.7. 基本オーバークラウドの作成

OpenStack 環境の最後の段階として、環境作成に必要とされるコマンドを実行します。デフォルトプランでは、コントローラーノードとコンピュートノードが 1 つずつインストールされます。



注記

Red Hat カスタマーポータルには、オーバークラウド作成前の設定検証に役立つラボがあります。このラボは、<https://access.redhat.com/labs/ospec/> で利用できます。ラボについての説明は、<https://access.redhat.com/labsinfo/ospec> に記載されています。

以下のコマンドを実行して、基本的なオーバークラウドの作成を開始します。

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml -e /home/stack/templates/network-environment.yaml -e /home/stack/templates/storage-environment.yaml --control-flavor control --compute-flavor compute --ntp-server pool.ntp.org --neutron-network-type vxlan --neutron-tunnel-types vxlan
```

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** にある 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プランに別の環境ファイルを追加します。この場合、これ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 /home/stack/templates/network-environment.yaml: -e** オプションはオーバークラウドプランに別の環境ファイルを追加します。この場合は「[基本オーバークラウドのネットワーク環境のテンプレート](#)」で作成したネットワーク環境ファイルです。

- **-e /home/stack/templates/storage-environment.yaml**: **-e** のオプションにより、オーバークラウドプランに環境ファイルが追加されます。上記の例では、「[NFS ストレージの設定](#)」で作成したストレージの環境ファイルを指定しています。
- **--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**: 対象のコンピューター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。
- **--neutron-network-type vxlan**: オーバークラウドの Neutron ネットワークに Virtual Extensible LAN (VXLAN) を使用します。
- **--neutron-tunnel-types vxlan**: オーバークラウドの Neutron トンネリングに Virtual Extensible LAN (VXLAN) を使用します。



注記

オプションの完全な一覧を表示するには、以下を実行します。

```
$ openstack help overcloud deploy
```

パラメーターの例は「[付録I デプロイメントパラメーター](#)」も参照してください。

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。



警告

-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。director は、「[7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ことになる場合があります。

オーバークラウドの設定を後で変更する予定がある場合は、カスタム環境ファイルと Heat テンプレートのパラメーターを変更し、**openstack overcloud deploy** の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。



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。

6.1.8. 基本オーバークラウドへのアクセス

director は、アンダークラウドからオーバークラウドに対話する際の設定/認証を行うファイルを作成し、このファイル (**overcloudrc**) を **stack** ユーザーのホームディレクトリーに保存します。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
```

これにより、director 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 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
```

6.1.9. 基本オーバークラウドの完了

これで基本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、「[7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。

6.2. 高度なシナリオ: CEPH STORAGE ノードを使用する大型のオーバークラウドの作成

このシナリオでは、Red Hat Ceph Storage ノードを利用する大規模なエンタープライズレベルの OpenStack Platform 環境を作成します。本シナリオでは、オーバークラウド内で 9 つのノードを使用します。

- 高可用性のコントローラーノード x 3
- コンピュートノード x 3
- クラスター内の Red Hat Ceph Storage ノード x 3

マシンはすべて、電源管理に IPMI を使用するベアメタルシステムです。このシナリオでは、今後コンピュートノードのスケーリングが可能な、実稼動レベルの Red Hat Enterprise Linux OpenStack Platform 環境を作成する director の機能を紹介することが目的です。また、本シナリオではコマンドラインツールを使用して、Automated Health Check ツールを用いたロール照合や高度なネットワーク分離など、director の高度な機能の一部を紹介します。

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およびベンチマークを検証します。
3. Automated Health Check (AHC) ツールを使用して自動的にノードをロールにタグ付けするポリシーを定義します。
4. フレーバーを作成してロールにタグ付けします。
5. 環境ファイルを使用して、Ceph Storage を設定します。
6. Heat テンプレートを作成して全ネットワークを分離します。
7. デフォルトの Heat テンプレートコレクションと追加のネットワーク分離テンプレートを使用してオーバークラウド環境を作成します。
8. 高可用性クラスター内の各コントローラーノードに関するフェンシング情報を追加します。

要件

- 「[3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
- ベアメタルマシン 9 台。これらのマシンは、コントローラーノード、コンピュートノード、Ceph Storage ノードに設定された要件に準拠する必要があります。要件については以下を参照してください。
 - 「[コントローラーノードの要件](#)」
 - 「[コンピュートノードの要件](#)」
 - 「[Ceph Storage ノードの要件](#)」

director により Red Hat Enterprise Linux 7 のイメージが各ノードにコピーされるため、これらのノードではオペレーティングシステムは必要ありません。

- 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6.3 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	
コントローラー 1	定義済みの DHCP	b1:b1:b1:b1:b1:b1	192.0.2.205
コントローラー 2	定義済みの DHCP	b2:b2:b2:b2:b2:b2	192.0.2.206
コントローラー 3	定義済みの DHCP	b3:b3:b3:b3:b3:b3	192.0.2.207
コンピュート 1	定義済みの DHCP	c1:c1:c1:c1:c1:c1	192.0.2.208
コンピュート 2	定義済みの DHCP	c2:c2:c2:c2:c2:c2	192.0.2.209
コンピュート 3	定義済みの DHCP	c3:c3:c3:c3:c3:c3	192.0.2.210
Ceph 1	定義済みの DHCP	d1:d1:d1:d1:d1:d1	192.0.2.211
Ceph 2	定義済みの DHCP	d2:d2:d2:d2:d2:d2	192.0.2.212
Ceph 3	定義済みの DHCP	d3:d3:d3:d3:d3:d3	192.0.2.213

- 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6.4 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
外部 / Floating IP	10.1.1.0/24	100

6.2.1. 高度なオーバークラウドのノード登録

本項では、ノード定義のテンプレートを作成します。このファイル (**instackenv.json**) は JSON ファイル形式で、9 つのノードのハードウェアおよび電源管理の詳細が含まれます。

このテンプレートでは、以下の属性を使用します。

mac

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。

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

cpu

ノード上の CPU 数

memory

メモリーサイズ (MB)

disk

ハードディスクのサイズ (GB)

arch

システムアーキテクチャー

例:

```
{
  "nodes": [
    {
      "mac": [
        "b1:b1:b1:b1:b1:b1"
      ],
      "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": [
        "b2:b2:b2:b2:b2:b2"
      ],
      "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"
    },
    {
        "mac": [
            "b3:b3:b3:b3:b3:b3"
        ],
        "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": [
            "c1:c1:c1:c1:c1:c1"
        ],
        "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"
    },
    {
        "mac": [
            "c2:c2:c2:c2:c2:c2"
        ],
        "cpu": "4",
        "memory": "6144",
        "disk": "40",
        "arch": "x86_64",
        "pm_type": "pxe_ipmitool",
        "pm_user": "admin",
        "pm_password": "p@55w0rd!",
        "pm_addr": "192.0.2.209"
    },
    {
        "mac": [
            "c3:c3:c3:c3:c3:c3"
        ],
        "cpu": "4",
        "memory": "6144",
        "disk": "40",
        "arch": "x86_64",
        "pm_type": "pxe_ipmitool",
        "pm_user": "admin",
        "pm_password": "p@55w0rd!",
        "pm_addr": "192.0.2.210"
    },
    {

```

```

        "mac": [
            "d1:d1:d1:d1:d1:d1"
        ],
        "cpu": "4",
        "memory": "6144",
        "disk": "40",
        "arch": "x86_64",
        "pm_type": "pxe_ipmitool",
        "pm_user": "admin",
        "pm_password": "p@55w0rd!",
        "pm_addr": "192.0.2.211"
    },
    {
        "mac": [
            "d2:d2:d2:d2:d2:d2"
        ],
        "cpu": "4",
        "memory": "6144",
        "disk": "40",
        "arch": "x86_64",
        "pm_type": "pxe_ipmitool",
        "pm_user": "admin",
        "pm_password": "p@55w0rd!",
        "pm_addr": "192.0.2.212"
    },
    {
        "mac": [
            "d3:d3:d3:d3:d3:d3"
        ],
        "cpu": "4",
        "memory": "6144",
        "disk": "40",
        "arch": "x86_64",
        "pm_type": "pxe_ipmitool",
        "pm_user": "admin",
        "pm_password": "p@55w0rd!",
        "pm_addr": "192.0.2.213"
    }
]
}

```



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C 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 (**instackenv.json**) にファイルを保存してから、director にインポートします。これには、以下のコマンドを実行します。

```
$ 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。以下のコマンドを使用して、CLI でこれらのノードの一覧を表示します。

```
$ openstack baremetal list
```

6.2.2. ノードのハードウェアの検証

ノードの登録後には、各ノードのハードウェア属性を検証します。このシナリオでは、Automated Health Check (AHC) ツールで使用するノードを基準に従って評価し、その情報をデプロイメントプロファイルにノードを自動的にタグ付けする際に使用します。これらのプロファイルタグにより、ノードがフレーバーに対して照合され、そのフレーバーはデプロイメントロールに割り当てられます。

重要

ベンチマーク機能を利用するには、最初に director の設定の際に、**discovery_runbench** オプションを **true** に設定する必要があります (「[director の設定](#)」を参照)。

director のインストール後にベンチマーク機能を有効化する場合がある場合には、`/httpboot/discoverd.ipxe` を編集して **RUNBENCH** カーネルパラメーターを **1** に設定します。

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-discoverd -u openstack-ironic-discoverd-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。ノードをメンテナンス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、メンテナンスモードから元に戻します

```
$ ironic node-set-maintenance [NODE UUID] true
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-maintenance [NODE UUID] false
```

6.2.3. Automated Health Check (AHC) ツールを使用したノードの自動タグ付け

検出プロセスでベンチマークテストが完了したら、一連のレポートを生成して、低パフォーマンス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分離します。本項では、これらのレポートの生成方法を検証してポリシーを作成し、特定のロールにノードを自動的にタグ付けしま

す。

下記のコマンドを使用して、以下の Automated Health Check (AHC) ツールをインストールします。

```
$ sudo yum install -y ahc-tools
```

このパッケージには 2 つのツールが含まれます。

- **ahc-report**: ベンチマークテストからのレポートを提供します。
- **ahc-match**: ポリシーに応じて、ノードを特定のロールにタグ付けします。

重要

これらのツールでは、`/etc/ahc-tools/ahc-tools.conf` ファイルで設定した Ironic と Swift の認証情報が必要になります。認証情報は、`/etc/ironic-discoverd/discoverd.conf` と同じです。以下のコマンドを使用して、設定ファイルをコピーして `/etc/ahc-tools/ahc-tools.conf` にカスタマイズします。

```
$ sudo -i
# mkdir /etc/ahc-tools
# sed 's/\[discoverd/\[ironic/' /etc/ironic-discoverd/discoverd.conf > /etc/ahc-tools/ahc-tools.conf
# chmod 0600 /etc/ahc-tools/ahc-tools.conf
# exit
```

6.2.3.1. ahc-report

ahc-report のスクリプトは、ノードに関するさまざまなレポートを作成します。完全なレポートを表示するには **--full** オプションを使用します。

```
$ sudo ahc-report --full
```

ahc-report コマンドは、レポートの特定の場所にフォーカスすることも可能です。たとえば、**--categories** を使用して、ハードウェア別にノードを分類することができます (プロセッサ、ネットワークインターフェース、ファームウェア、メモリー、さまざまなハードウェアコントローラー)。またこのコマンドは、同様のハードウェアプロファイルのノードをグループ化します。たとえば、2 つのサンプルノードの **Processors** セクションは以下のような一覧になります。

```
#####
##### Processors #####
2 identical systems :
[u'7F8831F1-0D81-464E-A767-7577DF49AAA5', u'7884BC95-6EF8-4447-BDE5-D19561718B29']
[(u'cpu', u'logical', u'number', u'4'),
 (u'cpu', u'physical', u'number', u'4'),
 (u'cpu',
  u'physical_0',
  u'flags',
  u'fpu fpu_exception wp de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pse36 clflush mmx fxsr sse sse2 syscall nx x86-64 rep_good nopl pni
cx16 hypervisor lahf_lm'),
 (u'cpu', u'physical_0', u'frequency', u'2000000000'),
```



```
(u'cpu', u'physical_0', u'physid', u'0'),
(u'cpu', u'physical_0', u'product', u'Intel(R) Xeon(TM) CPU          E3-
1271v3 @ 3.6GHz'),
(u'cpu', u'physical_0', u'vendor', u'GenuineIntel'),
(u'cpu',
  u'physical_1',
  u'flags',
  u'fpu fpu_exception wp de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pse36 clflush mmx fxsr sse sse2 syscall nx x86-64 rep_good nopl pni
cx16 hypervisor lahf_lm'),
(u'cpu', u'physical_0', u'frequency', u'2000000000'),
(u'cpu', u'physical_0', u'physid', u'0'),
(u'cpu', u'physical_0', u'product', u'Intel(R) Xeon(TM) CPU          E3-
1271v3 @ 3.6GHz'),
(u'cpu', u'physical_0', u'vendor', u'GenuineIntel')
...
]
```

ahc-report ツールもノードコレクションの外れ値を特定します。--outliers スイッチを使用して、この機能を有効化します。

```
$ sudo ahc-report --outliers
```

```
Group 0 : Checking logical disks perf
standalone_randread_4k_KBps : INFO : sda : Group performance :
min=45296.00, mean=53604.67, max=67923.00, stddev=12453.21
standalone_randread_4k_KBps : ERROR : sda : Group's variance is too
important : 23.23% of 53604.67 whereas limit is set to 15.00%
standalone_randread_4k_KBps : ERROR : sda : Group performance :
UNSTABLE
standalone_read_1M_IOPS : INFO : sda : Group performance : min=
1199.00, mean= 1259.00, max= 1357.00, stddev= 85.58
standalone_read_1M_IOPS : INFO : sda : Group performance =
1259.00 : CONSISTENT
standalone_randread_4k_IOPS : INFO : sda : Group performance :
min=11320.00, mean=13397.33, max=16977.00, stddev= 3113.39
standalone_randread_4k_IOPS : ERROR : sda : Group's variance is too
important : 23.24% of 13397.33 whereas limit is set to 15.00%
standalone_randread_4k_IOPS : ERROR : sda : Group performance :
UNSTABLE
standalone_read_1M_KBps : INFO : sda : Group performance :
min=1231155.00, mean=1292799.67, max=1393152.00, stddev=87661.11
standalone_read_1M_KBps : INFO : sda : Group performance =
1292799.67 : CONSISTENT
...
```

上記の例では **ahc-report** は、全ノードの標準偏差が許容可能な閾値よりも高いため、**standalone_randread_4k_KBps** および **standalone_randread_4k_IOPS** のディスクメトリックを不安定としてマークしました。この例では、2つのノードのディスク転送速度が大きく違う場合に、このような結果になる可能性があります。

ノードコレクションの外れ値を特定すると、高パフォーマンスのノードをより適切なタスクに割り当てることができるので役立ちます。たとえば、ディスク転送速度の高いノードは、より優れたストレージノードになり、メモリーのパフォーマンスが高いノードはコンピュートノードにより適しています。各

ノードのハードウェアパフォーマンスを特定したら、ポリシーのセットを作成して、**ahc-match** コマンドを使用してノードに特定のロールを割り当てます。

6.2.3.2. ahc-match

ahc-match コマンドは、ノードを特定のロールに割り当てられるように、オーバークラウドプランにポリシーセットを適用します。このコマンドを使用する前に、適切なノードとロールを照合するポリシーセットを作成します。

ahc-tools パッケージは、**/etc/ahc-tools/edeploy** に、以下のような一連のポリシーファイルをインストールします。

- **state**: 各ロールのノード数を記載する状態ファイル
- **compute.specs**、**control.specs**: コンピュートノードとコントローラーノードを照合するポリシーファイル
- **compute.cmdb.sample** および **control.cmdb.sample**: Sample Configuration Management Database (CMDB) ファイル。RAID および BIOS の ready-state 設定 (Dell DRAC のみ) のためのキー/値の設定値が含まれます。

状態ファイル

state ファイルは、各ロールのノード数を指定します。デフォルトの設定ファイルは以下のとおりです。

```
[('control', '1'), ('compute', '*')]
```

これは、**ahc-match** により 1 つのコントローラーノードと任意数のコンピュートノードが割り当てられます。このシナリオでは、このファイルを以下のように編集します。

```
[('control', '3'), ('ceph-storage', '3'), ('compute', '*')]
```

これにより、コントローラーノード 3 つ、Red Hat Ceph Storage ノード 3 つ、無限数のコンピュートノードが割り当てられます。

ポリシーファイル

compute.specs および **control.specs** ファイルは、対象ロールごとに割り当てルールを一覧表示します。ファイルの内容は、以下の例のようなタプル形式です。

```
[
  ('cpu', 'logical', 'number', 'ge(2)'),
  ('disk', '$disk', 'size', 'gt(4)'),
  ('network', '$eth', 'ipv4', 'network(192.0.2.0/24)'),
  ('memory', 'total', 'size', 'ge(4294967296)'),
]
```

これは、ハードウェアのパラメーターに基づいた割り当てルールを定義する方法を提供します。利用可能なパラメーターの完全なリファレンスは、「[付録D Automated Health Check \(AHC\) ツールのパラメーター](#)」を参照してください。

また、このポリシーファイルは、値の範囲を照合する場合もヘルパー関数を使用します。これらの関数は以下のとおりです。

- **network()**: 指定のネットワーク内にあるネットワークインターフェース

- **gt()**、**ge()**: 指定値以上
- **lt()**、**le()**: 指定値以下
- **in()**: 照合するアイテムは指定のセットに含まれる必要があります。
- **regexp()**: 正規表現に一致するもの
- **or()**、**and()**、**not()**: Boolean 関数。**or()**、**and()** に指定可能なパラメーターは 2 つ、**not()** のパラメーターは 1 つです。

たとえば、このシナリオでは、「[ahc-report](#)」からの **standalone_randread_4k_KBps** と **standalone_randread_4k_Iops** の値を使用して、平均よりもディスクアクセス速度が早いノードだけにコントローラーロールを制限します。各値のルールは、以下のとおりです。

```
[
  ('disk', '$disk', 'standalone_randread_4k_KBps', 'gt(53604)'),
  ('disk', '$disk', 'standalone_randread_4k_Iops', 'gt(13397)')
]
```

他のロールに追加のポリシープロファイルを作成することも可能です。たとえば、Red Hat Ceph Storage 専用の **ceph-storage.spec** プロファイルを作成します。これらのファイル名 (拡張子なし) は **state** ファイルに含まれるようにします。

ready-state のファイル (Dell DRAC のみ)

ready-state 設定により、デプロイメントに向けたベアメタルリソースの準備を行います。これには、事前設定済みプロファイル用の BIOS および RAID の設定が含まれます。

BIOS の設定を定義するには、**bios_settings** キーの各設定とターゲット値を定義する JSON タプルを定義します。以下に例を示します。

```
[
  {
    'bios_settings': {'ProcVirtualization': 'Enabled', 'ProcCores': 4}
  }
]
```

RAID の設定は、次の 2 つの方法で定義します。

- **物理ディスク ID を一覧表示する方法**:
controller、**size_gb**、**raid_level**、**physical_disks** の属性を使用して物理ディスク ID の一覧を指定します。**controller** には、DRAC によって割り当てられる RAID コントローラーの FQDD を指定する必要があります。同様に、**physical_disks** の一覧には、DRAC カードによって割り当てられる物理ディスクの FQDD の一覧を指定する必要があります。

```
[
  {
    'logical_disks': [
      {'controller': 'RAID.Integrated.1-1',
       'size_gb': 100,
       'physical_disks': [
         'Disk.Bay.0:Enclosure.Internal.0-1:RAID.Integrated.1-1',
         'Disk.Bay.1:Enclosure.Internal.0-1:RAID.Integrated.1-1',
         'Disk.Bay.2:Enclosure.Internal.0-1:RAID.Integrated.1-1'],
       'raid_level': '5'},
    ]
  }
]
```

```
    ]
  }
]
```

- **Ironic** により物理ディスクを **RAID** ボリュームに割り当てる方法:
controller、**size_gb**、**raid_level**、**number_of_physical_disks** の属性が必要となります。**controller** には、DRAC カードによって割り当てられる RAID コントローラーの FQDD を指定する必要があります。

```
[
  {
    'logical_disks': [
      {
        'controller': 'RAID.Integrated.1-1',
        'size_gb': 50,
        'raid_level': '1',
        'number_of_physical_disks': 2},
    ]
  }
]
```

照合ツールの実行

ルールを定義した後は、**ahc-match** ツールを実行してノードを割り当てます。

```
$ sudo ahc-match
```

このコマンドは、全ノードを **/etc/ahc-tools/edeploy/state** に定義したロールと照合します。ノードがロールに一致する場合は、**ahc-match** により、Ironic でノードにロールが 1 つのケイパビリティとして追加されます。

```
$ ironic node-show b73fb5fa-1a2c-49c6-b38e-8de41e3c0532 | grep properties
-A2
| properties      | {u'memory_mb': u'6144', u'cpu_arch': u'x86_64',
u'local_gb': u'40',      |
|                  | u'cpus': u'4', u'capabilities':
u'profile:control,boot_option:local'} |
| instance_uuid  | None
|
```

director は、各ノードに付いたこの **profile** タグを使用して、同じタグを持つロールとフレーバーを照合します。

RAID と BIOS の ready-state 設定を行った場合には、以下のコマンドを実行して、各ノードでこれらを設定します。

```
$ instack-ironic-deployment --configure-nodes
```

6.2.4. ハードウェアプロファイルの作成

director には、登録ノードのハードウェアプロファイルまたはフレーバーのセットが必要です。このシナリオでは、コンピュートノードとコントローラーノード、Ceph Storage ノードごとにプロファイルを作成します。

```
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 control
```

```
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 compute
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 ceph-storage
```



重要

このシナリオの3つのフレーバーの値は、例示のみを目的としています。実際の値には、AHC ツールで特定したハードウェアの仕様を使用してください。

これにより、ノードに3つのフレーバーが作成されます。また、各フレーバーに追加のプロパティも設定します。

```
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="compute" compute
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="control" control
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="ceph-storage" ceph-storage
```

capabilities:boot_option はフレーバーのブートモードを設定し、**capabilities:profile** は使用するプロファイルを定義します。



重要

使用していないロールにも **baremetal** というデフォルトのフレーバーが必要です。このフレーバーがない場合には作成します。

```
$ openstack flavor create --id auto --ram 4096 --disk 40 --
vcpus 1 baremetal
```

6.2.5. Ceph Storage の設定

本項では、OpenStack で使用するために director を使用して Red Hat Ceph Storage をインストール/設定する方法を説明します。このインストール/設定プロセスは、Heat テンプレートと Puppet 設定の組み合わせをベースにしています。

オーバークラウドのイメージにはすでに、コントローラークラスターに Ceph OSD ノードと Ceph Monitor の両方を自動で設定する Ceph Storage ソフトウェアと必要な Puppet モジュールが含まれています。オーバークラウドの Heat テンプレートコレクションには、Ceph Storage の設定を有効にするための設定も含まれています。

Ceph Storage クラスターには、特に Ceph Storage ノード上のディスクレイアウトなどのマイナーな設定が必要となる場合があります。この情報を渡すには、**storage-environment.yaml** 環境ファイルを **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/storage-
environment.yaml ~/templates/.
```

storage-environment.yaml のコピーで、以下のオプションを修正します。

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。 **true** 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。 **false** 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。 **true** に設定します。

GlanceBackend

Glance で使用するバックエンドを定義します。イメージに Ceph Storage を使用するには、 **rbd** に設定します。



注記

storage-environment.yaml には、Heat を直接使用して Ceph Storage を設定するためのオプションも含まれています。ただし、director はこれらのノードを作成して、このシナリオでは、自動的に設定値を定義するので、これらのオプションは必要ありません。

以下の内容を含む追加のセクションをこの環境ファイルに追加します。

```
parameter_defaults:
  ExtraConfig:
    ceph::profile::params::osds:
```

これにより、オーバークラウドに Hiera データがさらに追加されます。このデータは、Puppet の設定に使用されます。詳しくは、[「Puppet 設定データのカスタマイズ」](#)を参照してください。

ceph::profile::params::osds パラメーターを使用して、関連するジャーナルのパーティションとディスクをマッピングします。たとえば、ディスクが 4 つある Ceph ノードは、以下のように割り当てることができます。

- **/dev/sda**: オーバークラウドのイメージを含む root ディスク
- **/dev/sdb**: ディスクにはジャーナルのパーティションが含まれます。これは通常、システムパフォーマンスの向上に役立つソリッドステートドライブ (SSD) です。
- **/dev/sdc** および **/dev/sdd**: OSD ディスク

この例では、マッピングには以下を含めることができます。

```
ceph::profile::params::osds:
  '/dev/sdc':
    journal: '/dev/sdb'
```

```

'/dev/sdd':
  journal: '/dev/sdb'

```

ジャーナル用に別のディスクを使用しない場合には、OSD ディスクに併置されているジャーナルを使用します。**journal** パラメーターには空の値を渡します。

```

ceph::profile::params::osds:
  '/dev/sdb': {}
  '/dev/sdc': {}
  '/dev/sdd': {}

```

storage-environment.yaml ファイルのオプションは、以下の例のように設定されるはずです。

```

parameters:
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: true
  CinderEnableNfsBackend: false
  NovaEnableRbdBackend: true

parameter_defaults:
  ExtraConfig:
    ceph::profile::params::osds:
      '/dev/sdc':
        journal: '/dev/sdb'
      '/dev/sdd':
        journal: '/dev/sdb'

```

変更が完了したら、**storage-environment.yaml** を保存して、オーバークラウドのデプロイ時に Ceph Storage ノードにディスクマッピングとカスタム設定が使用されるようにします。また、デプロイメント内にこのファイルを追加して、ストレージに必要な設定が開始されるようにします。

重要

Ceph Storage OSD のディスクはパーティション分割せず、GPT ディスクラベルを付ける必要があります。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
```

6.2.6. VLAN への全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、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。

このシナリオでは、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: 外部、Floating IP (オーバークラウドの作成後にマッピング)

以下の項では、Heat テンプレートを作成してすべてのネットワーク種別を分離する方法を説明します。その他のネットワーク設定例は、「[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。

6.2.6.1. 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 (「[5章 Heat テンプレートについて](#)」の[理解](#)」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。

高度なオーバークラウドのシナリオでは、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ローカルの Heat テンプレートが作成され、このテンプレートで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します。各テンプレートには、標準の **parameters**、**resources**、**output** 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 のボンディングを定義します。ボンディングでは、2 つ以上の **interfaces** を結合して、冗長性や帯域幅を向上させます。

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

ovs_bridge

Open vSwitch のブリッジを定義します。ブリッジは、複数の **interface**、**bond**、**vlan** オブジェクトを接続します。

```
- 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_bridge

Linux ブリッジを定義します。Open vSwitch ブリッジと同様に、このブリッジは、複数の **interface**、**bond**、**vlan** オブジェクトを接続します。

```
- type: linux_bridge
  name: bridge1
  members:
    - type: interface
      name: nic1
      primary: true
    - type: vlan
      device: bridge1
      vlan_id: {get_param: ExternalNetworkVlanID}
      addresses:
        - ip_netmask: {get_param: ExternalIpSubnet}
```

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。

高度なシナリオでは、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。

```
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:
```

```

        - ip_netmask: 0.0.0.0/0
          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}

```

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。

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で作成した環境ファイルで定義します。

重要

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

```

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

```

6.2.6.2. 高度なオーバークラウドのネットワーク環境ファイルの作成

ネットワーク環境ファイルは Heat の環境ファイルで、オーバークラウドのネットワーク環境を記述し、前のセクションのネットワークインターフェース設定テンプレートを参照します。IP アドレス範囲と合わせてネットワークのサブネットおよび VLAN を定義します。また、これらの値をローカル環境

用にカスタマイズします。

このシナリオでは、`/home/stack/templates/network-environment.yaml` で保存した下記のネットワーク環境ファイルを使用します。

```
resource_registry:
  OS::Triple0::BlockStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/cinder-storage.yaml
  OS::Triple0::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::Triple0::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
  OS::Triple0::ObjectStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/swift-storage.yaml
  OS::Triple0::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
  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'}]
  # 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
  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"
```

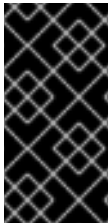
resource_registry セクションには、各ノードロールのネットワークインターフェーステンプレートへのリンクが含まれます。

parameter_defaults セクションには、各ネットワーク種別のネットワークオプションを定義するパラメーター一覧が含まれます。これらのオプションについての詳しい参考情報は「[付録G ネットワーク環境のオプション](#)」を参照してください。

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

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

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

6.2.6.3. OpenStack サービスの分離ネットワークへの割り当て

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

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

...

```
parameter_defaults:
```

```
  ServiceNetMap:
```

```
    NeutronTenantNetwork: tenant
    CeilometerApiNetwork: internal_api
    MongoDBNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    GlanceApiNetwork: storage
    GlanceRegistryNetwork: internal_api
    KeystoneAdminApiNetwork: internal_api
```

```

KeystonePublicApiNetwork: internal_api
NeutronApiNetwork: internal_api
HeatApiNetwork: internal_api
NovaApiNetwork: internal_api
NovaMetadataNetwork: internal_api
NovaVncProxyNetwork: internal_api
SwiftMgmtNetwork: storage_mgmt
SwiftProxyNetwork: storage
HorizonNetwork: internal_api
MemcachedNetwork: internal_api
RabbitMqNetwork: internal_api
RedisNetwork: internal_api
MysqlNetwork: internal_api
CephClusterNetwork: storage_mgmt
CephPublicNetwork: storage
# Define which network will be used for hostname resolution
ControllerHostnameResolveNetwork: internal_api
ComputeHostnameResolveNetwork: internal_api
BlockStorageHostnameResolveNetwork: internal_api
ObjectStorageHostnameResolveNetwork: internal_api
CephStorageHostnameResolveNetwork: storage

```

これらのパラメーターを **storage** に変更すると、対象のサービスが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、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 **parameter_defaults** セットを定義するだけで結構です。

6.2.7. 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。

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、[「VLAN への全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B SSL/TLS 証明書の設定」](#)を参照してください。

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。

parameter_defaults:

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:

以下のように、秘密鍵の内容を **SSLKey** パラメーターにコピーします。

```
parameter_defaults:
...
SSLKey: |
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0lV09hkJZnGP6qb6wtYUoy1bVP7
...
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B 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。

resource_registry:

OS::TripleO::NodeTLSData:

OS::TripleO::NodeTLSData: のリソース URL を絶対 URL に変更します。

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

ルート証明書の注入

自己署名証明書を使用する場合または、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、証明書をオーバークラウドのイメージに注入します。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。

parameter_defaults:

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 CzA JBgNV
    ...
    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。

resource_registry:

OS::TripleO::NodeTLSCAData:

OS::TripleO::NodeTLSCAData: のリソース URL を絶対 URL に変更します。

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

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。

parameter_defaults:

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"]
```

オーバークラウド作成時の環境ファイルの追加

「[高度な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
```

6.2.8. 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。

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。「[付録I デプロイメントパラメーター](#)」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、「[高度な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 [...]
```

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 Subscription Manager 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 5 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 6 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 **rhnreg_ks** に登録します。

rhel_reg_server_url

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

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)。

「[高度な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。~/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。

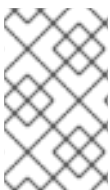
```
$ openstack overcloud deploy --templates [...] -e
/home/stack/templates/rhel-registration/environment-rhel-registration.yaml
-e /home/stack/templates/rhel-registration/rhel-registration-resource-
registry.yaml
```

**重要**

登録は、**OS::TripleO::NodeExtraConfig** Heat リソースのように設定されます。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。

6.2.9. 高度なオーバークラウドの作成

OpenStack 環境の最後の段階として、環境作成に必要とされるコマンドを実行します。このコマンドを使用して、3つのコントローラーノード、3つのコンピューターノード、3つの Ceph Storage ノードをインストールします。

**注記**

Red Hat カスタマーポータルには、オーバークラウド作成前の設定検証に役立つラボがあります。このラボは、<https://access.redhat.com/labs/ospec/> で利用できます。ラボについての説明は、<https://access.redhat.com/labsinfo/ospec> に記載されています。

以下のコマンドを実行して、高度なオーバークラウドの作成を開始します。

```
$ 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 --neutron-network-type vxlan --neutron-tunnel-types vxlan
```

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、[「高度なオーバークラウドの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 つにスケーリングします。
- **--compute-scale 3:** コンピュートノードを 3 つにスケーリングします。
- **--ceph-storage-scale 3:** Ceph Storage ノードを 3 つ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:** 対象の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。
- **--neutron-network-type vxlan:** オーバークラウドの Neutron ネットワークに Virtual Extensible LAN (VXLAN) を使用します。
- **--neutron-tunnel-types vxlan:** オーバークラウドの Neutron トンネリングに Virtual Extensible LAN (VXLAN) を使用します。



注記

オプションの完全な一覧を表示するには、以下を実行します。

```
$ openstack help overcloud deploy
```

また、パラメーターの例については「[付録I デプロイメントパラメーター](#)」、登録の詳細については「[オーバークラウドの登録](#)」も参照してください。

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。



警告

-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。director は、「[7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ことになる場合があります。

オーバークラウドの設定を後で変更する予定がある場合は、カスタム環境ファイルと Heat テンプレートのパラメーターを変更し、**openstack overcloud deploy** の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。

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 \
  --neutron-network-type vxlan \
  --neutron-tunnel-types vxlan \
  --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。



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。

6.2.10. 高度な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
```

これにより、director 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 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
```

6.2.11. コンピュート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 と呼ばれるツールを使用して、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、クラスターが実行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6.5 フェンスエージェント

デバイス	種別
<code>fence_ipmilan</code>	Intelligent Platform Management Interface (IPMI)
<code>fence_idrac</code> 、 <code>fence_drac5</code>	Dell Remote Access Controller (DRAC)
<code>fence_ilo</code>	Integrated Lights-Out (iLO)
<code>fence_ucs</code>	Cisco UCS: 詳しい情報は「 Configuring Cisco Unified Computing System (UCS) Fencing on an OpenStack High Availability Environment 」の記事を参照してください。
<code>fence_xvm</code> 、 <code>fence_virt</code>	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 1 の場合:

```
$ sudo pcs stonith create my-ipmilan-for-controller01 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-controller01 avoids  
overcloud-controller-0
```

コントローラーノード 2 の場合:

```
$ sudo pcs stonith create my-ipmilan-for-controller02 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-controller02 avoids  
overcloud-controller-1
```

コントローラーノード 3 の場合:

```
$ sudo pcs stonith create my-ipmilan-for-controller03 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-controller03 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
```

6.2.12. 高度なオーバークラウドの完了

これで高度な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、「[7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7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。

7.1. 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 |
+-----+-----+-----+
+-----+
```

7.2. オーバークラウドの外部ネットワークの作成

基本的/高度なオーバークラウドのシナリオでは、ノードのインターフェースが外部ネットワークを使用するように設定しましたが、Floating IP アドレスをインスタンスに割り当てできるように、オーバークラウド上に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 nova --router:external --provider:network_type flat -
-provider:physical_network datacentre
$ neutron subnet-create --name nova --enable_dhcp=False --allocation-
pool=start=10.1.1.51,end=10.1.1.250 --gateway=10.1.1.1 nova 10.1.1.0/24
```

以下の例では、**nova** 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 **datacenter** 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、**datacenter** 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 nova --router:external --provider:network_type vlan -
-provider:physical_network datacentre --provider:segmentation_id 104
$ neutron subnet-create --name nova --enable_dhcp=False --allocation-
pool=start=10.1.1.51,end=10.1.1.250 --gateway=10.1.1.1 nova 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... | nova          | 01c5f621-1e0f-4b9d-9c30-
7dc59592a52f 10.1.1.0/24 |
+-----+-----+-----+
+-----+
```

7.3. 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
datacenter:br-ex,floating:br-floating
```

以下のコマンドを使用して、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
```

7.4. 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データセンター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仮想マシン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。コンピュータノードは、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 tp が DHCP サービスをテナントの仮想マシン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
```

7.5. 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 を使用してお使いの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
```

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インストールします。

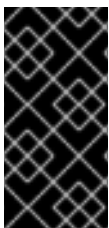
```
$ mkdir ~/tempest
$ cd ~/tempest
$ /usr/share/openstack-tempest-kilo/tools/configure-tempest-directory
```

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、インターネットへのアクセスがあるか、インターネットアクセスのあるプロキシを使用す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、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、出力では以下のように失敗したテストについて表示する場合があります。

```
{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、応答ヘッダーとボディが含まれます。

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
```

7.6. 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、「[高度な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 --neutron-network-type vxlan --neutron-tunnel-types vxlan
```

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、**-e** オプションを指定して **openstack overcloud deploy** コマンドを実行しファイルを追加します。

```
$ 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 --neutron-network-type vxlan --neutron-tunnel-types vxlan
```

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 が後ほど上書きしてしまう可能性がありますので、オーバークラウドの設定に手動で変更を加えないように推奨しています。

7.7. オーバークラウドへの仮想マシンのインポート

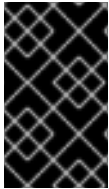
既存の OpenStack 環境があり、仮想マシンを Red Hat OpenStack Platform 環境に移行する予定がある場合には、以下の手順を使用します。

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。

7.8. 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。

手順7.1 コンピュートノードの SSH キーの設定

ホストの各 **nova** ユーザーが移行プロセス中にアクセスできるように、全コンピュートノードには共有 SSH キーが必要です。以下の手順を使用して、各コンピュートノードで SSH キーペアを設定します。

1. SSH キーを生成します。

```
$ ssh-keygen -t rsa -f nova_id_rsa
```

2. 各コンピュートノード上の **nova** ユーザーのホームディレクトリーに、SSH キーをコピーします。
3. **nova** ユーザーとして各コンピュートノードにログインして、以下のスクリプトを実行し、キーを設定します。

```
NOVA_SSH=/var/lib/nova/.ssh
mkdir ${NOVA_SSH}

cp nova_id_rsa ${NOVA_SSH}/id_rsa
chmod 600 ${NOVA_SSH}/id_rsa
cp nova_id_rsa.pub ${NOVA_SSH}/id_rsa.pub
cp nova_id_rsa.pub ${NOVA_SSH}/authorized_keys

chown -R nova.nova ${NOVA_SSH}

# enable login for nova user on compute hosts:
usermod -s /bin/bash nova

# add ssh keys of overcloud nodes into known hosts:
ssh-keyscan -t rsa `os-apply-config --key hosts --type raw --key-default '' | awk '{print $1}'` >> /etc/ssh/ssh_known_hosts
```

手順7.2 コンピュートノードからの仮想マシンの移行

1. director で、**overcloudrc** を読み込み、現在の nova サービスの一覧を取得します。

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

2. 移行するノードで **nova-compute** サービスを無効にします。


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

これにより、仮想マシンはそのノード上でスケジュールされなくなります。

3. ノードから仮想マシンを移行するプロセスを開始します。

```
$ nova host-servers-migrate [hostname]
```

4. 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
```

5. 各仮想マシン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]
```

これにより、ホストからすべての仮想マシンが移行されます。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
```

7.9. オーバークラウドの削除防止

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。

7.10. 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。

手順7.3 オーバークラウドの削除

1. 既存のオーバークラウドを削除します。

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

2. 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い、オーバークラウドを再度作成します。

第8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ト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。

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8.1 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	N	N	
Compute	Y	Y	
Ceph Storage ノード	Y	N	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。
Cinder Storage ノード	N	N	
Swift Storage ノード	N	N	



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 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。

8.1. コンピュートノードまたは CEPH STORAGE ノードの追加

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、[「高度なオーバークラウドのノード登録」](#)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-list
$ ironic node-set-maintenance [NODE UUID] true
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-maintenance [NODE UUID] false

```

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'

```

または、Automated Health Check (AHC) ツールを使用して、新規ノードを必要なロールに自動的にタグ付けすることもできます。詳しくは、[「Automated Health Check \(AHC\) ツールを使用したノードの自動タグ付け」](#)を参照してください。

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          |
+-----+-----+

```

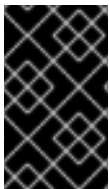
新規ノードの **deploy_kernel** および **deploy_ramdisk** 設定にこれらの UUID を設定します。

```
$ 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。

8.2. 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 service-list  
$ neutron service-delete [openvswitch-service-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。

8.3. 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。

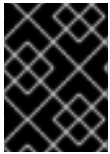
1. 既存のコンピュートノードからワークロードを移行して、ノードをシャットダウンします。この手順は「[オーバークラウドのコンピュートノードからの仮想マシンの移行](#)」を参照してください。
2. オーバークラウドからコンピュートノードを削除します。この手順は「[コンピュートノードの削除](#)」を参照してください。
3. 新しいコンピュートノードでオーバークラウドをスケーリングアウトします。この手順は「[8章 オーバークラウドのスケーリング](#)」を参照してください。

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。

8.4. 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を置き換えるための要求で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処理中のどの時点かで、**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ます。この後に **openstack overcloud deploy** は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。

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

8.4.2. 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-ae5d-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 がエラー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 ([「10章 カスタム設定の作成」](#)を参照)、以下のコマンドを **overcloud-without-mergepy.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-without-mergepy.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::Triple0::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::Triple0::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]}
```

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]
```

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、**-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
```

8.4.3. 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 **odelist** が表示されます (**overcloud-controller-1**)。

```
odelist {
  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、後で使用するのための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@192.168.201.47 "sudo pcs cluster
localnode remove overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.201.47 "sudo pcs cluster
reload corosync"
[stack@director] ssh heat-admin@192.168.201.46 "sudo pcs cluster
localnode remove overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.201.46 "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@192.168.201.47
[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。

10. 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。

11. 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

```

12. 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** 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、そ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"
```

13. Pacemaker を使用して、いくつかのサービスを有効化して再起動します。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
```

14. 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 overcloud-controller-3
```

15. 全ノードで Keystone サービスが起動するのを待ちます。

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs status | grep
keystone -A1
Clone Set: openstack-keystone-clone [openstack-keystone]
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 openstack-keystone-clone overcloud-controller-3
```

16. 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** ファイルは必要ない点に注意してください。

8.4.4. 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。

フェンシングが無効化されている場合には有効にします。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有効にします。

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs property set stonith-
enabled=true
```

director を終了します。

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

8.4.5. オーバークラウドのネットワークエージェントの最終処理

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
```

8.4.6. 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@192.168.201.47
[heat-admin@overcloud-controller-0 ~]$ pcs resource restart openstack-
nova-consoleauth
```

8.4.7. 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。

8.5. CEPH STORAGE ノードの置き換え

Ceph Storage ノードに障害が発生する可能性があります。このような状況では、データが失われないように、問題のあるノードを無効化してリバランスしてから、オーバークラウドから削除するようにしてください。以下の手順では、Ceph Storage ノードを置き換えるプロセスについて説明します。



注記

以下の手順では、『Red Hat Ceph Storage Administration Guide』からの手順を使用して、手動で Ceph Storage ノードを削除します。Ceph Storage ノードの手動での削除に関する詳しい情報は、『Red Hat Ceph Storage Administration Guide』の「[Chapter 15. Removing OSDs \(Manual\)](#)」を参照してください。

1. **heat-admin** ユーザーとして、コントローラーノードまたは Ceph Storage ノードにログインします。director の **stack** ユーザーには、**heat-admin** ユーザーにアクセスするための SSH キーがあります。
2. OSD ツリーを一覧表示して、ノードの OSD を検索します。たとえば、削除するノードには、以下の OSD が含まれる場合があります。

```
-2 0.09998      host overcloud-cephstorage-0
0 0.04999      osd.0                      up  1.00000
1.00000
1 0.04999      osd.1                      up  1.00000
1.00000
```

3. Ceph Storage ノードの OSD を無効化します。今回は、OSD ID は 0 と 1 です。

```
[heat-admin@overcloud-controller-0 ~]$ sudo ceph osd out 0
[heat-admin@overcloud-controller-0 ~]$ sudo ceph osd out 1
```

Ceph Storage Cluster がリバランスを開始します。このプロセスが完了するまで待ってください。以下のコマンドを使用して、ステータスを確認できます。

■

```
[heat-admin@overcloud-controller-0 ~]$ sudo ceph -w
```

4. Ceph クラスターのリバランスが完了したら、**heat-admin** ユーザーとして、問題のある Ceph Storage ノードにログインして、このノードを停止します。

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo /etc/init.d/ceph stop
osd.0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo /etc/init.d/ceph stop
osd.1
```

5. これ以上データを受信しないように、CRUSH マップからこの Ceph Storage ノードを削除します。

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd crush remove
osd.0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd crush remove
osd.1
```

6. OSD 認証キーを削除します。

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph auth del osd.0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph auth del osd.1
```

7. クラスターから OSD を削除します。

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd rm 0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd rm 1
```

8. ノードからログアウトして、**stack** ユーザーとして director ホストに戻ります。

```
[heat-admin@overcloud-cephstorage-0 ~]$ exit
[stack@director ~]$
```

9. director が再度プロビジョニングしないように、Ceph Storage ノードを無効にします。

```
[stack@director ~]$ ironic node-list
[stack@director ~]$ ironic node-set-maintenance [UUID] true
```

10. Ceph Storage 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。

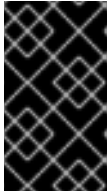
```
$ heat stack-list
```

削除する Ceph Storage 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]
```



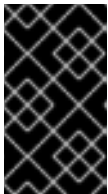
重要

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、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、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渡します。

stack が更新を完了するまで待ちます。 **heat stack-list --show-nested** を使用して、stack の更新を監視します。

11. 「[コンピューターノードまたは Ceph Storage ノードの追加](#)」の手順に従って新しいノードを director のノードプールに追加し、Ceph Storage ノードとしてデプロイします。 **--ceph-storage-scale** を使用してオーバークラウド内の Ceph Storage ノードの合計数を定義します。たとえば、3 つのノードで構成されるクラスターから、問題があって削除したノードを置き換える場合には、 **--ceph-storage-scale 3** を使用すると Ceph Storage ノードの数が元の値に戻ります。

```
$ openstack overcloud deploy --templates --ceph-storage-scale 3 -e
[ENVIRONMENT_FILES]
```



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、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、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渡します。

director は、新しいノードをプロビジョニングして、新しいノードの詳細を用いて stack 全体を更新します。

12. **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、Ceph Storage ノードのステータスを確認します。以下に例を示します。

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

osdmap セクションの値が、クラスターに必要なノード数と一致していることを確認します。

エラーの発生した Ceph Storage ノードが新規ノードに置き換えられました。

第9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

9.1. 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
$ nova list
$ ironic node-list
$ heat stack-list
```

9.2. 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. 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。

9.3. 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 ノードすべて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
```

9.4. 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。

アンダークラウドから、全コンピュートノードとそれらの UUID を一覧表示します。

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

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。

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

4. リブートする次のノードを選択します。

9.5. 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10章 カスタム設定の作成

Red Hat Enterprise Linux OpenStack Platform 環境と統合する追加のアプリケーションを設定する必要がある場合があります。このようなカスタム設定には、Heat テンプレートをオーバークラウドのスタックに追加する必要があります。本項では、利用可能なカスタム設定操作の一部について考察します。

10.1. 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、全ノード上の IP アドレスを使用してネームサーバーを更新することを目的とします。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、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します。

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

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

初回起動の設定を追加するには、最初にオーバークラウドを作成する際に、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、最初に使用した Heat テンプレートの内容が上書きされてしまいます。

10.2. 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、変数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。

```
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_identifiser: {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。

ExtraPreDeployments

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_identifiser** 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

```

この設定を追加するには、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、全ノードに設定が追加されます。



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。複数で使用すると、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。

10.3. 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更新時に設定を追加する必要とな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、変数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。

```
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。

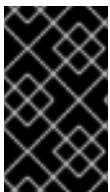
```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

この設定を追加するには、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、全ノードに設定が追加されます。



重要

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

10.4. PUPPET 設定データのカスタマイズ

オーバークラウドの機能をカスタマイズするための Puppet 設定データを渡すには 2 つの方法があります。

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

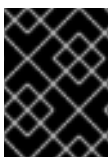
CephStorageExtraConfig

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、以下のように設定します。

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

openstack overcloud deploy を実行する際に、この環境ファイルを含めます。



重要

各パラメーターは 1 回のみ定義することが可能です。1 回以上使用すると、以前の値が上書きされます。

10.5. 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。基本例として、各ノードに **motd** をインストールします。このインストールのプロセスでは、まず Heat テンプレート

(`/home/stack/templates/custom_puppet_config.yaml`) を作成して、Puppet 設定を起動します。

```
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** を含め、ノードが設定されるようにこのファイルを渡します。**motd.pp** ファイル自体には、**motd** をインストールして設定する 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。

10.6. カスタムのオーバークラウド HEAT テンプレートの使用

オーバークラウドの作成時に、director はデフォルトの Heat テンプレートセットを使用します。ただし、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。これは、オーバークラウドの特定の部分をカスタマイズする際に便利です。

/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。これは、本シナリオの後半に記載しています (「[高度な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 では、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。

- 「[初回起動での設定のカスタマイズ](#)」
- 「[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](#)」

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11章 環境の更新

本章では、選択したオーバークラウドの作成後に環境を更新する方法について考察します。これには、アンダークラウドとオーバークラウドの両方の機能の更新が含まれます。

11.1. DIRECTOR パッケージの更新

director は、環境の更新には標準の RPM メソッドを利用します。このメソッドでは、director のホストが最新のパッケージを使用していることを **yum** で確認します。

```
$ sudo yum update
```

重要

パッケージの更新後には、director ですべての OpenStack サービスが実行されていることを確認します。また、**ironic-api** および **ironic-discoverd** サービスが実行されていることを確認します。実行されていない場合には、起動してください。

```
$ sudo systemctl restart openstack-ironic-api openstack-ironic-discoverd
```

同様に、データベースが利用できない場合には、アンダークラウドの **heat-engine** が起動に失敗する可能性があります。起動に失敗した場合には、更新後に **heat-engine** を手動で再起動してください。

```
$ sudo systemctl start openstack-heat-engine.service
```

11.2. オーバークラウドと検出イメージの更新

以下の手順に従って、ノードの検出とオーバークラウドのデプロイメント用に最新のイメージを用意します。これらのイメージは、Red Hat カスタマーポータルの https://access.redhat.com/downloads/content/191/ver=7.0/rhel---7/7.0/x86_64/product-downloads の Red Hat Enterprise Linux OpenStack Platform ダウンロードページから取得してください。イメージアーカイブの取得と抽出についての詳しい情報は、「[オーバークラウドノードのイメージの取得](#)」を参照してください。

stack ユーザーのホームディレクトリー上の **images** ディレクトリーにこれらのイメージをダウンロードします (**/home/stack/images**)。イメージの取得後には、以下の手順に従い、イメージを置き換えます。

手順11.1 イメージの更新

1. director から既存のイメージを削除します。

```
$ openstack image list
$ openstack image delete [IMAGE-UUID] [IMAGE-UUID] [IMAGE-UUID]
[IMAGE-UUID] [IMAGE-UUID]
```

2. 最新のイメージを director にインポートします。

```
$ cd ~/images
$ openstack overcloud image upload --update-existing
$ openstack baremetal configure boot
```

director が更新され、最新のパッケージとイメージを使用するようになりました。更新後にサービスを再起動する必要はありません。

11.3. オーバークラウドの更新

本項には、オーバークラウドの更新に必要な手順を記載します。各セクションを順番に従って進み、お使いの環境に関連するセクションのみを適用します。

11.3.1. 設定エージェント



重要

以下のセクションは、Red Hat Enterprise Linux OpenStack Platform 7.0 または 7.1 を Red Hat Enterprise Linux OpenStack Platform 7.2 以降のバージョンに更新する場合のみ必要です。

既知の問題 ([BZ#1278181](#) を参照) が原因で、オーバークラウドは設定エージェントを一部手動で設定する必要があります。これには、設定エージェントのスクリプトの新しいバージョンを director からオーバークラウド内の各ノードにコピーする作業を伴います。

stack ユーザーとして director ホストにログインし、アンダークラウドの設定ファイルを source コマンドで読み込みます。

```
$ source ~/stackrc
```

設定エージェント (**55-heat-config**) を各オーバークラウドノードにコピーします。この操作を行うには、以下のコマンドを使用して全ホストに対して実行します。

```
$ for i in `nova list|awk '/Running/ {print $(NF-1)}'|awk -F=" " '{print $NF}'`; do echo $i; scp -o StrictHostKeyChecking=no /usr/share/openstack-heat-templates/software-config/elements/heat-config/os-refresh-config/configure.d/55-heat-config heat-admin@${i}: ; ssh -o StrictHostKeyChecking=no heat-admin@${i} 'sudo /bin/bash -c "cp /home/heat-admin/55-heat-config /usr/libexec/os-refresh-config/configure.d/55-heat-config"'; done
```

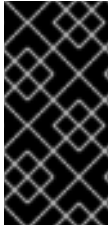
これにより、設定エージェントは最新の状態となります。

このオーバークラウドは、デプロイ後のファイルをいくつか再作成する必要もあります。director には、これをアーカイブするためのスクリプトが含まれています。**heat-config-rebuild-deployed** スクリプトをコピーして実行します。この操作は、以下のコマンドを使用して全ノードに対して実行します。

```
$ for i in `nova list|awk '/Running/ {print $(NF-1)}'|awk -F=" " '{print $NF}'`; do echo $i; scp -o StrictHostKeyChecking=no /usr/share/openstack-heat-templates/software-config/elements/heat-config/bin/heat-config-rebuild-deployed heat-admin@${i}: ; ssh -o StrictHostKeyChecking=no heat-admin@${i} 'sudo /bin/bash -c "mkdir -p /usr/share/openstack-heat-
```

```
templates/software-config/elements/heat-config/bin ; cp heat-config-rebuild-deployed /usr/share/openstack-heat-templates/software-config/elements/heat-config/bin/heat-config-rebuild-deployed ; chmod +x /usr/share/openstack-heat-templates/software-config/elements/heat-config/bin/heat-config-rebuild-deployed ; /usr/share/openstack-heat-templates/software-config/elements/heat-config/bin/heat-config-rebuild-deployed" ; done
```

11.3.2. 変更済みのオーバークラウドテンプレート



重要

本セクションは、「[カスタムのオーバークラウド Heat テンプレートの使用](#)」にある変更済みのテンプレートコレクションを使用する場合のみに必要です。これは、そのコピーが、**/usr/share/openstack-tripleo-heat-templates/**にある元の Heat テンプレートコレクションの静的なスナップショットであるためです。

変更済みテンプレートコレクションを更新するには、以下の手順を実行する必要があります。

1. 既存のカスタムテンプレートコレクションを保存します。

```
$ mv ~/templates/my-overcloud/ ~/templates/my-overcloud.bak
```

2. **/usr/share/openstack-tripleo-heat-templates** にあるテンプレートコレクションの新しいバージョンを置き換えます。

```
# sudo cp -rv /usr/share/openstack-tripleo-heat-templates  
~/templates/my-overcloud/
```

3. 古いバージョンと新しいバージョンのカスタムテンプレートコレクションの差異を確認します。これらの2つの間の変更を確認するには、以下の **diff** コマンドを使用します。

```
# diff -Nary ~/templates/my-overcloud.bak/ ~/templates/my-overcloud/
```

これは、新規テンプレートコレクションに取り入れることが可能な旧テンプレートコレクションのカスタマイズを特定するのに役立ちます。

4. 新規テンプレートにカスタマイズを取り入れます。

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。

- 「初回起動での設定のカスタマイズ」
- 「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」

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。

11.3.3. 新規環境のパラメーター

更新した Heat テンプレートコレクションには、元の Red Hat Enterprise Linux OpenStack Platform 7.0 リリースには含まれていなかった新しいパラメーターがいくつか必要です。これらのパラメーターは今後の更新で追加することを推奨します。

カスタムの環境ファイル (`~/templates/param-updates.yaml`) に以下にあげる追加のパラメーターを記載します。

表11.1 含めるべき追加のパラメーター

新規パラメーター	説明
ControlPlaneDefaultRoute	コントロールプレーンネットワークのデフォルトルート
EC2MetadataIp	EC2 メタデータサーバーの IP アドレス

例

```
parameter_defaults:
  ControlPlaneDefaultRoute: 192.168.1.1
  EC2MetadataIp: 169.254.169.254
```

オーバークラウドの今後の更新時にこのファイルを追加するようにしてください。

11.3.4. バージョン固有の注意事項

本項には、director およびオーバークラウドの初期バージョン固有の注意事項を記載します。本項の記載内容は、今後の更新時にオーバークラウドスタックに追加する際の参考資料として利用してください。

OpenStack Platform director 7.0 を使用しており、**OpenStack Platform director 7.2** 以降にアップグレードする場合:

- 初回のクラウドデプロイメントに使用したのと同じ値の **ServiceNetMap** パラメーターを必ず使用するようにしてください (「[OpenStack サービスの分離ネットワークへの割り当て](#)」を参照)。初回のデプロイメントでカスタム値が使用されていた場合には、それと同じカスタム値を指定してください。7.0 からの更新でカスタムの **ServiceNetMap** 値を初回デプロイメントで使用しなかった場合には、以下の環境ファイルを更新のコマンドに追加して、7.0 の値を保持します。

```
/usr/share/openstack-tripleo-heat-templates/environments/updates/update-from-keystone-admin-internal-api.yaml
```

オーバークラウドの今後の更新時にこのファイルを追加するようにしてください。

オーバークラウド作成後の **ServiceNetMap** 値の変更は、現在サポートされていません。

- オーバークラウドに単一のネットワークを使用する場合 (例: オリジナルのデプロイメントに **network-isolation.yaml** が含まれていなかった場合) には、更新のコマンドに以下の環境ファイルを追加します。

```
/usr/share/openstack-tripleo-heat-templates/environments/updates/update-from-publicvip-on-ctlplane.yaml
```

オーバークラウドの今後の更新時にこのファイルを追加するようにしてください。外部のロードバランサーを使用する場合にはこのファイルは必要ない点に注意してください。

OpenStack Platform director 7.1 を使用しており、OpenStack Platform director 7.2 以降にアップグレードする場合:

- 初回のクラウドデプロイメントに使用したのと同じ値の **ServiceNetMap** パラメーターを必ず使用するようにしてください (「[OpenStack サービスの分離ネットワークへの割り当て](#)」を参照)。初回のデプロイメントでカスタム値が使用されていた場合には、それと同じカスタム値を指定してください。7.1 からの更新でカスタムの **ServiceNetMap** の値を使用しなかった場合には、**ServiceNetMap** に追加の環境ファイルや値を指定する必要はありません。オーバークラウドの作成後の **ServiceNetMap** 値の変更は現在サポートされていません。
- 更新のコマンドに以下の環境ファイルを追加して、仮想 IP リソースが **vip.yaml** にマップされたままの状態となるようにします。

```
/usr/share/openstack-tripleo-heat-templates/environments/updates/update-from-vip.yaml
```

オーバークラウドの今後の更新時にこのファイルを追加するようにしてください。外部のロードバランサーを使用する場合にはこのファイルは必要ない点に注意してください。

- 7.1 からの更新で外部のロードバランサーを使用しない場合には、**control_virtual_ip** の入力パラメーターにコントロール仮想 IP を指定します。これは、アップグレード中にリソースが置き換えられることが理由です。そのためには、以下のコマンドで、現在の **control_virtual_ip** アドレスを確認します。

```
$ neutron port-show control_virtual_ip | grep ip_address
{"subnet_id": "3d7c11e0-53d9-4a54-a9d7-55865fcc1e47", "ip_address": "192.0.2.21"} |
```

この仮想 IP を「[新規環境のパラメーター](#)」に記載の **~/templates/param-updates.yaml** の例のようなカスタムの環境ファイルに追加します。


```
parameters:
  ControlFixedIPs: [{'ip_address': '192.0.2.21'}]
```

オーバークラウドの今後の更新時にこのファイルを追加するようにしてください。外部のロードバランサーを使用する場合にはこのファイルは必要ない点に注意してください。

既存の Neutron ポートを削除します。

```
$ neutron port-delete control_virtual_ip
```

更新プロセスにより、この仮想 IP は、元の IP アドレスを使用する新しいポートに置き換えられます。

OpenStack Platform director 7.2 から OpenStack Platform director 7.3 以降にアップグレードする場合:

- アンダークラウドの Heat では、既知の問題 ([BZ#1305947](#) を参照) に対応するために、RPC 応答のタイムアウトを引き伸ばす必要があります。`/etc/heat/heat.conf` を編集して、以下のパラメーターに設定してください。

```
rpc_response_timeout=600
```

次にすべての Heat サービスを再起動します。

```
$ systemctl restart openstack-heat-api.service
$ systemctl restart openstack-heat-api-cfn.service
$ systemctl restart openstack-heat-engine.service
```

11.3.5. オーバークラウドパッケージの更新

オーバークラウドでは、環境の更新に標準の RPM メソッドを利用します。このメソッドでは、director から **openstack overcloud update** を使用して全ノードを更新します。

`-e` を使用して、オーバークラウドと関連のある環境ファイルとアップグレードパスを追加します。後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。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 **overcloud-resource-registry-puppet.yaml** ファイル。このファイルは、**openstack overcloud deploy** コマンドを実行する場合には自動的に含まれますが、**openstack overcloud update** コマンドを実行する場合は必ず指定する必要があります。
- 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、「[VLAN への全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
- ストレージの環境ファイル
- Red Hat CDN または Satellite 登録用の環境ファイル
- 「[バージョン固有の注意事項](#)」のバージョン固有の環境ファイル

- その他のカスタム環境ファイル

全ノードで並行して更新を実行すると問題が発生する可能性があります。たとえば、パッケージの更新には、サービスの再起動が必要となる場合があります、その操作によって他のノードが中断される可能性があります。そのため、この更新プロセスでは、一連のブレイクポイントを設けて、ノードごとに更新します。1つのノードでパッケージの更新が完了すると、更新プロセスは次のノードに移ります。更新のプロセスには、各ブレイクポイントで確認が要求される対話モードでコマンドを実行するための **-i** オプションが必要です。-i オプションを使用しない場合には、更新は最初のブレイクポイントで一時停止の状態のままとなります。

director 7.1 から 7.2 に更新するコマンドの例を以下に示します。

```
$ openstack overcloud update stack overcloud -i \
  --templates \
  -e /usr/share/openstack-tripleo-heat-templates/overcloud-resource-
registry-puppet.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-
isolation.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/storage-environment.yaml \
  -e /home/stack/templates/rhel-registration/environment-rhel-
registration.yaml \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/updates/update-from-vip.yaml \
  -e /home/stack/templates/param-updates.yaml
```

このコマンドを実行すると更新のプロセスが開始します。このプロセス中に、director はブレイクポイントを通過するためのプロンプトを定期的に表示します。以下に例を示します。

```
not_started: [u'overcloud-controller-0', u'overcloud-controller-1',
u'overcloud-controller-2']
on_breakpoint: [u'overcloud-compute-0']
Breakpoint reached, continue?
```

Enter を押すと、**on_breakpoint** 一覧の最後のノードからブレイクポイントを通過します。これで、そのノードの更新が開始します。また、ノード名を入力して特定のノードでブレイクポイントを通過したり、複数のノードで一度にブレイクポイントを通過するための正規表現を入力することも可能です。ただし、複数のコントローラーノードで同時にブレイクポイントを通過することはお勧めしません。全ノードが更新を完了するまで、このプロセスを継続します。

重要

更新のプロセスを実行する場合には、オーバークラウド内のノードは自動的に再起動されません。インスタンスのダウンタイムなしで環境を再起動する手順については、[9 章 オーバークラウドのリブート](#)を参照してください。

重要

コントローラーノードにフェンシングを設定している場合には、更新プロセスによってその設定が無効になる場合があります。更新プロセスの完了時には、コントローラーノードの1つで以下のコマンドを実行してフェンシングを再度有効にします。

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


第12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-discoverd** は **openstack-ironic-discoverd** と **openstack-ironic-discoverd-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-discoverd -u openstack-ironic-discoverd-dnsmasq
```

- **ironic-discoverd** は、**/var/log/ironic-discoverd/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。

12.1. 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。

手順12.1 誤った MAC アドレスの修正

1. 割り当てられたポートの UUID を確認します。

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

2. MAC アドレスを更新します。

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

手順12.2 誤った IPMI アドレスの修正

- 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]
```

12.2. 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

検出および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-discoverd**) 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

一般的には、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-discoverd** 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
```

検出プロセスの停止

現在 **ironic-discoverd** は直接検出プロセスを停止することができません。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-discoverd/discoverd.conf** の **timeout** 設定を別のタイムアウト時間 (分) 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。

手順12.3 検出プロセスの停止

1. 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
```

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

```
$ rm /var/lib/ironic-discoverd/discoverd.sqlite  
$ sudo systemctl restart openstack-ironic-discoverd
```

12.3. 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。

12.3.1. オーケストレーション

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、初期のオーケストレーション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。

12.3.2. 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** の欄を確認します。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
 - **Provision State** 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 State** 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ってください。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、ログを使用して説明します。

```
$ 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。

12.3.3. 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。

手順12.4 デプロイメント後の設定の問題の診断

1. 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。

2. 問題のあるリソースを表示します。

```
$ heat resource-show overcloud [FAILED RESOURCE]
```

resource_status_reason の情報で診断に役立つ可能性のあるものを確認します。

3. **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
```

4. **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
```

5. 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/***

6. 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
```

12.4. プロビジョニングネットワーク上での IP アドレスの競合の回避

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。

手順12.5 アクティブな IP アドレスの特定

1. **nmap** をインストールします。

```
# yum install nmap
```

2. **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
```

3. **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
```

12.5. "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 により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が検出できなかったことを意味します。さらに通常、Nova が検出するはずのリソースと、Ironic が Nova に通知するリソースに差異が生じていることも意味します。

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。
6. 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
```

12.6. 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。

12.6.1. 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、「[オーバークラウドの作成のトラブルシューティング](#)」に記載したアドバイスに従います。これら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

List all events related to the **overcloud** スタック内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。

12.6.2. 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 Enterprise Linux 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、それぞれのコンポーネントのログを `/var/log/` で確認します。

12.6.3. Compute Service のエラー

Compute ノードは、OpenStack Nova Compute Service 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、コンピュートノードのメインの診断が、このサービスを中心に回っ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プライマリーログファイルは `/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、既存の仮想マシン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。

12.6.4. Ceph Storage サービスのエラー

Red Hat Ceph Storage クラスターで発生した問題については、『Red Hat Ceph Storage Configuration Guide』の「[Part X. 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。

12.7. 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。

- OpenStack 認証サービス (**keystone**) は、トークンベースのシステムを使用して、他の OpenStack サービスにアクセスします。一定の期間が経過すると、データベースは未使用のトークンを多数累積します。データベース内のトークンテーブルをフラッシュする cron ジョブを作成することを推奨します。たとえば、毎日午前 4 時にトークンテーブルをフラッシュするには、以下のように設定します。

```
0 04 * * * /bin/keystone-manage token_flush
```

- 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

データベースのトランザクションが、行のロック待ちを中断するまでの時間 (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 エンジンを実行します。

```
$ sudo systemctl restart openstack-heat-engine
```

12.8. 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12.1 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	/var/log/nova/* /var/log/heat/* /var/log/ironic/*
イントロスペクション	アンダークラウド	/var/log/ironic/* /var/log/ironic-discoverd/*
プロビジョニング	アンダークラウド	/var/log/ironic/*
cloud-init ログ	オーバークラウド	/var/log/cloud-init.log
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	/var/lib/puppet/state/last_run_summary.yaml
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	/var/lib/puppet/state/last_run_report.yaml
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	/var/lib/puppet/reports/overcloud-*/*

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 コンポーネント

本項には、director が使用するコンポーネント一覧が含まれています。

共有ライブラリー

diskimage-builder

diskimage-builder は、イメージ構築ツールです。

dib-utils

dib-utils には、**diskimage-builder** が使用するツールが含まれています。

os-collect-config, os-refresh-config, os-apply-config, os-net-config

インスタンスの設定に使用するツールスイートです。

tripleo-image-elements

tripleo-image-elements は、さまざまなソフトウェアコンポーネントをインストールするための **diskimage-builder** スタイル要素のリポジトリです。

インストーラー

instack

instack は、現在のシステム上で **diskimage-builder** スタイル要素を実行します。これにより、**diskimage-builder** がイメージビルドに適用するのと同じ方法で、現在実行中のシステムに要素を適用することができます。

instack-undercloud

instack-undercloud は、**instack** をベースにしたアンダークラウドインストーラーです。

ノード管理

ironic

OpenStack Ironic プロジェクトは、Bare Metal インスタンスのプロビジョニングと管理を行う役割を果たします。

ironic-discoverd

ironic-discoverd は、新しく登録されたノードのハードウェアプロパティを検出します。

デプロイメントプラン

tuskar

OpenStack Tuskar プロジェクトは、デプロイメントのプランニングを行う役割を果たします。

デプロイメントおよびオーケストレーション

heat

OpenStack Heat プロジェクトは、オーケストレーションツールを提供します。このツールは、OpenStack 環境のリソースを記述する YAML ファイルを読み込み、それらのリソースを希望の状態に設定します。

heat-templates

openstack-heat-templates リポジトリには、Heat を使用して Puppet 設定のディスクイメージを作成するためのイメージ要素が追加で含まれます。

tripleo-heat-templates

openstack-tripleo-heat-templates リポジトリには、OpenStack 環境を定義する Heat Orchestration テンプレートの YAML ファイルおよび Puppet マニフェストが含まれています。これらのテンプレートは Tuskar によって処理され、Heat によって実際の環境が作成されます。

puppet-modules

OpenStack Puppet モジュールは、**tripleo-heat-templates** で OpenStack 環境を設定するのに使用します。

tripleo-puppet-elements

tripleo-puppet-elements は、director が Red Hat Enterprise Linux OpenStack Platform をインストールする際に使用するディスクイメージの内容を記述します。

ユーザーインターフェース

tuskar-ui

OpenStack をインストール/管理するための GUI を提供します。Horizon Dashboard のプラグインとして実装されます。

tuskar-ui-extras

tuskar-ui の GUI 拡張機能を提供します。Horizon Dashboard のプラグインとして実装されます。

python-openstackclient

python-openstackclient は、複数の OpenStack サービスやクライアントを管理する CLI ツールです。

python-rdomanager-oscplugin

python-rdomanager-oscplugin は、**python-openstackclient** に組み込まれた CLI ツールで、**instack** インストールと初期設定に関連する機能を提供します。

付録B 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/TLS 証明書を使用する場合には、その証明書には特定の設定をして使用する必要があります。

認証局の作成

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

たとえば、キーと証明書のペアを生成して、認証局として機能するようにします。

```
$ openssl genrsa -out ca.key.pem 4096
$ openssl req -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -
out ca.crt.pem
```

openssl req コマンドは、認証局に関する特定の情報を要求します。それらの情報を指定してください。

これにより、**ca.crt.pem** という名前の証明書ファイルが作成されます。Red Hat Openstack Platform 環境にアクセスする予定の各クライアントにこのファイルをコピーしてから、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
```

SSL/TLS 証明書の作成

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用の署名済み証明書を作成します。

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 = 192.168.0.1
DNS.2 = instack.localdomain
DNS.3 = vip.localdomain
```

重要

commonName_default をパブリック API の IP アドレスに設定します。

- アンダークラウドでは、**undercloud.conf** の **undercloud_public_vip** パラメーターを使用します。
- 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。

alt_names セクションの IP エントリーおよび DNS エントリーとして、同じパブリック API の IP アドレスを追加します。DNS も使用する場合は、同じセクションに DNS エントリーとしてそのサーバーのホスト名を追加します。**openssl.cnf** の詳しい情報は **man openssl.cnf** を実行してください。

以下のコマンドを実行して、キー (**server.key.pem**)、証明書の署名要求 (**server.csr.pem**)、および署名済みの証明書 (**server.crt.pem**) を生成します。

```
$ openssl genrsa -out server.key.pem 2048
$ openssl req -config openssl.cnf -key server.key.pem -new -out
server.csr.pem
$ sudo openssl ca -config openssl.cnf -extensions v3_req -days 3650 -in
server.csr.pem -out server.crt.pem -cert ca.cert.pem
```

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。

このキーペアを使用して、アンダークラウドまたはオーバークラウドのいずれかの SSL/TTL 証明書を作成します。

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を作成します。


```
$ cat server.crt.pem server.key.pem > undercloud.pem
```

これにより、**undercloud_service_certificate** オプションに使用する **undercloud.pem** が作成されます。このファイルは、HAProxy ツールが読み取ることができるように、特別な SELinux コンテキストが必要です。以下の例をガイドとして利用してください。

```
$ 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
```

アンダークラウドの信頼済みの認証局 (CA) 一覧に証明書を追加します。

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

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
```

「[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。

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](#)」の **enable-tls.yaml** ファイルと合わせてこの証明書を使用します。

付録C 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。「[基本オーバークラウドへのノードの登録](#)」または「[高度なオーバークラウドのノード登録](#)」には、以下の電源管理設定を使用します。

C.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

このオプションを **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

C.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

このオプションを **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

補注

- **/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 年度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。

C.3. IBOOT

Dataprobe の iBoot は、システムのリモート電源管理機能を提供する電源ユニットです。

pm_type

このオプションを **pxe_iboot** に設定します。

pm_user, pm_password

iBoot のユーザー名およびパスワード

pm_addr

iBoot インターフェースの IP アドレス

pm_relay_id (オプション)

ホストの iBoot リレー ID。デフォルトは 1 です。

pm_port (オプション)

iBoot ポート。デフォルトは 9100 です。

補注

- **/etc/ironic/ironic.conf** ファイルを編集して、**enabled_drivers** オプションに **pxe_iboot** を追加し、このドライバーを有効化します。

C.4. 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

このオプションを **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"
```

補注

- `/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。
- director には、UCS 用に追加のユーティリティーセットも必要です。`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
```

C.5. 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 `scc` です。デフォルトは `ipmitool` です。

補注

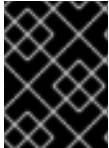
- `/etc/ironic/ironic.conf` ファイルを編集して、`enabled_drivers` オプションに `pxe_irmc` を追加し、このドライバーを有効化します。

- 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ーセットも必要です。**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
```

C.6. SSH および 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 Enterprise Linux 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

このオプションを **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

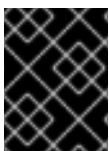
virsh ホストの IP アドレス

補注

- 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。

C.7. 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、イントロスペ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 Enterprise Linux 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

このオプションを **fake_pxe** に設定します。

補注

- 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。
- 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D AUTOMATED HEALTH CHECK (AHC) ツールのパラメーター

以下の表では、AHC ツールのポリシーで使用可能な各種パラメーターのリファレンスを紹介しています。

D.1. ハードドライブ

AHC ツールは、以下からのディスクのプロパティをレポートします。

1. 通常の SATA コントローラーまたは RAID コントローラーからの論理ドライブ
2. Hewlett Packard RAID コントローラーにアタッチされたディスク

表D.1 通常の SATA コントローラーのパラメーター

値	説明	サンプル設定	識別レベル
size	ディスクのサイズ	('disk', 'sda', 'size', '899')	Medium
vendor	ディスクのベンダー	('disk', 'sda', 'vendor', 'HP')	Medium
model	ディスクのモデル	('disk', 'sda', 'model', 'LOGICAL VOLUME')	High
rev	ディスクのファームウェアのバージョン	('disk', 'sda', 'rev', '3.42')	Medium
WCE	書き込みキャッシュの有効化 (Write Cache Enabled)	('disk', 'sda', 'WCE', '1')	Low
RCD	読み取りキャッシュの無効化 (Read Cache Disabled)	('disk', 'sda', 'RCD', '1')	Low

表D.2 Hewlett Packard RAID コントローラーのパラメーター

値	説明	サンプル設定	識別レベル
size	Raw ディスクのサイズ	('disk', '1l:1:1', 'size', '300')	Medium
type	Raw ディスクの種別	('disk', '1l:1:1', 'type', 'SAS')	Low
slot	Raw ディスクのスロット id	('disk', '1l:1:1', 'slot', '0')	Medium

D.2. システム

製品情報は、ホストの DMI 構造により提供されます。この情報は必ずしもハードウェアメーカーが提供するわけではありません。

表D.3 システムの製品パラメーター

値	説明	サンプル設定	識別レベル
serial	ハードウェアのシリアル番号	('system', 'product', 'serial', 'XXXXXX')	Unique*
name	製品名	('system', 'product', 'name', 'ProLiant DL360p Gen8 (654081-B21)')	High
vendor	ベンダー名	('system', 'product', 'vendor', 'HP')	Medium

表D.4 システムの IPMI パラメーター

値	説明	サンプル設定	識別レベル
ipmi	IPMI チャンネル番号	('system', 'ipmi', 'channel', 2)	Low
ipmi-fake	テスト用のフェイク IPMI インターフェース	('system', 'ipmi-fake', 'channel', '0')	Low

D.3. ファームウェア

ファームウェア情報は、ホストの DMI 構造により提供されます。この情報は必ずしもハードウェアメーカーが提供するわけではありません。

表D.5 ファームウェアのパラメーター

値	説明	サンプル設定	識別レベル
version	BIOS のバージョン	('firmware', 'bios', 'version', 'G1ET73WW (2.09)')	Medium
date	BIOS のリリース日	('firmware', 'bios', 'date', '10/19/2012')	Medium
vendor	ベンダー	('firmware', 'bios', 'vendor', 'LENOVO')	Low

D.4. ネットワーク

表D.6 ネットワークパラメーター

値	説明	サンプル設定	識別レベル
serial	MAC アドレス	('network', 'eth0', 'serial', 'd8:9d:67:1b:07:e4')	Unique
vendor	NIC のベンダー	('network', 'eth0', 'vendor', 'Broadcom Corporation')	Low
product	NIC の説明	('network', 'eth0', 'product', 'NetXtreme BCM5719 Gigabit Ethernet PCIe')	Medium
size	リンクの許容サイズ (ビット/秒)	('network', 'eth0', 'size', '1000000000')	Low
ipv4	IPv4 アドレス	('network', 'eth0', 'ipv4', '10.66.6.136')	High
ipv4-netmask	IPv4 ネットマスク	('network', 'eth0', 'ipv4-netmask', '255.255.255.0')	Low
ipv4-cidr	IPv4 の CIDR	('network', 'eth0', 'ipv4-cidr', '24')	Low
ipv4-network	IPv4 ネットワークアドレス	('network', 'eth0', 'ipv4-network', '10.66.6.0')	Medium
link	物理的なリンクのステータス	('network', 'eth0', 'link', 'yes')	Medium
driver	NIC のドライバー名	('network', 'eth0', 'driver', 'tg3')	Low
duplex	NIC のデュプレックス種別	('network', 'eth0', 'duplex', 'full')	Low
speed	NIC の現在のリンク速度	('network', 'eth0', 'speed', '10Mbit/s')	Medium
latency	ネットワークデバイスの PCI レイテンシー	('network', 'eth0', 'latency', '0')	Low
autonegotiation	NIC のオートネゴシエーション	('network', 'eth0', 'autonegotiation', 'on')	Low

D.5. CPU

表D.7 CPU パラメーター別

値	説明	サンプル設定	識別レベル
physid	CPU の物理 ID	('cpu', 'physical_0', 'physid', '1')	Low
cores	CPU コア数	('cpu', 'physical_0', 'cores', '2')	Medium
enabled_cores	CPU の有効なコア数	('cpu', 'physical_0', 'enabled_cores', '2')	Medium
threads	CPU のスレッド数	('cpu', 'physical_0', 'threads', '4')	Medium
product	CPU の識別文字列	('cpu', 'physical_0', 'product', 'Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz')	High
vendor	CPU ベンダー	('cpu', 'physical_0', 'vendor', 'Intel Corp.')	Low
frequency	CPU 内部周波数 (Hz)	('cpu', 'physical_0', 'frequency', '1200000000')	Low
clock	CPU のクロック周波数 (Hz)	('cpu', 'physical_0', 'clock', '100000000')	Low

表D.8 CPU 累計パラメーター

値	説明	サンプル設定	識別レベル
number (physical)	物理 CPU の数	('cpu', 'physical', 'number', 2)	Medium
number (logical)	論理 CPU の数	('cpu', 'logical', 'number', '8')	Medium

D.6. メモリー

メモリー情報は、ホストの DMI 構造により提供されます。この情報は必ずしもハードウェアメーカーが提供するわけではありません。

表D.9 メモリーパラメーター

値	説明	サンプル設定	識別レベル
total	ホストのメモリー容量 (バイト)	('memory', 'total', 'size', '17179869184')	High

値	説明	サンプル設定	識別レベル
size	バンクサイズ (バイト)	('memory', 'bank:0', 'size', '4294967296')	Medium
clock	メモリーのクロック速度 (Hz)	('memory', 'bank:0', 'clock', '667000000')	Low
description	メモリーの詳細	('memory', 'bank:0', 'description', 'FB-DIMM DDR2 FB-DIMM Synchronous 667 MHz (1.5 ns)')	Medium
vendor	メモリーのベンダー	('memory', 'bank:0', 'vendor', 'Nanya Technology')	Medium
serial	メモリーのシリアル番号	('memory', 'bank:0', 'serial', 'C7590943')	Unique*
slot	このバンクの物理スロット	('memory', 'bank:0', 'slot', 'DIMM1')	High
banks	メモリーのバンク数	('memory', 'banks', 'count', 8)	Medium

D.7. INFINIBAND

表D.10 Infiniband ベースのカードのパラメーター

値	説明	サンプル設定	識別レベル
card_type	カードの種別	('infiniband', 'card0', 'card_type', 'mlx4_0')	Medium
device_type	カードのデバイス種別	('infiniband', 'card0', 'device_type', 'MT4099')	Medium
fw_version	カードのファームウェアバージョン	('infiniband', 'card0', 'fw_version', '2.11.500')	High
hw_version	カードのハードウェアバージョン	('infiniband', 'card0', 'hw_version', '0')	Low
nb_ports	ポート数	('infiniband', 'card0', 'nb_ports', '2')	Low
sys_guid	カードのグローバル一意識別子	('infiniband', 'card0', 'sys_guid', '0x0002c90300ea7183')	Unique

値	説明	サンプル設定	識別レベル
node_guid	ノードのグローバル一意識別子	('infiniband', 'card0', 'node_guid', '0x0002c90300ea7180')	Unique

表D.11 Infiniabnd ベースのポートのパラメーター

値	説明	サンプル設定	識別レベル
state	インターフェースの状態	('infiniband', 'card0_port1', 'state', 'Down')	High
physical_state	リンクの物理的な状態	('infiniband', 'card0_port1', 'physical_state', 'Down')	High
rate	速度 (Gbit/秒)	('infiniband', 'card0_port1', 'rate', '40')	High
base_lid	ポートのベースローカル ID	('infiniband', 'card0_port1', 'base_lid', '0')	Low
lmc	ローカル ID のマスク数	('infiniband', 'card0_port1', 'lmc', '0')	Low
sm_lid	サブネットマネージャーのローカル ID	('infiniband', 'card0_port1', 'sm_lid', '0')	Low
port_guid	ポートのグローバル一意識別子	('infiniband', 'card0_port1', 'port_guid', '0x0002c90300ea7181')	Unique

付録E ネットワークインターフェースのパラメーター

以下の表では、ネットワークインターフェース種別に対する Heat テンプレートのパラメーターを定義します。

表E.1 インターフェースオプション

オプション	デフォルト	説明
名前		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。

表E.2 VLAN オプション

オプション	デフォルト	説明
vlan_id		VLAN ID
device		VLAN をアタッチする 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 をアタッチします。
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)

オプション	デフォルト	説明
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。

表E.3 OVS ボンディングオプション

オプション	デフォルト	説明
名前		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。

表E.4 OVS ブリッジオプション

オプション	デフォルト	説明
名前		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。

オプション	デフォルト	説明
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。

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、全く同じである必要はありません。たとえば、あるホストに **em1** と **em2** のインターフェースが指定されており、別のホストには **eno1** と **eno2** が指定されていても、両ホストの NIC は **nic1** および **nic2** として参照することができます。

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

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

番号付きの NIC スキームは、ライブのインターフェース (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。

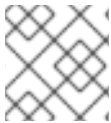
F.2. ルートおよびデフォルトルートの設定

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ことになります。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、変更する必要があるのはコントローラーのテンプレートだけです。反対に、ストレージネットワークは全ロールにアタッチされているため、ストレージネットワークがデフォルトの VLAN の場合には、全ロールを変更する必要があります。

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、デフォルトで 1500 に指定されているスイッチが多いです。

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	
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

パラメーター	説明	例
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ネットワークとサブネット	ControlPlaneSubnetCidr: 192.0.2.0/24
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"]
NeutronExternalNetworkBridge	外部ネットワークに使用するブリッジを定義します。ブリッジ br-ex 上のネイティブ VLAN で Floating IP を使用する場合は "br-ex" と設定します。	NeutronExternalNetworkBridge: "br-ex"
BondInterfaceOvsOptions	ボンディングインターフェースのオプション	bond_mode=balance-tcp lacp=active other-config:lacp-fallback-ab=true"

付録H ボンディングオプション

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 を介してネットワークを提供します。「高度なオーバークラウドの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オプションを使用して、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"
```



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。

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。また、LACP を Linux ボンディングで使用することは可能です。

表H.1 ボンディングオプション

bond_mode=balance-tcp	<p>このモードは、レイヤー 2 とレイヤー 4 のデータを考慮して、ロードバランシングを実行します。たとえば、宛先の MAC アドレス、IP アドレス、および TCP ポート、また balance-tcp には、スイッチ上で LACP が設定されている必要があります。このモードは、Linux ボンディングドライバで使用するモード 4 のボンディングと同様です。LACP はリンクの障害を検出するための高度な耐障害性と、ボンディングについての追加の診断情報を提供するので、可能な場合には、balance-tcp を使用することを推奨されます。</p> <p>LACP に balance-tcp を設定するのは、推奨オプションですが、物理スイッチと LACP をネゴシエーションできない場合には active-backup にフォールバックします。</p>
bond_mode=balance-slb	<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。balance-slb とのボンディングにより、リモートスイッチについての知識や協力なしに限定された形態のロードバランシングが可能となります。SLB は送信元 MAC アドレスと VLAN の各ペアをリンクに割り当て、そのリンクを介して、対象の MAC アドレスと LAN からのパケットをすべて伝送します。このモードはトラフィックパターンの変化に応じて定期的にリバランスを行う、送信元 MAC アドレスと VLAN の番号に基づいた、簡単なハッシュアルゴリズムを使用します。これは、Linux ボンディングドライバで使用されているモード 2 のボンディングと同様で、スイッチはボンディングで設定されているが、LACP (動的なボンディングではなく静的なボンディング) を使用するように設定されていない場合に使用されます。</p>

bond_mode=active-backup	このモードは、アクティブな接続が失敗した場合にスタンバイの NIC がネットワーク操作を再開するアクティブ/スタンバイ構成のフェイルオーバーを提供します。物理スイッチに提示される MAC アドレスは 1 つのみです。このモードには、特別なスイッチのサポートや設定は必要なく、リンクが別のスイッチに接続された際に機能します。このモードは、ロードバランシングは提供しません。
lacp=[active passive off]	Link Aggregation Control Protocol (LACP) の動作を制御します。LACP をサポートしているのは特定のスイッチのみです。お使いのスイッチが LACP に対応していない場合には bond_mode=balance-slb または bond_mode=active-backup を使用してください。
other_config: lacp-fallback-ab=true	フォールバックとして bond_mode=active-backup に切り替わるように LACP の動作を設定します。
other_config: lacp-time=[fast slow]	LACP のハートビートを 1 秒 (高速) または 30 秒 (低速) に設定します。デフォルトは低速です。
other_config: bond-detect-mode=[miimon carrier]	リンク検出に miimon ハートビート (miimon) またはモニターキャリア (carrier) を設定します。デフォルトは carrier です。
other_config: bond-miimon-interval=100	miimon を使用する場合には、ハートビートの間隔をミリ秒単位で設定します。
other_config: bond_updelay=1000	アクティブ化してフラッピングを防ぐためにリンクが Up の状態である必要のある時間 (ミリ秒単位)
other_config: bond-rebalance-interval=10000	ボンディングメンバー間のリバランシングフローの間隔 (ミリ秒単位)。無効にするにはゼロに設定します。

重要

Linux のボンディングをプロバイダーネットワークと併用してパケットのドロップやパフォーマンス上の問題が発生した場合には、スレーブ/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。

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。ポートフラッピングが発生したり、接続が切れたりする可能性があります。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が原因です。

付録I デプロイメントパラメーター

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I.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 /usr/share/openstack-tripleo-heat-templates/ を使用します。	~/templates/my-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トノード数	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

パラメーター	説明	例
--neutron-flat-networks [NEUTRON_FLAT_NETWORKS]	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「datacenter」に設定されています。	datacentre
--neutron-physical-bridge [NEUTRON_PHYSICAL_BRIDGE]	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、通常この値の変更は推奨していません。	br-ex
--neutron-bridge-mappings [NEUTRON_BRIDGE_MAPPINGS]	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 (datacenter) 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]	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
--hypervisor-neutron-public-interface [HYPERVISOR_NEUTRON_PUBLIC_INTERFACE]	各ハイパーバイザーでブリッジに追加するインターフェース	nic1、eth0
--neutron-network-type [NEUTRON_NETWORK_TYPE]	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]	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]	テナントネットワークを割り当てるに使用できる GRE トンネリングの ID 範囲	1:1000
--neutron-vni-ranges [NEUTRON_VNI_RANGES]	テナントネットワークを割り当てるに使用できる VXLAN VNI の ID 範囲	1:1000
--neutron-disable-tunneling	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。	

パラメーター	説明	例
<code>--neutron-network-vlan-ranges [NEUTRON_NETWORK_VLAN_RANGES]</code>	サポートされる Neutron ML2 および Open vSwitch VLAN マッピングの範囲。デフォルトでは、物理ネットワーク「datacentre」上の VLAN を許可するように設定されています。	datacentre:1:1000
<code>--neutron-mechanism-drivers [NEUTRON_MECHANISM_DRIVERS]</code>	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'
<code>--libvirt-type [LIBVIRT_TYPE]</code>	ハイパーバイザーに使用する仮想化タイプ	kvm、qemu
<code>--ntp-server [NTP_SERVER]</code>	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: <code>--ntp-server 0.centos.pool.org,1.centos.pool.org</code>)。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>--cinder-lvm</code>	Cinder ストレージには LVM iSCSI ドライバーを使用します。	
<code>--tripleo-root [TRIPLEO_ROOT]</code>	director の設定ファイルを保存するディレクトリー。デフォルトのままにします。	
<code>--nodes-json [NODES_JSON]</code>	ノード登録に使用するオリジナルの JSON ファイル。オーバークラウドの作成後には、director によりこのファイルが変更されます。デフォルトは、instackenv.json です。	
<code>--no-proxy [NO_PROXY]</code>	環境変数 no_proxy 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。	

パラメーター	説明	例
-O [OUTPUT DIR], --output-dir [OUTPUT DIR]	Tuskar テンプレートファイルを書き込むディレクトリー。存在しない場合は作成されます。ディレクトリーが指定されていない場合は、一時ディレクトリーが使用されます。	~/templates/plan-templates
-e [EXTRA HEAT TEMPLATE], --extra-template [EXTRA HEAT TEMPLATE]	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。 openstack overcloud deploy 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。	-e ~/templates/my-config.yaml
--validation-errors-fatal	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。	
--validation-warnings-fatal	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。	
--rhel-reg	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。	
--reg-method	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 satellite 、カスタマーポータルは portal
--reg-org [REG_ORG]	登録に使用する組織	
--reg-force	すでに登録済みの場合でもシステムを登録します。	

パラメーター	説明	例
<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、<code>https://satellite.example.com</code> ではなく <code>http://satellite.example.com</code> 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 5 サーバーの場合は、オーバークラウドは <code>katello-ca-consumer-latest.noarch.rpm</code> ファイルを取得して <code>subscription-manager</code> に登録し、<code>katello-agent</code> をインストールします。Red Hat Satellite 6 サーバーの場合はオーバークラウドは <code>RHN-ORG-TRUSTED-SSL-CERT</code> ファイルを取得して <code>rhnreg_ks</code> 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	

付録J 改訂履歴

改訂 7.3-18.1 翻訳ファイルを XML ソースバージョン 7.3-18 と同期	Sun Oct 15 2017	Red Hat Localization Services
改訂 7.3-18 単一のブリッジ上の複数のボンディングについての注記を追加	Thu Jun 15 2017	Dan Macpherson
改訂 7.3-17 stack update コマンドについて強調	Thu Mar 30 2017	Dan Macpherson
改訂 7.3-16 別の OSD マッピングを修正	Wed Sep 21 2016	Dan Macpherson
改訂 7.3-15 OSD レイアウトに対する修正	Wed Sep 21 2016	Dan Macpherson
改訂 7.3-14 アンダークラウドのディスクスペースについての注記を追加	Mon Aug 22 2016	Dan Macpherson
改訂 7.3-13 フェンシングのセクションが明確になるように修正	Tue Aug 16 2016	Dan Macpherson
改訂 7.3-12 OSP 8 の『director のインストールと使用方法』ガイドから、スケーリングの章をバックポート	Thu Aug 4 2016	Dan Macpherson
改訂 7.3-11 SAN サポートについての注記を追加	Thu Jun 16 2016	Dan Macpherson
改訂 7.3-10 ノードの UEFI ブートモードサポートについての注記を追加	Fri May 27 2016	Dan Macpherson
改訂 7.3-9 マイナーな更新	Tue Apr 26 2016	Dan Macpherson
改訂 7.3-8 削除防止についてのセクションを追加	Fri Apr 8 2016	Dan Macpherson
改訂 7.3-7 一部の Ceph コマンドに含まれていなかった sudo アクセスを追加	Fri Apr 8 2016	Dan Macpherson
改訂 7.3-6 テストビルド	Tue Apr 5 2016	Dan Macpherson
改訂 7.3-5 Ceph Storage の置き換え手順を追加	Tue Apr 5 2016	Dan Macpherson
改訂 7.3-4 Satellite のリポジトリ要件を追加	Thu Mar 3 2016	Dan Macpherson
改訂 7.3-3 SSL の DNS リストを若干変更	Wed Mar 2 2016	Dan Macpherson
改訂 7.3-2 ノードのスケーリングについての内容を再構成	Tue Mar 1 2016	Dan Macpherson
改訂 7.3-1	Thu Feb 18 2016	Dan Macpherson

OpenStack Platform director 7.2 についての新たな内容を追加
オーバークラウド向けの SSL/TLS の手順を追加
オーバークラウド向けの Satellite 5 の登録に関する情報を追加
ログの一覧を追加
数カ所を若干修正

改訂 7.2-1	Sun Dec 20 2015	Dan Macpherson
新しい更新手順、登録情報、その他のマイナーな追加事項を含む、OpenStack Platform director 7.2 についての新たな記載内容を追加		
改訂 7.1-14	Wed Dec 16 2015	Dan Macpherson
SELinux コンテキスト用の NFS オプションを追加		
改訂 7.1-13	Tue Dec 15 2015	Dan Macpherson
Puppet のカスタム設定のセクションを若干変更		
改訂 7.1-12	Fri Dec 11 2015	Dan Macpherson
nova migration コマンドを修正		
改訂 7.1-11	Fri Dec 11 2015	Dan Macpherson
LACP の問題についての注記を追加		
改訂 7.1-10	Tue Dec 8 2015	Dan Macpherson
OVS での LACP の使用が非推奨であることについての説明を追加		
改訂 7.1-9	Wed Dec 2 2015	Dan Macpherson
カスタム事前設定の内容を修正		
改訂 7.1-7	Tue Dec 1 2015	Dan Macpherson
ExtraConfig リソースと hiera データパラメーターを追加		
改訂 7.1-6	Mon Nov 30 2015	Dan Macpherson
HA コントローラーノードルーチンを置き換えるための修正を追加		
改訂 7.1-5	Thu Nov 19 2015	Dan Macpherson
OSPd 7y2 についての説明を更新		
改訂 7.1-4	Wed Oct 14 2015	Dan Macpherson
スタックの更新手順を追加 UCS フェンシングリンクを追加		
改訂 7.1-2	Fri Oct 9 2015	Martin Lopes
デプロイメント中のエンタイトルメントの消費についての注記を追加		
改訂 7.1-2	Fri Oct 9 2015	Dan Macpherson
アップグレードのプロセスを変更		
改訂 7.1-1	Fri Oct 9 2015	Dan Macpherson
スタックの更新手順を追加		
改訂 7.1-1	Fri Oct 9 2015	Dan Macpherson
ダウンロードのリンクを修正		
改訂 7.1-0	Thu Oct 8 2015	Dan Macpherson
プロビジョニングネットワーク用の静的IP アドレス 新しい検証メソッド		

改訂 7.0-18 --reg-sat-url パラメーターについての明確な説明を追加	Wed Oct 7 2015	Dan Macpherson
改訂 7.0-17 ネットワークおよびイメージのアップロードについての明確な説明を追加	Tue Oct 6 2015	Dan Macpherson
改訂 7.0-16 プロビジョニングネットワーク内で IP アドレスが重複している場合のトラブルシューティングに関するアドバイスを追加	Tues Oct 6 2015	Martin Lopes
改訂 7.0-15 若干の修正	Fri Oct 2 2015	Dan Macpherson
改訂 7.0-14 AHC ツールの sudo アクセスを追加	Thu Oct 1 2015	Dan Macpherson
改訂 7.0-13 フェンシングデバイスを追加	Mon Sep 28 2015	Dan Macpherson
改訂 7.0-12 admin とパブリックIP の設定を修正 リポジトリとパッケージの競合を回避するための yum-plugin-priorities についての説明を追加 さまざまな箇所にマイナーな修正を行い、改訂	Fri Sep 25 2015	Dan Macpherson
改訂 7.0-11 スケーリングのための新規ノードの登録と設定についての新しい説明を追加 デブロイの後の操作について説明した項で外部ネットワークの作成についての内容を修正 若干の修正	Thu Sep 24 2015	Dan Macpherson
改訂 7.0-10 単一ノードのイントロスペクションと追加ノード登録のコマンドを追加	Fri Sep 18 2015	Dan Macpherson
改訂 7.0-9 オーバークラウド作成後のトラブルシューティングの推奨事項を追加	Fri Sep 11 2015	Dan Macpherson
改訂 7.0-8 pxe_ssh の手順を修正 基本的/高度なシナリオの NeutronExternalNetworkBridge 構文を修正 以前は記載されていなかったベアメタルのフレーバー作成についての内容を追加 SELinux context for SSL 証明書用の正しい SELinux コンテキストを記載 イントロスペクションの進捗状況を確認するコマンドを追加 Ceph のカスタマイズの要件を追加	Tue Sep 8 2015	Dan Macpherson
改訂 7.0-7 GUI のテストシナリオに修正を追加 Tuskar ベースのコンテンツを削除 (テストシナリオを除く)	Mon Aug 24 2015	Dan Macpherson
改訂 7.0-6 イメージのアーカイブを展開する手順を追加 若干の修正	Mon Aug 17 2015	Dan Macpherson
改訂 7.0-5 追加のストレージ環境に関する修正を追加 若干の修正	Mon Aug 10 2015	Dan Macpherson
改訂 7.0-4 オーバークラウドパッケージを更新する際の -i オプションについての内容を修正	Thu Aug 6 2015	Dan Macpherson

改訂 7.0-3**Thu Aug 6 2015****Dan Macpherson**

環境ファイルについての警告を更新
SSL 用の SELinux ルールを修正
オーバークラウドのイメージのリンクを更新

改訂 7.0-2**Wed Aug 5 2015****Dan Macpherson**

並び順を新たに変更してビルド

改訂 7.0-1**Wed May 20 2015****Dan Macpherson**

ドキュメントの初回ドラフト