Red Hat OpenStack Platform 16.1

director のインストールと使用方法

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

Last Updated: 2023-11-07

Red Hat OpenStack Platform 16.1 director のインストールと使用方法

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

OpenStack Team
rhos-docs@redhat.com

法律上の通知

概要

エンタープライズ環境で Red Hat OpenStack Platform director を使用して Red Hat OpenStack Platform 16 をインストールします。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

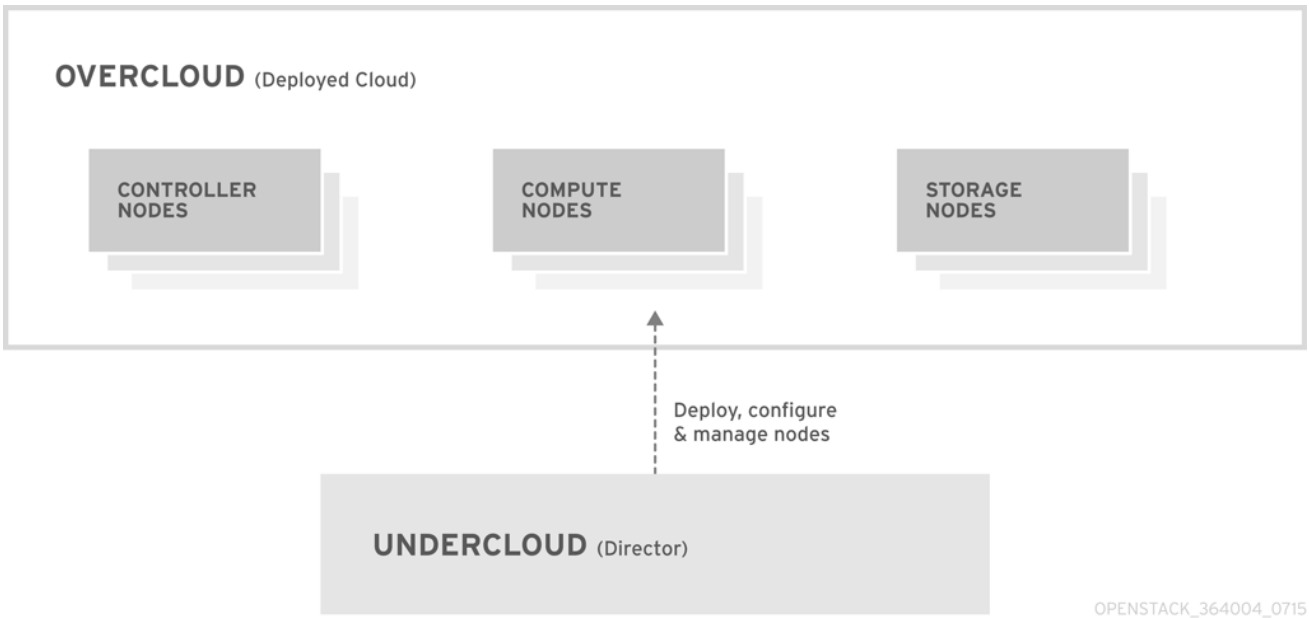
多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ を参照してください。

第1章 DIRECTOR の概要

Red Hat OpenStack Platform (RHOSP) director は、完全な RHOSP 環境のインストールおよび管理を行うためのツールセットです。director は、主に OpenStack プロジェクト TripleO をベースとしています。director により、完全に機能するスリムで堅牢な RHOSP 環境をインストールすることができます。この環境を使用して、OpenStack ノードして使用するベアメタルシステムのプロビジョニングおよび制御を行うことができます。

director は、アンダークラウドとオーバークラウドという 2 つの主要な概念を採用しています。まずアンダークラウドをインストールし、続いてアンダークラウドをツールとして使用してオーバークラウドをインストールおよび設定します。



1.1. アンダークラウドを理解する

アンダークラウドは、Red Hat OpenStack Platform director ツールセットが含まれる主要管理ノードです。OpenStack をインストールした単一システムで、OpenStack 環境 (オーバークラウド) を設定する OpenStack ノードをプロビジョニング/管理する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設定するコンポーネントは、さまざまな機能を持ちます。

環境のプランニング

アンダークラウドには、特定のノードロールを作成して割り当てるのに使用できるプランニング機能が含まれます。アンダークラウドには、Compute、Controller、さまざまな Storage ロールなど、特定のノードに割り当てることのできるデフォルトのノードロールセットが含まれます。また、カスタムロールを設定することもできます。さらに、各ノードロールにどの Red Hat OpenStack Platform サービスを含めるかを選択でき、新しいノード種別をモデル化するか、独自のホストで特定のコンポーネントを分離する方法を提供します。

ベアメタルシステムの制御

アンダークラウドは、各ノードの帯域外管理インターフェイス (通常 Intelligent Platform Management Interface (IPMI)) を使用して電源管理機能を制御し、PXE ベースのサービスを使用してハードウェア属性を検出し、各ノードに OpenStack をインストールします。この機能を使用して、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ことができます。電源管理ドライバーの全リストについては、30章 *電源管理ドライバー*を参照してください。

オーケストレーション

アンダークラウドには、環境のプランのセットに対応する YAML テンプレートセットが含まれます。アンダークラウドは、これらのプランをインポートして、その指示に従い、目的の OpenStack

環境を作成します。このプランに含まれるフックを使用して、環境作成プロセスの特定のポイントとして、カスタマイズを組み込むこともできます。

アンダークラウドのコンポーネント

アンダークラウドは、OpenStack のコンポーネントをベースのツールセットとして使用します。各コンポーネントは、アンダークラウドの個別のコンテナー内で動作します。

- OpenStack Identity (keystone): director コンポーネントの認証および認可
- OpenStack Bare Metal (ironic) および OpenStack Compute (nova): ベアメタルノードの管理
- OpenStack Networking (neutron) および Open vSwitch: ベアメタルノードのネットワークの制御
- OpenStack Image サービス (glance): director がベアメタルマシンに書き込むイメージの格納
- OpenStack Orchestation (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 Workflow サービス (mistral): プランのインポートやデプロイなど、特定の director 固有のアクションに対してワークフローセットを提供します。
- OpenStack Messaging Service (zaqar): OpenStack Workflow サービスのメッセージサービスを提供します。
- OpenStack Object Storage (swift): 以下のさまざまな OpenStack Platform のコンポーネントに対してオブジェクトストレージを提供します。
  - OpenStack Image サービスのイメージストレージ
  - OpenStack Bare Metal のイントロスペクションデータ
  - OpenStack Workflow サービスのデプロイメントプラン

1.2. オーバークラウドについて

オーバークラウドは、アンダークラウドが構築することで得られる Red Hat OpenStack Platform (RHOSP) 環境です。オーバークラウドは、さまざまなロールを持つ複数のノードで設定されます。このノード設定は、希望する OpenStack Platform 環境をベースに定義されます。アンダークラウドには、以下に示すオーバークラウドのデフォルトノードロールセットが含まれます。

コントローラー

コントローラーノードは、OpenStack 環境に管理、ネットワーク、高可用性の機能を提供します。コントローラーノード 3 台で高可用性クラスターを設定する OpenStack 環境が推奨されます。デフォルトのコントローラーノードロールは、以下のコンポーネントをサポートします。これらのサービスがすべてデフォルトで有効化されている訳ではありません。これらのコンポーネントの中には、有効にするのにカスタムの環境ファイルまたは事前にパッケージ化された環境ファイルを必要とするものがあります。

- OpenStack Dashboard (horizon)
- OpenStack Identity (keystone)
- OpenStack Compute (nova) API

- OpenStack Networking (neutron)
- OpenStack Image サービス (glance)
- OpenStack Block Storage (cinder)
- OpenStack Object Storage (swift)
- OpenStack Orchestration (heat)
- OpenStack Shared File Systems (manila)
- OpenStack Bare Metal (ironic)
- OpenStack Load Balancing-as-a-Service (octavia)
- OpenStack Key Manager (barbican)
- MariaDB
- Open vSwitch
- 高可用性サービス向けの Pacemaker および Galera

コンピュート

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

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

ストレージ

ストレージノードは OpenStack 環境にストレージを提供します。以下のリストで、RHOSP のさまざまなストレージノード種別について説明します。

- Ceph Storage ノード: ストレージクラスターを設定するために使用します。それぞれのノードには、Ceph Object Storage Daemon (OSD) が含まれます。また、環境の一部として Ceph Storage ノードをデプロイする場合には、director により Ceph Monitor がコントローラーノードにインストールされます。
- Block Storage (cinder): 高可用性コントローラーノードの外部 Block Storage として使用します。このノードには、以下のコンポーネントが含まれます。
  - OpenStack Block Storage (cinder) ボリューム
  - OpenStack Telemetry エージェント
  - Open vSwitch
- Object Storage (swift): これらのノードは、OpenStack Swift の外部ストレージ層を提供します。コントローラーノードは、Swift プロキシーを介してオブジェクトストレージノード

にアクセスします。オブジェクトストレージノードには、以下のコンポーネントが含まれます。

- OpenStack Object Storage (swift) のストレージ
- OpenStack Telemetry エージェント
- Open vSwitch

1.3. RED HAT OPENSTACK PLATFORM での高可用性について

Red Hat OpenStack Platform (RHOSP) director は、OpenStack Platform 環境に高可用性サービスを提供するためにコントローラーノードクラスターを使用します。それぞれのサービスについて、director はすべてのコントローラーノードに同じコンポーネントをインストールし、コントローラーノードをまとめて単一のサービスとして管理します。このタイプのクラスター設定では、1 つのコントローラーノードが機能しなくなった場合にフォールバックが可能です。これにより、OpenStack のユーザーには一定レベルの運用継続性が提供されます。

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

- Pacemaker: Pacemaker は、クラスターのリソースを管理します。Pacemaker は、クラスター内の全ノードにわたって OpenStack コンポーネントの可用性を管理および監視します。
- HA Proxy: クラスターに負荷分散およびプロキシーサービスを提供します。
- Galera: クラスター全体の RHOSP データベースを複製します。
- Memcached: データベースのキャッシュを提供します。

注記

- バージョン 13 から、director を使用してコンピュートインスタンスの高可用性 (インスタンス HA) をデプロイできるようになりました。インスタンス HA により、コンピュートノードで障害が発生した際にコンピュートノードからインスタンスを自動的に退避させることができます。

1.4. RED HAT OPENSTACK PLATFORM でのコンテナー化について

アンダークラウドおよびオーバークラウド上の各 OpenStack Platform サービスは、対応するノード上の個別の Linux コンテナー内で実行されます。このコンテナー化により、サービスを分離し、環境を維持し、Red Hat OpenStack Platform (RHOSP) をアップグレードすることができます。

Red Hat OpenStack Platform 16.1 では、Red Hat Enterprise Linux 8.2 オペレーティングシステムへのインストールがサポートされます。Red Hat Enterprise Linux 8.2 には Docker が含まれなくなり、Docker エコシステムに代わる新たなツールセットが用意されています。したがって、OpenStack Platform 16.1 では、Docker に代わるこれらの新しいツールにより、OpenStack Platform のデプロイメントおよびアップグレードを行います。

Podman

Pod Manager (Podman) はコンテナー管理用のツールです。このツールには、ほとんどすべての Docker CLI コマンドが実装されています。ただし、Docker Swarm に関連するコマンドは含まれません。Podman は、Pod、コンテナー、およびコンテナーイメージを管理します。Podman と

Docker の主な違いの 1 つは、Podman がバックグラウンドでデーモンを実行せずにリソースを管理できることです。
Podman についての詳しい情報は、Podman の Web サイト を参照してください。

Buildah

Buildah は Open Containers Initiative (OCI) イメージのビルドに特化したツールで、Podman と共に使用します。Buildah コマンドは、Dockerfile の内容と等価です。Buildah は、コンテナーイメージをビルドするための低レベル **coreutils** インターフェイスも提供します。したがって、コンテナーをビルドするのに Dockerfile は必要ありません。また、Buildah は他のスクリプト言語を使用してコンテナーイメージをビルドしますが、その際にデーモンは必要ありません。
Buildah についての詳しい情報は、Buildah の Web サイト を参照してください。

Skopeo

Skopeo により、運用者はリモートコンテナーイメージを検査することができます。これは、director がイメージをプルする際にデータを収集するのに役立ちます。この機能に加えて、コンテナーイメージをあるレジストリーから別のレジストリーにコピーしたり、イメージをレジストリーから削除したりすることもできます。

Red Hat では、オーバークラウド用コンテナーイメージの管理に関して、以下の方法をサポートしています。

- コンテナーイメージを Red Hat Container Catalog からアンダークラウド上の **image-serve** レジストリーにプルし、続いてそのイメージを **image-serve** レジストリーからプルする。イメージをアンダークラウドにプルする際には、複数のオーバークラウドノードが外部接続を通じて同時にコンテナーイメージをプルする状況を避けてください。

- Satellite 6 サーバーからコンテナーイメージをプルする。ネットワークトラフィックは内部になるため、これらのイメージを Satellite から直接プルすることができます。

本ガイドでは、コンテナーイメージレジストリー情報の設定および基本的なコンテナー操作の実施について説明します。

1.5. RED HAT OPENSTACK PLATFORM での CEPH STORAGE の使用

通常、Red Hat OpenStack Platform (RHOSP) を使用する大規模な組織では、数千単位またはそれ以上のクライアントにサービスを提供します。Block Storage リソースの消費に関して、それぞれの OpenStack クライアントは固有のニーズを持つのが一般的です。glance (イメージ)、cinder (ボリューム)、および nova (コンピュート) を単一ノードにデプロイすると、数千単位のクライアントがある大規模なデプロイメントでの管理ができなくなる可能性があります。このような課題は、OpenStack をスケールアウトすることによって解決できます。

ただし、実際には、Red Hat Ceph Storage などのソリューションを活用して、ストレージ層を仮想化する必要もでてきます。ストレージ層の仮想化により、RHOSP 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リング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、大きな 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優れたパフォーマンスを示します。Ceph ブロックデバイスでは、パフォーマンスを強化するために、キャッシュ、Copy On Write クローン、Copy On Read クローンもサポートされています。

Red Hat Ceph Storage についての詳細な情報は、Red Hat Ceph Storage を参照してください。

注記

マルチアーキテクチャークラウドでは、Red Hat は事前にインストール済みの Ceph 実装または外部の Ceph 実装のみをサポートします。詳細は、Integrating an Overcloud with an Existing Red Hat Ceph Cluster と Configuring the CPU architecture for the overcloud を参照してください。

第2章 アンダークラウドのプランニング

アンダークラウドに director を設定してインストールする前に、アンダークラウドホストを計画して、特定の要件を満たしていることを確認する必要があります。

2.1. コンテナー化されたアンダークラウド

アンダークラウドは、最終的な Red Hat OpenStack Platform (RHOSP) 環境 (オーバークラウドと呼ばれる) の設定、インストール、および管理をコントロールするノードです。アンダークラウドは、各 RHOSP コンポーネントサービスをコンテナーとして実行します。アンダークラウドは、これらのコンテナー化されたサービスを使用して、director という名前のツールセットを作成します。このツールセットは、オーバークラウドの作成と管理に使用されます。

アンダークラウドおよびオーバークラウドの両方でコンテナーが使用されているので、どちらも同じアーキテクチャーを使用してコンテナーをプル、設定、および実行します。このアーキテクチャーは、OpenStack Orchestration サービス (heat) をベースにノードをプロビジョニングし、Ansible を使用してサービスおよびコンテナーを設定します。heat および Ansible に関する知識を習得していると、異常発生時のトラブルシューティングに役立ちます。

2.2. アンダークラウドネットワークの準備

アンダークラウドでは、2 つの主要ネットワークへのアクセスが必要です。

- **プロビジョニングまたはコントロールプレーンネットワーク**: director は、このネットワークを使用してノードをプロビジョニングし、Ansible 設定の実行時に SSH 経由でこれらのノードにアクセスします。このネットワークでは、アンダークラウドからオーバークラウドノードへの SSH アクセスも可能です。アンダークラウドには、このネットワーク上の他のノードのイントロスペクションおよびプロビジョニング用 DHCP サービスが含まれます。つまり、このネットワーク上にその他の DHCP サービスは必要ありません。director がこのネットワークのインターフェイスを設定します。

- **External ネットワーク**: このネットワークにより、OpenStack Platform リポジトリー、コンテナーイメージソース、および DNS サーバーや NTP サーバー等の他のサーバーにアクセスすることができます。ご自分のワークステーションからアンダークラウドへの標準アクセスには、このネットワークを使用します。External ネットワークにアクセスするためには、アンダークラウド上でインターフェイスを手動で設定する必要があります。

アンダークラウドには、少なくとも 2 枚の 1 Gbps ネットワークインターフェイスカードが必要です。1 枚は **プロビジョニングまたはコントロールプレーンネットワーク** 用で、残りの 1 枚は **External ネットワーク** 用です。

ネットワークを計画する際には、以下のガイドラインを確認してください。

- Red Hat は、プロビジョニングとコントロールプレーンに 1 つのネットワークを使用し、データプレーンに別のネットワークを使用することを推奨します。OVS ブリッジの上にプロビジョニングおよびコントロールプレーンネットワークを作成しないでください。

- プロビジョニングおよびコントロールプレーンネットワークは、Linux ボンディング上または個々のインターフェイスで設定できます。Linux ボンディングを使用する場合は、アクティブバックアップボンディングタイプとして設定します。

    - 非コントローラーノードでは、プロビジョニングおよびコントロールプレーンネットワークのトラフィック量は比較的少なく、高帯域幅や負荷分散は必要ありません。

    - コントローラーでは、プロビジョニングおよびコントロールプレーンネットワークに追加

の帯域幅が必要です。帯域幅が増加する理由は、コントローラーが他のロールで多くのノードにサービスを提供するためです。環境に頻繁に変更を加える場合も、より多くの帯域幅が必要になります。
最高のパフォーマンスを得るには、50 を超えるコンピュートノードを備えたコントローラー (または 4 つを超えるベアメタルノードが同時にプロビジョニングされている場合) は、非コントローラーノードのインターフェイスの 4 ～ 10 倍の帯域幅を備えている必要があります。

- 50 を超えるオーバークラウドノードがプロビジョニングされる場合、アンダークラウドはプロビジョニングネットワークへのより高い帯域幅の接続を持つ必要があります。
- ワークステーションから director マシンへのアクセスに使用する NIC を、プロビジョニングまたはコントロールプレーン NIC として使用しないでください。director のインストールでは、プロビジョニング NIC を使用してブリッジが作成され、リモート接続はドロップされます。director システムへリモート接続する場合には、外部 NIC を使用します。
- プロビジョニングネットワークには、環境のサイズに適した IP 範囲が必要です。以下のガイドラインを使用して、この範囲に含めるべき IP アドレスの総数を決定してください。
  - イントロスペクション中は、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ごとに少なくとも 1 つの一時 IP アドレスを含めます。
  - デプロイメント中は、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ごとに少なくとも 1 つの永続的な IP アドレスを含めます。
  - プロビジョニングネットワーク上のオーバークラウド高可用性クラスターの仮想 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に、この範囲にさらに IP アドレスを追加します。
- コントローラーノードのネットワークカードまたはネットワークスイッチの異常がオーバークラウドサービスの可用性を阻害するのを防ぐには、keystone 管理エンドポイントがボンディングされたネットワークカードまたはネットワークハードウェアの冗長性を使用するネットワークに配置されるようにしてください。keystone エンドポイントを **internal_api** などの別のネットワークに移動する場合は、アンダークラウドが VLAN またはサブネットに到達できるようにします。詳細は、Red Hat ナレッジベースのソリューション Keystone Admin Endpoint を internal_api network に移行する方法 を参照してください。

2.3. 環境規模の判断

アンダークラウドをインストールする前に、環境の規模を判断します。環境をプランニングする際には、以下の要素を考慮してください。

オーバークラウドにデプロイするノードの数

アンダークラウドは、オーバークラウド内の各ノードを管理します。オーバークラウドノードのプロビジョニングには、アンダークラウドのリソースが使用されます。アンダークラウドには、すべてのオーバークラウドノードを適切にプロビジョニングし管理するのに十分なリソースを提供する必要があります。

アンダークラウドで実行する同時操作の数

アンダークラウド上の OpenStack サービスの多くは、ワーカーのセットを使用します。それぞれのワーカーは、そのサービスに固有の操作を実行します。複数のワーカーを用いると、同時に操作を実行することができます。アンダークラウドのデフォルトのワーカー数は、アンダークラウドの合計 CPU スレッド数の半分です。ここでは、スレッド数とは CPU コア数にハイパースレッディング

の値を掛けたものを指します。たとえば、アンダークラウドの CPU スレッド数が 16 の場合には、デフォルトでは、director のサービス により 8 つのワーカーが提供されます。デフォルトでは、director のサービスに最小および最大のワーカー数も適用されます。

サービス	最小値	最大値
OpenStack Orchestration (heat)	4	24
その他すべてのサービス	2	12

アンダークラウドの CPU およびメモリーの最低要件を以下に示します。

- Intel 64 または AMD64 CPU 拡張機能をサポートする、8 スレッド 64 ビット x86 プロセッサー。これにより、各アンダークラウドサービスに 4 つのワーカーが提供されます。
- 最小 24 GB の RAM
    - **ceph-ansible** Playbook は、アンダークラウドがデプロイするホスト 10 台につき 1 GB の常駐セットサイズ (RSS) を消費します。デプロイメントで新規または既存の Ceph クラスターを使用する場合は、それに応じてアンダークラウド用 RAM をプロビジョニングする必要があります。

多数のワーカーを使用するには、以下の推奨事項に従ってアンダークラウドの仮想 CPU 数およびメモリー容量を増やします。

- **最小値:** 1 スレッドあたり 1.5 GB のメモリーを使用します。たとえば、48 スレッドのマシンの場合、heat 用 24 ワーカーおよびその他のサービス用 12 ワーカーを最低限動作させるのに、72 GB の RAM が必要です。
- **推奨値:** 1 スレッドあたり 3 GB のメモリーを使用します。たとえば、48 スレッドのマシンの場合、heat 用 24 ワーカーおよびその他のサービス用 12 ワーカーを推奨される状態で動作させるのに、144 GB の RAM が必要です。

2.4. アンダークラウドのディスクサイズ

アンダークラウドのディスクサイズとして、ルートディスク上に少なくとも **100 GB** の空きディスク領域があることが推奨されます。

- コンテナーイメージ用に 20 GB
- QCOW2 イメージの変換とノードのプロビジョニングプロセスのキャッシュ用に 10 GB
- 一般用途、ログの記録、メトリック、および将来の拡張用に 70 GB 以上

2.5. 仮想化のサポート

Red Hat は、以下のプラットフォームでのみ仮想アンダークラウドをサポートします。

プラットフォーム	説明

プラットフォーム	説明
Kernel-based Virtual Machine (KVM)	認定済みのハイパーバイザーとしてリストされている Red Hat Enterprise Linux 8 でホストされていること
Red Hat Virtualization	認定済みのハイパーバイザーとしてリストされている Red Hat Virtualization 4.x でホストされていること
Microsoft Hyper-V	Red Hat Customer Portal Certification Catalogueに記載の Hyper-V のバージョンでホストされている。
VMware ESX および ESXi	Red Hat Customer Portal Certification Catalogueに記載の ESX および ESXi のバージョンでホストされていること

重要

Red Hat OpenStack Platform director には、ホストオペレーティングシステムとして、最新バージョンの Red Hat Enterprise Linux 8 がインストールされている必要があります。このため、仮想化プラットフォームは下層の Red Hat Enterprise Linux バージョンもサポートする必要があります。

仮想マシンの要件

仮想アンダークラウドのリソース要件は、ベアメタルのアンダークラウドの要件と似ています。ネットワークモデル、ゲスト CPU 機能、ストレージのバックエンド、ストレージのフォーマット、キャッシュモードなどプロビジョニングの際には、さまざまなチューニングオプションを検討してください。

ネットワークの考慮事項

電源管理

アンダークラウド仮想マシン (VM) は、オーバークラウドノードの電源管理デバイスにアクセスする必要があります。これには、ノードの登録の際に、**pm_addr** パラメーターに IP アドレスを設定してください。

プロビジョニングネットワーク

プロビジョニングネットワーク (**ctlplane**) に使用する NIC には、オーバークラウドのベアメタルノードの NIC に対する DHCP 要求をブロードキャストして、対応する機能が必要です。仮想マシンの NIC をベアメタル NIC と同じネットワークに接続するブリッジを作成します。

不明なアドレスからのトラフィックを許可する

アンダークラウドをブロックしているハイパーバイザーが未知のアドレスからトラフィックを送信しないように、仮想アンダークラウドハイパーバイザーを設定する必要があります。設定は、仮想アンダークラウドに使用しているプラットフォームによって異なります。

- Red Hat Enterprise Virtualization: **anti-mac-spoofing** パラメーターを無効にします。
- VMware ESX または ESXi:
  - IPv4 **ctlplane** ネットワーク: 偽造送信を許可します。

- IPv6 **ctlplane** ネットワーク: 偽造送信、MAC アドレスの変更、無差別モード操作を許可します。
  VMware ESX または ESXi を設定する方法の詳細については、VMware ドキュメント Web サイトの vSphere 標準スイッチのセキュリティー保護 を参照してください。

これらの設定を適用したら、director 仮想マシンの電源を一旦オフにしてから再投入する必要があります。仮想マシンをリブートするだけでは不十分です。

2.6. 文字のエンコーディング設定

Red Hat OpenStack Platform には、ロケール設定の一部として特殊文字のエンコーディングに関する要件があります。

- すべてのノードで UTF-8 エンコーディングを使用します。すべてのノードで **LANG** 環境変数を **en_US.UTF-8** に設定するようにします。
- Red Hat OpenStack Platform リソース作成の自動化に Red Hat Ansible Tower を使用している場合は、非 ASCII 文字を使用しないでください。

2.7. プロキシーを使用してアンダークラウドを実行する際の考慮事項

プロキシーを使用してアンダークラウドを実行する場合は特定の制限があります。Red Hat は、レジストリーおよびパッケージ管理に Red Hat Satellite を使用することを推奨します。

ただし、ご自分の環境でプロキシーを使用している場合は、以下の考慮事項を確認して、Red Hat OpenStack Platform の一部とプロキシーを統合する際のさまざまな設定手法、およびそれぞれの手法の制限事項を十分に理解するようにしてください。

システム全体のプロキシー設定

アンダークラウド上のすべてのネットワークトラフィックに対してプロキシー通信を設定するには、この手法を使用します。プロキシー設定を定義するには、**/etc/environment** ファイルを編集して以下の環境変数を設定します。

http_proxy
: 標準の HTTP リクエストに使用するプロキシー

https_proxy
: HTTPs リクエストに使用するプロキシー

no_proxy
: プロキシー通信から除外するドメインのコンマ区切りリスト

システム全体のプロキシー手法には、以下の制限事項があります。

- **no_proxy** 変数は、主にドメイン名 (**www.example.com**)、ドメイン接尾辞 (**example.com**)、およびワイルドカード付きのドメイン (***.example.com**) を使用します。ほとんどの Red Hat OpenStack Platform サービスは **no_proxy** の IP アドレスを解釈しますが、コンテナーのヘルスチェックなどの特定のサービスは、cURL と **wget** による制限のため **no_proxy** 環境変数の IP アドレスを解釈しません。アンダークラウドでシステム全体のプロキシーを使用するには、インストール中に **undercloud.conf** ファイルの **container_healthcheck_disabled** パラメーターを使用してコンテナーヘルスチェックを無効にします。詳細については、BZ#1837458: Container health checks fail to honor no_proxy CIDR notation を参照してください。

- **pam_env** PAM モジュールのバッファーが固定されているため、**no_proxy** の最大長は 1024 文字です。
- 一部のコンテナーでは、**/etc/environments** の環境変数が正しくバインドおよび解析されないため、これらのサービスの実行時に問題が発生します。詳細については、BZ#1916070: proxy configuration updates in /etc/environment files are not being picked up in containers correctly および BZ#1918408: mistral_executor container fails to properly set no_proxy environment parameter を参照してください。

dnf プロキシー設定

すべてのトラフィックがプロキシーを通過するように **dnf** を設定するには、この手法を使用します。プロキシー設定を定義するには、**/etc/dnf/dnf.conf** ファイルを編集して以下のパラメーターを設定します。

proxy
: プロキシーサーバーの URL

proxy_username
: プロキシーサーバーへの接続に使用するユーザー名

proxy_password
: プロキシーサーバーへの接続に使用するパスワード

proxy_auth_method
: プロキシーサーバーが使用する認証方法

これらのオプションの詳細については、**man dnf.conf** を実行してください。

dnf プロキシー手法には、以下の制限事項があります。

- この手法では、**dnf** に対してのみプロキシーがサポートされます。
- **dnf** プロキシー手法には、特定のホストをプロキシー通信から除外するオプションは含まれていません。

Red Hat Subscription Manager プロキシー

すべてのトラフィックがプロキシーを通過するように Red Hat Subscription Manager を設定するには、この手法を使用します。プロキシー設定を定義するには、**/etc/rhsm/rhsm.conf** ファイルを編集して以下のパラメーターを設定します。

proxy_hostname
: プロキシーのホスト

proxy_scheme
: プロキシーをリポジトリー定義に書き出す際のプロキシーのスキーム

proxy_port
: プロキシーのポート

proxy_username
: プロキシーサーバーへの接続に使用するユーザー名

proxy_password
: プロキシーサーバーへの接続に使用するパスワード

no_proxy
: プロキシー通信から除外する特定ホストのホスト名接尾辞のコンマ区切りリスト

これらのオプションの詳細については、**man rhsm.conf** を実行してください。

Red Hat Subscription Manager プロキシー手法には、以下の制限事項があります。

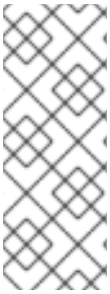
- この手法では、Red Hat Subscription Manager に対してのみプロキシーがサポートされます。
- Red Hat Subscription Manager プロキシー設定の値は、システム全体の環境変数に設定されたすべての値をオーバーライドします。

透過プロキシー

アプリケーション層のトラフィックを管理するのにネットワークで透過プロキシーが使用される場合は、プロキシー管理が自動的に行われるため、アンダークラウド自体をプロキシーと対話するように設定する必要はありません。透過プロキシーは、Red Hat OpenStack Platform のクライアントベースのプロキシー設定に関連する制限に対処するのに役立ちます。

2.8. アンダークラウドのリポジトリー

Red Hat OpenStack Platform (RHOSP) 16.1 は、Red Hat Enterprise Linux 8.2 上で動作します。したがって、これらのリポジトリーからのコンテンツを該当する Red Hat Enterprise Linux バージョンにロックする必要があります。



注記

リポジトリーを Red Hat Satellite と同期する場合は、特定バージョンの Red Hat Enterprise Linux リポジトリーを有効にすることができます。ただし、選択したバージョンに関係なく、リポジトリーラベルは同じままです。たとえば、BaseOS リポジトリーの 8.4 バージョンを有効にした場合、リポジトリー名には有効にした特定のバージョンが含まれますが、リポジトリーラベルは依然として **rhel-8-for-x86_64-baseos-eus-rpms** です。

警告

ここで指定する以外のリポジトリーは、サポートされません。別途推奨されない限り、以下の表に記載されている以外の製品またはリポジトリーを有効にしないでください。有効にすると、パッケージの依存関係の問題が発生する可能性があります。Extra Packages for Enterprise Linux (EPEL) を有効にしないでください。

コアリポジトリー

以下の表には、アンダークラウドをインストールするためのコアリポジトリーをまとめています。

名前	リポジトリー	要件の説明
Red Hat Enterprise Linux 8.4 for x86_64 - BaseOS (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-baseos-tus-rpms**	x86_64 システム用ベースオペレーティングシステムのリポジトリー

名前	リポジトリー	要件の説明
Red Hat Enterprise Linux 9 for x86_64 - AppStream (RPMs)	**rhel-8-for-x86_64-appstream-tus-rpms**	Red Hat OpenStack Platform の依存関係が含まれます。
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-highavailability-tus-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 Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Red Hat Enterprise Linux 用 Ansible エンジン。最新バージョンの Ansible を提供するために使用されます。
Advanced Virtualization for RHEL 8 x86_64 (RPMs)	**advanced-virt-for-rhel-8-x86_64-eus-rpms**	OpenStack Platform 用仮想化パッケージを提供します。
Red Hat Satellite Tools for RHEL 8 Server RPMs x86_64	**satellite-tools-6.5-for-rhel-8-x86_64-rpms**	Red Hat Satellite 6 でのホスト管理ツール
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-x86_64-rpms**	Red Hat OpenStack Platform のコアリポジトリー。Red Hat OpenStack Platform director のパッケージが含まれます。
Red Hat Fast Datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	OpenStack Platform 用 Open vSwitch (OVS) パッケージを提供します。

Ceph リポジトリー

以下の表には、アンダークラウド用の Ceph Storage 関連のリポジトリーをまとめています。

名前	リポジトリー	要件の説明
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	Ceph Storage クラスターと通信するためのノード用のツールを提供します。オーバークラウドで Ceph Storage を使用する場合、または既存の Ceph Storage クラスターと統合する場合、アンダークラウドにはこのリポジトリーからの **ceph-ansible** パッケージが必要です。

IBM POWER 用リポジトリー

次の表には、POWER PC アーキテクチャー上の RHOSP のリポジトリーのリストが含まれています。コアリポジトリーの該当リポジトリーの代わりに、これらのリポジトリーを使用してください。

名前	リポジトリー	要件の説明
Red Hat Enterprise Linux for IBM Power, little endian - BaseOS (RPMs)	**rhel-8-for-ppc64le-baseos-rpms**	ppc64le システム用ベースオペレーティングシステムのリポジトリー
Red Hat Enterprise Linux 8 for IBM Power, little endian - AppStream (RPMs)	**rhel-8-for-ppc64le-appstream-rpms**	Red Hat OpenStack Platform の依存関係が含まれます。
Red Hat Enterprise Linux 8 for IBM Power, little endian - High Availability (RPMs)	**rhel-8-for-ppc64le-highavailability-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 Fast Datapath for RHEL 8 IBM Power, little endian (RPMS)	**fast-datapath-for-rhel-8-ppc64le-rpms**	OpenStack Platform 用 Open vSwitch (OVS) パッケージを提供します。
Red Hat Ansible Engine 2.8 for RHEL 8 IBM Power, little endian (RPMs)	**ansible-2.8-for-rhel-8-ppc64le-rpms**	Red Hat Enterprise Linux 用 Ansible エンジン。最新バージョンの Ansible を提供します。
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-ppc64le-rpms**	ppc64le システム用 Red Hat OpenStack Platform のコアリポジトリー

第3章 DIRECTOR インストールの準備

director をインストールおよび設定するには、アンダークラウドを Red Hat Customer Portal または Red Hat Satellite サーバーに登録し、director パッケージをインストールし、インストール中にコンテナーイメージを取得するために director のコンテナーイメージソースを設定するなどの準備作業を完了する必要があります。

3.1. アンダークラウドの準備

director をインストールする前に、ホストマシンでいくつかの基本設定を完了する必要があります。

手順

1. お使いのアンダークラウドに **root** ユーザーとしてログインします。
2. **stack** ユーザーを作成します。

```
[root@director ~]# useradd stack
```

3. ユーザーのパスワードを設定します。

```
[root@director ~]# passwd stack
```

4. **sudo** を使用する場合にパスワードを要求されないようにします。

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

5. 新規作成した **stack** ユーザーに切り替えます。

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

6. システムイメージおよび heat テンプレート用のディレクトリーを作成します。

```
[stack@director ~]$ mkdir ~/images
[stack@director ~]$ mkdir ~/templates
```

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。ローカルファイルシステムの管理を容易にするために、Red Hat ではこれらのディレクトリーを作成することを推奨します。

7. アンダークラウドのベースおよび完全なホスト名を確認します。

```
[stack@director ~]$ hostname
[stack@director ~]$ hostname -f
```

上記のコマンドのいずれかで正しい完全修飾ホスト名が出力されない、またはエラーが表示される場合には、**hostnamectl** でホスト名を設定します。

```
[stack@director ~]$ sudo hostnamectl set-hostname undercloud.example.com
[stack@director ~]$ sudo hostnamectl set-hostname --transient undercloud.example.com
```

8. アンダークラウドホストの完全修飾ドメイン名 (FQDN) を解決できる DNS サーバーを使用していない場合は、**/etc/hosts** を編集してシステムホスト名のエントリーを追加します。**/etc/hosts** の IP アドレスは、アンダークラウドのパブリック API に使用する予定のアドレスと一致する必要があります。たとえば、システムの FQDN に **undercloud.example.com** が使用され、IP アドレスに **10.0.0.1** を使用する場合には、**/etc/hosts** ファイルに以下の行を追加します。

```
10.0.0.1  undercloud.example.com undercloud
```

9. Red Hat OpenStack Platform director をオーバークラウドまたはその認証プロバイダーとは別のドメインに配置する予定の場合には、追加のドメインを /etc/resolv.conf に加える必要があります。

```
search overcloud.com idp.overcloud.com
```

3.2. アンダークラウドの登録およびサブスクリプションのアタッチ

director をインストールする前に、**subscription-manager** を実行し、アンダークラウドを登録して有効な Red Hat OpenStack Platform サブスクリプションをアタッチする必要があります。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. Red Hat コンテンツ配信ネットワークまたは Red Hat Satellite のどちらかにシステムを登録します。たとえば、システムをコンテンツ配信ネットワークに登録するには、以下のコマンドを実行します。要求されたら、カスタマーポータルのユーザー名およびパスワードを入力します。

```
[stack@director ~]$ sudo subscription-manager register
```

3. Red Hat OpenStack Platform (RHOSP) director のエンタイトルメントプール ID を検索します。

```
[stack@director ~]$ sudo subscription-manager list --available --all --matches="Red Hat
OpenStack"
Subscription Name:   Name of SKU
Provides:            Red Hat Single Sign-On
                     Red Hat Enterprise Linux Workstation
                     Red Hat CloudForms
                     Red Hat OpenStack
                     Red Hat Software Collections (for RHEL Workstation)
                     Red Hat Virtualization
SKU:                 SKU-Number
Contract:            Contract-Number
Pool ID:             Valid-Pool-Number-123456
Provides Management: Yes
Available:           1
Suggested:           1
Service Level:       Support-level
Service Type:        Service-Type
Subscription Type:   Sub-type
Ends:                End-date
System Type:         Physical
```

4. **Pool ID** の値を特定して、Red Hat OpenStack Platform 16.1 のエンタイトルメントをアタッチします。

```
[stack@director ~]$ sudo subscription-manager attach --pool=Valid-Pool-Number-123456
```

5. アンダークラウドを Red Hat Enterprise Linux 8.2 にロックします。

```
$ sudo subscription-manager release --set=8.2
```

3.3. アンダークラウド用リポジトリーの有効化

アンダークラウドに必要なリポジトリーを有効にし、システムパッケージを最新バージョンに更新します。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. デフォルトのリポジトリーをすべて無効にしてから、必要な Red Hat Enterprise Linux リポジトリーを有効にします。

```
[stack@director ~]$ sudo subscription-manager repos --disable=*
[stack@director ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-tus-rpms --enable=rhel-8-for-x86_64-appstream-tus-rpms --enable=rhel-8-for-x86_64-highavailability-tus-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms --enable=openstack-16.1-for-rhel-8-x86_64-rpms --enable=fast-datapath-for-rhel-8-x86_64-rpms --enable=advanced-virt-for-rhel-8-x86_64-eus-rpms
```

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

3. **container-tools** リポジトリーモジュールをバージョン **2.0** に設定します。

```
[stack@director ~]$ sudo dnf module reset container-tools
[stack@director ~]$ sudo dnf module enable -y container-tools:2.0
```

4. 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。

```
[stack@director ~]$ sudo dnf update -y
[stack@director ~]$ sudo reboot
```

3.4. DIRECTOR パッケージのインストール

Red Hat OpenStack Platform director に関連するパッケージをインストールします。

手順

1. director のインストールと設定を行うためのコマンドラインツールをインストールします。

```
[stack@director ~]$ sudo dnf install -y python3-tripleoclient
```

3.5. CEPH-ANSIBLE のインストール

Red Hat OpenStack Platform で Ceph Storage を使用する場合、**ceph-ansible** パッケージが必要です。

手順

1. Ceph Tools リポジトリーを有効にします。

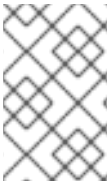
```
[stack@director ~]$ sudo subscription-manager repos --enable=rhceph-4-tools-for-rhel-8-x86_64-rpms
```

2. **ceph-ansible** パッケージをインストールします。

```
[stack@director ~]$ sudo dnf install -y ceph-ansible
```

3.6. コンテナーイメージの準備

アンダークラウドのインストールには、コンテナーイメージの取得先およびその保存方法を定義するための環境ファイルが必要です。この環境ファイルを生成してカスタマイズし、コンテナーイメージの準備に使用します。



注記

アンダークラウド用に特定のコンテナーイメージバージョンを設定する必要がある場合は、イメージを特定のバージョンに固定する必要があります。詳しい情報は、Pinning container images for the undercloud を参照してください。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. デフォルトのコンテナーイメージ準備ファイルを生成します。

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

上記のコマンドでは、以下の追加オプションを使用しています。

- **--local-push-destination**: コンテナーイメージの保管場所として、アンダークラウド上のレジストリーを設定します。つまり、director は必要なイメージを Red Hat Container Catalog からプルし、それをアンダークラウド上のレジストリーにプッシュします。director はこのレジストリーをコンテナーイメージのソースとして使用します。Red Hat Container Catalog から直接プルする場合には、このオプションを省略します。
- **--output-env-file**: 環境ファイルの名前です。このファイルには、コンテナーイメージを準備するためのパラメーターが含まれます。ここでは、ファイル名は **containers-prepare-parameter.yaml** です。

注記

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

3. 要件に合わせて **containers-prepare-parameter.yaml** を変更します。

3.7. コンテナーイメージ準備のパラメーター

コンテナー準備用のデフォルトファイル (**containers-prepare-parameter.yaml**) には、**ContainerImagePrepare** heat パラメーターが含まれます。このパラメーターで、イメージのセットを準備するためのさまざまな設定を定義します。

```
parameter_defaults:
  ContainerImagePrepare:
  - (strategy one)
  - (strategy two)
  - (strategy three)
  ...
```

それぞれの設定では、サブパラメーターのセットにより使用するイメージやイメージの使用方法を定義することができます。以下の表には、**ContainerImagePrepare** ストラテジーの各設定で使用することのできるサブパラメーターの情報をまとめています。

パラメーター	説明
excludes	設定からイメージ名を除外する正規表現のリスト
includes	設定に含める正規表現のリスト。少なくとも 1 つのイメージ名が既存のイメージと一致している必要があります。**includes** パラメーターを指定すると、**excludes** の設定はすべて無視されます。
modify_append_tag	対象となるイメージのタグに追加する文字列。たとえば、16.1.3-5.161 のタグが付けられたイメージをプルし、**modify_append_tag** を **-hotfix** に設定すると、director は最終イメージを 16.1.3-5.161-hotfix とタグ付けします。
modify_only_with_labels	変更するイメージを絞り込むイメージラベルのディクショナリー。イメージが定義したラベルと一致する場合には、director はそのイメージを変更プロセスに含めます。
modify_role	イメージのアップロード中 (ただし目的のレジストリーにプッシュする前) に実行する Ansible ロール名の文字列
modify_vars	**modify_role** に渡す変数のディクショナリー

パラメーター	説明
push_destination	アップロードプロセス中にイメージをプッシュするレジストリーの名前空間を定義します。 ● **true** に設定すると、**push_destination** はホスト名を使用してアンダークラウドレジストリーの名前空間に設定されます。これが推奨される方法です。 ● **false** に設定すると、ローカルレジストリーへのプッシュは実行されず、ノードはソースから直接イメージをプルします。 ● カスタムの値に設定すると、director はイメージを外部のローカルレジストリーにプッシュします。 実稼働環境でこのパラメーターを **false** に設定した場合、イメージを Red Hat Container Catalog から直接プルする際に、すべてのオーバークラウドノードが同時に外部接続を通じて Red Hat Container Catalog からイメージをプルするため、帯域幅の問題が発生する可能性があります。コンテナーイメージをホストする Red Hat Satellite Server から直接プルする場合にのみ、**false** を使用します。 **push_destination** パラメーターが **false** に設定されているか、定義されておらずリモートレジストリーで認証が必要な場合は、**ContainerImageRegistryLogin** パラメーターを **true** に設定し、**ContainerImageRegistryCredentials** パラメーターで認証情報を追加します。
pull_source	元のコンテナーイメージをプルするソースレジストリー
set	初期イメージの取得場所を定義する、**キー: 値** 定義のディクショナリー
tag_from_label	指定したコンテナーイメージのメタデータラベルの値を使用して、すべてのイメージのタグを作成し、そのタグが付けられたイメージをプルします。たとえば、**tag_from_label: {version}-{release}** を設定すると、director は **version** および **release** ラベルを使用して新しいタグを作成します。あるコンテナーについて、**version** を 16.1.3 に設定し、**release** を **5.161** に設定した場合、タグは 16.1.3-5.161 となります。**set** ディクショナリーで **tag** を定義していない場合に限り、director はこのパラメーターを使用します。

重要

イメージをアンダークラウドにプッシュする場合は、**push_destination: UNDERCLOUD_IP:PORT** の代わりに **push_destination: true** を使用します。**push_destination: true** 手法を使用することで、IPv4 アドレスおよび IPv6 アドレスの両方で一貫性が確保されます。

set パラメーターには、複数の **キー: 値** 定義を設定することができます。

キー	説明
ceph_image	Ceph Storage コンテナーイメージの名前
ceph_namespace	Ceph Storage コンテナーイメージの名前空間
ceph_tag	Ceph Storage コンテナーイメージのタグ
ceph_alertmanager_image **ceph_alertmanager_namespace** **ceph_alertmanager_tag**	Ceph Storage Alert Manager コンテナーイメージの名前、namespace、およびタグ。
ceph_grafana_image **ceph_grafana_namespace** **ceph_grafana_tag**	Ceph Storage Grafana コンテナーイメージの名前、namespace、およびタグ。
ceph_node_exporter_image **ceph_node_exporter_namespace** **ceph_node_exporter_tag**	Ceph Storage Node Exporter コンテナーイメージの名前、namespace、およびタグ。
ceph_prometheus_image **ceph_prometheus_namespace** **ceph_prometheus_tag**	Ceph Storage Prometheus コンテナーイメージの名前、namespace、およびタグ。
name_prefix	各 OpenStack サービスイメージの接頭辞
name_suffix	各 OpenStack サービスイメージの接尾辞
namespace	各 OpenStack サービスイメージの名前空間
neutron_driver	使用する OpenStack Networking (neutron) コンテナーを定義するのに使用するドライバー。標準の **neutron-server** コンテナーに設定するには、null 値を使用します。OVN ベースのコンテナーを使用するには、**ovn** に設定します。

キー	説明
tag	ソースからの全イメージに特定のタグを設定します。定義されていない場合は、director は Red Hat OpenStack Platform のバージョン番号をデフォルト値として使用します。このパラメーターは、**tag_from_label** の値よりも優先されます。

注記

コンテナーイメージでは、Red Hat OpenStack Platform のバージョンに基づいたマルチストリームタグが使用されます。したがって、今後 **latest** タグは使用されません。

3.8. コンテナーイメージタグ付けのガイドライン

Red Hat コンテナーレジストリーでは、すべての Red Hat OpenStack Platform コンテナーイメージをタグ付けするのに、特定のバージョン形式が使用されます。この形式は、**version-release** のように各コンテナーのラベルのメタデータに従います。

version

Red Hat OpenStack Platform のメジャーおよびマイナーバージョンに対応します。これらのバージョンは、1 つまたは複数のリリースが含まれるストリームとして機能します。

release

バージョンストリーム内の、特定コンテナーイメージバージョンのリリースに対応します。

たとえば、Red Hat OpenStack Platform の最新バージョンが 16.1.3 で、コンテナーイメージのリリースが **5.161** の場合、コンテナーイメージのタグは 16.1.3-5.161 となります。

Red Hat コンテナーレジストリーでは、コンテナーイメージバージョンの最新リリースとリンクするメジャーおよびマイナー **version** タグのセットも使用されます。たとえば、16.1 と 16.1.3 の両方が、16.1.3 コンテナーストリームの最新 **release** とリンクします。16.1 の新規マイナーリリースが公開されると、16.1 タグは新規マイナーリリースストリームの最新 **release** とリンクします。一方、16.1.3 タグは、引き続き 16.1.3 ストリーム内の最新 **release** とリンクします。

ContainerImagePrepare パラメーターには 2 つのサブパラメーターが含まれ、これを使用してダウンロードするコンテナーイメージを定義することができます。これらのサブパラメーターは、**set** ディクショナリー内の **tag** パラメーターおよび **tag_from_label** パラメーターです。以下のガイドラインを使用して、**tag** または **tag_from_label** のどちらを使用するかを判断してください。

- **tag** のデフォルト値は、お使いの OpenStack Platform のメジャーバージョンです。本バージョンでは、16.1 です。これは常に最新のマイナーバージョンおよびリリースに対応します。

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 16.1
      ...
```

- OpenStack Platform コンテナーイメージの特定マイナーバージョンに変更するには、タグをマイナーバージョンに設定します。たとえば、16.1.2 に変更するには、**tag** を 16.1.2 に設定します。

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 16.1.2
      ...
```

- **tag** を設定すると、インストールおよび更新時に、director は必ず **tag** で設定したバージョンの最新のコンテナーイメージ **release** をダウンロードします。
- **tag** を設定しないと、director は最新のメジャーバージョンと共に **tag_from_label** の値を使用します。

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      # tag: 16.1
      ...
    tag_from_label: '{version}-{release}'
```

- **tag_from_label** パラメーターは、Red Hat コンテナーレジストリーから検査する最新コンテナーイメージリリースのラベルメタデータからタグを生成します。たとえば、特定のコンテナーのラベルは、以下の **version** および **release** メタデータを使用します。

```
"Labels": {
  "release": "5.161",
  "version": "16.1.3",
  ...
}
```

- **tag_from_label** のデフォルト値は **{version}-{release}** です。これは、各コンテナーイメージのバージョンおよびリリースのメタデータラベルに対応します。たとえば、コンテナーイメージの **version** に 16.1.3 が、**release** に 5.161 が、それぞれ設定されている場合、コンテナーイメージのタグは 16.1.3-5.161 となります。
- **tag** パラメーターは、常に **tag_from_label** パラメーターよりも優先されます。**tag_from_label** を使用するには、コンテナー準備の設定で **tag** パラメーターを省略します。
- **tag** および **tag_from_label** の主な相違点は、次のとおりです。director が **tag** を使用してイメージをプルする場合は、Red Hat コンテナーレジストリーがバージョンストリーム内の最新イメージリリースとリンクするメジャーまたはマイナーバージョンのタグだけに基づきます。一方、**tag_from_label** を使用する場合は、director がタグを生成して対応するイメージをプルできるように、各コンテナーイメージのメタデータの検査を行います。

3.9. プライベートレジストリーからのコンテナーイメージの取得

registry.redhat.io レジストリーにアクセスしてイメージをプルするには、認証が必要です。**registry.redhat.io** およびその他のプライベートレジストリーで認証するには、**containers-prepare-parameter.yaml** ファイルに **ContainerImageRegistryCredentials** および **ContainerImageRegistryLogin** パラメーターを含めます。

ContainerImageRegistryCredentials

一部のコンテナーイメージレジストリーでは、イメージにアクセスするのに認証が必要です。そのよう

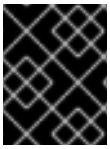
な場合には、**containers-prepare-parameter.yaml** 環境ファイルの **ContainerImageRegistryCredentials** パラメーターを使用します。**ContainerImageRegistryCredentials** パラメーターは、プライベートレジストリーの URL に基づくキーのセットを使用します。それぞれのプライベートレジストリーの URL は、独自のキーと値のペアを使用して、ユーザー名 (キー) およびパスワード (値) を定義します。これにより、複数のプライベートレジストリーに対して認証情報を指定することができます。

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

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

複数のレジストリーの認証情報を指定するには、**ContainerImageRegistryCredentials** でレジストリーごとに複数のキー/ペアの値を設定します。

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
      namespace: registry.redhat.io/...
      ...
  - push_destination: true
    set:
      namespace: registry.internalsite.com/...
      ...
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
    registry.internalsite.com:
      myuser2: '0th3rp@55w0rd!'
    '192.0.2.1:8787':
      myuser3: '@n0th3rp@55w0rd!'
```



重要

デフォルトの **ContainerImagePrepare** パラメーターは、認証が必要な **registry.redhat.io** からコンテナーイメージをプルします。

詳細は、Red Hat コンテナーレジストリーの認証 を参照してください。

ContainerImageRegistryLogin

ContainerImageRegistryLogin パラメーターを使用して、コンテナーイメージを取得するために、オーバークラウドノードシステムがリモートレジストリーにログインする必要があるかどうかを制御します。このような状況は、アンダークラウドを使用してイメージをホストする代わりに、オーバークラ

ウドノードがイメージを直接プルする場合に発生します。

特定の設定について、**push_destination** が false に設定されている、または使用されていない場合は、**ContainerImageRegistryLogin** を **true** に設定する必要があります。

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      namespace: registry.redhat.io/...
      ...
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
  ContainerImageRegistryLogin: true
```

ただし、オーバークラウドノードに **ContainerImageRegistryCredentials** で定義されたレジストリーホストへのネットワーク接続がなく、**ContainerImageRegistryLogin** を **true** に設定すると、ログインを試みる際にデプロイメントが失敗する可能性があります。オーバークラウドノードに **ContainerImageRegistryCredentials** で定義されたレジストリーホストへのネットワーク接続がない場合、**push_destination** を **true** に、**ContainerImageRegistryLogin** を **false** に設定して、オーバークラウドノードがアンダークラウドからイメージをプルできるようにします。

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
      namespace: registry.redhat.io/...
      ...
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
  ContainerImageRegistryLogin: false
```

3.10. イメージ準備エントリーの階層化

ContainerImagePrepare パラメーターの値は YAML リストです。したがって、複数のエントリーを指定することができます。以下の例で、2 つのエントリーを指定するケースを説明します。この場合、director はすべてのイメージの最新バージョンを使用しますが、**nova-api** イメージについてのみ、**16.2-44** とタグ付けされたバージョンを使用します。

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

```
- nova-api
set:
  namespace: registry.redhat.io/rhosp-rhel8
  tag: 16.2-44
```

includes および **excludes** のパラメーターでは、各エントリーのイメージの絞り込みをコントロールするのに正規表現が使用されます。**includes** 設定と一致するイメージが、**excludes** と一致するイメージに優先します。イメージが一致するとみなされるためには、名前に **includes** または **excludes** の正規表現の値が含まれている必要があります。

3.11. CEPH STORAGE コンテナーイメージの除外

デフォルトのオーバークラウドロール設定では、デフォルトの Controller ロール、Compute ロール、および Ceph Storage ロールが使用されます。ただし、デフォルトのロール設定を使用して Ceph Storage ノードを持たないオーバークラウドをデプロイする場合、director は引き続き Ceph Storage コンテナーイメージを Red Hat コンテナーレジストリーからプルします。イメージがデフォルト設定の一部として含まれているためです。

オーバークラウドで Ceph Storage コンテナーが必要ない場合は、Red Hat コンテナーレジストリーから Ceph Storage コンテナーイメージをプルしないように director を設定することができます。

手順

1. **containers-prepare-parameter.yaml** ファイルを編集し、Ceph Storage コンテナーを除外します。

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    excludes:
      - ceph
      - prometheus
    set:
      …
```

   excludes パラメーターは正規表現を使用して、**ceph** または **prometheus** 文字列を含むコンテナーイメージを除外します。

2. **containers-prepare-parameter.yaml** ファイルを保存します。

3.12. 準備プロセスにおけるイメージの変更

イメージの準備中にイメージを変更し、変更したそのイメージで直ちにオーバークラウドをデプロイすることが可能です。

注記

Red Hat OpenStack Platform (RHOSP) ディレクターは、Ceph コンテナーではなく、RHOSP コンテナーの準備中にイメージを変更することをサポートします。

イメージを変更するシナリオを以下に示します。

- デプロイメント前にテスト中の修正でイメージが変更される、継続的インテグレーションのパイプラインの一部として。

- ローカルの変更をテストおよび開発のためにデプロイしなければならない、開発ワークフローの一部として。
- 変更をデプロイしなければならないが、イメージビルドパイプラインでは利用することができない場合。たとえば、プロプライエタリーアドオンの追加または緊急の修正など。

準備プロセス中にイメージを変更するには、変更する各イメージで Ansible ロールを呼び出します。ロールはソースイメージを取得して必要な変更を行い、その結果をタグ付けします。prepare コマンドでイメージを目的のレジストリーにプッシュし、変更したイメージを参照するように heat パラメーターを設定することができます。

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

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

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

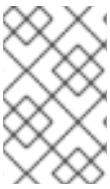
イメージを変更する **ContainerImagePrepare** エントリーを開発およびテストする場合には、イメージが想定どおりに変更されることを確認するために、追加のオプションを指定せずにイメージの準備コマンドを実行します。

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

重要

openstack tripleo container image prepare コマンドを使用するには、アンダークラウドに実行中の **image-serve** レジストリーが含まれている必要があります。したがって、**image-serve** レジストリーがインストールされないため、新しいアンダークラウドのインストール前にこのコマンドを実行することはできません。アンダークラウドが正常にインストールされた後に、このコマンドを実行することができます。

3.13. コンテナーイメージの既存パッケージの更新



注記

Red Hat OpenStack Platform (RHOSP) ディレクターは、Ceph コンテナーではなく、RHOSP コンテナーのコンテナーイメージ上の既存のパッケージの更新をサポートします。

手順

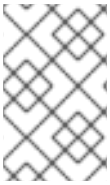
- 以下の **ContainerImagePrepare** エントリーは、アンダークラウドホストの dnf リポジトリー設定を使用してコンテナーイメージのパッケージをすべて更新する例です。

  -

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

3.14. コンテナーイメージへの追加 RPM ファイルのインストール

RPM ファイルのディレクトリーをコンテナーイメージにインストールすることができます。この機能は、ホットフィックスやローカルパッケージビルドなど、パッケージリポジトリーからは入手できないパッケージのインストールに役立ちます。



注記

Red Hat OpenStack Platform (RHOSP) ディレクターは、Ceph コンテナーではなく、RHOSP コンテナーのコンテナーイメージへの追加の RPM ファイルのインストールをサポートします。

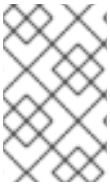
手順

- 次の **ContainerImagePrepare** エントリーの例では、いくつかのホットフィックスパッケージを **nova-compute** イメージにのみインストールします。

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

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

Dockerfile を含むディレクトリーを指定して、必要な変更を加えることができます。**tripleo-modify-image** ロールを呼び出すと、ロールは **Dockerfile.modified** ファイルを生成し、これにより **FROM** ディレクティブが変更され新たな **LABEL** ディレクティブが追加されます。



注記

Red Hat OpenStack Platform (RHOSP) ディレクターは、Ceph コンテナーではなく、RHOSP コンテナー用のカスタム Dockerfile を使用したコンテナーイメージの変更をサポートします。

手順

1. 以下の例では、**nova-compute** イメージでカスタム Dockerfile が実行されます。

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

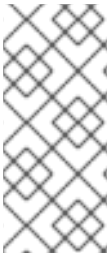
2. **/home/stack/nova-custom/Dockerfile** ファイルの例を以下に示します。**USER** root ディレクティブを実行した後は、元のイメージのデフォルトユーザーに戻す必要があります。

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

USER "root"

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

USER "nova"
```

3.16. コンテナーイメージ管理用 SATELLITE サーバーの準備

Red Hat Satellite 6 には、レジストリーの同期機能が備わっています。これにより、複数のイメージを Satellite Server にプルし、アプリケーションライフサイクルの一環として管理することができます。また、他のコンテナー対応システムも Satellite をレジストリーとして使うことができます。コンテナーイメージ管理についての詳細は、**Red Hat Satellite 6 コンテンツ管理ガイド**の コンテナーイメージの管理 を参照してください。

以下の手順は、Red Hat Satellite 6 の **hammer** コマンドラインツールを使用した例を示しています。組織には、例として **ACME** という名称を使用しています。この組織は、実際に使用する Satellite 6 の組織に置き換えてください。



注記

この手順では、**registry.redhat.io** のコンテナーイメージにアクセスするために認証情報が必要です。Red Hat では、個人のユーザー認証情報を使用する代わりに、レジストリーサービスアカウントを作成し、それらの認証情報を使用して **registry.redhat.io** コンテンツにアクセスすることを推奨します。詳しくは、Red Hat コンテナーレジストリーの認証 を参照してください。

手順

1. すべてのコンテナーイメージのリストを作成します。

```
$ sudo podman search --limit 1000 "registry.redhat.io/rhosp-rhel8/openstack" --format="{{
.Name }}" | sort > satellite_images
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-4-dashboard-
```

```
rhel8
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-4-rhel8
$ sudo podman search --limit 1000 "registry.redhat.io/openshift" | grep ose-prometheus
```

- Ceph をインストールして Ceph Dashboard を有効にする場合は、次の ose-prometheus コンテナーが必要です。

```
registry.redhat.io/openshift4/ose-prometheus-node-exporter:v4.6
registry.redhat.io/openshift4/ose-prometheus:v4.6
registry.redhat.io/openshift4/ose-prometheus-alertmanager:v4.6
```

2. Satellite 6 の **hammer** ツールがインストールされているシステムに **satellite_images** ファイルをコピーします。あるいは、Hammer CLI ガイド に記載の手順に従って、アンダークラウドに **hammer** ツールをインストールします。

3. 以下の **hammer** コマンドを実行して、実際の Satellite 組織に新規製品 (**OSP16.1 Containers**) を作成します。

```
$ hammer product create \
  --organization "ACME" \
  --name "OSP Containers"
```

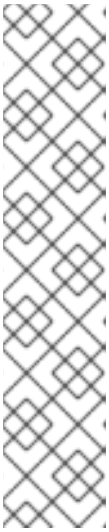
このカスタム製品に、イメージを保管します。

4. **satellite_images** ファイルからオーバークラウドのコンテナーイメージを追加します。

```
$ while read IMAGE; do \
  IMAGE_NAME=$(echo $IMAGE | cut -d"/" -f3 | sed "s/openstack-//g") ; \
  IMAGE_NOURL=$(echo $IMAGE | sed "s/registry.redhat.io\///g") ; \
  hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name $IMAGE_NOURL \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name $IMAGE_NAME ; done < satellite_images
```

5. Ceph Storage 4 コンテナーイメージを追加します。

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph/rhceph-4-rhel8 \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name rhceph-4-rhel8
```



注記

Ceph Dashboard をインストールする場合は、**hammer repository create** コマンドに **--name rhceph-4-dashboard-rhel8** を含めます。

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph/rhceph-4-dashboard-rhel8 \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name rhceph-4-dashboard-rhel8
```

6. コンテナーイメージを同期します。

```
$ hammer product synchronize \
  --organization "ACME" \
  --name "OSP Containers"
```

Satellite Server が同期を完了するまで待ちます。

注記

設定によっては、**hammer** から Satellite Server のユーザー名およびパスワードが要求される場合があります。設定ファイルを使用して自動的にログインするように **hammer** を設定することができます。詳しくは、**Red Hat Satellite Hammer CLI ガイド**の 認証 セクションを参照してください。

7. お使いの Satellite 6 サーバーでコンテンツビューが使われている場合には、新たなバージョンのコンテンツビューを作成してイメージを反映し、アプリケーションライフサイクルの環境に従ってプロモートします。この作業は、アプリケーションライフサイクルをどのように設定するかに大きく依存します。たとえば、ライフサイクルに **production** という名称の環境があり、その環境でコンテナーイメージを利用可能にする場合には、コンテナーイメージを含むコンテンツビューを作成し、そのコンテンツビューを **production** 環境にプロモートします。詳しくは、Red Hat Satellite コンテンツ管理ガイドの コンテンツビューの管理 を参照してください。

8. **base** イメージに使用可能なタグを確認します。

```
$ hammer docker tag list --repository "base" \
  --organization "ACME" \
  --lifecycle-environment "production" \
  --product "OSP Containers"
```

このコマンドにより、特定環境のコンテンツビューでの OpenStack Platform コンテナーイメージのタグが表示されます。

9. アンダークラウドに戻り、Satellite サーバーをソースとして使用して、イメージを準備するデフォルトの環境ファイルを生成します。以下のサンプルコマンドを実行して環境ファイルを生成します。

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

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

10. **containers-prepare-parameter.yaml** ファイルを編集して以下のパラメーターを変更します。

    - **push_destination**: 選択したコンテナーイメージの管理手段に応じて、これを **true** または **false** に設定します。このパラメーターを **false** に設定すると、オーバークラウドノードはイメージを直接 Satellite からプルします。このパラメーターを **true** に設定すると、director はイメージを Satellite からアンダークラウドレジストリーにプルし、オーバークラウドはそのイメージをアンダークラウドレジストリーからプルします。

    - **namespace**: Satellite サーバー上のレジストリーの URL およびポート。Red Hat Satellite のデフォルトのレジストリーポートは 443 です。

    - **name_prefix**: 接頭辞は Satellite 6 の命名規則に基づきます。これは、コンテンツビューを使用するかどうかによって異なります。

      - コンテンツビューを使用する場合、設定は **[org]-[environment]-[content view]-[product]-** です。例: **acme-production-myosp16-osp_containers-**。

      - コンテンツビューを使用しない場合、設定は **[org]-[product]-** です。例: **acme-osp_containers-**。

    - **ceph_namespace**、**ceph_image**、**ceph_tag**: Ceph Storage を使用する場合には、Ceph Storage コンテナーイメージの場所を定義するこれらの追加パラメーターを指定します。**ceph_image** に Satellite 固有の接頭辞が追加された点に注意してください。この接頭辞は、**name_prefix** オプションと同じ値です。

Satellite 固有のパラメーターが含まれる環境ファイルの例を、以下に示します。

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      ceph_image: acme-production-myosp16_1-osp_containers-rhceph-4
      ceph_namespace: satellite.example.com:443
      ceph_tag: latest
      name_prefix: acme-production-myosp16_1-osp_containers-
      name_suffix: ''
      namespace: satellite.example.com:443
      neutron_driver: null
      tag: '{osp_curr_ver_no_beta}'
      ...
```

注記

Red Hat SatelliteServer に保存されている特定のコンテナーイメージのバージョンを使用するには、**tag** のキーと値のペアを **set** ディクショナリー内の特定のバージョンに設定します。たとえば、{osp_curr_ver_no_beta}.2 イメージストリームを使用するには、**set** された辞書に **tag: {osp_curr_ver_no_beta}.2** を設定します。

undercloud.conf 設定ファイルで **containers-prepare-parameter.yaml** 環境ファイルを定義する必要があります。定義しないと、アンダークラウドはデフォルト値を使用します。

```
container_images_file = /home/stack/containers-prepare-parameter.yaml
```

第4章 アンダークラウドへの DIRECTOR のインストール

Director を設定してインストールするには、**undercloud.conf** ファイルに適切なパラメーターを設定し、undercloud installation コマンドを実行します。director をインストールしたら、ノードのプロビジョニング中に director がベアメタルノードへの書き込みに使用するオーバークラウドイメージをインポートします。

4.1. DIRECTOR の設定

director のインストールプロセスでは、director が **stack** ユーザーのホームディレクトリーから読み取る **undercloud.conf** 設定ファイルに、特定の設定が必要になります。設定のベースとするためにデフォルトのテンプレートをコピーするには、以下の手順を実施します。

手順

1. デフォルトのテンプレートを **stack** ユーザーのホームディレクトリーにコピーします。

```
[stack@director ~]$ cp \
  /usr/share/python-tripleoclient/undercloud.conf.sample \
  ~/undercloud.conf
```

2. **undercloud.conf** ファイルを編集します。このファイルには、アンダークラウドを設定するための設定値が含まれています。パラメーターを省略したり、コメントアウトした場合には、アンダークラウドのインストールでデフォルト値が使用されます。

4.2. DIRECTOR の設定パラメーター

以下のリストで、**undercloud.conf** ファイルを設定するパラメーターについて説明します。エラーを避けるために、パラメーターは決して該当するセクションから削除しないでください。

重要

少なくとも、コンテナーイメージの設定が含まれる環境ファイルに **container_images_file** パラメーターを設定する必要があります。このパラメーターに適切なファイルを正しく設定しないと、director は **ContainerImagePrepare** パラメーターからコンテナーイメージのルールセットを取得すること や、**ContainerImageRegistryCredentials** パラメーターからコンテナーレジストリーの認証情報を取得することができなくなります。

デフォルト

undercloud.conf ファイルの **[DEFAULT]** セクションで定義されているパラメーターを以下に示します。

additional_architectures

オーバークラウドがサポートする追加の (カーネル) アーキテクチャーのリスト。現在、オーバークラウドは、デフォルトの **x86_64** アーキテクチャーに加えて **ppc64le** アーキテクチャーをサポートしています。

certificate_generation_ca

要求した証明書に署名する CA の **certmonger** のニックネーム。**generate_service_certificate** パラメーターを設定した場合に限り、このオプションを使用します。**local** CA を選択する場合は、certmonger はローカルの CA 証明書を **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** に抽出し、証明書をトラストチェーンに追加します。

clean_nodes

デプロイメントを再実行する前とイントロスペクションの後にハードドライブを消去するかどうかを定義します。

cleanup

一時的なファイルを削除します。デプロイメント中に使用される一時ファイルを保持するには、これを **False** に設定します。一時ファイルは、エラーが発生した場合にデプロイメントのデバッグに役立ちます。

container_cli

コンテナー管理用の CLI ツール。このパラメーターは、**podman** に設定したままにしてください。Red Hat Enterprise Linux 8.2 がサポートするのは **podman** だけです。

container_healthcheck_disabled

コンテナー化されたサービスのヘルスチェックを無効にします。Red Hat は、ヘルスチェックを有効にし、このオプションを **false** に設定したままにすることを推奨します。

container_images_file

コンテナーイメージ情報が含まれる heat 環境ファイル。このファイルには、以下のエントリーを含めることができます。

- 必要なすべてのコンテナーイメージのパラメーター
- 必要なイメージの準備を実施する **ContainerImagePrepare** パラメーター。このパラメーターが含まれるファイルの名前は、通常 **containers-prepare-parameter.yaml** です。

container_insecure_registries

podman が使用するセキュアではないレジストリーのリスト。プライベートコンテナーレジストリー等の別のソースからイメージをプルする場合には、このパラメーターを使用します。多くの場合、**podman** は Red Hat Container Catalog または Satellite サーバー (アンダークラウドが Satellite に登録されている場合) のいずれかからコンテナーイメージをプルするための証明書を持ちます。

container_registry_mirror

設定により **podman** が使用するオプションの **registry-mirror**

custom_env_files

アンダークラウドのインストールに追加する新たな環境ファイル

deployment_user

アンダークラウドをインストールするユーザー。現在のデフォルトユーザー **stack** を使用する場合には、このパラメーターを未設定のままにします。

discovery_default_driver

自動的に登録されたノードのデフォルトドライバーを設定します。**enable_node_discovery** パラメーターを有効にし、**enabled_hardware_types** のリストにドライバーを含める必要があります。

enable_ironic、enable_ironic_inspector、enable_mistral、enable_nova、enable_tempest、enable_validations、enable_zaqar

director で有効にするコアサービスを定義します。これらのパラメーターは **true** に設定されたままにします。

enable_node_discovery

introspection ramdisk を PXE ブートする不明なノードを自動的に登録します。新規ノードは、**fake** ドライバーをデフォルトとして使用しますが、**discovery_default_driver** を設定して上書きすることもできます。また、イントロスペクションルールを使用して、新しく登録したノードにドライバーの情報を指定することもできます。

enable_novajoin

アンダークラウドに **novajoin** メタデータサービスをインストールするかどうかを定義します。

enable_routed_networks

ルーティングされたコントロールプレーンネットワークのサポートを有効にするかどうかを定義します。

enable_swift_encryption

保存データの Swift 暗号化を有効にするかどうかを定義します。

enable_telemetry

アンダークラウドに OpenStack Telemetry サービス (gnocchi、aodh、panko) をインストールするかどうかを定義します。Telemetry サービスを自動的にインストール/設定するには、**enable_telemetry** パラメーターを **true** に設定します。デフォルト値は **false** で、アンダークラウド上の telemetry が無効になります。このパラメーターは、メトリックデータを消費する Red Hat CloudForms などの他の製品を使用する場合に必要です。

警告

RBAC はすべてのコンポーネントでサポートされているわけではありません。Alarming サービス (aodh) と Gnocchi は、安全な RBAC ルールを考慮していません。

enabled_hardware_types

アンダークラウドで有効にするハードウェアタイプのリスト

generate_service_certificate

アンダークラウドのインストール時に SSL/TLS 証明書を生成するかどうかを定義します。これは **undercloud_service_certificate** パラメーターに使用します。アンダークラウドのインストールで、作成された証明書 **/etc/pki/tls/certs/undercloud-[undercloud_public_vip].pem** を保存します。**certificate_generation_ca** パラメーターで定義される CA はこの証明書に署名します。

heat_container_image

使用する heat コンテナーイメージの URL。未設定のままにします。

heat_native

heat-all を使用してホストベースのアンダークラウド設定を実行します。**true** のままにします。

hieradata_override

director に Puppet hieradata を設定するための **hieradata** オーバーライドファイルへのパス。これにより、サービスに対して **undercloud.conf** パラメーター以外のカスタム設定を行うことができます。設定すると、アンダークラウドのインストールでこのファイルが **/etc/puppet/hieradata** ディレクトリーにコピーされ、階層の最初のファイルに設定されます。この機能の使用についての詳細は、Director Installation and Usage の Configuring hieradata on the undercloud を参照してください。

inspection_extras

イントロスペクション時に追加のハードウェアコレクションを有効化するかどうかを定義します。このパラメーターを使用するには、イントロスペクションイメージに **python-hardware** または **python-hardware-detect** パッケージが必要です。

inspection_interface

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

inspection_runbench

ノードイントロスペクション時に一連のベンチマークを実行します。ベンチマークを有効にするには、このパラメーターを **true** に設定します。このオプションは、登録ノードのハードウェアを検査する際にベンチマーク分析を実行する場合に必要です。

ipa_otp

IPA サーバーにアンダークラウドノードを登録するためのワンタイムパスワードを定義します。これは、**enable_novajoin** が有効な場合に必要です。

ipv6_address_mode

アンダークラウドのプロビジョニングネットワーク用の IPv6 アドレス設定モード。このパラメーターに設定できる値のリストを以下に示します。

- dhcpv6-stateless: ルーター広告 (RA) を使用するアドレス設定と DHCPv6 を使用するオプションの情報
- dhcpv6-stateful: DHCPv6 を使用するアドレス設定とオプションの情報

ipxe_enabled

iPXE か標準の PXE のいずれを使用するか定義します。デフォルトは **true** で iPXE を有効化します。標準の PXE を使用するには、このパラメーターを **false** に設定します。

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イス。director は、DHCP および PXE ブートサービスにもこのデバイスを使用します。この値を選択したデバイスに変更します。接続されているデバイスを確認するには、**ip addr** コマンドを使用します。**ip addr** コマンドの出力結果の例を、以下に示します。

```
2: em0: <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 em0
       valid_lft 3462sec preferred_lft 3462sec
    inet6 fe80::5054:ff:fe75:2409/64 scope link
       valid_lft forever preferred_lft forever
3: em1: <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 **em0** を使用し、プロビジョニング NIC は、現在設定されていない **em1** を使用します。この場合は、**local_interface** を **em1** に設定します。この設定スクリプトにより、このインターフェイスが **inspection_interface** パラメーターで定義したカスタムのブリッジにアタッチされます。

local_ip

director のプロビジョニング NIC 用に定義する IP アドレス。director は、DHCP および PXE ブートサービスにもこの IP アドレスを使用します。この IP アドレスが環境内の既存の IP アドレスまたはサブネットと競合するなどの理由により、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、この値をデフォルトの **192.168.24.1/24** のままにします。
IPv6 の場合、ステートフル接続とステートレス接続の両方をサポートするには、ローカル IP アドレス接頭辞接頭辞の長さを **/64** にする必要があります。

local_mtu

local_interface に使用する最大伝送単位 (MTU)。アンダークラウドでは 1500 以下にします。

local_subnet

PXE ブートと DHCP インターフェイスに使用するローカルサブネット。**local_ip** アドレスがこのサブネットに含まれている必要があります。デフォルトは **ctlplane-subnet** です。

net_config_override

ネットワーク設定のオーバーライドテンプレートへのパス。このパラメーターを設定すると、アンダークラウドは JSON または YAML 形式のテンプレートを使用して、**os-net-config** でネットワークを設定し、**undercloud.conf** で設定されたネットワークパラメーターを無視します。ボンディングを設定する場合、またはインターフェイスにオプションを追加する場合に、このパラメーターを使用します。アンダークラウドネットワークインターフェイスのカスタマイズの詳細については、Configuring undercloud network interfaces を参照してください。

networks_file

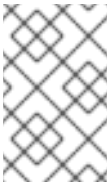
heat をオーバーライドするネットワークファイル

output_dir

状態、処理された heat テンプレート、および Ansible デプロイメントファイルを出力するディレクトリー

overcloud_domain_name

オーバークラウドをデプロイする際に使用する DNS ドメイン名



注記

オーバークラウドを設定する際に、**CloudDomain** にこのパラメーターと同じ値を設定する必要があります。オーバークラウドを設定する際に、環境ファイルでこのパラメーターを設定します。

roles_file

アンダークラウドのインストールで、デフォルトロールファイルを上書きするのに使用するロールファイル。director のインストールにデフォルトのロールファイルが使用されるように、このパラメーターは未設定のままにすることを強く推奨します。

scheduler_max_attempts

スケジューラーがインスタンスのデプロイを試行する最大回数。スケジューリング時に競合状態にならないように、この値を 1 度にデプロイする予定のベアメタルノードの数以上に指定する必要があります。

service_principal

この証明書を使用するサービスの Kerberos プリンシパル。FreeIPA など CA で Kerberos プリンシパルが必要な場合に限り、このパラメーターを使用します。

subnets

プロビジョニングおよびイントロスペクション用のルーティングネットワークのサブネットのリスト。デフォルト値に含まれるのは、**ctlplane-subnet** サブネットのみです。詳細は、サブネット を参照してください。

templates

上書きする heat テンプレートファイル

undercloud_admin_host

SSL/TLS 経由の director の管理 API エンドポイントに定義する IP アドレスまたはホスト名。director の設定により、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 director のソフトウェアブリッジに接続されます。

undercloud_admin_host が **local_ip** と同じ IP ネットワークにない場合は、Undercloud の管理 API がリッスンするインターフェイスに **ControlVirtualInterface** パラメーターを設定する必要があります。デフォルトでは、管理 API は **br-ctlplane** インターフェイスでリッスンしま

す。**ControlVirtualInterface** パラメーターをカスタム環境ファイルに設定し、**custom_env_files** パラメーターを設定して、**undercloud.conf** ファイルにカスタム環境ファイルを含めます。

アンダークラウドネットワークインターフェイスのカスタマイズについての詳細は、Configuring undercloud network interfaces 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。**DEBUG** ログレベルを有効にするには、この値を **true** に設定します。

undercloud_enable_selinux

デプロイメント時に、SELinux を有効または無効にします。問題をデバッグする場合以外は、この値を **true** に設定したままにすることを強く推奨します。

undercloud_hostname

アンダークラウドの完全修飾ホスト名を定義します。本パラメーターを指定すると、アンダークラウドのインストールでホスト名すべてに設定されます。指定しないと、アンダークラウドは現在のホスト名を使用しますが、システムのホスト名すべてを適切に設定しておく必要があります。

undercloud_log_file

アンダークラウドのインストールログおよびアップグレードログを保管するログファイルへのパス。デフォルトでは、ログファイルはホームディレクトリー内の **install-undercloud.log** です。たとえば、**/home/stack/install-undercloud.log** のようになります。

undercloud_nameservers

アンダークラウドのホスト名解決に使用する DNS ネームサーバーのリスト

undercloud_ntp_servers

アンダークラウドの日付と時刻を同期できるようにする Network Time Protocol サーバーのリスト

undercloud_public_host

SSL/TLS 経由の director のパブリック API エンドポイントに定義する IP アドレスまたはホスト名。director の設定により、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 director のソフトウェアブリッジに接続されます。
undercloud_public_host が **local_ip** と同じ IP ネットワークにない場合は、**PublicVirtualInterface** パラメーターを、アンダークラウド上のパブリック API がリッスンする公開インターフェイスに設定する必要があります。デフォルトでは、パブリック API は **br-ctlplane** インターフェイスでリッスンします。カスタム環境ファイルに **PublicVirtualInterface** パラメーターを設定し、**custom_env_files** パラメーターを設定して、**undercloud.conf** ファイルにカスタム環境ファイルを含めます。

アンダークラウドネットワークインターフェイスのカスタマイズについての詳細は、Configuring undercloud network interfaces を参照してください。

undercloud_service_certificate

OpenStack SSL/TLS 通信の証明書の場所とファイル名。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、独自の自己署名の証明書を生成します。

undercloud_timezone

アンダークラウド用ホストのタイムゾーン。タイムゾーンを指定しなければ、director は既存のタイムゾーン設定を使用します。

undercloud_update_packages

アンダークラウドのインストール時にパッケージを更新するかどうかを定義します。

サブネット

undercloud.conf ファイルには、各プロビジョニングサブネットの名前が付いたセクションがあります。たとえば、**ctlplane-subnet** という名前のサブネットを作成するには、**undercloud.conf** ファイルで以下のような設定を使用します。

```
[ctlplane-subnet]
cidr = 192.168.24.0/24
dhcp_start = 192.168.24.5
dhcp_end = 192.168.24.24
inspection_iprange = 192.168.24.100,192.168.24.120
gateway = 192.168.24.1
masquerade = true
```

プロビジョニングネットワークは、環境に応じて、必要なだけ指定することができます。

重要

director がサブネットを作成した後、director はサブネットの IP アドレスを変更することはできません。

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、アンダークラウドの **neutron** サービスが管理する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 (**192.168.24.0/24**) のままにします。

masquerade

外部アクセスのために、**cidr** で定義したネットワークをマスカレードするかどうかを定義します。このパラメーターにより、director 経由で外部アクセスすることができるように、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) のメカニズムが提供されます。

注記

director 設定は、適切な **sysctl** カーネルパラメーターを使用して IP フォワーディングも自動的に有効化します。

dhcp_start、dhcp_end

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

dhcp_exclude

DHCP 割り当て範囲で除外する IP アドレス

dns_nameservers

サブネットに固有の DNS ネームサーバー。サブネットにネームサーバーが定義されていない場合には、サブネットは **undercloud_nameservers** パラメーターで定義されるネームサーバーを使用します。

gateway

オーバークラウドインスタンスのゲートウェイ。External ネットワークにトラフィックを転送するアンダークラウドのホストです。director に別の IP アドレスを使用する場合または直接外部ゲートウェイを使用する場合以外は、この値はデフォルト (**192.168.24.1**) のままにします。

host_routes

このネットワーク上のオーバークラウドインスタンス用の neutron が管理するサブネットのホストルート。このパラメーターにより、アンダークラウド上の **local_subnet** のホストルートも設定されます。

inspection_iprange

検査プロセス中に使用するこのネットワーク上のノードの一時的な IP 範囲。この範囲は、**dhcp_start** と **dhcp_end** で定義された範囲と重複することはできませんが、同じ IP サブネット内になければなりません。

実際の設定に応じて、これらのパラメーターの値を変更してください。完了したら、ファイルを保存します。

4.3. 環境ファイルを使用したアンダークラウドの設定

undercloud.conf ファイルを使用して、アンダークラウドの主要なパラメーターを設定します。heat パラメーターが含まれる環境ファイルを使用して、アンダークラウドの追加設定を行うこともできます。

手順

1. **/home/stack/templates/custom-undercloud-params.yaml** という名前で環境ファイルを作成します。
2. このファイルを編集して、必要な heat パラメーターを追加します。たとえば、特定の OpenStack Platform サービスのデバッグを有効にするには、**custom-undercloud-params.yaml** ファイルに以下のスニペットを追加します。

```
parameter_defaults:
  Debug: True
```

   完了したら、このファイルを保存します。
3. **undercloud.conf** ファイルを編集し、**custom_env_files** パラメーターまでスクロールします。作成した **custom-undercloud-params.yaml** 環境ファイルをポイントするようにパラメーターを編集します。

```
custom_env_files = /home/stack/templates/custom-undercloud-params.yaml
```

注記

コンマ区切りリストを使用して、複数の環境ファイルを指定することができます。

アンダークラウドの次回インストールまたはアップグレード操作時に、この環境ファイルが director のインストールに追加されます。

4.4. アンダークラウド設定用の標準 HEAT パラメーター

以下の表には、アンダークラウド用のカスタム環境ファイルで設定する標準の heat パラメーターをまとめています。

パラメーター	説明
AdminPassword	アンダークラウドの **admin** ユーザーのパスワードを設定します。
AdminEmail	アンダークラウドの **admin** ユーザーの電子メールアドレスを設定します。
Debug	デバッグモードを有効にします。

カスタム環境ファイルの **parameter_defaults** セクションで、これらのパラメーターを設定します。

```
parameter_defaults:
  Debug: True
  AdminPassword: "myp@ssw0rd!"
  AdminEmail: "admin@example.com"
```

4.5. アンダークラウドへの HIERADATA の設定

director に Puppet hieradata を設定して、サービスに対して利用可能な **undercloud.conf** パラメーター以外のカスタム設定を行うことができます。

手順

1. hieradata オーバーライドファイルを作成します (例: **/home/stack/hieradata.yaml**)。
2. カスタマイズした hieradata をファイルに追加します。たとえば、Compute (nova) サービスのパラメーター **force_raw_images** をデフォルト値の **True** から **False** に変更するには、以下のスニペットを追加します。

   ```
   nova::compute::force_raw_images: False
   ```

   設定するパラメーターに Puppet 実装がない場合には、以下の手段によりパラメーターを設定します。

   ```
   nova::config::nova_config:
     DEFAULT/<parameter_name>:
       value: <parameter_value>
   ```

   以下に例を示します。

   ```
   nova::config::nova_config:
     DEFAULT/network_allocate_retries:
       value: 20
     ironic/serial_console_state_timeout:
       value: 15
   ```

3. **undercloud.conf** ファイルの **hieradata_override** パラメーターを、新しい **/home/stack/hieradata.yaml** ファイルのパスに設定します。

```
hieradata_override = /home/stack/hieradata.yaml
```

4.6. IPV6 を使用してベアメタルをプロビジョニングするためのアンダークラウド設定

IPv6 ノードおよびインフラストラクチャーがある場合には、IPv4 ではなく IPv6 を使用するようにアンダークラウドおよびプロビジョニングネットワークを設定することができます。これにより、director は IPv6 ノードに Red Hat OpenStack Platform をプロビジョニングおよびデプロイすることができます。ただし、いくつかの考慮事項があります。

- デュアルスタック IPv4/6 は利用できません。
- tempest 検証が正しく動作しない可能性があります。
- アップグレード時に IPv4 から IPv6 に移行することはできません。

undercloud.conf ファイルを変更して、Red Hat OpenStack Platform で IPv6 プロビジョニングを有効にします。

前提条件

- アンダークラウドの IPv6 アドレス。詳しい情報は、**IPv6 Networking for the Overcloud**の Configuring an IPv6 Address on the Undercloud を参照してください。

手順

1. **undercloud.conf** ファイルを開きます。
2. IPv6 アドレスモードをステートレスまたはステートフルのいずれかに指定します。

```
[DEFAULT]
ipv6_address_mode = <address_mode>
...
```

   - NIC がサポートするモードに基づいて、**<address_mode>** を **dhcpv6-stateless** または **dhcpv6-stateful** に置き換えます。

注記

ステートフルアドレスモードを使用する場合、ファームウェア、チェーンローダー、およびオペレーティングシステムは、DHCP サーバーが追跡する ID を生成するために異なるアルゴリズムを使用する場合があります。DHCPv6 は MAC によってアドレスを追跡せず、リクエスターからの ID 値が変更されても、MAC アドレスが同じままである場合、同じアドレスを提供しません。したがって、ステートフル DHCPv6 を使用する場合は、次の手順を実行してネットワークインターフェイスを設定する必要もあります。

3. ステートフル DHCPv6 を使用するようにアンダークラウドを設定した場合は、ベアメタルノードに使用するネットワークインターフェイスを指定します。

```
[DEFAULT]
ipv6_address_mode = dhcpv6-stateful
ironic_enabled_network_interfaces = neutron,flat
```

```
...
```

4. ベアメタルノードのデフォルトのネットワークインターフェイスを設定します。

```
[DEFAULT]
...
ironic_default_network_interface = neutron
...
```

5. アンダークラウドがプロビジョニングネットワーク上にルーターを作成するかどうかを指定します。

```
[DEFAULT]
...
enable_routed_networks: <true/false>
...
```

- **<true/false>** を **true** に置き換えて、ルーティングされたネットワークを有効にし、アンダークラウドがプロビジョニングネットワーク上にルーターを作成しないようにします。**true** の場合、データセンタールーターはルーターアドバタイズメントを提供する必要があります。

- **<true/false>** を **false** に置き換えて、ルーティングされたネットワークを無効にし、プロビジョニングネットワーク上にルーターを作成します。

6. ローカル IP アドレス、および SSL/TLS を介した director Admin API および Public API エンドポイントの IP アドレスを設定します。

```
[DEFAULT]
...
local_ip = <ipv6_address>
undercloud_admin_host = <ipv6_address>
undercloud_public_host = <ipv6_address>
...
```

- **<ipv6_address>** をアンダークラウドの IPv6 アドレスに置き換えます。

7. オプション: director がインスタンスの管理に使用するプロビジョニングネットワークを設定します。

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
...
```

- **<ipv6_address>** を、デフォルトのプロビジョニングネットワークを使用していないときにインスタンスの管理に使用するネットワークの IPv6 アドレスに置き換えます。

- **<ipv6_prefix>** を、デフォルトのプロビジョニングネットワークを使用していないときにインスタンスの管理に使用するネットワークの IP アドレス接頭辞に置き換えます。

8. プロビジョニングノードの DHCP 割り当て範囲を設定します。

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
```

```
dhcp_end = <ipv6_address_dhcp_end>
...
```

- **<ipv6_address_dhcp_start>** を、オーバークラウドノードに使用するネットワーク範囲の開始点の IPv6 アドレスに置き換えます。
- **<ipv6_address_dhcp_end>** を、オーバークラウドノードに使用するネットワーク範囲の終わりの IPv6 アドレスに置き換えます。

9. オプション: トラフィックを External ネットワークに転送するようにゲートウェイを設定します。

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
...
```

- デフォルトゲートウェイを使用しない場合は、**<ipv6_gateway_address>** をゲートウェイの IPv6 アドレスに置き換えます。

10. 検査プロセス中に使用する DHCP 範囲を設定します。

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
inspection_iprange = <ipv6_address_inspection_start>,<ipv6_address_inspection_end>
...
```

- **<ipv6_address_inspection_start>** を、検査プロセス中に使用するネットワーク範囲の開始点の IPv6 アドレスに置き換えます。
- **<ipv6_address_inspection_end>** を、検査プロセス中に使用するネットワーク範囲の終わりの IPv6 アドレスに置き換えます。

注記

この範囲は、**dhcp_start** と **dhcp_end** で定義された範囲と重複することはできませんが、同じ IP サブネット内になければなりません。

11. サブネットの IPv6 ネームサーバーを設定します。

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
inspection_iprange = <ipv6_address_inspection_start>,<ipv6_address_inspection_end>
dns_nameservers = <ipv6_dns>
```

- **<ipv6_dns>** をサブネットに固有の DNS ネームサーバーに置き換えます。

4.7. アンダークラウドネットワークインターフェイスの設定

特定のネットワーク機能を持つアンダークラウドをインストールするには、**undercloud.conf** ファイルにカスタムネットワーク設定を追加します。たとえば、一部のインターフェイスは DHCP を持ちません。このような場合は、アンダークラウドのインストールプロセス中に **os-net-config** が設定を適用できるように、**undercloud.conf** ファイルでこれらのインターフェイスの DHCP を無効にする必要があります。

手順

1. アンダークラウドのホストにログインします。
2. 新規ファイル **undercloud-os-net-config.yaml** を作成し、必要なネットワーク設定を追加します。
   詳細は、ネットワークインターフェイスのリファレンス を参照してください。

   以下に例を示します。

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers: 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  members:
  - type: interface
    name: nic2
```

   特定のインターフェイスのネットワークボンディングを作成するには、次のサンプルを使用します。

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers: 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  members:
  - name: bond-ctlplane
    type: linux_bond
    use_dhcp: false
    bonding_options: "mode=active-backup"
    mtu: 1500
    members:
    - type: interface
```

```
      name: nic2
    - type: interface
      name: nic3
```

3. **undercloud.conf** ファイルの **net_config_override** パラメーターに、**undercloud-os-net-config.yaml** ファイルへのパスを追加します。

```
[DEFAULT]
...
net_config_override=undercloud-os-net-config.yaml
...
```

注記

director は、**net_config_override** パラメーターに追加するファイルをテンプレートとして使用し、**/etc/os-net-config/config.yaml** ファイルを生成します。**os-net-config** はテンプレートで定義するインターフェイスを管理するので、このファイルですべてのアンダークラウドネットワークインターフェイスのカスタマイズを実行する必要があります。

4. アンダークラウドをインストールします。

検証

- アンダークラウドのインストールが正常に完了したら、**/etc/os-net-config/config.yaml** ファイルに該当する設定が含まれていることを確認します。

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers: 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  members:
  - type: interface
    name: nic2
```

4.8. DIRECTOR のインストール

director をインストールして基本的なインストール後タスクを行うには、以下の手順を実施します。

手順

1. 以下のコマンドを実行して、アンダークラウドに director をインストールします。

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

このコマンドにより、director の設定スクリプトが起動します。director は追加のパッケージをインストールし、**undercloud.conf** の設定に従ってサービスを設定し、すべての RHOSP サービスコンテナーを起動します。このスクリプトは、完了までに数分かかります。

スクリプトにより、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リスト
- **stackrc**: director 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

2. RHOSP サービスコンテナーが実行中であることを確認します。

```
[stack@director ~]$ sudo podman ps -a --format "{{.Names}} {{.Status}}"
```

次のコマンド出力は、RHOSP サービスコンテナーが実行中 (**Up**) であることを示しています。

```
memcached Up 3 hours (healthy)
haproxy Up 3 hours
rabbitmq Up 3 hours (healthy)
mysql Up 3 hours (healthy)
iscsid Up 3 hours (healthy)
keystone Up 3 hours (healthy)
keystone_cron Up 3 hours (healthy)
neutron_api Up 3 hours (healthy)
logrotate_crond Up 3 hours (healthy)
neutron_dhcp Up 3 hours (healthy)
neutron_l3_agent Up 3 hours (healthy)
neutron_ovs_agent Up 3 hours (healthy)
ironic_api Up 3 hours (healthy)
ironic_conductor Up 3 hours (healthy)
ironic_neutron_agent Up 3 hours (healthy)
ironic_pxe_tftp Up 3 hours (healthy)
ironic_pxe_http Up 3 hours (unhealthy)
ironic_inspector Up 3 hours (healthy)
ironic_inspector_dnsmasq Up 3 hours (healthy)
neutron-dnsmasq-qdhcp-30d628e6-45e6-499d-8003-28c0bc066487 Up 3 hours
...
```

3. **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。

```
[stack@director ~]$ source ~/stackrc
```

プロンプトには、OpenStack コマンドがアンダークラウドに対して認証および実行されることが表示されるようになります。

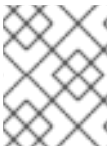
```
(undercloud) [stack@director ~]$
```

director のインストールが完了しました。これで、director コマンドラインツールが使用できるようになりました。

4.9. オーバークラウド用の CPU アーキテクチャーの設定

Red Hat OpenStack Platform (RHOSP) は、オーバークラウドの CPU アーキテクチャーをデフォルト

で x86_64 として設定します。POWER (ppc64le) ハードウェアにオーバークラウドコンピュートノードをデプロイすることもできます。コンピュートノードのクラスターには、同じアーキテクチャーのシステム、または x86_64 と ppc64le が混在するシステムを使用することができます。

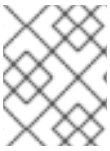


注記

アンダークラウド、コントローラーノード、Ceph Storage ノード、およびその他のシステムは、すべて x86_64 ハードウェアでのみサポートされています。

4.9.1. オーバークラウドの単一 CPU アーキテクチャーとしての POWER (ppc64le) の設定

オーバークラウド上のコンピュートノードのデフォルトの CPU アーキテクチャーは x86_64 です。オーバークラウドコンピュートノードを POWER (ppc64le) ハードウェアにデプロイするには、アーキテクチャーを ppc64le に変更できます。



注記

アーキテクチャーに POWER (ppc64le) ノードが含まれている場合、RHOSP 16.1 は PXE ブートのみをサポートします。

手順

1. **undercloud.conf** ファイルで iPXE を無効にします。

```
[DEFAULT]
ipxe_enabled = False
```

注記

この設定により、デプロイメント内の x86_64 ノードも PXE/レガシーモードで起動します。

2. アンダークラウドをインストールします。

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

詳細は、Installing director on the undercloud を参照してください。

3. インストールスクリプトが完了するまで待ちます。

4. オーバークラウドノードのイメージを取得してアップロードします。詳細は、Obtaining images for overcloud nodes を参照してください。

4.9.2. マルチアーキテクチャーオーバークラウドでの Ceph Storage の使用

マルチアーキテクチャークラウドにおいて外部 Ceph へのアクセスを設定する場合には、**CephAnsiblePlaybook** パラメーターを **/usr/share/ceph-ansible/site.yml.sample** に設定し、クライアントキーおよびその他の Ceph 固有パラメーターを含めます。

以下に例を示します。

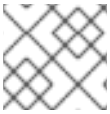
```
parameter_defaults:
```

```
CephAnsiblePlaybook: /usr/share/ceph-ansible/site.yml.sample
CephClientKey: AQDLOh1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==
CephClusterFSID: 4b5c8c0a-ff60-454b-a1b4-9747aa737d19
CephExternalMonHost: 172.16.1.7, 172.16.1.8
```

4.9.3. マルチアーキテクチャーオーバークラウドでのコンポーザブルサービスの使用

一般にコントローラーノードの一部となる以下のサービスは、テクノロジープレビューとしてカスタムロールでの使用が可能です。

- Block Storage サービス (cinder)
- Image サービス (glance)
- Identity サービス (keystone)
- Networking サービス (neutron)
- Object Storage サービス (swift)



注記

Red Hat は、テクノロジープレビューの機能をサポートしていません。

コンポーザブルサービスについての詳しい情報は、**オーバークラウドの高度なカスタマイズ**の コンポーザブルサービスとカスタムロール を参照してください。以下の例を使用して、上記のサービスをコントローラーノードから専用の ppc64le ノードに移動する方法を説明します。

```
(undercloud) [stack@director ~]$ rsync -a /usr/share/openstack-tripleo-heat-templates/. ~/templates
(undercloud) [stack@director ~]$ cd ~/templates/roles
(undercloud) [stack@director roles]$ cat <<EO_TEMPLATE >ControllerPPC64LE.yaml
###############################################################################
# Role: ControllerPPC64LE                                                     #
###############################################################################
- name: ControllerPPC64LE
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  tags:
    - primary
    - controller
  networks:
    - External
    - InternalApi
    - Storage
    - StorageMgmt
    - Tenant
  # For systems with both IPv4 and IPv6, you may specify a gateway network for
  # each, such as ['ControlPlane', 'External']
  default_route_networks: ['External']
  HostnameFormatDefault: '%stackname%-controllerppc64le-%index%'
  ImageDefault: ppc64le-overcloud-full
  ServicesDefault:
    - OS::TripleO::Services::Aide
```

```
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackendDellPs
    - OS::TripleO::Services::CinderBackendDellSc
    - OS::TripleO::Services::CinderBackendDellEMCUnity
    - OS::TripleO::Services::CinderBackendDellEMCVMAXISCSI
    - OS::TripleO::Services::CinderBackendDellEMCVNX
    - OS::TripleO::Services::CinderBackendDellEMCXTREMIOISCSI
    - OS::TripleO::Services::CinderBackendNetApp
    - OS::TripleO::Services::CinderBackendScaleIO
    - OS::TripleO::Services::CinderBackendVRTSHyperScale
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderHPELeftHandISCSI
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronApi
    - OS::TripleO::Services::NeutronBgpVpnApi
    - OS::TripleO::Services::NeutronSfcApi
    - OS::TripleO::Services::NeutronCorePlugin
    - OS::TripleO::Services::NeutronDhcpAgent
    - OS::TripleO::Services::NeutronL2gwAgent
    - OS::TripleO::Services::NeutronL2gwApi
    - OS::TripleO::Services::NeutronL3Agent
    - OS::TripleO::Services::NeutronLbaasv2Agent
    - OS::TripleO::Services::NeutronLbaasv2Api
    - OS::TripleO::Services::NeutronLinuxbridgeAgent
    - OS::TripleO::Services::NeutronMetadataAgent
    - OS::TripleO::Services::NeutronML2FujitsuCfab
    - OS::TripleO::Services::NeutronML2FujitsuFossw
    - OS::TripleO::Services::NeutronOvsAgent
    - OS::TripleO::Services::NeutronVppAgent
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::OpenDaylightOvs
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::SkydiveAgent
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
```

```
    - OS::TripleO::Services::SwiftProxy
    - OS::TripleO::Services::SwiftDispersion
    - OS::TripleO::Services::SwiftRingBuilder
    - OS::TripleO::Services::SwiftStorage
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    - OS::TripleO::Services::Vpp
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::OVNMetadataAgent
    - OS::TripleO::Services::Ptp
EO_TEMPLATE
(undercloud) [stack@director roles]$ sed -i~ -e '/OS::TripleO::Services::\
(Cinder\|Glance\|Swift\|Keystone\|Neutron\)/d' Controller.yaml
(undercloud) [stack@director roles]$ cd ../
(undercloud) [stack@director templates]$ openstack overcloud roles generate \
   --roles-path roles -o roles_data.yaml \
   Controller Compute ComputePPC64LE ControllerPPC64LE BlockStorage ObjectStorage
CephStorage
```

4.10. 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のに、複数のディスクイメージが必要です。

- イントロスペクションカーネルおよび ramdisk: PXE ブートでのベアメタルシステムのイントロスペクション用
- 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用
- オーバークラウドカーネル、ramdisk、完全なイメージで、director がノードのハードディスクに書き込むベースオーバークラウドシステムを形成しています。

CPU アーキテクチャーに基づいて、必要なイメージを取得してインストールできます。他の Red Hat OpenStack Platform (RHOSP) サービスを実行したくない場合、またはサブスクリプションエンタイトルメントの 1 つを使用したくない場合は、basic イメージを取得してインストールし、ベア OS をプロビジョニングすることもできます。

4.10.1. シングル CPU アーキテクチャーのオーバークラウドイメージ

Red Hat OpenStack Platform (RHOSP) のインストールには、director 用に次のオーバークラウドイメージを提供するパッケージが含まれています。

- **overcloud-full**
- **overcloud-full-initrd**
- **overcloud-full-vmlinuz**

CPU アーキテクチャーがデフォルトの x86-64 の場合には、オーバークラウドのデプロイメントに以下のイメージおよび手順が必要です。これらのイメージを director にインポートすると、イントロスペクションイメージも director PXE サーバーにインストールされます。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
[stack@director ~]$ source ~/stackrc
```

3. **rhosp-director-images** および **rhosp-director-images-ipa-x86_64** パッケージをインストールします。

```
(undercloud) [stack@director ~]$ sudo dnf install rhosp-director-images rhosp-director-images-ipa-x86_64
```

4. **stack** ユーザーのホームディレクトリー (**/home/stack/images**) に **images** ディレクトリーを作成します。

```
(undercloud) [stack@director ~]$ mkdir /home/stack/images
```

5. イメージアーカイブを **images** ディレクトリーにデプロイメントします。

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ for i in /usr/share/rhosp-director-images/overcloud-full-latest-16.1.tar /usr/share/rhosp-director-images/ironic-python-agent-latest-16.1.tar; do tar -xvf $i; done
```

6. イメージを director にインポートします。

```
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path /home/stack/images/
```

7. イメージがアップロードされていることを確認します。

```
(undercloud) [stack@director images]$ openstack image list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full         |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd  |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz |
+--------------------------------------+------------------------+
```

8. Director がイントロスペクション PXE イメージを **/var/lib/ironic/httpboot** にコピーしたことを確認します。

```
(undercloud) [stack@director images]$ ls -l /var/lib/ironic/httpboot
total 417296
-rwxr-xr-x. 1 root  root    6639920 Jan 29 14:48 agent.kernel
-rw-r--r--. 1 root  root  420656424 Jan 29 14:48 agent.ramdisk
-rw-r--r--. 1 42422 42422       758 Jan 29 14:29 boot.ipxe
-rw-r--r--. 1 42422 42422       488 Jan 29 14:16 inspector.ipxe
```

4.10.2. 複数の CPU アーキテクチャーのオーバークラウドイメージ

Red Hat OpenStack Platform (RHOSP) のインストールには、デフォルトの CPU アーキテクチャーである x86-64 を使用したオーバークラウドのデプロイに必要な次のイメージを提供するパッケージが含まれています。

- **overcloud-full**
- **overcloud-full-initrd**
- **overcloud-full-vmlinuz**

RHOSP のインストールには、POWER (ppc64le) CPU アーキテクチャーを使用したオーバークラウドのデプロイに必要な次のイメージを提供するパッケージも含まれています。

- **ppc64le-overcloud-full**

これらのイメージを director にインポートすると、イントロスペクションイメージも director PXE サーバーにインストールされます。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
[stack@director ~]$ source ~/stackrc
```

3. **rhosp-director-images-all** パッケージをインストールします。

```
(undercloud) [stack@director ~]$ sudo dnf install rhosp-director-images-all
```

4. アーカイブをアーキテクチャー個別のディレクトリーにデプロイメントします。ここでは、**stack** ユーザーのホームディレクトリー下の **images** ディレクトリー (**/home/stack/images**) です。

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ for arch in x86_64 ppc64le ; do mkdir $arch ; done
(undercloud) [stack@director images]$ for arch in x86_64 ppc64le ; do for i in
/usr/share/rhosp-director-images/overcloud-full-latest-16.1-${arch}.tar /usr/share/rhosp-
director-images/ironic-python-agent-latest-16.1-${arch}.tar ; do tar -C $arch -xf $i ; done ;
done
```

5. イメージを director にインポートします。

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/ppc64le --architecture ppc64le --whole-disk --http-boot
/var/lib/ironic/tftpboot/ppc64le
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/ppc64le --architecture ppc64le --whole-disk --image-type ironic-python-agent --
http-boot /var/lib/ironic/httpboot/ppc64le
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/x86_64/ --architecture x86_64 --http-boot /var/lib/ironic/tftpboot
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path
~/images/x86_64 --architecture x86_64 --image-type ironic-python-agent --http-boot
/var/lib/ironic/httpboot
```

6. イメージがアップロードされていることを確認します。

```
(undercloud) [stack@director images]$ openstack image list
+--------------------------------------+-------------------------+--------+
| ID                                   | Name                    | Status |
+--------------------------------------+-------------------------+--------+
| 6a6096ba-8f79-4343-b77c-4349f7b94960 | overcloud-full          | active |
| de2a1bde-9351-40d2-bbd7-7ce9d6eb50d8 | overcloud-full-initrd   | active |
| 67073533-dd2a-4a95-8e8b-0f108f031092 | overcloud-full-vmlinuz  | active |
| f0fedcd0-3f28-4b44-9c88-619419007a03 | ppc64le-overcloud-full  | active |
+--------------------------------------+-------------------------+--------+
```

7. Director がイントロスペクション PXE イメージを **/var/lib/ironic/tftpboot** にコピーしたことを確認します。

```
(undercloud) [stack@director images]$ ls -l /var/lib/ironic/tftpboot
/var/lib/ironic/tftpboot/ppc64le/
/var/lib/ironic/tftpboot:
total 422624
-rwxr-xr-x. 1 root  root     6385968 Aug  8 19:35 agent.kernel
-rw-r--r--. 1 root  root   425530268 Aug  8 19:35 agent.ramdisk
-rwxr--r--. 1 42422 42422      20832 Aug  8 02:08 chain.c32
-rwxr--r--. 1 42422 42422     715584 Aug  8 02:06 ipxe.efi
-rw-r--r--. 1 root  root          22 Aug  8 02:06 map-file
drwxr-xr-x. 2 42422 42422         62 Aug  8 19:34 ppc64le
-rwxr--r--. 1 42422 42422      26826 Aug  8 02:08 pxelinux.0
drwxr-xr-x. 2 42422 42422         21 Aug  8 02:06 pxelinux.cfg
-rwxr--r--. 1 42422 42422      69631 Aug  8 02:06 undionly.kpxe

/var/lib/ironic/tftpboot/ppc64le/:
total 457204
-rwxr-xr-x. 1 root  root  19858896 Aug  8 19:34 agent.kernel
-rw-r--r--. 1 root  root  448311235 Aug  8 19:34 agent.ramdisk
-rw-r--r--. 1 42422 42422       336 Aug  8 02:06 default
```

4.10.3. 最小限のオーバークラウドイメージ

overcloud-minimal イメージを使用すると、他の Red Hat OpenStack Platform (RHOSP) サービスを実行したり、サブスクリプションエンタイトメントを消費したりしたくないベア OS をプロビジョニングすることが可能です。

RHOSP のインストールには、director 用に次のオーバークラウドイメージを提供する **overcloud-minimal** パッケージが含まれています。

- **overcloud-minimal**
- **overcloud-minimal-initrd**
- **overcloud-minimal-vmlinuz**

注記

デフォルトの **overcloud-full.qcow2** イメージは、フラットなパーティションイメージです。ただし、完全なディスクイメージをインポートして使用することも可能です。詳細は、*24章 完全なディスクイメージの作成* を参照してください。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
[stack@director ~]$ source ~/stackrc
```

3. **overcloud-minimal** パッケージをインストールします。

```
(undercloud) [stack@director ~]$ sudo dnf install rhosp-director-images-minimal
```

4. イメージのアーカイブを、**stack** ユーザーのホームディレクトリー下の **images** ディレクトリー (**/home/stack/images**) にデプロイメントします。

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ tar xf /usr/share/rhosp-director-images/overcloud-minimal-latest-16.1.tar
```

5. イメージを director にインポートします。

```
(undercloud) [stack@director images]$ openstack overcloud image upload --image-path /home/stack/images/ --image-type os --os-image-name overcloud-minimal.qcow2
```

6. イメージがアップロードされていることを確認します。

```
(undercloud) [stack@director images]$ openstack image list
+--------------------------------------+-------------------------+
| ID                                   | Name                    |
+--------------------------------------+-------------------------+
| ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full          |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd   |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz  |
| 32cf6771-b5df-4498-8f02-c3bd8bb93fdd | overcloud-minimal       |
| 600035af-dbbb-4985-8b24-a4e9da149ae5 | overcloud-minimal-initrd |
| d45b0071-8006-472b-bbcc-458899e0d801 | overcloud-minimal-vmlinuz |
+--------------------------------------+-------------------------+
```

4.11. コントロールプレーン用のネームサーバーの設定

オーバークラウドで **cdn.redhat.com** などの外部のホスト名を解決する予定の場合は、オーバークラウドノード上にネームサーバーを設定してください。ネットワークを分離していない標準のオーバークラウドの場合には、ネームサーバーはアンダークラウドのコントロールプレーンのサブネットを使用して定義されます。環境でネームサーバーを定義するには、以下の手順を実施します。

手順

1. source コマンドで **stackrc** ファイルを読み込み、director コマンドラインツールを有効にします。

```
[stack@director ~]$ source ~/stackrc
```

2. **ctlplane-subnet** サブネット用のネームサーバーを設定します。

```
(undercloud) [stack@director images]$ openstack subnet set --dns-nameserver
[nameserver1-ip] --dns-nameserver [nameserver2-ip] ctlplane-subnet
```

各ネームサーバーに **--dns-nameserver** オプションを使用します。

3. サブネットを表示してネームサーバーを確認します。

```
(undercloud) [stack@director images]$ openstack subnet show ctlplane-subnet
+-------------------+-----------------------------------------------+
| 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ファイルの **DnsServers** パラメーターを使用する必要があります。また、コントロールプレーンのネームサーバーと **DnsServers** パラメーターを同じ DNS サーバーに設定する必要があります。

4.12. アンダークラウド設定の更新

新たな要件に合わせて、アンダークラウドの設定を変更する必要がある場合は、該当する設定ファイルを編集し、**openstack undercloud install** コマンドを再度実行して、インストール後のアンダークラウド設定に変更を加えることができます。

手順

1. アンダークラウドの設定ファイルを変更します。以下の例では、**undercloud.conf** ファイルを編集して、有効なハードウェア種別のリストに **idrac** ハードウェア種別を追加しています。

```
enabled_hardware_types = ipmi,redfish,idrac
```

2. **openstack undercloud install** コマンドを実行し、新たな変更を反映させてアンダークラウドを更新します。

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

コマンドの実行が完了するまで待ちます。

3. **stack** ユーザーを初期化し、コマンドラインツールを使用します。

```
[stack@director ~]$ source ~/stackrc
```

プロンプトには、OpenStack コマンドがアンダークラウドに対して認証および実行されることが表示されるようになります。

```
(undercloud) [stack@director ~]$
```

4. director が新しい設定を適用していることを確認します。この例では、有効なハードウェア種別のリストを確認しています。

```
(undercloud) [stack@director ~]$ openstack baremetal driver list
+---------------------+---------------------+
| Supported driver(s) | Active host(s)      |
+---------------------+---------------------+
| idrac               | director.example.com |
| ipmi                | director.example.com |
| redfish             | director.example.com |
+---------------------+---------------------+
```

アンダークラウドの再設定が完了しました。

4.13. アンダークラウドのコンテナーレジストリー

Red Hat Enterprise Linux 8.2 には、Docker Registry v2 をインストールするための **docker-distribution** パッケージが含まれなくなりました。互換性および同じ機能レベルを維持するために、director のインストールでは Apache Web サーバーおよび **image-serve** という仮想ホストが作成され、これによりレジストリーが提供されます。このレジストリーでも、SSL を無効にしたポート 8787/TCP が使用されます。Apache ベースのレジストリーはコンテナー化されていません。したがって、以下のコマンドを実行してレジストリーを再起動する必要があります。

```
$ sudo systemctl restart httpd
```

コンテナーレジストリーのログは、以下の場所に保存されます。

- /var/log/httpd/image_serve_access.log
- /var/log/httpd/image_serve_error.log.

イメージのコンテンツは、**/var/lib/image-serve** から提供されます。この場所では、レジストリー REST API のプル機能を実装するために、特定のディレクトリーレイアウトおよび **apache** 設定が使用されています。

Apache ベースのレジストリーでは、**podman push** コマンドも **buildah push** コマンドもサポートされません。つまり、従来の方法を使用してコンテナーイメージをプッシュすることはできません。デプロイ中にイメージを変更するには、**ContainerImagePrepare** パラメーターなどのコンテナー準備ワークフローを使用します。コンテナーイメージを管理するには、コンテナー管理コマンドを使用します。

openstack tripleo container image list
　レジストリーに保存されているすべてのイメージをリスト表示します。

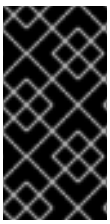
openstack tripleo container image show
　レジストリーの特定イメージのメタデータを表示します。

openstack tripleo container image push
　イメージをリモートレジストリーからアンダークラウドレジストリーにプッシュします。

openstack tripleo container image delete
　レジストリーからイメージを削除します。

第5章 アンダークラウドミニオンのインストール

追加のアンダークラウドミニオンをデプロイして、OpenStack Platform director のサービスを複数ホストにわたってスケーリングすることができます。これにより、大規模なオーバークラウドをデプロイする際にパフォーマンスが向上します。この機能はオプションです。



重要

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

5.1. アンダークラウドミニオン

アンダークラウドミニオンにより、別のホスト上に **heat-engine** サービスおよび **ironic-conductor** サービスが追加されます。これらの追加サービスは、アンダークラウドのオーケストレーションおよびプロビジョニング操作をサポートします。アンダークラウドの操作を複数ホスト間に分散することにより、オーバークラウドのデプロイメントにより多くのリソースを割り当てることができ、結果として大規模なデプロイメントをより迅速に実施することができます。

5.2. アンダークラウドミニオンの要件

サービスの要件

スケーリングされたアンダークラウドミニオン上の **heat-engine** サービスおよび **ironic-conductor** サービスは、ワーカーのセットを使用します。それぞれのワーカーは、そのサービスに固有の操作を実行します。複数のワーカーを用いると、同時に操作を実行することができます。ミニオンのデフォルトのワーカー数は、ミニオンホストの合計 CPU スレッド数の半分です。ここでは、合計スレッド数とは CPU コア数にハイパースレッディングの値を掛けたものを指します。たとえば、ミニオンの CPU スレッド数が 16 の場合には、デフォルトでは、ミニオンによりサービスごとに 8 つのワーカーが提供されます。デフォルトでは、ミニオンのサービスに最小および最大のワーカー数も適用されます。

サービス	最小値	最大値
heat-engine	4	24
ironic-conductor	2	12

アンダークラウドミニオンの CPU およびメモリーの最低要件を以下に示します。

- Intel 64 または AMD64 CPU 拡張機能をサポートする、8 スレッド 64 ビット x86 プロセッサー。このプロセッサーにより、各アンダークラウドサービスに 4 つのワーカーが提供されます。
- 最小 16 GB の RAM

多数のワーカーを使用するには、CPU スレッドごとに 2 GB の RAM の比率で、アンダークラウド上の仮想 CPU 数およびメモリー容量を増やします。たとえば、48 スレッドのマシンには 96 GB の RAM が必要です。これにより、**heat-engine** 用 24 ワーカーおよび **ironic-conductor** 用 12 ワーカーが提供されます。

コンテナーイメージの要件

アンダークラウドミニオンは、内部コンテナーイメージレジストリーをホストしません。したがって、以下のいずれかの方法を使用してコンテナーイメージを取得するようにミニオンを設定する必要があります。

- イメージを Red Hat Container Image Registry (**registry.redhat.io**) から直接プルします。
- Red Hat Satellite Server でホストするイメージをプルする。

どちらの方法でも、**containers-prepare-parameter.yaml** ファイルの **ContainerImagePrepare** heat パラメーターの一部として、**push_destination: false** を設定する必要があります。

5.3. ミニオンの準備

ミニオンをインストールする前に、ホストマシンでいくつかの基本設定を完了する必要があります。

- コマンドを実行するための非 root ユーザー
- 解決可能なホスト名
- Red Hat サブスクリプション
- イメージの準備およびミニオンのインストールを行うためのコマンドラインツール

手順

1. ミニオンホストに **root** ユーザーとしてログインします。
2. **stack** ユーザーを作成します。

```
[root@minion ~]# useradd stack
```

3. **stack** ユーザーのパスワードを設定します。

```
[root@minion ~]# passwd stack
```

4. **sudo** を使用する場合にパスワードを要求されないようにします。

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

5. 新規作成した **stack** ユーザーに切り替えます。

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

6. ミニオンのベースおよび完全なホスト名を確認します。

```
[stack@minion ~]$ hostname
[stack@minion ~]$ hostname -f
```

上記のコマンドのいずれかで正しい完全修飾ホスト名が出力されない、またはエラーが表示される場合には、**hostnamectl** でホスト名を設定します。

```
[stack@minion ~]$ sudo hostnamectl set-hostname minion.example.com
[stack@minion ~]$ sudo hostnamectl set-hostname --transient minion.example.com
```

7. **/etc/hosts** ファイルを編集して、システムホスト名のエントリーを追加します。たとえば、システムの名前が **minion.example.com** で、IP アドレスに **10.0.0.1** を使用する場合には、**/etc/hosts** ファイルに以下の行を追加します。

```
10.0.0.1  minion.example.com manager
```

8. Red Hat コンテンツ配信ネットワークまたは Red Hat Satellite のどちらかにシステムを登録します。たとえば、システムをコンテンツ配信ネットワークに登録するには、以下のコマンドを実行します。要求されたら、カスタマーポータルのユーザー名およびパスワードを入力します。

```
[stack@minion ~]$ sudo subscription-manager register
```

9. Red Hat OpenStack Platform (RHOSP) director のエンタイトルメントプール ID を検索します。

```
[stack@minion ~]$ sudo subscription-manager list --available --all --matches="Red Hat
OpenStack"
Subscription Name:   Name of SKU
Provides:            Red Hat Single Sign-On
                     Red Hat Enterprise Linux Workstation
                     Red Hat CloudForms
                     Red Hat OpenStack
                     Red Hat Software Collections (for RHEL Workstation)
                     Red Hat Virtualization
SKU:                 SKU-Number
Contract:            Contract-Number
Pool ID:             Valid-Pool-Number-123456
Provides Management: Yes
Available:           1
Suggested:           1
Service Level:       Support-level
Service Type:        Service-Type
Subscription Type:   Sub-type
Ends:                End-date
System Type:         Physical
```

10. **Pool ID** の値を特定して、Red Hat OpenStack Platform 16.1 のエンタイトルメントをアタッチします。

```
[stack@minion ~]$ sudo subscription-manager attach --pool=Valid-Pool-Number-123456
```

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

```
[stack@minion ~]$ sudo subscription-manager repos --disable=*
[stack@minion ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-
eus-rpms --enable=rhel-8-for-x86_64-appstream-eus-rpms --enable=rhel-8-for-x86_64-
highavailability-eus-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms --enable=openstack-
16.1-for-rhel-8-x86_64-rpms --enable=fast-datapath-for-rhel-8-x86_64-rpms
```

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

12. 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。

```
[stack@minion ~]$ sudo dnf update -y
[stack@minion ~]$ sudo reboot
```

13. ミニオンのインストールと設定を行うためのコマンドラインツールをインストールします。

```
[stack@minion ~]$ sudo dnf install -y python3-tripleoclient
```

5.4. アンダークラウド設定ファイルのミニオンへのコピー

ミニオンには、アンダークラウドからの設定ファイルがいくつか必要です。これにより、ミニオンのインストールでミニオンサービスを設定し、それらを director に登録することができます。

- **tripleo-undercloud-outputs.yaml**
- **tripleo-undercloud-passwords.yaml**

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. ファイルをアンダークラウドからミニオンにコピーします。

```
$ scp ~/tripleo-undercloud-outputs.yaml ~/tripleo-undercloud-passwords.yaml
stack@<minion-host>:~/.
```

   - **<minion-host>** は、ミニオンのホスト名または IP アドレスに置き換えます。

5.5. アンダークラウドの認証局のコピー

アンダークラウドがエンドポイントの暗号化に SSL/TLS を使用する場合は、ミニオンホストにアンダークラウドの SSL/TLS 証明書に署名した認証局が含まれている必要があります。アンダークラウドの設定により、この認証局は以下のいずれかになります。

- ミニオンホストに事前に証明書を読み込む外部の認証局。対応の必要はありません。
- director が **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** に生成する自己署名認証局。このファイルをミニオンホストにコピーし、ファイルをミニオンホストの信頼済み認証局の一部として追加します。この手順では、このファイルを例として使用します。
- OpenSSL で作成するカスタムの自己署名認証局。本書の例では、このファイルを **ca.crt.pem** と呼びます。このファイルをミニオンホストにコピーし、ファイルをミニオンホストの信頼済み認証局の一部として追加します。

手順

1. ミニオンホストに **root** ユーザーとしてログインします。
2. 認証局ファイルをアンダークラウドからミニオンにコピーします。

```
[root@minion ~]# scp \
  root@<undercloud-host>:/etc/pki/ca-trust/source/anchors/cm-local-ca.pem \
  /etc/pki/ca-trust/source/anchors/undercloud-ca.pem
```

- **<undercloud-host>** は、アンダークラウドのホスト名または IP アドレスに置き換えます。

3. ミニオンホストの信頼済み認証局を更新します。

```
[root@minion ~]# update-ca-trust enable
[root@minion ~]# update-ca-trust extract
```

5.6. ミニオンの設定

ミニオンのインストールプロセスでは、ミニオンが **stack** ユーザーのホームディレクトリーから読み取る **minion.conf** 設定ファイルに、特定の設定が必要になります。デフォルトのテンプレートを設定のベースとして使用するには、以下の手順を実施します。

手順

1. ミニオンホストに **stack** ユーザーとしてログインします。
2. デフォルトのテンプレートを **stack** ユーザーのホームディレクトリーにコピーします。

```
[stack@minion ~]$ cp \
  /usr/share/python-tripleoclient/minion.conf.sample \
  ~/minion.conf
```

3. **minion.conf** ファイルを編集します。このファイルには、ミニオンを設定するためのパラメーターが含まれています。パラメーターを省略したり、コメントアウトした場合には、ミニオンのインストールでデフォルト値が使用されます。以下の推奨パラメーターを確認してください。
   - **minion_hostname**: ミニオンのホスト名に設定します。
   - **minion_local_interface**: プロビジョニングネットワークを介してアンダーグラウンドに接続するインターフェイスに設定します。
   - **minion_local_ip**: プロビジョニングネットワークのフリー IP アドレスに設定します。
   - **minion_nameservers**: ミニオンがホスト名を解決できるように DNS ネームサーバーに設定します。
   - **enable_ironic_conductor**: **ironic-conductor** サービスを有効にするかどうかを定義します。
   - **enable_heat_engine**: **heat-engine** サービスを有効にするかどうかを定義します。

注記

デフォルトの **minion.conf** ファイルでは、ミニオンの **heat-engine** サービスだけが有効になります。**ironic-conductor** サービスを有効にするには、**enable_ironic_conductor** パラメーターを **true** に設定します。

5.7. ミニオンの設定パラメーター

以下のリストで、**minion.conf** ファイルを設定するパラメーターについて説明します。エラーを避けるために、パラメーターは決して該当するセクションから削除しないでください。

デフォルト

minion.conf ファイルの **[DEFAULT]** セクションで定義されているパラメーターを以下に示します。

cleanup
: 一時ファイルをクリーンナップします。コマンド実行後もデプロイメント時に使用した一時ファイルをそのまま残すには、このパラメーターを **False** に設定します。ファイルを残すと、生成されたファイルのデバッグを行う場合やエラーが発生した場合に役に立ちます。

container_cli
: コンテナー管理用の CLI ツール。このパラメーターは、**podman** に設定したままにしてください。Red Hat Enterprise Linux 8.2 がサポートするのは **podman** だけです。

container_healthcheck_disabled
: コンテナー化されたサービスのヘルスチェックを無効にします。Red Hat は、ヘルスチェックを有効にし、このオプションを **false** に設定したままにすることを推奨します。

container_images_file
: コンテナーイメージ情報が含まれる heat 環境ファイル。このファイルには、以下のエントリーを含めることができます。

  - 必要なすべてのコンテナーイメージのパラメーター
  - 必要なイメージの準備を実施する **ContainerImagePrepare** パラメーター。このパラメーターが含まれるファイルの名前は、通常 **containers-prepare-parameter.yaml** です。

container_insecure_registries
: **podman** が使用するセキュアではないレジストリーのリスト。プライベートコンテナーレジストリー等の別のソースからイメージをプルする場合には、このパラメーターを使用します。多くの場合、**podman** は Red Hat Container Catalog または Satellite サーバー (ミニオンが Satellite に登録されている場合) のいずれかからコンテナーイメージをプルするための証明書を持ちます。

container_registry_mirror
: 設定により **podman** が使用するオプションの **registry-mirror**

custom_env_files
: ミニオンのインストールに追加する新たな環境ファイル

deployment_user
: ミニオンをインストールするユーザー。現在のデフォルトユーザー **stack** を使用する場合には、このパラメーターを未設定のままにします。

enable_heat_engine
: ミニオンに heat engine サービスをインストールするかどうかを定義します。デフォルトは **true** です。

enable_ironic_conductor
: ミニオンに ironic conductor サービスをインストールするかどうかを定義します。デフォルト値は **false** です。ironic conductor サービスを有効にするには、この値を **true** に設定します。

heat_container_image
: 使用する heat コンテナーイメージの URL。未設定のままにします。

heat_native
: ネイティブの heat テンプレートを使用します。**true** のままにします。

hieradata_override

director に Puppet hieradata を設定するための **hieradata** オーバーライドファイルへのパス。これにより、サービスに対して **minion.conf** パラメーター以外のカスタム設定を行うことができます。設定すると、ミニオンのインストールでこのファイルが **/etc/puppet/hieradata** ディレクトリーにコピーされ、階層の最初のファイルに設定されます。

minion_debug

ミニオンサービスの **DEBUG** ログレベルを有効にするには、この値を **true** に設定します。

minion_enable_selinux

デプロイメント時に、SELinux を有効または無効にします。問題をデバッグする場合以外は、この値を **true** に設定したままにすることを強く推奨します。

minion_enable_validations

minion で検証サービスを有効にします。

minion_hostname

ミニオンの完全修飾ホスト名を定義します。設定すると、ミニオンのインストールで全システムのホスト名が設定されます。未設定のままにすると、ミニオンは現在のホスト名を使用しますが、システムのホスト名設定をすべて適切に定義する必要があります。

minion_local_interface

アンダークラウドのプロビジョニング NIC 用に選択するインターフェイス。ミニオンは、DHCP および PXE ブートサービスにもこのデバイスを使用します。この値を選択したデバイスに変更します。接続されているデバイスを確認するには、**ip addr** コマンドを使用します。**ip addr** コマンドの出力結果の例を、以下に示します。

```
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、このインターフェイスが **inspection_interface** パラメーターで定義したカスタムのブリッジにアタッチされます。

minion_local_ip

アンダークラウドのプロビジョニング NIC 用に定義する IP アドレス。ミニオンは、DHCP および PXE ブートサービスにもこの IP アドレスを使用します。デフォルトの IP アドレスが環境内の既存の IP アドレスまたはサブネットと競合するなどの理由により、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、この値をデフォルトの **192.168.24.1/24** のままにします。

minion_local_mtu

local_interface に使用する最大伝送単位 (MTU)。ミニオンでは 1500 以下にします。

minion_log_file

ミニオンのインストールログおよびアップグレードログを保管するログファイルへのパス。デフォルトでは、ログファイルはホームディレクトリー内の **install-minion.log** です。たとえば、**/home/stack/install-minion.log** のようになります。

minion_nameservers

ミニオンのホスト名解決に使用する DNS ネームサーバーのリスト

minion_ntp_servers

ミニオンの日付と時刻を同期できるようにする Network Time Protocol サーバーのリスト

minion_password_file

ミニオンがアンダークラウドサービスに接続するためのパスワードが含まれるファイル。このパラメーターは、アンダークラウドからコピーした **tripleo-undercloud-passwords.yaml** ファイルに設定したままにしておきます。

minion_service_certificate

OpenStack SSL/TLS 通信の証明書の場所とファイル名。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、独自の自己署名の証明書を生成します。

minion_timezone

ミニオン用ホストのタイムゾーン。タイムゾーンを指定しなければ、ミニオンは既存のタイムゾーン設定を使用します。

minion_undercloud_output_file

ミニオンがアンダークラウドサービスに接続するのに使用できるアンダークラウド設定情報が含まれるファイル。このパラメーターは、アンダークラウドからコピーした **tripleo-undercloud-outputs.yaml** ファイルに設定したままにします。

net_config_override

ネットワーク設定のオーバーライドテンプレートへのパス。このパラメーターを設定すると、ミニオンは JSON 形式のテンプレートを使用して **os-net-config** でネットワークを設定し、**minion.conf** で設定したネットワークパラメーターを無視します。**/usr/share/python-tripleoclient/minion.conf.sample** の例を参照してください。

networks_file

heat をオーバーライドするネットワークファイル

output_dir

状態、処理された heat テンプレート、および Ansible デプロイメントファイルを出力するディレクトリー

roles_file

ミニオンのインストールで、デフォルトロールファイルを上書きするのに使用するロールファイル。ミニオンのインストールにデフォルトのロールファイルが使用されるように、このパラメーターは未設定のままにすることを強く推奨します。

templates

上書きする heat テンプレートファイル

5.8. ミニオンのインストール

ミニオンをインストールするには、以下の手順を実施します。

手順

1. ミニオンホストに **stack** ユーザーとしてログインします。
2. 以下のコマンドを実行して、ミニオンをインストールします。

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

このコマンドによりミニオンの設定スクリプトが起動し、追加のパッケージがインストールされ、**minion.conf** ファイルの設定に応じてミニオンサービスが設定されます。このスクリプトは、完了までに数分かかります。

5.9. ミニオンのインストールの検証

ミニオンのインストールが正常に行われたことを確認するには、以下の手順を実施します。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
[stack@director ~]$ source ~/stackrc
```

3. ミニオンで heat engine サービスを有効にしている場合には、ミニオンからの **heat-engine** サービスがアンダークラウドサービスのリストに表示されることを確認します。

```
[stack@director ~]$ $ openstack orchestration service list
```

このコマンド出力には、アンダークラウドとミニオン両方の **heat-engine** ワーカーが記載された表が表示されます。

4. ミニオンで ironic conductor サービスを有効にしている場合には、ミニオンからの **ironic-conductor** サービスがアンダークラウドサービスのリストに表示されることを確認します。

```
[stack@director ~]$ $ openstack baremetal conductor list
```

このコマンド出力には、アンダークラウドとミニオン両方の **ironic-conductor** ワーカーが記載された表が表示されます。

第6章 オーバークラウドのプランニング

以下の項で、Red Hat OpenStack Platform (RHOSP) 環境のさまざまな要素をプランニングする際のガイドラインを説明します。これには、ノードロールの定義、ネットワークトポロジーのプランニング、ストレージなどが含まれます。

重要

デプロイ後は、オーバークラウドノードの名前を変更しないでください。デプロイメント後にノードの名前を変更すると、インスタンスの管理に問題が生じます。

6.1. ノードロール

director には、オーバークラウドを作成するために、以下に示すデフォルトノード種別が含まれます。

コントローラー

環境を制御するための主要なサービスを提供します。これには、Dashboard (horizon)、認証 (keystone)、イメージストレージ (glance)、ネットワーク (neutron)、オーケストレーション (heat)、高可用性サービスが含まれます。高可用性に対応した実稼働レベルの環境の場合は、Red Hat OpenStack Platform (RHOSP) 環境にコントローラーノードが 3 台必要です。

注記

1 台のコントローラーノードで設定される環境は、実稼働用ではなくテスト目的にのみ使用してください。2 台のコントローラーノードまたは 4 台以上のコントローラーノードで設定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機能を持つ物理サーバー。基本的な RHOSP 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。

Swift Storage

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

以下の表には、オーバークラウドの設定例と各シナリオで使用するノード種別の定義をまとめています。

表6.1 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	合計
小規模のオーバークラウド	3	1	-	-	4
中規模のオーバークラウド	3	3	-	-	6

追加のオブジェクトストレージがある中規模のオーバークラウド	3	3	-	3	9
Ceph Storage クラスターがある中規模のオーバークラウド	3	3	3	-	9

さらに、個別のサービスをカスタムのロールに分割するかどうかを検討します。コンポーザブルロールのアーキテクチャーについての詳しい情報は、**オーバークラウドの高度なカスタマイズ**の コンポーザブルサービスとカスタムロール を参照してください。

表6.2 概念検証用デプロイメントに使用するノードデプロイメントロール

	アンダークラウド	コントローラー	コンピュート	Ceph Storage	合計
概念実証	1	1	1	1	4

警告

Red Hat OpenStack Platform は、Day 2 操作中、稼働状態にある Ceph Storage クラスターを維持します。そのため、MON ノードまたはストレージノードの数が 3 未満のデプロイメントでは、一部のDay 2 操作 (Ceph Storage クラスターのアップグレードまたはマイナー更新等) を行うことができません。単一のコントローラーノードまたは単一の Ceph Storage ノードを使用している場合は、Day 2 操作に失敗します。

6.2. オーバークラウドネットワーク

ロールとサービスを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ーおよびサブネットのプランニングを行うことが重要です。Red Hat OpenStack Platform (RHOSP) 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 Openstack Networking (neutron) サービスを使用します。

デフォルトでは、director は接続に **プロビジョニング/コントロールプレーン** を使用するようにノードを設定します。ただし、ネットワークトラフィックを一連のコンポーザブルネットワークに分離し、カスタマイズしてサービスを割り当てることができます。

一般的な RHOSP 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ために、オーバークラウドは VLAN タグ付けを使用して、それぞれのインターフェイスに複数のネットワークを提供します。ネットワークの多くは独立したサブネットですが、一部のネットワークには、インターネットアクセスまたはインフラストラク

チャーにネットワーク接続ができるようにルーティングを提供するレイヤー 3 のゲートウェイが必要です。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 を提供する必要があります。

注記

デプロイ時にトンネリングを無効にした neutron VLAN モード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 Tenant ネットワークを作成することは変わりませんが、Tenant 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、Tenant 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 Tenant 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 には、NIC を分離コンポーザブルネットワークと連携させるのに使用できるテンプレートセットも含まれています。デフォルトの設定は以下のとおりです。

- シングル NIC 設定: ネイティブ VLAN 上のプロビジョニングネットワークと、オーバークラウドネットワークの種別ごとのサブネットを使用するタグ付けされた VLAN 用に NIC を 1 つ。
- ボンディングされた NIC 設定: ネイティブ VLAN 上のプロビジョニングネットワーク用に NIC を 1 つと、オーバークラウドネットワークの種別ごとのタグ付けされた VLAN 用にボンディング設定の 2 つの NIC。
- 複数 NIC 設定 - 各 NIC は、オーバークラウドネットワークの種別ごとのサブセットを使用します。

専用のテンプレートを作成して、特定の NIC 設定をマッピングすることもできます。

ネットワーク設定を検討する上で、以下の点を考慮することも重要です。

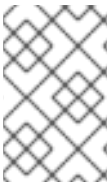
- オーバークラウドの作成時には、全オーバークラウドマシンで 1 つの名前を使用して NIC を参照します。理想としては、混乱を避けるため、対象のネットワークごとに、各オーバークラウドノードで同じ NIC を使用してください。たとえば、プロビジョニングネットワークにはプライマリー NIC を使用して、OpenStack サービスにはセカンダリー NIC を使用します。
- すべてのオーバークラウドシステムをプロビジョニング NIC から PXE ブートするように設定して、同システム上の外部 NIC およびその他の NIC の PXE ブートを無効にします。また、プロビジョニング NIC の PXE 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、ブート順序の最上位に指定するようにします。
- director が各ノードの電源管理を制御できるように、すべてのオーバークラウドベアメタルシステムには、Intelligent Platform Management Interface (IPMI) などのサポート対象の電源管理インターフェイスが必要です。
- 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、後でオーバークラウドノードを設定する際に役立ちます。
- 外部のインターネットからインスタンスにアクセス可能でなければならない場合、パブリックネットワークから Floating IP アドレスを確保して、その Floating IP アドレスをインスタンスに割り当てることができます。インスタンスはプライベートの IP アドレスを確保しますが、ネットワークトラフィックは NAT を使用して、Floating IP アドレスに到達します。Floating IP アドレスは、複数のプライベート IP アドレスではなく、単一のインスタンスにのみ割り当て可

能である点に注意してください。ただし、Floating IP アドレスは、単一のテナントでのみ使用するように確保されます。したがって、そのテナントは必要に応じて Floating IP アドレスを特定のインスタンスに割り当てまたは割り当てを解除することができます。この設定では、お使いのインフラストラクチャーが外部のインターネットに公開されるので、適切なセキュリティー確保手段に従う必要があります。

- あるブリッジのメンバーを単一のインターフェイスまたは単一のボンディングだけにすると、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イ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。

- Red Hat では、オーバークラウドノードが Red Hat コンテンツ配信ネットワークやネットワークタイムサーバーなどの外部のサービスに接続できるように、DNS によるホスト名解決を使用することを推奨します。

- Red Hat では、プロビジョニングインターフェイス、外部インターフェイス、Floating IP インターフェイスの 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を転送できないためです。



注記

Red Hat Virtualization (RHV) を使用している場合には、オーバークラウドのコントロールプレーンを仮想化することができます。詳細は、Creating virtualized control planes を参照してください。

6.3. オーバークラウドのストレージ

注記

任意のドライバーまたはバックエンド種別のバックエンド cinder ボリュームを使用するゲストインスタンスで LVM を使用すると、パフォーマンス、ボリュームの可視性/可用性、およびデータ破損の問題が生じます。可視性、可用性、およびデータ破損の問題を軽減するには、LVM フィルターを使用します。詳しい情報は、**Storage Guide** の section 2 Block Storage and Volumes と KCS アーティクル 3213311 "Using LVM on a cinder volume exposes the data to the compute host." を参照してください。

director には、オーバークラウド環境用にさまざまなストレージオプションが含まれています。

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、以下のストレージ種別にこのノードを使用します。

- **イメージ**: Image サービス (glance) は仮想マシンのイメージを管理します。イメージを変更することはできません。OpenStack はイメージバイナリーブロブとして処理し、それに応じてイメージをダウンロードします。Image サービス (glance) を使用して、Ceph ブロックデバイスにイメージを保管することができます。

- **ボリューム**: OpenStack は Block Storage サービス (cinder) を使用してボリュームを管理します。Block Storage サービス (cinder) ボリュームはブロックデバイスです。OpenStack では、ボリュームを使用して仮想マシンをブートしたり、ボリュームを実行中の仮想マシンにアタッチしたりします。Block Storage サービスを使用して、イメージの Copy-on-Write クローンで仮想マシンをブートすることができます。

- **ファイルシステム**: OpenStack は Shared File Systems サービス (manila) を使用して共有ファイルシステムを管理します。ファイル共有は、ファイルシステムによりバッキングされます。manila を使用して、Ceph Storage ノードにデータを保管する CephFS ファイルシステムにバッキングされる共有を管理することができます。

- **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、Compute サービス (nova) で仮想マシンをブートすると、仮想マシンの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>/** 内)。Ceph 内にあるすべての仮想マシンは、Block Storage サービス (cinder) を使用せずにブートすることができます。これにより、ライブマイグレーションのプロセスを使用して、簡単にメンテナンス操作を実施することができます。また、ハイパーバイザーに障害が発生した場合には、**nova evacuate** をトリガーして仮想マシンを別の場所で実行することもできるので便利です。

重要

サポートされるイメージ形式の詳細は、**Creating and Managing Images** ガイドの Image Service を参照してください。

Ceph Storage についての詳しい情報は、Red Hat Ceph Storage アーキテクチャーガイド を参照してください。

Swift Storage ノード

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

6.4. オーバークラウドのセキュリティー

OpenStack Platform の実装のセキュリティーレベルは、お使いの環境のセキュリティーレベルと同等でしかありません。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を正しく制御するようにします。

- 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およびパスワードの使用を徹底してください。
- 必ず SELinux を有効にします。

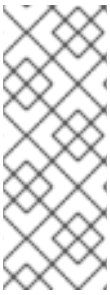
システムのセキュリティー保護についての詳細は、以下の Red Hat ガイドを参照してください。

- Red Hat Enterprise Linux 8 の セキュリティーの強化
- Red Hat Enterprise Linux 8 の SELinux の使用

6.5. オーバークラウドの高可用性

高可用性なオーバークラウドをデプロイするために、director は複数のコントローラー、コンピュート、およびストレージノードを単一のクラスターとして連携するように設定します。ノードで障害が発生すると、障害が発生したノードのタイプに応じて、自動フェンシングおよび再起動プロセスがトリ

ガーされます。オーバークラウドの高可用性アーキテクチャーおよびサービスに関する情報は、高可用性デプロイメントと使用方法 を参照してください。



注記

STONITH を使用しない高可用性オーバークラウドのデプロイはサポートの対象外です。高可用性オーバークラウドの Pacemaker クラスターの一部である各ノードには、STONITH デバイスを設定する必要があります。STONITH および Pacemaker の詳細は、Fencing in a Red Hat High Availability Cluster および Support Policies for RHEL High Availability Clusters - General Requirements for Fencing/STONITH を参照してください。

director を使用して、コンピュートインスタンスの高可用性 (インスタンス HA) を設定することもできます。この高可用性のメカニズムにより、ノードで障害が発生するとコンピュートノード上のインスタンスが自動的に退避および再起動されます。インスタンス HA に対する要件は通常のオーバークラウドの要件と同じですが、環境をデプロイメント用に準備するために追加のステップを実施する必要があります。インスタンス HA およびそのインストール手順についての情報は、コンピュートインスタンスの高可用性 を参照してください。

6.6. コントローラーノードの要件

コントローラーノードは、Red Hat OpenStack Platform 環境の中核となるサービス (例: Dashboard (horizon)、バックエンドのデータベースサーバー、Identity サービス (keystone) の認証、および高可用性サービスなど) をホストします。

プロセッサー

Intel 64 または AMD64 CPU 拡張機能をサポートする 64 ビット x86 プロセッサー。

メモリー

最小のメモリー容量は 32 GB です。ただし、推奨のメモリー容量は、仮想 CPU の数 (CPU コアの数をハイパースレッディングの値で乗算した数値に基づく) によって異なります。以下の計算により、RAM の要件を決定します。

- **コントローラーの最小メモリー容量の算出:**
  - 各仮想 CPU ごとに 1.5 GB のメモリーを使用します。たとえば、仮想 CPU が 48 個あるマシンにはメモリーは 72 GB 必要です。
- **コントローラーの推奨メモリー容量の算出:**
  - 各仮想 CPU ごとに 3 GB のメモリーを使用します。たとえば、仮想 CPU が 48 個あるマシンにはメモリーは 144 GB 必要です。

メモリーの要件に関する詳しい情報は、Red Hat カスタマーポータルで Red Hat OpenStack Platform Hardware Requirements for Highly Available Controllers を参照してください。

ディスクストレージとレイアウト

Object Storage サービス (swift) がコントローラーノード上で実行されていない場合には、最小で 50 GB のストレージが必要です。ただし、Telemetry および Object Storage サービスはいずれもコントローラーにインストールされ、ルートディスクを使用するように設定されます。これらのデフォルトは、市販のハードウェア上に構築される小型のオーバークラウドのデプロイに適しています。これらの環境は、概念検証およびテストの標準的な環境です。これらのデフォルトを使用すれば、最小限のプランニングでオーバークラウドをデプロイすることができますが、ワークロードキャパシティーとパフォーマンスの面ではあまり優れていません。
ただし、Telemetry が絶えずストレージにアクセスするため、エンタープライズ環境の場合、デフォ

ルト設定では大きなボトルネックが生じる可能性があります。これにより、ディスク I/O が過度に使用されて、その他すべてのコントローラーサービスに深刻な影響をもたらします。このタイプの環境では、オーバークラウドのプランニングを行って、適切に設定する必要があります。

Red Hat は、Telemetry と Object Storage の両方の推奨設定をいくつか提供しています。詳しくは、Deployment Recommendations for Specific Red Hat OpenStack Platform Services を参照してください。

ネットワークインターフェイスカード

最小 2 枚の 1Gbps ネットワークインターフェイスカード。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イス向けには、追加のネットワークインターフェイスを使用します。

電源管理

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

仮想化のサポート

Red Hat では、Red Hat Virtualization プラットフォーム上の仮想コントローラーノードのみをサポートします。詳細は、Creating virtualized control planes を参照してください。

6.7. 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ロールを果たします。コンピュートノードには、ハードウェアの仮想化をサポートするベアメタルシステムが必要です。また、ホストする仮想マシンインスタンスの要件をサポートするのに十分なメモリーとディスク容量も必要です。

プロセッサー

- Intel 64 または AMD64 CPU 拡張機能をサポートする 64 ビット x86 プロセッサーで、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。このプロセッサーには最小でも 4 つのコアが搭載されていることを推奨しています。
- IBM POWER 8 プロセッサー

メモリー

ホストオペレーティングシステム用に最低 6GB の RAM と、次の考慮事項に対応するための追加メモリー。

- 仮想マシンインスタンスで使用できるようにするメモリーを追加します。
- メモリーを追加して、追加のカーネルモジュール、仮想スイッチ、モニターソリューション、その他の追加のバックグラウンドタスクなど、ホスト上で特別な機能や追加のリソースを実行します。
- Non-Uniform Memory Access (NUMA) を使用する場合、Red Hat は CPU ソケットノードあたり 8 GB、または 256 GB を超える物理 RAM がある場合はソケットノードあたり 16 GB を推奨します。
- 少なくとも 4 GB のスワップスペースを設定します。

ディスク容量

最小 50 GB の空きディスク領域

ネットワークインターフェイスカード

最小 1 枚の 1 Gbps ネットワークインターフェイスカード (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イス向けには、追加のネットワークインターフェイスを使用します。

電源管理

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

6.8. CEPH STORAGE ノードの要件

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

Ceph Storage ノードのプロセッサー、メモリー、ネットワークインターフェイスカード (NIC)、およびディスクレイアウトを選択する方法の情報は、**Red Hat Ceph Storage ハードウェアガイド**で Red Hat Ceph Storage におけるハードウェア選択に関する推奨事項 を確認してください。各 Ceph Storage ノードにも、Intelligent Platform Management Interface (IPMI) 機能などのサポート対象の電源管理インターフェイスがサーバーのマザーボードに搭載されている必要があります。

注記

Red Hat OpenStack Platform (RHOSP) director は **ceph-ansible** を使用しますが、Ceph Storage ノードのルートディスクへの OSD インストールには対応しません。したがって、サポートされる Ceph Storage ノードには少なくとも 2 つのディスクが必要になります。

Ceph Storage ノードと RHEL の互換性

- RHOSP 16.1 は RHEL 8.2 でサポートされています。ただし、Ceph Storage ロールにマップされているホストは、最新のメジャー RHEL リリースに更新されます。RHOSP 16.1 以降にアップグレードする前に、Red Hat ナレッジベースの記事 Red Hat Ceph Storage: Supported configurations を確認してください。

配置グループ (PG)

- デプロイメントの規模によらず、動的で効率的なオブジェクトの追跡を容易に実施するために、Ceph Storage では配置グループ (PG) が使用されています。OSD の障害やクラスターのリバランスの際には、Ceph は配置グループおよびその内容を移動または複製することができるので、Ceph Storage クラスターは効率的にリバランスおよび復旧を行うことができます。

- director が作成するデフォルトの配置グループ数が常に最適とは限らないので、実際の要件に応じて正しい配置グループ数を計算することが重要です。配置グループの計算ツールを使用して、正しい配置グループ数を計算することができます。PG の計算ツールを使用するには、Ceph クラスターに関するその他の属性 (OSD の数など) と共に、サービスごとに予測されるストレージ使用量をパーセンテージで入力します。計算ツールは、プールごとに最適な PG 数を返します。詳細は、Ceph Placement Groups (PGs) per Pool Calculator を参照してください。

- 自動スケーリングは、配置グループを管理するもう 1 つの方法です。自動スケーリング機能では、具体的な配置グループ数ではなく、サービスごとに予想される Ceph Storage 要件をパーセンテージで設定します。Ceph は、クラスターの使用状況に応じて配置グループを自動的にスケーリングします。詳細は、**Red Hat Ceph Storage ストレージストラテジーガイド**の 配置グループの自動スケーリング を参照してください。

プロセッサー

- Intel 64 または AMD64 CPU 拡張機能をサポートする 64 ビット x86 プロセッサー。

ネットワークインターフェイスカード

- 最小 1 枚の 1 Gbps ネットワークインターフェイスカード (NIC)。ただし、Red Hat では実稼働環境の場合には最低でも NIC を 2 枚使用することを推奨します。ボンディングインターフェイス向けやタグ付けされた VLAN トラフィックを委譲する場合は、追加の NIC を使用します。特に大量のトラフィックを処理する Red Hat OpenStack Platform (RHOSP) 環境を構築する場合には、ストレージノードに 10 Gbps インターフェイスを使用します。

電源管理

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

Ceph Storage クラスターを使用するオーバークラウドのインストールについての詳しい情報は、コンテナー化された Red Hat Ceph を持つオーバークラウドのデプロイ を参照してください。

6.9. オブジェクトストレージノードの要件

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

プロセッサー

Intel 64 または AMD64 CPU 拡張機能をサポートする 64 ビット x86 プロセッサー。

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、最低でも 1 GB のメモリーを使用します。最適なパフォーマンスを得るには、ハードディスク容量 1 TB あたり 2 GB のメモリーを使用することを推奨します (特に、ワークロードが 100 GB に満たないファイルの場合)。

ディスク容量

ストレージ要件は、ワークロードに必要とされる容量により異なります。アカウントとコンテナーのデータを保存するには SSD ドライブを使用することを推奨します。アカウントおよびコンテナーデータとオブジェクトの容量比率は、約 1% です。たとえば、ハードドライブの容量 100 TB ごとに、アカウントおよびコンテナーデータの SSD 容量は 1 TB 用意するようにします。
ただし、これは保存したデータの種類により異なります。保存するオブジェクトの大半が小さい場合には、SSD の容量がさらに必要です。オブジェクトが大きい場合には (ビデオ、バックアップなど)、SSD の容量を減らします。

ディスクのレイアウト

推奨されるノード設定には、以下の例に示すようなディスクレイアウトが必要です。

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

ネットワークインターフェイスカード

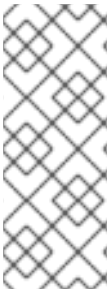
最小 2 枚の 1 Gbps ネットワークインターフェイスカード。タグ付けされた VLAN トラフィックを委譲する場合や、ボンディングインターフェイス向けには、追加のネットワークインターフェイスを使用します。

電源管理

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

6.10. オーバークラウドのリポジトリー

Red Hat OpenStack Platform (RHOSP) 16.1 は、Red Hat Enterprise Linux 8.2 上で動作します。したがって、これらのリポジトリーからのコンテンツを該当する Red Hat Enterprise Linux バージョンにロックする必要があります。



注記

リポジトリーを Red Hat Satellite と同期する場合は、特定バージョンの Red Hat Enterprise Linux リポジトリーを有効にすることができます。ただし、選択したバージョンに関係なく、リポジトリーラベルは同じままです。たとえば、BaseOS リポジトリーの 8.4 バージョンを有効にした場合、リポジトリー名には有効にした特定のバージョンが含まれますが、リポジトリーラベルは依然として **rhel-8-for-x86_64-baseos-eus-rpms** です。

警告

ここで指定する以外のリポジトリーは、サポートされません。別途推奨されない限り、以下の表に記載されている以外の製品またはリポジトリーを有効にしないでください。有効にすると、パッケージの依存関係の問題が発生する可能性があります。Extra Packages for Enterprise Linux (EPEL) を有効にしないでください。

コントローラーノード用リポジトリー

以下の表には、オーバークラウドのコントローラーノード用コアリポジトリーをまとめています。

名前	リポジトリー	要件の説明
Red Hat Enterprise Linux 8.4 for x86_64 - BaseOS (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-baseos-tus-rpms**	x86_64 システム用ベースオペレーティングシステムのリポジトリー
Red Hat Enterprise Linux 9 for x86_64 - AppStream (RPMs)	**rhel-8-for-x86_64-appstream-tus-rpms**	Red Hat OpenStack Platform の依存関係が含まれます。

名前	リポジトリー	要件の説明
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-highavailability-tus-rpms**	Red Hat Enterprise Linux の高可用性ツール。
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Red Hat Enterprise Linux 用 Ansible エンジン。最新バージョンの Ansible を提供するために使用されます。
Advanced Virtualization for RHEL 8 x86_64 (RPMs)	**advanced-virt-for-rhel-8-x86_64-eus-rpms**	OpenStack Platform 用仮想化パッケージを提供します。
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-x86_64-rpms**	Red Hat OpenStack Platform のコアリポジトリー
Red Hat Fast Datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	OpenStack Platform 用 Open vSwitch (OVS) パッケージを提供します。
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	Red Hat Enterprise Linux 8 での Red Hat Ceph Storage 4 用ツール
Red Hat Satellite Tools for RHEL 8 Server RPMs x86_64	**satellite-tools-6.5-for-rhel-8-x86_64-rpms**	Red Hat Satellite 6 でのホスト管理ツール

Compute ノードおよび ComputeHCI ノードのリポジトリー

以下の表に、オーバークラウド内の Compute ノードおよび ComputeHCI ノードのコアリポジトリーを示します。

名前	リポジトリー	要件の説明
Red Hat Enterprise Linux 8.4 for x86_64 - BaseOS (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-baseos-tus-rpms**	x86_64 システム用ベースオペレーティングシステムのリポジトリー
Red Hat Enterprise Linux 9 for x86_64 - AppStream (RPMs)	**rhel-8-for-x86_64-appstream-tus-rpms**	Red Hat OpenStack Platform の依存関係が含まれます。
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-highavailability-tus-rpms**	Red Hat Enterprise Linux の高可用性ツール。

名前	リポジトリー	要件の説明
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Red Hat Enterprise Linux 用 Ansible エンジン。最新バージョンの Ansible を提供するために使用されます。
Advanced Virtualization for RHEL 8 x86_64 (RPMs)	**advanced-virt-for-rhel-8-x86_64-eus-rpms**	OpenStack Platform 用仮想化パッケージを提供します。
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-x86_64-rpms**	Red Hat OpenStack Platform のコアリポジトリー
Red Hat Fast Datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	OpenStack Platform 用 Open vSwitch (OVS) パッケージを提供します。
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	Red Hat Enterprise Linux 8 での Red Hat Ceph Storage 4 用ツール
Red Hat Satellite Tools for RHEL 8 Server RPMs x86_64	**satellite-tools-6.5-for-rhel-8-x86_64-rpms**	Red Hat Satellite 6 でのホスト管理ツール

リアルタイムコンピュート用リポジトリー

以下の表には、リアルタイムコンピュート (RTC) 機能用リポジトリーをまとめています。

名前	リポジトリー	要件の説明
Red Hat Enterprise Linux 8 for x86_64 - Real Time (RPMs)	**rhel-8-for-x86_64-rt-rpms**	リアルタイム KVM (RT-KVM) のリポジトリー。リアルタイムカーネルを有効化するためのパッケージが含まれています。RT-KVM 対象の全コンピュートノードで、このリポジトリーを有効にします。注記: このリポジトリーにアクセスするには、別途 **Red Hat OpenStack Platform for Real Time** SKU のサブスクリプションが必要です。

名前	リポジトリー	要件の説明
Red Hat Enterprise Linux 8 for x86_64 - Real Time for NFV (RPMs)	**rhel-8-for-x86_64-nfv-rpms**	NFV 向けのリアルタイム KVM (RT-KVM) のリポジトリー。リアルタイムカーネルを有効化するためのパッケージが含まれています。RT-KVM 対象の全 NFV コンピュートノードで、このリポジトリーを有効にします。注記: このリポジトリーにアクセスするには、別途 **Red Hat OpenStack Platform for Real Time** SKU のサブスクリプションが必要です。

Ceph Storage ノード用リポジトリー

以下の表には、オーバークラウド用の Ceph Storage 関連のリポジトリーをまとめています。

名前	リポジトリー	要件の説明
Red Hat Enterprise Linux 8.4 for x86_64 - BaseOS (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-baseos-tus-rpms**	x86_64 システム用ベースオペレーティングシステムのリポジトリー
Red Hat Enterprise Linux 9 for x86_64 - AppStream (RPMs)	**rhel-8-for-x86_64-appstream-tus-rpms**	Red Hat OpenStack Platform の依存関係が含まれます。
Red Hat Enterprise Linux 8 for x86_64 - High Availability (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-highavailability-tus-rpms**	Red Hat Enterprise Linux の高可用性ツール。注記: Ceph Storage ロールに **overcloud-full** イメージを使用した場合は、このリポジトリーを有効にする必要があります。Ceph Storage ロールは、このリポジトリーを必要としない **overcloud-minimal** イメージを使用する必要があります。
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Red Hat Enterprise Linux 用 Ansible エンジン。最新バージョンの Ansible を提供するために使用されます。

名前	リポジトリー	要件の説明
Red Hat OpenStack Platform 16.1 Director Deployment Tools for RHEL 8 x86_64 (RPMs)	**openstack-16.1-deployment-tools-for-rhel-8-x86_64-rpms**	director が Ceph Storage ノードを設定するのに役立つパッケージ。このリポジトリーは、スタンドアロンの Ceph Storage サブスクリプションに含まれています。OpenStack Platform と Ceph Storage を組み合わせたサブスクリプションを使用する場合は、**openstack-16.1-for-rhel-8-x86_64-rpms** リポジトリーを使用します。
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-x86_64-rpms**	director が Ceph Storage ノードを設定するのに役立つパッケージ。このリポジトリーは、OpenStack Platform と Ceph Storage を組み合わせたサブスクリプションに含まれています。スタンドアロンの Ceph Storage サブスクリプションを使用する場合は、**openstack-16.1-deployment-tools-for-rhel-8-x86_64-rpms** リポジトリーを使用します。
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	Ceph Storage クラスターと通信するためのノード用のツールを提供します。
Red Hat Fast Datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	OpenStack Platform 用 Open vSwitch (OVS) パッケージを提供します。Ceph Storage ノードで OVS を使用している場合は、このリポジトリーをネットワークインターフェイス設定 (NIC) テンプレートに追加します。

IBM POWER 用リポジトリー

次の表に、POWER PC アーキテクチャー上の RHOSP のリポジトリーをまとめています。コアリポジトリーの該当リポジトリーの代わりに、これらのリポジトリーを使用してください。

名前	リポジトリー	要件の説明
Red Hat Enterprise Linux for IBM Power, little endian - BaseOS (RPMs)	**rhel-8-for-ppc64le-baseos-rpms**	ppc64le システム用ベースオペレーティングシステムのリポジトリー

名前	リポジトリー	要件の説明
Red Hat Enterprise Linux 8 for IBM Power, little endian - AppStream (RPMs)	**rhel-8-for-ppc64le-appstream-rpms**	Red Hat OpenStack Platform の依存関係が含まれます。
Red Hat Enterprise Linux 8 for IBM Power, little endian - High Availability (RPMs)	**rhel-8-for-ppc64le-highavailability-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 Fast Datapath for RHEL 8 IBM Power, little endian (RPMS)	**fast-datapath-for-rhel-8-ppc64le-rpms**	OpenStack Platform 用 Open vSwitch (OVS) パッケージを提供します。
Red Hat Ansible Engine 2.8 for RHEL 8 IBM Power, little endian (RPMs)	**ansible-2.8-for-rhel-8-ppc64le-rpms**	Red Hat Enterprise Linux 用 Ansible エンジン。最新バージョンの Ansible を提供するために使用されます。
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-ppc64le-rpms**	ppc64le システム用 Red Hat OpenStack Platform のコアリポジトリー

6.11. プロビジョニングの方法

Red Hat OpenStack Platform 環境にノードをプロビジョニングする場合、使用できる主な方法が 3 つあります。

director を使用したプロビジョニング

Red Hat OpenStack Platform director は、標準のプロビジョニング方法です。このシナリオでは、**openstack overcloud deploy** コマンドによって、デプロイメントのプロビジョニングと設定の両方を実行します。標準のプロビジョニングおよびデプロイメント方法についての詳しい情報は、*7章 基本的なオーバークラウドの設定* を参照してください。

OpenStack Bare Metal (ironic) サービスを使用したプロビジョニング

このシナリオでは、標準の director デプロイメントのプロビジョニングステージと設定ステージを、2 つの別のプロセスに分割することができます。この方法は、標準の director デプロイメントに伴うリスクの一部を軽減し、より効率的に障害点を特定するのに役立ちます。このシナリオについての詳しい情報は、*8章 オーバークラウドのデプロイ前に行うベアメタルノードのプロビジョニング* を参照してください。

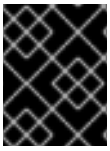
重要

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

外部ツールを使用したプロビジョニング

このシナリオでは、director は外部ツールを使用して事前にプロビジョニングされたノードでオー

バークラウドの設定を制御します。この方法は、電源管理制御を設定せずにオーバークラウドを作成する場合や、DHCP/PXE ブートの制限があるネットワークを使用する場合、あるいは QCOW2 **overcloud-full** イメージに依存しないカスタムのパーティションレイアウトを持つノードを使用する場合に便利です。このシナリオでは、ノードの管理に OpenStack Compute (nova)、OpenStack Bare Metal (ironic)、または OpenStack Image (glance) サービスを使用しません。
このシナリオについての詳しい情報は、*9章 事前にプロビジョニングされたノードを使用した基本的なオーバークラウドの設定* を参照してください。



重要

事前にプロビジョニングされたノードと director がプロビジョニングしたノードを組み合わせることはできません。

第7章 基本的なオーバークラウドの設定

基本設定のオーバークラウドには、カスタム機能は含まれません。基本的な Red Hat OpenStack Platform (RHOSP) 環境を設定するには、次のタスクを実行する必要があります。

- オーバークラウドのベアメタルノードを登録します。
- ベアメタルノードのハードウェアのインベントリーをディレクターに提供します。
- 各ベアメタルノードに、ノードを指定されたロールに一致させるリソースクラスでタグ付けします。

ヒント

この基本的なオーバークラウドに高度な設定オプションを追加して、仕様に合わせてカスタマイズできます。詳細は、Advanced Overcloud Customization を参照してください。

7.1. オーバークラウドノードの登録

ディレクターには、ノードのハードウェアと電源管理の詳細を指定するノード定義テンプレートが必要です。このテンプレートは、JSON 形式の **nodes.json** または YAML 形式の **nodes.yaml** で作成できます。

手順

1. ノードをリスト表示する **nodes.json** または **nodes.yaml** という名前のテンプレートを作成します。以下の例に示す JSON および YAML テンプレートを使用して、ノード定義のテンプレートを設定する方法を説明します。

   JSON テンプレートの例

```
{
 "nodes": [{
   "ports": [{
    "address": "aa:aa:aa:aa:aa:aa",
    "physical_network": "ctlplane",
    "local_link_connection": {
     "switch_id": "52:54:00:00:00:00",
     "port_id": "p0"
    }
   }],
   "name": "node01",
   "cpu": "4",
   "memory": "6144",
   "disk": "40",
   "arch": "x86_64",
   "pm_type": "ipmi",
   "pm_user": "admin",
   "pm_password": "p@55w0rd!",
   "pm_addr": "192.168.24.205"
  },
  {
   "ports": [{
    "address": "bb:bb:bb:bb:bb:bb",
    "physical_network": "ctlplane",
```

```
    "local_link_connection": {
     "switch_id": "52:54:00:00:00:00",
     "port_id": "p0"
    }
   }],
   "name": "node02",
   "cpu": "4",
   "memory": "6144",
   "disk": "40",
   "arch": "x86_64",
   "pm_type": "ipmi",
   "pm_user": "admin",
   "pm_password": "p@55w0rd!",
   "pm_addr": "192.168.24.206"
  }
 ]
}
```

YAML テンプレートの例

```
nodes:
  - ports:
      - address: aa:aa:aa:aa:aa:aa
        physical_network: ctlplane
        local_link_connection:
          switch_id: 52:54:00:00:00:00
          port_id: p0
    name: "node01"
    cpu: 4
    memory: 6144
    disk: 40
    arch: "x86_64"
    pm_type: "ipmi"
    pm_user: "admin"
    pm_password: "p@55w0rd!"
    pm_addr: "192.168.24.205"
  - ports:
      - address: bb:bb:bb:bb:bb:bb
        physical_network: ctlplane
        local_link_connection:
          switch_id: 52:54:00:00:00:00
          port_id: p0
    name: "node02"
    cpu: 4
    memory: 6144
    disk: 40
    arch: "x86_64"
    pm_type: "ipmi"
    pm_user: "admin"
    pm_password: "p@55w0rd!"
    pm_addr: "192.168.24.206"
```

このテンプレートには、以下の属性が含まれます。

name

ノードの論理名

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 (**ipmi**) を使用しています。

注記

IPMI が推奨されるサポート対象電源管理ドライバーです。サポート対象電源管理ドライバーの種別およびそのオプションに関する詳細は、30章 *電源管理ドライバー* を参照してください。それらの電源管理ドライバーが想定どおりに機能しない場合には、電源管理に IPMI を使用してください。

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

pm_port (オプション)

特定の IPMI デバイスにアクセスするためのポート

address

(オプション) ノード上のネットワークインターフェイスの MAC アドレスリスト。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。

physical_network

(オプション) プロビジョニング NIC に接続される物理ネットワーク

local_link_connection

(オプション) IPv6 プロビジョニングを使用し、イントロスペクション中に LLDP がローカルリンク接続を正しく反映しない場合は、**local_link_connection** パラメーターの **switch_id** および **port_id** フィールドにダミーのデータを含める必要があります。偽のデータを含める方法の詳細は、Using director introspection to collect bare metal node hardware information を参照してください。

cpu

(オプション) ノード上の CPU 数

memory

(オプション) メモリーサイズ (MB 単位)

disk

(オプション) ハードディスクのサイズ (GB 単位)

arch

(オプション) システムアーキテクチャー

重要

マルチアーキテクチャークラウドをビルドする場合には、**x86_64** アーキテクチャーを使用するノードと **ppc64le** アーキテクチャーを使用するノードを区別するために **arch** キーが必須です。

2. テンプレートを作成したら、以下のコマンドを実行してフォーマットおよび構文を検証します。

```
$ source ~/stackrc
(undercloud)$ openstack overcloud node import --validate-only ~/nodes.json
```

重要

マルチアーキテクチャーノードの場合は、**--http-boot /var/lib/ironic/tftpboot/** オプションも追加する必要があります。

3. **stack** ユーザーのホームディレクトリーにファイルを保存し (**/home/stack/nodes.json**)、続いて以下のコマンドを実行してテンプレートを director にインポートします。

```
(undercloud)$ openstack overcloud node import ~/nodes.json
```

このコマンドにより、それぞれのノードがテンプレートから director に登録されます。UEFI ブートモードを使用する場合は、各ノードでブートモードも設定する必要があります。UEFI ブートモードを設定せずにノードをイントロスペクトすると、ノードはレガシーモードでブートします。詳細は、Setting the boot mode to UEFI boot mode を参照してください。

4. ノードの登録および設定が完了するまで待ちます。完了したら、ノードが director に正しく登録されていることを確認します。

```
(undercloud)$ openstack baremetal node list
```

7.2. ベアメタルノードハードウェアのインベントリーの作成

ディレクターは、プロファイルのタグ付け、ベンチマーク、および手動のルートディスク割り当てのために、Red Hat OpenStack Platform (RHOSP) デプロイメント内のノードのハードウェアインベントリーを必要とします。

次のいずれかの方法を使用して、ハードウェアインベントリーをディレクターに提供できます。

- Automatic: 各ノードからハードウェア情報を収集するディレクターのイントロスペクションプロセスを使用できます。このプロセスは、各ノードでイントロスペクションエージェントを起動します。イントロスペクションエージェントは、ノードからハードウェアのデータを収集し、そのデータを director に送り返します。Director は、ハードウェアデータを OpenStack 内部データベースに保存します。

- Manual: 各ベアメタルマシンの基本的なハードウェアインベントリーを手動で設定できます。このインベントリーは、ベアメタルプロビジョニングサービス (ironic) に保存され、ベアメタルマシンの管理とデプロイに使用されます。

ディレクターの自動イントロスペクションプロセスには、ベアメタルプロビジョニングサービスポートを手動で設定する方法に比べて、次の利点があります。

- イントロスペクションは、接続されているすべてのポートをハードウェア情報に記録します。これには、**nodes.yaml** でまだ設定されていない場合に PXE ブートに使用するポートも含まれます。

- イントロスペクションは、属性が LLDP を使用して検出可能である場合、各ポートの **local_link_connection** 属性を設定します。手動による方法を使用する場合は、ノードを登録するときに各ポートに **local_link_connection** を設定する必要があります。

- イントロスペクションは、スパイン/リーフ型または DCN のアーキテクチャーをデプロイするときに、ベアメタルプロビジョニングサービスポートの **physical_network** 属性を設定します。

7.2.1. ディレクターのイントロスペクションを使用してベアメタルノードのハードウェア情報を収集する

物理マシンをベアメタルノードとして登録した後、ディレクターイントロスペクションを使用して、ハードウェアの詳細を自動的に追加し、イーサネット MAC アドレスごとにポートを作成できます。

ヒント

自動イントロスペクションの代わりに、ベアメタルノードのハードウェア情報をディレクターに手動で提供できます。詳細は、ベアメタルノードのハードウェア情報を手動で設定する を参照してください。

前提条件

- オーバークラウドのベアメタルノードを登録しました。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

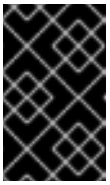
3. pre-introspection 検証グループを実行して、プリイントロスペクションの要件を確認します。

```
(undercloud)$ openstack tripleo validator run --group pre-introspection
```

4. 検証レポートの結果を確認します。
5. オプション: 特定の検証からの詳細な出力を確認します。

```
(undercloud)$ openstack tripleo validator show run --full <validation>
```

- **<validation>** を、確認するレポートの特定の検証の UUID に置き換えます。



重要

検証結果が **FAILED** であっても、Red Hat OpenStack Platform のデプロイや実行が妨げられることはありません。ただし、**FAILED** の検証結果は、実稼働環境で問題が発生する可能性があることを意味します。

6. 各ノードのハードウェア属性を検証します。すべてのノードまたは特定のノードのハードウェア属性を検査できます。
   - すべてのノードのハードウェア属性を検査します。

```
(undercloud)$ openstack overcloud node introspect --all-manageable --provide
```

- **--all-manageable** オプションを使用して、管理状態にあるノードのみをイントロスペクションします。ここでは、すべてのノードが管理状態にあります。
- **--provide** オプションを使用して、イントロスペクション後に全ノードを **available** の状態に再設定します。

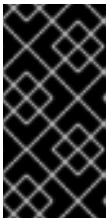
- 特定のノードのハードウェア属性を検査します。

```
(undercloud)$ openstack overcloud node introspect --provide <node1> [node2] [noden]
```

- **--provide** オプションを使用して、イントロスペクション後に指定されたすべてのノードを **available** 状態にリセットします。
- **<node1>**、**[node2]**、および **[noden]** までのすべてのノードを、イントロスペクションする各ノードの UUID に置き換えます。

7. 別のターミナルウィンドウで、イントロスペクションの進捗ログを監視します。

```
(undercloud)$ sudo tail -f /var/log/containers/ironic-inspector/ironic-inspector.log
```



重要

イントロスペクションプロセスが完了するまで実行されていることを確認します。イントロスペクションは通常、ベアメタルノードの場合 15 分かかります。ただし、イントロスペクションネットワークのサイズが正しくないと、時間がかかる可能性があり、イントロスペクションが失敗する可能性があります。

8. オプション: IPv6 を介したベアメタルプロビジョニング用にアンダークラウドを設定した場合は、LLDP がベアメタルプロビジョニングサービス (ironic) ポートの **local_link_connection** を設定していることも確認する必要があります。

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
```

- ベアメタルノードのポートに対して Local Link Connection フィールドが空の場合、**local_link_connection** 値に偽のデータを手動で入力する必要があります。次の例では、偽のスイッチ ID を **52:54:00:00:00:00** に設定し、偽のポート ID を **p0** に設定します。

```
(undercloud)$ openstack baremetal port set <port_uuid> \
--local-link-connection switch_id=52:54:00:00:00:00 \
--local-link-connection port_id=p0
```

- ローカルリンク接続フィールドにダミーのデータが含まれていることを確認します。

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
```

イントロスペクション完了後には、すべてのノードが **available** の状態に変わります。

7.2.2. ベアメタルノードのハードウェア情報を手動で設定する

物理マシンをベアメタルノードとして登録した後、ハードウェアの詳細を手動で追加し、イーサネット MAC アドレスごとにベアメタルポートを作成できます。オーバークラウドをデプロイする前に、少なくとも 1 つのベアメタルポートを作成する必要があります。

ヒント

手動イントロスペクションの代わりに、自動ディレクターイントロスペクションプロセスを使用して、ベアメタルノードのハードウェア情報を収集できます。詳細は、Using director introspection to collect bare metal node hardware information を参照してください。

前提条件

- オーバークラウドのベアメタルノードを登録しました。
- **nodes.json** の登録済みノードの各ポートに **local_link_connection** を設定しました。詳しい情報は、Registering nodes for the overcloud を参照してください。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

3. ノードの機能に **boot_option':'local** を追加して、登録されたノードごとにブートオプションを **local** に設定します。

```
(undercloud)$ openstack baremetal node set \
 --property capabilities="boot_option:local" <node>
```

- **<node>** をベアメタルノードの ID に置き換えてください。

4. ノードドライバーのデプロイカーネルとデプロイ ramdisk を指定します。

```
(undercloud)$ openstack baremetal node set <node> \
  --driver-info deploy_kernel=<kernel_file> \
  --driver-info deploy_ramdisk=<initramfs_file>
```

- **<node>** をベアメタルノードの ID に置き換えてください。
- **<kernel_file>** を **.kernel** イメージへのパス (例: **file:///var/lib/ironic/httpboot/agent.kernel**) に置き換えます。
- **<initramfs_file>** は、**.initramfs** イメージへのパス (例: **file:///var/lib/ironic/httpboot/agent.ramdisk**) に置き換えます。

5. ノードの属性を更新して、ノード上のハードウェアの仕様と一致するようにします。

```
(undercloud)$ openstack baremetal node set <node> \
  --property cpus=<cpu> \
  --property memory_mb=<ram> \
  --property local_gb=<disk> \
  --property cpu_arch=<arch>
```

- **<node>** をベアメタルノードの ID に置き換えてください。
- **<cpu>** は、CPU の数に置き換えます。
- **<ram>** を MB 単位の RAM に置き換えます。
- **<disk>** を GB 単位のディスクサイズに置き換えます。
- **<arch>** は、アーキテクチャータイプに置き換えます。

6. オプション: 各ノードの IPMI 暗号スイートを指定します。

```
(undercloud)$ openstack baremetal node set <node> \
 --driver-info ipmi_cipher_suite=<version>
```

- **<node>** をベアメタルノードの ID に置き換えてください。
- **<version>** は、ノードで使用する暗号スイートのバージョンに置き換えます。以下の有効な値のいずれかに設定します。
  - **3** - ノードは SHA1 暗号スイートで AES-128 を使用します。
  - **17** - ノードは SHA256 暗号スイートで AES-128 を使用します。

7. オプション: 複数のディスクがある場合は、ルートデバイスのヒントを設定して、デプロイメントに使用するディスクをデプロイ ramdisk に通知します。

```
(undercloud)$ openstack baremetal node set <node> \
 --property root_device='{"<property>": "<value>"}'
```

- **<node>** をベアメタルノードの ID に置き換えてください。
- **<property>** と **<value>** は、デプロイメントに使用するディスクの詳細に置き換えます (例: **root_device='{"size": "128"}'**)。
  RHOSP は、次のプロパティーをサポートしています。
  - **model** (文字列): デバイスの ID
  - **vendor** (文字列): デバイスのベンダー
  - **serial** (文字列): ディスクのシリアル番号
  - **hctl** (文字列): SCSI のホスト、チャンネル、ターゲット、Lun
  - **size** (整数): デバイスのサイズ (GB 単位)
  - **wwn** (文字列): 一意のストレージ ID
  - **wwn_with_extension** (文字列): ベンダー拡張子を追加した一意のストレージ ID
  - **wwn_vendor_extension** (文字列): 一意のベンダーストレージ ID
  - **rotational** (ブール値): 回転式デバイス (HDD) には true、そうでない場合 (SSD) には false
  - **name** (文字列): デバイス名 (例: /dev/sdb1)。このプロパティーは、永続デバイス名が付いたデバイスにのみ使用してください。

注記

複数のプロパティーを指定する場合には、デバイスはそれらの全プロパティーと一致する必要があります。

8. プロビジョニングネットワーク上の NIC の MAC アドレスを使用してポートを作成することにより、Bare Metal Provisioning サービスにノードのネットワークカードを通知します。

```
(undercloud)$ openstack baremetal port create --node <node_uuid> <mac_address>
```

- **<node_uuid>** をベアメタルノードの一意の ID に置き換えます。
- **<mac_address>** は、PXE ブートに使用する NIC の MAC アドレスに置き換えます。

9. ノードの設定を検証します。

```
(undercloud)$ openstack baremetal node validate <node>
+------------+--------+--------------------------------------------+
| Interface  | Result | Reason                                     |
+------------+--------+--------------------------------------------+
| boot       | False  | Cannot validate image information for node  |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b       |
|            |        | because one or more parameters are missing  |
|            |        | from its instance_info. Missing are:       |
|            |        | ['ramdisk', 'kernel', 'image_source']      |
| console    | None   | not supported                              |
| deploy     | False  | Cannot validate image information for node  |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b       |
|            |        | because one or more parameters are missing  |
|            |        | from its instance_info. Missing are:       |
|            |        | ['ramdisk', 'kernel', 'image_source']      |
| inspect    | None   | not supported                              |
| management | True   |                                            |
| network    | True   |                                            |
| power      | True   |                                            |
| raid       | True   |                                            |
| storage    | True   |                                            |
+------------+--------+--------------------------------------------+
```

有効出力の **Result** は、次のことを示しています。

- **False**: インターフェイスは検証に失敗しました。**instance_info** パラメーター **[\'ramdisk', \'kernel', and \'image_source']** が見つからない場合、Compute サービスがデプロイメントプロセスの最初にこれらのパラメーターを設定するので、この時点では設定されていない可能性があ ります。ディスクイメージ全体を使用している場合は、検証にパスするために **image_source** を設定するだけでよい場合があります。
- **True**: インターフェイスは検証にパスしました。
- **None**: インターフェイスはドライバーでサポートされていません。

7.3. プロファイルへのノードのタグ付け

各ノードのハードウェアを登録、検査した後には、特定のプロファイルにノードをタグ付けします。このプロファイルタグにより、ノードがフレーバーに照合され、そのフレーバーがデプロイメントロール

に割り当てられます。以下の例は、コントローラーノードのロール、フレーバー、プロファイル、ノード間の関係を示しています。

タイプ	説明
ロール	**Controller** ロールは、director がコントローラーノードをどのように設定するかを定義します。
フレーバー	**control** フレーバーは、ノードをコントローラーとして使用するためのハードウェアプロファイルを定義します。使用するノードを director が決定できるように、このフレーバーを **Controller** ロールに割り当てます。
プロファイル	**control** プロファイルは、**control** フレーバーに適用するタグです。これにより、フレーバーに属するノードが定義されます。
ノード	また、各ノードに **control** プロファイルタグを適用して、**control** フレーバーにグループ化します。これにより、director が **Controller** ロールを使用してノードを設定します。

アンダークラウドのインストール時に、デフォルトプロファイルのフレーバー **compute**、**control**、**swift-storage**、**ceph-storage**、**block-storage** が作成され、大半の環境で変更なしに使用することができます。

手順

1. 特定のプロファイルにノードをタグ付けする場合には、各ノードの **properties/capabilities** パラメーターに **profile** オプションを追加します。たとえば、特定のプロファイルを使用するように特定のノードをタグ付けするには、以下のコマンドを使用します。

```
(undercloud) $ NODE=<NODE NAME OR ID>
(undercloud) $ PROFILE=<PROFILE NAME>
(undercloud) $ openstack baremetal node set --property
capabilities="profile:$PROFILE,boot_option:local" $NODE
```

   - **$NODE** 変数にノードの名前または UUID を設定します。
   - **$PROFILE** 変数は、**control** または **compute** などの特定のプロファイルに設定します。
   - **properties/capabilities** の **profile** オプションには、**profile:control** または **profile:compute** などの対応するプロファイルとノードをタグ付けする **$PROFILE** 変数が含まれます。
   - 各ノードのブート方法を定義するには、**boot_option:local** オプションを設定します。

   追加の **openstack baremetal node show** コマンドおよび **jq** フィルタリングを使用して、既存の **capabilities** の値を保持することもできます。

```
(undercloud) $ openstack baremetal node set --property
capabilities="profile:$PROFILE,boot_option:local,$(openstack baremetal node show $NODE
-f json -c properties | jq -r .properties.capabilities | sed "s/boot_mode:[^,]*,//g")" $NODE
```

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

```
(undercloud) $ openstack overcloud profiles list
```

7.4. ブートモードを UEFI モードに設定する

デフォルトのブートモードはレガシー BIOS モードです。レガシー BIOS ブートモードの代わりに UEFI ブートモードを使用するように RHOSP デプロイメントのノードを設定できます。

警告

一部のハードウェアは、レガシー BIOS ブートモードをサポートしていません。レガシー BIOS ブートモードをサポートしていないハードウェアでレガシー BIOS ブートモードを使用しようとすると、デプロイメントが失敗する可能性があります。ハードウェアが正常にデプロイされるようにするには、UEFI ブートモードを使用します。

注記

UEFI ブートモードを有効にする場合は、ユーザーイメージとともに、パーティションレイアウトとブートローダーを含む独自のディスク全体のイメージを構築する必要があります。全ディスクイメージの作成の詳細については、Creating whole-disk images を参照してください。

手順

1. **undercloud.conf** ファイルで以下のパラメーターを設定します。

```
ipxe_enabled = True
```

2. **undercloud.conf** ファイルを保存して、アンダークラウドのインストールを実行します。

```
$ openstack undercloud install
```

インストールスクリプトが完了するまで待ちます。

3. 登録された各ノードの既存の機能を確認します。

```
$ openstack baremetal node show <node> -f json -c properties | jq -r .properties.capabilities
```

- **<node>** をベアメタルノードの ID に置き換えてください。

4. ノードの既存の機能に **boot_mode:uefi** を追加して、登録されている各ノードのブートモードを **uefi** に設定します。

```
$ openstack baremetal node set --property capabilities="boot_mode:uefi,<capability_1>,...,
<capability_n>" <node>
```

- **<node>** をベアメタルノードの ID に置き換えてください。
- **<capability_1>** および **<capability_n>** までのすべての機能を、手順 3 で取得した各機能に置き換えます。
  たとえば、次のコマンドを使用して、ローカルブートでブートモードを **uefi** に設定します。

```
$ openstack baremetal node set --property capabilities="boot_mode:uefi,boot_option:local"
<node>
```

5. ベアメタルフレーバーごとに、ブートモードを **uefi** に設定します。

```
$ openstack flavor set --property capabilities:boot_mode='uefi' <flavor>
```

7.5. 仮想メディアブートの有効化

重要

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

Redfish 仮想メディアブートを使用して、ノードの Baseboard Management Controller (BMC) にブートイメージを提供することができます。これにより、BMC はイメージを仮想ドライブのいずれかに挿入することができます。その後、ノードは仮想ドライブからイメージに存在するオペレーティングシステムにブートすることができます。

Redfish ハードウェア種別は、仮想メディアを通じたデプロイ、レスキュー、およびユーザーの各イメージのブートに対応しています。Bare Metal サービス (ironic) は、ノードのデプロイメント時に、ノードに関連付けられたカーネルイメージおよび ramdisk イメージを使用して、UEFI または BIOS ブートモード用のブート可能 ISO イメージをビルドします。仮想メディアブートの主な利点は、PXE の TFTP イメージ転送フェーズを排除し、HTTP GET 等の方法を使用することができる点です。

仮想メディアを通じて **redfish** ハードウェア種別のノードをブートするには、ブートインターフェイスを **redfish-virtual-media** に設定し、UEFI ノードの場合は EFI システムパーティション (ESP) イメージを定義します。続いて、登録したノードが Redfish 仮想メディアブートを使用するように設定します。

前提条件

- **undercloud.conf** ファイルの **enabled_hardware_types** パラメーターで、Redfish ドライバーが有効化されている。
- ベアメタルノードが登録されている。
- Image サービス (glance) に IPA およびインスタンスイメージがある。

- UEFI ノードの場合、EFI システムパーティション (ESP) イメージも Image サービス (glance) で利用可能でなければなりません。
- ベアメタルフレーバー
- クリーニングおよびプロビジョニング用ネットワーク

手順

1. Bare Metal サービス (ironic) のブートインターフェイスを **redfish-virtual-media** に設定します。

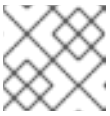
```
$ openstack baremetal node set --boot-interface redfish-virtual-media $NODE_NAME
```

   - **$NODE_NAME** はノード名に置き換えてください。

2. UEFI ノードの場合は、ブートモードを **uefi** に設定します。

```
NODE=<NODE NAME OR ID> ; openstack baremetal node set --property
capabilities="boot_mode:uefi,$(openstack baremetal node show $NODE -f json -c properties
| jq -r .properties.capabilities | sed "s/boot_mode:[^,]*,//g")" $NODE
```

   - **$NODE** はノード名に置き換えてください。



注記

BIOS ノードの場合は、このステップを実施しないでください。

3. UEFI ノードの場合は、EFI システムパーティション (ESP) イメージを定義します。

```
$ openstack baremetal node set --driver-info bootloader=$ESP $NODE_NAME
```

   - **$ESP** は glance イメージの UUID または ESP イメージの URL に、 **$NODE_NAME** はノードの名前に、それぞれ置き換えてください。

注記

BIOS ノードの場合は、このステップを実施しないでください。

4. ベアメタルノードにポートを作成し、そのポートをベアメタルノード上の NIC の MAC アドレスに関連付けます。

```
$ openstack baremetal port create --pxe-enabled True --node $UUID $MAC_ADDRESS
```

   - **$UUID** はベアメタルノードの UUID に、**$MAC_ADDRESS** はベアメタルノード上の NIC の MAC アドレスに、それぞれ置き換えてください。

7.6. マルチディスククラスターのルートディスクの定義

ほとんどの Ceph Storage ノードは複数のディスクを使用します。ノードが複数のディスクを使用する場合、director はルートディスクを識別する必要があります。デフォルトのプロビジョニングプロセスでは、director はルートディスクにオーバークラウドイメージを書き込みます。

この手順を使用して、シリアル番号でルートデバイスを識別します。ルートディスクを識別するために使用できるその他のプロパティーの詳細については、「ルートディスクを識別するプロパティー」 を参照してください。

手順

1. 各ノードのハードウェアイントロスペクションからのディスク情報を確認します。以下のコマンドを実行して、ノードのディスク情報を表示します。

```
(undercloud)$ openstack baremetal introspection data save 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 | jq ".inventory.disks"
```

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

```
[
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sda",
    "wwn_vendor_extension": "0x1ea4dcc412a9632b",
    "wwn_with_extension": "0x61866da04f3807001ea4dcc412a9632b",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380700",
    "serial": "61866da04f3807001ea4dcc412a9632b"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdb",
    "wwn_vendor_extension": "0x1ea4e13c12e36ad6",
    "wwn_with_extension": "0x61866da04f380d001ea4e13c12e36ad6",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380d00",
    "serial": "61866da04f380d001ea4e13c12e36ad6"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdc",
    "wwn_vendor_extension": "0x1ea4e31e121cfb45",
    "wwn_with_extension": "0x61866da04f37fc001ea4e31e121cfb45",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f37fc00",
    "serial": "61866da04f37fc001ea4e31e121cfb45"
  }
]
```

2. アンダークラウドで、ノードのルートディスクを設定します。ルートディスクを定義するのに最も適切なハードウェア属性値を指定します。

```
(undercloud)$ openstack baremetal node set --property root_device='{"serial":"<serial_number>"}' <node-uuid>
```

たとえば、ルートデバイスをシリアル番号が **61866da04f380d001ea4e13c12e36ad6** の disk 2 に設定するには、以下のコマンドを実行します。

```
(undercloud)$ openstack baremetal node set --property root_device='{"serial":
"61866da04f380d001ea4e13c12e36ad6"}' 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0
```

注記

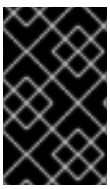
選択したルートディスクから起動するように各ノードの BIOS を設定します。最初にネットワークからのブートを試み、次にルートディスクからのブートを試みるように、ブート順序を設定します。

director は、ルートディスクとして使用する特定のディスクを把握します。**openstack overcloud deploy** コマンドを実行すると、director はオーバークラウドをプロビジョニングし、ルートディスクにオーバークラウドのイメージを書き込みます。

7.7. ルートディスクを識別するプロパティー

以下の属性を定義すると、director がルートディスクを特定するのに役立ちます。

- **model** (文字列): デバイスの ID
- **vendor** (文字列): デバイスのベンダー
- **serial** (文字列): ディスクのシリアル番号
- **hctl** (文字列): SCSI のホスト、チャンネル、ターゲット、Lun
- **size** (整数): デバイスのサイズ (GB 単位)
- **wwn** (文字列): 一意のストレージ ID
- **wwn_with_extension** (文字列): ベンダー拡張子を追加した一意のストレージ ID
- **wwn_vendor_extension** (文字列): 一意のベンダーストレージ ID
- **rotational** (ブール値): 回転式デバイス (HDD) には true、そうでない場合 (SSD) には false
- **name** (文字列): デバイス名 (例: /dev/sdb1)



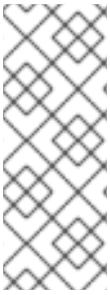
重要

name プロパティーは、永続デバイス名が付いたデバイスにのみ使用します。他のデバイスのルートディスクを設定する際に、**name** を使用しないでください。この値は、ノードのブート時に変更される可能性があります。

7.8. OVERCLOUD-MINIMAL イメージの使用による RED HAT サブスクリプションエンタイトルメントの使用回避

デフォルトでは、プロビジョニングプロセス中 director はルートディスクに QCOW2 **overcloud-full** イメージを書き込みます。**overcloud-full** イメージには、有効な Red Hat サブスクリプションが使用されます。ただし、**overcloud-minimal** イメージを使用して、たとえばベア OS をプロビジョニングすることもできます。この場合、他の OpenStack サービスは使用されないので、サブスクリプションエンタイトルメントは消費されません。

この典型的なユースケースは、Ceph デーモンのみを持つノードをプロビジョニングする場合です。この場合や類似のユースケースでは、**overcloud-minimal** イメージのオプションを使用して、有償の Red Hat サブスクリプションが限度に達するのを避けることができます。**overcloud-minimal** イメージの取得方法についての情報は、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。



注記

Red Hat OpenStack Platform (RHOSP) のサブスクリプションには Open vSwitch (OVS) が含まれますが、**overcloud-minimal** イメージを使用する場合には、OVS などのコアサービスは利用できません。Ceph Storage ノードをデプロイするのに OVS は必要ありません。**ovs_bond** を使用してボンディングを定義する代わりに、**linux_bond** を使用します。**linux_bond** の詳細は、オーバークラウドの高度なカスタマイズの Linux ボンディングのオプション を参照してください。

手順

1. **overcloud-minimal** イメージを使用するように director を設定するには、以下のイメージ定義を含む環境ファイルを作成します。

```
parameter_defaults:
  <roleName>Image: overcloud-minimal
```

2. **<roleName>** をロール名に置き換え、ロール名に **Image** を追加します。Ceph Storage ノードの **overcloud-minimal** イメージの例を以下に示します。

```
parameter_defaults:
  CephStorageImage: overcloud-minimal
```

3. **roles_data.yaml** ロール定義ファイルで、**rhsm_enforce** パラメーターを **False** に設定します。

```
rhsm_enforce: False
```

4. この環境ファイルを **openstack overcloud deploy** コマンドに渡します。

注記

overcloud-minimal イメージでは、標準の Linux ブリッジしかサポートされません。OVS は Red Hat OpenStack Platform のサブスクリプションエンタイトルメントが必要な OpenStack サービスなので、このイメージでは OVS はサポートされません。

7.9. アーキテクチャーに固有なロールの作成

マルチアーキテクチャークラウドを構築する場合には、**roles_data.yaml** ファイルにアーキテクチャー固有のロールを追加する必要があります。以下に示す例では、デフォルトのロールに加えて **ComputePPC64LE** ロールを追加しています。

```
openstack overcloud roles generate \
  --roles-path /usr/share/openstack-tripleo-heat-templates/roles -o ~/templates/roles_data.yaml \
  Controller Compute ComputePPC64LE BlockStorage ObjectStorage CephStorage
```

オーバークラウドの高度なカスタマイズの roles_data ファイルの作成 セクションには、ロールについての情報が記載されています。

7.10. 環境ファイル

アンダークラウドには、オーバークラウドの作成プランを形作るさまざまな heat テンプレートが含まれます。YAML フォーマットの環境ファイルを使用して、オーバークラウドの特性をカスタマイズすることができます。このファイルで、コア heat テンプレートコレクションのパラメーターおよびリソースを上書きします。必要に応じていくつでも環境ファイルを追加することができます。ただし、後で指定する環境ファイルで定義されるパラメーターとリソースが優先されることになるため、環境ファイルの順番は重要です。以下のリストは、環境ファイルの順序の例です。

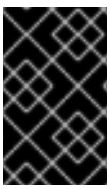
- 各ロールのノード数およびフレーバー。オーバークラウドを作成するには、この情報の追加は不可欠です。
- コンテナー化された OpenStack サービスのコンテナーイメージの場所
- 任意のネットワーク分離ファイル。heat テンプレートコレクションの初期化ファイル (**environments/network-isolation.yaml**) から開始して、次にカスタムの NIC 設定ファイル、最後に追加のネットワーク設定の順番です。詳しい情報は、**Advanced Overcloud Customization** の以下の章を参照してください。
  - 基本的なネットワーク分離
  - カスタムコンポーザブルネットワーク
  - カスタムネットワークインターフェイステンプレート
- 外部のロードバランサーを使用している場合には、外部の負荷分散機能の環境ファイル。詳しい情報は、External Load Balancing for the Overcloud を参照してください。
- Ceph Storage、NFS、または iSCSI 等のストレージ環境ファイル
- Red Hat CDN または Satellite 登録用の環境ファイル
- その他のカスタム環境ファイル

注記

Open Virtual Networking (OVN) は、Red Hat OpenStack Platform 16.1 におけるデフォルトのネットワークメカニズムドライバーです。分散仮想ルーター (DVR) で OVN を使用する場合には、**openstack overcloud deploy** コマンドに **environments/services/neutron-ovn-dvr-ha.yaml** ファイルを追加する必要があります。DVR なしで OVN を使用する場合には、**openstack overcloud deploy** コマンドに **environments/services/neutron-ovn-ha.yaml** ファイルを追加する必要があります。

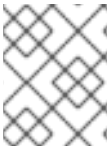
Red Hat では、カスタム環境ファイルを別のディレクトリーで管理することを推奨します (たとえば、**templates** ディレクトリー)。

オーバークラウドの高度な機能のカスタマイズについての詳しい情報は、Advanced Overcloud Customization を参照してください。



重要

基本的なオーバークラウドでは、ブロックストレージにローカルの LVM ストレージを使用しますが、この設定はサポートされません。ブロックストレージには、外部ストレージソリューション (Red Hat Ceph Storage 等) を使用することを推奨します。



注記

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

これ以降の数セクションで、オーバークラウドに必要な環境ファイルの作成について説明します。

7.11. ノード数とフレーバーを定義する環境ファイルの作成

デフォルトでは、director は **baremetal** フレーバーを使用して 1 つのコントローラーノードと 1 つのコンピュートノードを持つオーバークラウドをデプロイします。ただし、この設定は概念検証のためのデプロイメントにしか適しません。異なるノード数およびフレーバーを指定して、デフォルトの設定をオーバーライドすることができます。小規模な実稼働環境では、少なくとも 3 つのコントローラーノードと 3 つのコンピュートノードをデプロイし、特定のフレーバーを割り当ててノードが適切なリソース仕様を持つようにします。ノード数およびフレーバーの割り当てを定義する環境ファイル **node-info.yaml** を作成するには、以下の手順を実施します。

手順

1. **/home/stack/templates/** ディレクトリーに **node-info.yaml** ファイルを作成します。

```
(undercloud) $ touch /home/stack/templates/node-info.yaml
```

2. ファイルを編集し、必要なノード数およびフレーバーを設定します。以下の例には、3 台のコントローラーノードおよび 3 台のコンピュートノードが含まれます。

```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeFlavor: compute
  ControllerCount: 3
  ComputeCount: 3
```

7.12. アンダークラウド CA を信頼するための環境ファイルの作成

アンダークラウドで TLS を使用され認証局 (CA) が一般に信頼できない場合には、SSL エンドポイント暗号化にアンダークラウドが運用する CA を使用することができます。デプロイメントの他の要素からアンダークラウドのエンドポイントにアクセスできるようにするには、アンダークラウドの CA を信頼するようにオーバークラウドノードを設定します。

注記

この手法が機能するためには、オーバークラウドノードにアンダークラウドの公開エンドポイントへのネットワークルートが必要です。スパイン/リーフ型ネットワークに依存するデプロイメントでは、この設定を適用する必要があります。

アンダークラウドで使用することのできるカスタム証明書には、2 つのタイプがあります。

- **ユーザーの提供する証明書**: 自己の証明書を提供している場合がこれに該当します。自己の CA からの証明書、または自己署名の証明書がその例です。この証明書は **undercloud_service_certificate** オプションを使用して渡されます。この場合、自己署名の証明書または CA のどちらかを信頼する必要があります (デプロイメントによります)。

- **自動生成される証明書**: **certmonger** により自己のローカル CA を使用して証明書を生成する場

合がこれに該当します。**undercloud.conf** ファイルの **generate_service_certificate** オプションを使用して、証明書の自動生成を有効にします。この場合、director は CA 証明書 **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** を生成し、アンダークラウドの HAProxy インスタンスがサーバー証明書を使用するように設定します。この CA 証明書を OpenStack Platform に配置するには、証明書を **inject-trust-anchor-hiera.yaml** ファイルに追加します。

以下の例では、**/home/stack/ca.crt.pem** に保存された自己署名の証明書が使われています。自動生成される証明書を使用する場合には、代わりに **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** を使用してください。

手順

1. 証明書ファイルを開き、証明書部分だけをコピーします。鍵を含めないでください。

```
$ vi /home/stack/ca.crt.pem
```

必要となる証明書部分の例を、以下に示します。

```
-----BEGIN CERTIFICATE-----
MIIDITCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCSqGSIb3DQEBCwUAMGExCzAJBgNV
BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECgwH
UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBAMMCzE5Mi4xNjguMC4yMB4XDTE3
-----END CERTIFICATE-----
```

2. 以下に示す内容で **/home/stack/inject-trust-anchor-hiera.yaml** という名称の新たな YAML ファイルを作成し、PEM ファイルからコピーした証明書を追加します。

```
parameter_defaults:
  CAMap:
    undercloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDITCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCSqGSIb3DQEBCwUAMGExCzAJBgNV

BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECgwH

UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBAMMCzE5Mi4xNjguMC4yMB4XDTE3
        -----END CERTIFICATE-----
```

注記

証明書の文字列は、PEM の形式に従う必要があります。

注記

CAMap パラメーターには、SSL/TLS 設定に関連する他の証明書が含まれる場合があります。

director は、オーバークラウドのデプロイメント時に CA 証明書をそれぞれのオーバークラウドノードにコピーします。これにより、それぞれのノードはアンダークラウドの SSL エンドポイントが提示する暗号化を信頼するようになります。環境ファイルに関する詳しい情報は、「オーバークラウドデプロ

イメントへの環境ファイルの追加」 を参照してください。

7.13. 新規デプロイメントでの TSX の無効化

Red Hat Enterprise Linux 8.3 以降、カーネルは、デフォルトで Intel Transactional Synchronization Extensions (TSX) 機能のサポートを無効にします。

ワークロードまたはサードパーティーベンダー用に厳密に要求しない限り、新しいオーバークラウドで TSX を明示的に無効にする必要があります。

環境ファイルで **KernelArgs** heat パラメーターを設定します。

```
parameter_defaults:
  ComputeParameters:
    KernelArgs: "tsx=off"
```

openstack overcloud deploy コマンドを実行する際に、環境ファイルを指定します。

関連情報

- Guidance on Intel TSX impact on OpenStack guests (applies for RHEL 8.3 and above)

7.14. デプロイメントコマンド

OpenStack 環境作成における最後の段階では、**openstack overcloud deploy** コマンドを実行してオーバークラウドを作成します。このコマンドを実行する前に、キーオプションやカスタムの環境ファイルの追加方法を十分に理解しておいてください。

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。オーバークラウドの作成をバックグラウンドプロセスとして実行した場合、デプロイメントの途中で停止してしまう可能性があります。

7.15. デプロイメントコマンドのオプション

以下の表には、**openstack overcloud deploy** コマンドの追加パラメーターをまとめています。

重要

一部のオプションは、本リリースでは **テクノロジープレビュー** として提供されているため、Red Hat では全面的にはサポートしていません。これらはテスト目的にのみご利用いただく機能で、実稼働環境で使用すべきではありません。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

表7.1 デプロイメントコマンドのオプション

パラメーター	説明
--templates [TEMPLATES]	デプロイする heat テンプレートが含まれるディレクトリー。空欄にした場合には、デプロイメント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。
--stack STACK	作成または更新するスタックの名前
-t [TIMEOUT]、--timeout [TIMEOUT]	デプロイメントのタイムアウト時間 (分単位)
--libvirt-type [LIBVIRT_TYPE]	ハイパーバイザーに使用する仮想化タイプ
--ntp-server [NTP_SERVER]	時刻の同期に使用する Network Time Protocol (NTP) サーバー。コンマ区切りリストで複数の NTP サーバーを指定することも可能です (例: **--ntp-server 0.centos.pool.org,1.centos.pool.org**)。高可用性クラスターのデプロイメントの場合には、コントローラーノードが一貫して同じ時刻ソースを参照することが重要です。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。
--no-proxy [NO_PROXY]	環境変数 **no_proxy** のカスタム値を定義します。これにより、プロキシー通信から特定のホスト名は除外されます。
--overcloud-ssh-user OVERCLOUD_SSH_USER	オーバークラウドノードにアクセスする SSH ユーザーを定義します。通常、SSH アクセスは **heat-admin** ユーザーで実行されます。
--overcloud-ssh-key OVERCLOUD_SSH_KEY	オーバークラウドノードへの SSH アクセスに使用する鍵のパスを定義します。
--overcloud-ssh-network OVERCLOUD_SSH_NETWORK	オーバークラウドノードへの SSH アクセスに使用するネットワーク名を定義します。
-e [EXTRA HEAT TEMPLATE]、--extra-template [EXTRA HEAT TEMPLATE]	オーバークラウドのデプロイメントに渡す追加の環境ファイル。このオプションは複数回指定することができます。**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。
--environment-directory	デプロイメントに追加する環境ファイルが含まれるディレクトリー。デプロイメントコマンドでは、これらの環境ファイルは最初に番号順、その後にアルファベット順で処理されます。

パラメーター	説明
-r ROLES_FILE	ロールファイルを定義し、**--templates** ディレクトリーのデフォルトの **roles_data.yaml** を上書きします。ファイルの場所は、絶対パスまたは **--templates** に対する相対パスになります。
-n NETWORKS_FILE	ネットワークファイルを定義し、**--templates** ディレクトリーのデフォルトの network_data.yaml を上書きします。ファイルの場所は、絶対パスまたは **--templates** に対する相対パスになります。
-p PLAN_ENVIRONMENT_FILE	プラン環境ファイルを定義し、**--templates** ディレクトリーのデフォルトの **plan-environment.yaml** を上書きします。ファイルの場所は、絶対パスまたは **--templates** に対する相対パスになります。
--no-cleanup	デプロイメント後に一時ファイルを削除せず、それらの場所をログに記録するには、このオプションを使用します。
--update-plan-only	実際のデプロイメントを実行せずにプランを更新するには、このオプションを使用します。
--validation-errors-nonfatal	オーバークラウドの作成プロセスでは、デプロイメントの前に一連のチェックが行われます。このオプションは、デプロイメント前のチェックで何らかの致命的でないエラーが発生した場合に終了します。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は、デプロイメントの前に一連のチェックが行われます。このオプションは、デプロイメント前のチェックで何らかのクリティカルではない警告が発生した場合に終了します。
--dry-run	オーバークラウドを作成せずにオーバークラウドで検証チェックを実行するには、このオプションを使用します。
--run-validations	**openstack-tripleo-validations** パッケージで提供される外部検証を実行するには、このオプションを使用します。
--skip-postconfig	オーバークラウドデプロイ後の設定を省略するには、このオプションを使用します。

パラメーター	説明
--force-postconfig	オーバークラウドデプロイ後の設定を強制的に行うには、このオプションを使用します。
--skip-deploy-identifier	デプロイメントコマンドで **DeployIdentifier** パラメーターの一意の ID を生成するのを希望しない場合は、このオプションを使用します。ソフトウェア設定のデプロイメントステップは、実際に設定が変更された場合にしか実行されません。このオプションの使用には注意が必要です。特定のロールをスケールアウトする時など、ソフトウェア設定の実行が明らかに不要な場合にしか使用しないでください。
--answers-file ANSWERS_FILE	引数とパラメーターが記載された YAML ファイルへのパス
--disable-password-generation	オーバークラウドサービスのパスワード生成を無効にする場合は、このオプションを使用します。
--deployed-server	事前にプロビジョニングされたオーバークラウドノードをデプロイする場合は、このオプションを使用します。**--disable-validations** と併用されます。
--no-config-download, --stack-only	**config-download** ワークフローを無効にして、スタックおよび関連する OpenStack リソースだけを作成する場合は、このオプションを使用します。このコマンドによってオーバークラウドにソフトウェア設定が適用されることはありません。
--config-download-only	オーバークラウドスタックの作成を無効にして、ソフトウェア設定を適用する **config-download** ワークフローだけを実行する場合は、このオプションを使用します。
--output-dir OUTPUT_DIR	保存した **config-download** の出力に使用するディレクトリー。ディレクトリーは mistral ユーザーが書き込み可能でなければなりません。指定しない場合、director はデフォルトの **/var/lib/mistral/overcloud** を使用します。
--override-ansible-cfg OVERRIDE_ANSIBLE_CFG	Ansible 設定ファイルへのパス。このファイルの設定は、**config-download** がデフォルトで生成する設定を上書きします。
--config-download-timeout CONFIG_DOWNLOAD_TIMEOUT	**config-download** のステップに使用するタイムアウト時間 (分単位)。設定しなければ、スタックデプロイメント操作後の **--timeout** パラメーターの残り時間にかかわらず、director はデフォルトをその時間に設定します。

パラメーター	説明
--limit NODE1,NODE2	config-download Playbook の実行を特定のノードまたはノードセットに制限する場合は、このオプションを使用してノードのコンマ区切りリストを指定します。たとえば、**--limit** オプションは、スケールアップ操作時に新規ノード上でのみ config-download を実行する場合に役立ちます。この引数により、ホスト間のインスタンスのライブマイグレーションが失敗する可能性があります。Running config-download with the ansible-playbook-command.sh script を参照してください。
--tags TAG1,TAG2	**(テクノロジープレビュー)** config-download の特定のタスクセットでデプロイメントを実施する場合は、このオプションを使用して config-download Playbook からのタグのコンマ区切りリストを指定します。
--skip-tags TAG1,TAG2	**(テクノロジープレビュー)** config-download Playbook のタグの一部を省略する場合は、このオプションを使用して省略するタグのコンマ区切りリストを指定します。

オプションの全リストを表示するには、以下のコマンドを実行します。

```
(undercloud) $ openstack help overcloud deploy
```

環境ファイルの **parameter_defaults** セクションに追加する heat テンプレートのパラメーターの使用が優先されるため、一部のコマンドラインパラメーターは古いか非推奨となっています。以下の表では、非推奨となったパラメーターと、それに相当する heat テンプレートのパラメーターを対比しています。

表7.2 非推奨の CLI パラメーターと heat テンプレートのパラメーターの対照表

パラメーター	説明	heat テンプレートのパラメーター
--control-scale	スケールアウトするコントローラーノード数	**ControllerCount**
--compute-scale	スケールアウトするコンピュートノード数	**ComputeCount**

パラメーター	説明	heat テンプレートのパラメーター
--ceph-storage-scale	スケールアウトする Ceph Storage ノードの数	**CephStorageCount**
--block-storage-scale	スケールアウトする Block Storage (cinder) ノード数	**BlockStorageCount**
--swift-storage-scale	スケールアウトする Object Storage (swift) ノード数	**ObjectStorageCount**
--control-flavor	コントローラーノードに使用するフレーバー	**OvercloudControllerFlavor**
--compute-flavor	コンピュートノードに使用するフレーバー	**OvercloudComputeFlavor**
--ceph-storage-flavor	Ceph Storage ノードに使用するフレーバー	**OvercloudCephStorageFlavor**
--block-storage-flavor	Block Storage (cinder) ノードに使用するフレーバー	**OvercloudBlockStorageFlavor**
--swift-storage-flavor	Object Storage (swift) ノードに使用するフレーバー	**OvercloudSwiftStorageFlavor**
--validation-errors-fatal	オーバークラウドの作成プロセスでは、デプロイメントの前に一連のチェックが行われます。このオプションは、デプロイメント前のチェックで何らかの致命的なエラーが発生した場合に終了します。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。	パラメーターのマッピングなし
--disable-validations	デプロイメント前の検証を完全に無効にします。これらの検証は、デプロイメント前の検証として組み込まれていましたが、**openstack-tripleo-validations** パッケージで提供される外部検証に置き換えられています。	パラメーターのマッピングなし

パラメーター	説明	heat テンプレートのパラメーター
--config-download	**config-download** のメカニズムを使用してデプロイメントを実行します。これは現在のデフォルトであり、この CLI のオプションは今後廃止される可能性があります。	パラメーターのマッピングなし
--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する場合は、このオプションを使用します。	**RhsmVars**
--reg-method	このオプションを使用して、オーバークラウドノードの登録方法を定義します。Red Hat Satellite 6 または Red Hat Satellite 5 の場合は **satellite**、カスタマーポータルの場合は **portal** に設定します。	**RhsmVars**
--reg-org [REG_ORG]	登録に使用する組織。	**RhsmVars**
--reg-force	すでに登録済みでもシステムを登録する場合は、このオプションを使用します。	**RhsmVars**

パラメーター	説明	heat テンプレートのパラメーター
--reg-sat-url [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。サーバーが Red Hat Satellite 6 サーバーの場合は、オーバークラウドは **katello-ca-consumer-latest.noarch.rpm** ファイルを取得して **subscription-manager** に登録し、**katello-agent** をインストールします。サーバーが Red Hat Satellite 5 サーバーの場合に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。	**RhsmVars**
--reg-activation-key [REG_ACTIVATION_KEY]	登録に使用するアクティベーションキーを定義する場合は、このオプションを使用します。	**RhsmVars**

これらのパラメーターは、Red Hat OpenStack Platform の今後のリリースで廃止される予定です。

7.16. オーバークラウドデプロイメントへの環境ファイルの追加

オーバークラウドをカスタマイズするための環境ファイルを追加するには、**-e** オプションを使用します。必要に応じていくつでも環境ファイルを追加することができます。ただし、後で指定する環境ファイルで定義されるパラメーターとリソースが優先されることになるため、環境ファイルの順番は重要です。

-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。

以下のコマンドは、本シナリオの初期に定義した環境ファイルを使用してオーバークラウドの作成を開始する方法の一例です。

```
(undercloud) $ openstack overcloud deploy --templates \
  -e /home/stack/templates/node-info.yaml\
  -e /home/stack/containers-prepare-parameter.yaml \
```

```
-e /home/stack/inject-trust-anchor-hiera.yaml \
-r /home/stack/templates/roles_data.yaml \
```

上記のコマンドでは、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ベースとして使用し、オーバークラウドを作成します。

-e /home/stack/templates/node-info.yaml

各ロールに使用するノード数とフレーバーを定義する環境ファイルを追加します。

-e /home/stack/containers-prepare-parameter.yaml

コンテナーイメージ準備の環境ファイルを追加します。このファイルはアンダークラウドのインストール時に生成したもので、オーバークラウドの作成に同じファイルを使用することができます。

-e /home/stack/inject-trust-anchor-hiera.yaml

アンダークラウドにカスタム証明書をインストールする環境ファイルを追加します。

-r /home/stack/templates/roles_data.yaml

(オプション) カスタムロールを使用する、またはマルチアーキテクチャークラウドを有効にする場合に生成されるロールデータ。詳細は、「アーキテクチャーに固有なロールの作成」 を参照してください。

director は、再デプロイおよびデプロイ後の機能にこれらの環境ファイルを必要とします。これらのファイルが含まれていない場合には、オーバークラウドが破損する可能性があります。

これ以降の段階でオーバークラウド設定を変更するには、以下の操作を実施します。

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

オーバークラウドの設定は直接編集しないでください。手動で設定しても、オーバークラウドスタックの更新時に、director が設定を上書きするためです。

7.17. デプロイ前の検証の実行

pre-deployment 検証グループを実行して、デプロイメント要件を確認します。

手順

1. source コマンドで **stackrc** ファイルを読み込みます。

```
$ source ~/stackrc
```

2. この検証には、オーバークラウドプランのコピーが必要です。必要なすべての環境ファイルと共に、オーバークラウドプランをアップロードします。プランのみをアップロードするには、**--update-plan-only** オプションを指定して **openstack overcloud deploy** コマンドを実行します。

```
$ openstack overcloud deploy --templates \
    -e environment-file1.yaml \
    -e environment-file2.yaml \
    ...
    --update-plan-only
```

3. **--group pre-deployment** オプションを指定して、**openstack tripleo validator run** コマンドを実行します。

```
$ openstack tripleo validator run --group pre-deployment
```

4. オーバークラウドにデフォルトのプラン名 **overcloud** 以外の名前を使用する場合は、**--plan** オプションでプラン名を設定します。

```
$ openstack tripleo validator run --group pre-deployment \
    --plan myovercloud
```

5. 検証レポートの結果を確認します。特定の検証からの詳細出力を表示するには、レポートからの特定検証の UUID を指定して **openstack tripleo validator show run --full** コマンドを実行します。

```
$ openstack tripleo validator show run --full <UUID>
```

重要

検証結果が **FAILED** であっても、Red Hat OpenStack Platform のデプロイや実行が妨げられることはありません。ただし、**FAILED** の検証結果は、実稼働環境で問題が発生する可能性があることを意味します。

7.18. オーバークラウドデプロイメントの出力

オーバークラウドの作成が完了すると、オーバークラウドを設定するために実施された Ansible のプレイの概要が director により提示されます。

```
PLAY RECAP *********************************************************************
overcloud-compute-0     : ok=160  changed=67   unreachable=0    failed=0
overcloud-controller-0  : ok=210  changed=93   unreachable=0    failed=0
undercloud              : ok=10   changed=7    unreachable=0    failed=0

Tuesday 15 October 2018  18:30:57 +1000 (0:00:00.107) 1:06:37.514 ******
===============================================================================
```

director により、オーバークラウドへのアクセス情報も提供されます。

```
Ansible passed.
Overcloud configuration completed.
Overcloud Endpoint: http://192.168.24.113:5000
Overcloud Horizon Dashboard URL: http://192.168.24.113:80/dashboard
Overcloud rc file: /home/stack/overcloudrc
Overcloud Deployed
```

7.19. 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と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ます。director は、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。

```
(undercloud) $ source ~/overcloudrc
```

このコマンドにより、アンダークラウド CLI からオーバークラウドと対話するのに必要な環境変数が読み込まれます。コマンドプロンプトが変わり、オーバークラウドと対話していることが示されます。

```
(overcloud) $
```

アンダークラウドとの対話に戻るには、以下のコマンドを実行します。

```
(overcloud) $ source ~/stackrc
(undercloud) $
```

7.20. デプロイ後の検証の実行

post-deployment 検証グループを実行し、デプロイメント後の状態を確認します。

手順

1. source コマンドで **stackrc** ファイルを読み込みます。

```
$ source ~/stackrc
```

2. **--group post-deployment** オプションを指定して、 **openstack tripleo validator run** コマンドを実行します。

```
$ openstack tripleo validator run --group post-deployment
```

3. オーバークラウドにデフォルトのプラン名 **overcloud** 以外の名前を使用する場合は、**--plan** オプションでプラン名を設定します。

```
$ openstack tripleo validator run --group post-deployment \
    --plan myovercloud
```

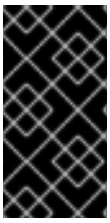
4. 検証レポートの結果を確認します。特定の検証からの詳細出力を表示するには、レポートからの特定検証の UUID を指定して **openstack tripleo validator show run --full** コマンドを実行します。

```
$ openstack tripleo validator show run --full <UUID>
```

重要

検証結果が **FAILED** であっても、Red Hat OpenStack Platform のデプロイや実行が妨げられることはありません。ただし、**FAILED** の検証結果は、実稼働環境で問題が発生する可能性があることを意味します。

第8章 オーバークラウドのデプロイ前に行うベアメタルノードのプロビジョニング



重要

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

オーバークラウドのデプロイメントプロセスには、2 つの主要な操作があります。

- ノードのプロビジョニング
- オーバークラウドのデプロイ

これらの操作を別個のプロセスに分割すると、このプロセスに伴うリスクの一部を軽減し、より効率的に障害点を特定することができます。

1. ベアメタルノードをプロビジョニングする。
    a. ノード定義ファイルを yaml 形式で作成します。
    b. ノード定義ファイルを指定して、プロビジョニングコマンドを実行します。
2. オーバークラウドをデプロイする。
    a. プロビジョニングコマンドにより生成される heat 環境ファイルを指定して、デプロイメントコマンドを実行します。

プロビジョニングプロセスにより、ノードがプロビジョニングされ、ノード数、予測可能なノード配置、カスタムイメージ、カスタム NIC 等のさまざまなノード仕様が含まれる heat 環境ファイルが生成されます。オーバークラウドをデプロイする際に、このファイルをデプロイメントコマンドに追加します。

重要

事前にプロビジョニングされたノードと director がプロビジョニングしたノードを組み合わせることはできません。

8.1. オーバークラウドノードの登録

ディレクターには、ノードのハードウェアと電源管理の詳細を指定するノード定義テンプレートが必要です。このテンプレートは、JSON 形式の **nodes.json** または YAML 形式の **nodes.yaml** で作成できます。

手順

1. ノードをリスト表示する **nodes.json** または **nodes.yaml** という名前のテンプレートを作成します。以下の例に示す JSON および YAML テンプレートを使用して、ノード定義のテンプレートを設定する方法を説明します。

    JSON テンプレートの例

```
{
 "nodes": [{
   "ports": [{
    "address": "aa:aa:aa:aa:aa:aa",
    "physical_network": "ctlplane",
    "local_link_connection": {
     "switch_id": "52:54:00:00:00:00",
     "port_id": "p0"
    }
   }],
   "name": "node01",
   "cpu": "4",
   "memory": "6144",
   "disk": "40",
   "arch": "x86_64",
   "pm_type": "ipmi",
   "pm_user": "admin",
   "pm_password": "p@55w0rd!",
   "pm_addr": "192.168.24.205"
  },
  {
   "ports": [{
    "address": "bb:bb:bb:bb:bb:bb",
    "physical_network": "ctlplane",
    "local_link_connection": {
     "switch_id": "52:54:00:00:00:00",
     "port_id": "p0"
    }
   }],
   "name": "node02",
   "cpu": "4",
   "memory": "6144",
   "disk": "40",
   "arch": "x86_64",
   "pm_type": "ipmi",
   "pm_user": "admin",
   "pm_password": "p@55w0rd!",
   "pm_addr": "192.168.24.206"
  }
 ]
}
```

YAML テンプレートの例

```
nodes:
  - ports:
      - address: aa:aa:aa:aa:aa:aa
        physical_network: ctlplane
        local_link_connection:
          switch_id: 52:54:00:00:00:00
          port_id: p0
    name: "node01"
    cpu: 4
    memory: 6144
    disk: 40
```

```
    arch: "x86_64"
    pm_type: "ipmi"
    pm_user: "admin"
    pm_password: "p@55w0rd!"
    pm_addr: "192.168.24.205"
  - ports:
      - address: bb:bb:bb:bb:bb:bb
        physical_network: ctlplane
        local_link_connection:
          switch_id: 52:54:00:00:00:00
          port_id: p0
    name: "node02"
    cpu: 4
    memory: 6144
    disk: 40
    arch: "x86_64"
    pm_type: "ipmi"
    pm_user: "admin"
    pm_password: "p@55w0rd!"
    pm_addr: "192.168.24.206"
```

このテンプレートには、以下の属性が含まれます。

name

ノードの論理名

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 (**ipmi**) を使用しています。

注記

IPMI が推奨されるサポート対象電源管理ドライバーです。サポート対象電源管理ドライバーの種別およびそのオプションに関する詳細は、30章 *電源管理ドライバー* を参照してください。それらの電源管理ドライバーが想定どおりに機能しない場合には、電源管理に IPMI を使用してください。

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

pm_port (オプション)

特定の IPMI デバイスにアクセスするためのポート

address

(オプション) ノード上のネットワークインターフェイスの MAC アドレスリスト。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。

physical_network

(オプション) プロビジョニング NIC に接続される物理ネットワーク

local_link_connection

(オプション) IPv6 プロビジョニングを使用し、イントロスペクション中に LLDP がローカルリンク接続を正しく反映しない場合は、**local_link_connection** パラメーターの **switch_id** および **port_id** フィールドにダミーのデータを含める必要があります。偽のデー

タを含める方法の詳細は、Using director introspection to collect bare metal node hardware information を参照してください。

cpu

(オプション) ノード上の CPU 数

memory

(オプション) メモリーサイズ (MB 単位)

disk

(オプション) ハードディスクのサイズ (GB 単位)

arch

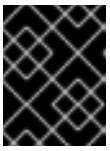
(オプション) システムアーキテクチャー

重要

マルチアーキテクチャークラウドをビルドする場合には、**x86_64** アーキテクチャーを使用するノードと **ppc64le** アーキテクチャーを使用するノードを区別するために **arch** キーが必須です。

2. テンプレートを作成したら、以下のコマンドを実行してフォーマットおよび構文を検証します。

```
$ source ~/stackrc
(undercloud)$ openstack overcloud node import --validate-only ~/nodes.json
```



重要

マルチアーキテクチャーノードの場合は、**--http-boot /var/lib/ironic/tftpboot/** オプションも追加する必要があります。

3. **stack** ユーザーのホームディレクトリーにファイルを保存し (**/home/stack/nodes.json**)、続いて以下のコマンドを実行してテンプレートを director にインポートします。

```
(undercloud)$ openstack overcloud node import ~/nodes.json
```

このコマンドにより、それぞれのノードがテンプレートから director に登録されます。UEFI ブートモードを使用する場合は、各ノードでブートモードも設定する必要があります。UEFI ブートモードを設定せずにノードをイントロスペクトすると、ノードはレガシーモードでブートします。詳細は、Setting the boot mode to UEFI boot mode を参照してください。

4. ノードの登録および設定が完了するまで待ちます。完了したら、ノードが director に正しく登録されていることを確認します。

```
(undercloud)$ openstack baremetal node list
```

8.2. ベアメタルノードハードウェアのインベントリーの作成

ディレクターは、プロファイルのタグ付け、ベンチマーク、および手動のルートディスク割り当てのために、Red Hat OpenStack Platform (RHOSP) デプロイメント内のノードのハードウェアインベントリーを必要とします。

次のいずれかの方法を使用して、ハードウェアインベントリーをディレクターに提供できます。

- Automatic: 各ノードからハードウェア情報を収集するディレクターのイントロスペクションプロセスを使用できます。このプロセスは、各ノードでイントロスペクションエージェントを起動します。イントロスペクションエージェントは、ノードからハードウェアのデータを収集し、そのデータを director に送り返します。Director は、ハードウェアデータを OpenStack 内部データベースに保存します。

- Manual: 各ベアメタルマシンの基本的なハードウェアインベントリーを手動で設定できます。このインベントリーは、ベアメタルプロビジョニングサービス (ironic) に保存され、ベアメタルマシンの管理とデプロイに使用されます。

ディレクターの自動イントロスペクションプロセスには、ベアメタルプロビジョニングサービスポートを手動で設定する方法に比べて、次の利点があります。

- イントロスペクションは、接続されているすべてのポートをハードウェア情報に記録します。これには、**nodes.yaml** でまだ設定されていない場合に PXE ブートに使用するポートも含まれます。

- イントロスペクションは、属性が LLDP を使用して検出可能である場合、各ポートの **local_link_connection** 属性を設定します。手動による方法を使用する場合は、ノードを登録するときに各ポートに **local_link_connection** を設定する必要があります。

- イントロスペクションは、スパイン/リーフ型または DCN のアーキテクチャーをデプロイするときに、ベアメタルプロビジョニングサービスポートの **physical_network** 属性を設定します。

8.2.1. ディレクターのイントロスペクションを使用してベアメタルノードのハードウェア情報を収集する

物理マシンをベアメタルノードとして登録した後、ディレクターイントロスペクションを使用して、ハードウェアの詳細を自動的に追加し、イーサネット MAC アドレスごとにポートを作成できます。

ヒント

自動イントロスペクションの代わりに、ベアメタルノードのハードウェア情報をディレクターに手動で提供できます。詳細は、ベアメタルノードのハードウェア情報を手動で設定する を参照してください。

前提条件

- オーバークラウドのベアメタルノードを登録しました。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

3. pre-introspection 検証グループを実行して、プリイントロスペクションの要件を確認します。

```
(undercloud)$ openstack tripleo validator run --group pre-introspection
```

4. 検証レポートの結果を確認します。

5. オプション: 特定の検証からの詳細な出力を確認します。

```
(undercloud)$ openstack tripleo validator show run --full <validation>
```

- **<validation>** を、確認するレポートの特定の検証の UUID に置き換えます。

重要

検証結果が **FAILED** であっても、Red Hat OpenStack Platform のデプロイや実行が妨げられることはありません。ただし、**FAILED** の検証結果は、実稼働環境で問題が発生する可能性があることを意味します。

6. 各ノードのハードウェア属性を検証します。すべてのノードまたは特定のノードのハードウェア属性を検査できます。

- すべてのノードのハードウェア属性を検査します。

```
(undercloud)$ openstack overcloud node introspect --all-manageable --provide
```

  - **--all-manageable** オプションを使用して、管理状態にあるノードのみをイントロスペクションします。ここでは、すべてのノードが管理状態にあります。
  - **--provide** オプションを使用して、イントロスペクション後に全ノードを **available** の状態に再設定します。

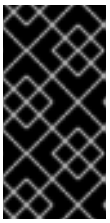
- 特定のノードのハードウェア属性を検査します。

```
(undercloud)$ openstack overcloud node introspect --provide <node1> [node2] [noden]
```

  - **--provide** オプションを使用して、イントロスペクション後に指定されたすべてのノードを **available** 状態にリセットします。
  - **<node1>**、**[node2]**、および **[noden]** までのすべてのノードを、イントロスペクションする各ノードの UUID に置き換えます。

7. 別のターミナルウィンドウで、イントロスペクションの進捗ログを監視します。

```
(undercloud)$ sudo tail -f /var/log/containers/ironic-inspector/ironic-inspector.log
```



重要

イントロスペクションプロセスが完了するまで実行されていることを確認します。イントロスペクションは通常、ベアメタルノードの場合 15 分かかります。ただし、イントロスペクションネットワークのサイズが正しくないと、時間がかかる可能性があり、イントロスペクションが失敗する可能性があります。

8. オプション: IPv6 を介したベアメタルプロビジョニング用にアンダークラウドを設定した場合は、LLDP がベアメタルプロビジョニングサービス (ironic) ポートの **local_link_connection** を設定していることも確認する必要があります。

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
```

- ベアメタルノードのポートに対して Local Link Connection フィールドが空の場合、**local_link_connection** 値に偽のデータを手動で入力する必要があります。次の例では、偽のスイッチ ID を **52:54:00:00:00:00** に設定し、偽のポート ID を **p0** に設定します。

```
(undercloud)$ openstack baremetal port set <port_uuid> \
--local-link-connection switch_id=52:54:00:00:00:00 \
--local-link-connection port_id=p0
```

- ローカルリンク接続フィールドにダミーのデータが含まれていることを確認します。

```
(undercloud)$ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
```

イントロスペクション完了後には、すべてのノードが **available** の状態に変わります。

8.2.2. ベアメタルノードのハードウェア情報を手動で設定する

物理マシンをベアメタルノードとして登録した後、ハードウェアの詳細を手動で追加し、イーサネット MAC アドレスごとにベアメタルポートを作成できます。オーバークラウドをデプロイする前に、少なくとも 1 つのベアメタルポートを作成する必要があります。

ヒント

手動イントロスペクションの代わりに、自動ディレクターイントロスペクションプロセスを使用して、ベアメタルノードのハードウェア情報を収集できます。詳細は、Using director introspection to collect bare metal node hardware information を参照してください。

前提条件

- オーバークラウドのベアメタルノードを登録しました。
- **nodes.json** の登録済みノードの各ポートに **local_link_connection** を設定しました。詳しい情報は、Registering nodes for the overcloud を参照してください。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

3. ノードの機能に **boot_option':'local** を追加して、登録されたノードごとにブートオプションを **local** に設定します。

```
(undercloud)$ openstack baremetal node set \
 --property capabilities="boot_option:local" <node>
```

- **<node>** をベアメタルノードの ID に置き換えてください。

4. ノードドライバーのデプロイカーネルとデプロイ ramdisk を指定します。

```
(undercloud)$ openstack baremetal node set <node> \
 --driver-info deploy_kernel=<kernel_file> \
 --driver-info deploy_ramdisk=<initramfs_file>
```

- **<node>** をベアメタルノードの ID に置き換えてください。
- **<kernel_file>** を **.kernel** イメージへのパス (例: **file:///var/lib/ironic/httpboot/agent.kernel**) に置き換えます。
- **<initramfs_file>** は、**.initramfs** イメージへのパス (例: **file:///var/lib/ironic/httpboot/agent.ramdisk**) に置き換えます。

5. ノードの属性を更新して、ノード上のハードウェアの仕様と一致するようにします。

```
(undercloud)$ openstack baremetal node set <node> \
 --property cpus=<cpu> \
 --property memory_mb=<ram> \
 --property local_gb=<disk> \
 --property cpu_arch=<arch>
```

- **<node>** をベアメタルノードの ID に置き換えてください。
- **<cpu>** は、CPU の数に置き換えます。
- **<ram>** を MB 単位の RAM に置き換えます。
- **<disk>** を GB 単位のディスクサイズに置き換えます。
- **<arch>** は、アーキテクチャータイプに置き換えます。

6. オプション: 各ノードの IPMI 暗号スイートを指定します。

```
(undercloud)$ openstack baremetal node set <node> \
 --driver-info ipmi_cipher_suite=<version>
```

- **<node>** をベアメタルノードの ID に置き換えてください。
- **<version>** は、ノードで使用する暗号スイートのバージョンに置き換えます。以下の有効な値のいずれかに設定します。
  - **3** - ノードは SHA1 暗号スイートで AES-128 を使用します。
  - **17** - ノードは SHA256 暗号スイートで AES-128 を使用します。

7. オプション: 複数のディスクがある場合は、ルートデバイスのヒントを設定して、デプロイメントに使用するディスクをデプロイ ramdisk に通知します。

```
(undercloud)$ openstack baremetal node set <node> \
 --property root_device='{"<property>": "<value>"}'
```

- **<node>** をベアメタルノードの ID に置き換えてください。
- **<property>** と **<value>** は、デプロイメントに使用するディスクの詳細に置き換えます (例: **root_device='{"size": "128"}'**)。
  RHOSP は、次のプロパティーをサポートしています。

- **model** (文字列): デバイスの ID
- **vendor** (文字列): デバイスのベンダー
- **serial** (文字列): ディスクのシリアル番号
- **hctl** (文字列): SCSI のホスト、チャンネル、ターゲット、Lun
- **size** (整数): デバイスのサイズ (GB 単位)
- **wwn** (文字列): 一意のストレージ ID
- **wwn_with_extension** (文字列): ベンダー拡張子を追加した一意のストレージ ID
- **wwn_vendor_extension** (文字列): 一意のベンダーストレージ ID
- **rotational** (ブール値): 回転式デバイス (HDD) には true、そうでない場合 (SSD) には false
- **name** (文字列): デバイス名 (例: /dev/sdb1)。このプロパティーは、永続デバイス名が付いたデバイスにのみ使用してください。

注記

複数のプロパティーを指定する場合には、デバイスはそれらの全プロパティーと一致する必要があります。

8. プロビジョニングネットワーク上の NIC の MAC アドレスを使用してポートを作成することにより、Bare Metal Provisioning サービスにノードのネットワークカードを通知します。

```
(undercloud)$ openstack baremetal port create --node <node_uuid> <mac_address>
```

- **<node_uuid>** をベアメタルノードの一意の ID に置き換えます。
- **<mac_address>** は、PXE ブートに使用する NIC の MAC アドレスに置き換えます。

9. ノードの設定を検証します。

```
(undercloud)$ openstack baremetal node validate <node>
+------------+--------+----------------------------------------------+
| Interface  | Result | Reason                                       |
+------------+--------+----------------------------------------------+
| boot       | False  | Cannot validate image information for node   |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b         |
|            |        | because one or more parameters are missing   |
|            |        | from its instance_info. Missing are:         |
|            |        | ['ramdisk', 'kernel', 'image_source']        |
| console    | None   | not supported                                |
| deploy     | False  | Cannot validate image information for node   |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b         |
|            |        | because one or more parameters are missing   |
|            |        | from its instance_info. Missing are:         |
|            |        | ['ramdisk', 'kernel', 'image_source']        |
| inspect    | None   | not supported                                |
| management | True   |                                              |
| network    | True   |                                              |
```

```
| power      | True   |                                      |
| raid       | True   |                                      |
| storage    | True   |                                      |
+------------+--------+--------------------------------------+
```

有効出力の **Result** は、次のことを示しています。

- **False**: インターフェイスは検証に失敗しました。**instance_info** パラメーター **[\'ramdisk', \'kernel', and \'image_source']** が見つからない場合、Compute サービスがデプロイメントプロセスの最初にこれらのパラメーターを設定するので、この時点では設定されていない可能性があ ります。ディスクイメージ全体を使用している場合は、検証にパスするために **image_source** を設定するだけでよい場合があります。
- **True**: インターフェイスは検証にパスしました。
- **None**: インターフェイスはドライバーでサポートされていません。

8.3. ベアメタルノードのプロビジョニング

新しい YAML ファイル ~/**overcloud-baremetal-deploy.yaml** を作成し、デプロイするベアメタルノードの数と属性を定義して、これらのノードにオーバークラウドロールを割り当てます。プロビジョニングプロセスにより heat 環境ファイルが作成され、そのファイルを **openstack overcloud deploy** コマンドに追加することができます。

前提条件

- アンダークラウドがインストールされます。詳しくは、Installing director を参照してください。
- イントロスペクション済みで、プロビジョニングおよびデプロイメントに利用可能なベアメタルノード。詳細は、Registering nodes for the overcloud および Creating an inventory of the bare metal node hardware を参照してください。

手順

1. source コマンドで **stackrc** アンダークラウド認証情報ファイルを読み込みます。

```
$ source ~/stackrc
```

2. 新しい ~/**overcloud-baremetal-deploy.yaml** ファイルを作成し、プロビジョニングする各ロールのノード数を定義します。たとえば、コントローラーノード 3 つとコンピュートノード 3 つをプロビジョニングするには、以下の構文を使用します。

```
- name: Controller
  count: 3
- name: Compute
  count: 3
```

3. ~/**overcloud-baremetal-deploy.yaml** ファイルで、予測可能なノード配置、カスタムイメージ、カスタム NIC、またはノードに割り当てるその他の属性を定義します。たとえば、以下の例の構文を使用すると、3 つのコントローラーノードがノード **node00**、**node01**、および **node02** に、3 つのコンピュートノードがノード **node04**、**node05**、および **node06** に、それぞれプロビジョニングされます。

```
- name: Controller
  count: 3
  instances:
  - hostname: overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
  - hostname: overcloud-controller-2
    name: node02
- name: Compute
  count: 3
  instances:
  - hostname: overcloud-novacompute-0
    name: node04
  - hostname: overcloud-novacompute-1
    name: node05
  - hostname: overcloud-novacompute-2
    name: node06
```

デフォルトでは、プロビジョニングプロセスには **overcloud-full** イメージが使用されます。**instances** パラメーターで **image** 属性を使用して、カスタムイメージを定義することができます。

```
- name: Controller
  count: 3
  instances:
  - hostname: overcloud-controller-0
    name: node00
    image:
      href: overcloud-custom
```

ノードエントリーごとに手動でノードを定義するのを避けるために、**defaults** パラメーターでデフォルトのパラメーター値を上書きすることもできます。

```
- name: Controller
  count: 3
  defaults:
    image:
      href: overcloud-custom
  instances:
  - hostname :overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
  - hostname: overcloud-controller-2
    name: node02
```

ノード定義ファイルで使用できるパラメーター、属性、および値の詳細は、Bare metal node provisioning attributes を参照してください。

4. ~/**overcloud-baremetal-deploy.yaml** ファイルを指定し、**--output** オプションで出力ファイルを定義して、プロビジョニングコマンドを実行します。

```
(undercloud)$ openstack overcloud node provision \
--stack stack \
```

```
--output ~/overcloud-baremetal-deployed.yaml \
~/overcloud-baremetal-deploy.yaml
```

プロビジョニングプロセスにより、**--output** オプションで指定する名前の heat 環境ファイルが生成されます。このファイルには、ノード定義が含まれます。オーバークラウドをデプロイする際に、このファイルをデプロイメントコマンドに追加します。

5. 別のターミナルでノードをモニタリングし、プロビジョニングが正常に行われていることを確認します。プロビジョニングプロセスでは、ノードの状態が **available** から **active** に変わります。

```
(undercloud)$ watch openstack baremetal node list
```

metalsmith ツールを使用して、割り当てや neutron ポートなどを含むノードの統合ビューを取得します。

```
(undercloud)$ metalsmith list
```

openstack baremetal allocation コマンドを使用して、ノードのホスト名への関連付けを確認することもできます。

```
(undercloud)$ openstack baremetal allocation list
```

ノードが正常にプロビジョニングされると、オーバークラウドをデプロイすることができます。詳細は、Configuring a basic overcloud with pre-provisioned nodes を参照してください。

8.4. ベアメタルノードのスケールアップ

既存オーバークラウドのベアメタルノード数を増やすには、**~/overcloud-baremetal-deploy.yaml** ファイルのノード数を増やして、オーバークラウドを再デプロイします。

前提条件

- アンダークラウドの正常なインストール。詳しくは、Installing director を参照してください。
- オーバークラウドの正常なデプロイメント。詳細は、Configuring a basic overcloud with pre-provisioned nodes を参照してください。
- イントロスペクション済みで、プロビジョニングおよびデプロイメントに利用可能なベアメタルノード。詳細については、Registering nodes for the overcloud と Creating an inventory of the bare-metal node hardware を参照してください。

手順

1. source コマンドで **stackrc** アンダークラウド認証情報ファイルを読み込みます。

```
$ source ~/stackrc
```

2. ベアメタルノードのプロビジョニングに使用した **~/overcloud-baremetal-deploy.yaml** ファイルを編集し、スケールアップするロールの **count** パラメーターを増やします。たとえば、オーバークラウドにコンピュートノードが 3 つある場合に、以下のスニペットを使用してコンピュートノード数を 10 に増やします。

```
- name: Controller
  count: 3
- name: Compute
  count: 10
```

instances パラメーターを使用して、予測可能なノード配置を追加することもできます。使用可能なパラメーターと属性の詳細は、Bare metal node provisioning attributes を参照してください。

3. ~/**overcloud-baremetal-deploy.yaml** ファイルを指定し、**--output** オプションで出力ファイルを定義して、プロビジョニングコマンドを実行します。

```
(undercloud)$ openstack overcloud node provision \
--stack stack \
--output ~/overcloud-baremetal-deployed.yaml \
~/overcloud-baremetal-deploy.yaml
```

4. **openstack baremetal node list** コマンドを使用して、プロビジョニングの進捗をモニタリングします。

5. デプロイメントに該当するその他の環境ファイルと共に、プロビジョニングコマンドによって生成される ~/**overcloud-baremetal-deployed.yaml** ファイルを指定して、オーバークラウドをデプロイします。

```
(undercloud)$ openstack overcloud deploy \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-
environment.yaml \
  -e ~/overcloud-baremetal-deployed.yaml \
  --deployed-server \
  --disable-validations \
  ...
```

8.5. ベアメタルノードのスケールダウン

スタックから削除するノードを ~/**overcloud-baremetal-deploy.yaml** ファイルでタグ付けし、オーバークラウドを再デプロイしてから、**--baremetal-deployment** オプションを指定して **openstack overcloud node delete** コマンドにこのファイルを追加します。

前提条件

- アンダークラウドの正常なインストール。詳細は、*4章 アンダークラウドへの director のインストール* を参照してください。
- オーバークラウドの正常なデプロイメント。詳細は、*9章 事前にプロビジョニングされたノードを使用した基本的なオーバークラウドの設定* を参照してください。
- スタックから削除する 1 つ以上のベアメタルノード

手順

1. source コマンドで **stackrc** アンダークラウド認証情報ファイルを読み込みます。

```
$ source ~/stackrc
```

2. ベアメタルノードのプロビジョニングに使用した ~/**overcloud-baremetal-deploy.yaml** ファイルを編集し、スケールダウンするロールの **count** パラメーターを減らします。また、スタックから削除するノードごとに以下の属性を定義する必要もあります。

   - ノードの名前
   - ノードに関連付けられたホスト名
   - **provisioned: false** 属性
     たとえば、スタックからノード **overcloud-controller-1** を削除するには、~/**overcloud-baremetal-deploy.yaml** ファイルに以下のスニペットを追加します。

     ```
     - name: Controller
       count: 2
       instances:
       - hostname: overcloud-controller-0
         name: node00
       - hostname: overcloud-controller-1
         name: node01
         # Removed from cluster due to disk failure
         provisioned: false
       - hostname: overcloud-controller-2
         name: node02
     ```

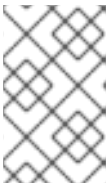
3. ~/**overcloud-baremetal-deploy.yaml** ファイルを指定し、**--output** オプションで出力ファイルを定義して、プロビジョニングコマンドを実行します。

   ```
   (undercloud)$ openstack overcloud node provision \
   --stack stack \
   --output ~/overcloud-baremetal-deployed.yaml \
   ~/overcloud-baremetal-deploy.yaml
   ```

4. デプロイメントに該当するその他の環境ファイルと共に、プロビジョニングコマンドによって生成される ~/**overcloud-baremetal-deployed.yaml** ファイルを指定して、オーバークラウドを再デプロイします。

   ```
   (undercloud)$ openstack overcloud deploy \
     ...
     -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-
   environment.yaml \
     -e ~/overcloud-baremetal-deployed.yaml \
     --deployed-server \
     --disable-validations \
     ...
   ```

   オーバークラウドの再デプロイ後、**provisioned: false** 属性で定義したノードがスタックには存在しなくなります。ただし、これらのノードは provisioned の状態で稼働したままです。



注記

スタックから一時的にノードを削除する場合は、オーバークラウドを属性 **provisioned: false** でデプロイしてから属性 **provisioned: true** で再デプロイすることで、ノードをスタックに戻すことができます。

5. **--baremetal-deployment** オプションで **~/overcloud-baremetal-deploy.yaml** ファイルを指定して、**openstack overcloud node delete** コマンドを実行します。

```
(undercloud)$ openstack overcloud node delete \
--stack stack \
--baremetal-deployment ~/overcloud-baremetal-deploy.yaml
```

注記

スタックから削除するノードを、**openstack overcloud node delete** コマンドのコマンド引数に含めないでください。

8.6. ベアメタルノードプロビジョニングの属性

以下の表で、**openstack baremetal node provision** コマンドでベアメタルノードをプロビジョニングする際に使用できるパラメーター、属性、および値について説明します。

表8.1 ロールパラメーター

パラメーター	値
name	ロール名 (必須)
count	このロール用にプロビジョニングするノード数。デフォルト値は **1** です。
defaults	**instances** エントリープロパティーのデフォルト値のディクショナリー。**instances** エントリーのプロパティーは、**defaults** パラメーターで指定したデフォルトを上書きします。
instances	特定のノードの属性を指定するために使用可能な値のディクショナリー。**instances** パラメーターでサポートされるプロパティーについての詳しい情報は、 表8.2「**instances** および **defaults** パラメーター」 を参照してください。このリストの長さは、**count** パラメーターの値よりも大きくすることはできません。
hostname_format	このロールのデフォルトのホスト名形式を上書きします。デフォルトの形式では、小文字のロール名を使用します。たとえば、Controller ロールのデフォルト形式は、**%stackname%-controller-%index%** です。Compute ロールだけは、ロール名のルールに従いません。Compute のデフォルトの形式は、**%stackname%-novacompute-%index%** です。

構文の例

以下の例では、**name** はノードの論理名を指し、**hostname** は、オーバークラウドスタック名、ロール、および増分インデックスから派生して生成されたホスト名を指します。すべてのコントローラーサーバーは、デフォルトのカスタムイメージ **overcloud-full-custom** を使用し、予測可能なノード上に

あります。コンピュートサーバーの 1 つは、カスタムホスト名 **overcloud-compute-special** の **node04** 上に予測的に配置され、残りの 99 のコンピュートサーバーは、使用可能なノードのプールから自動的に割り当てられるノード上にあります。

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
  instances:
  - hostname: overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
  - hostname: overcloud-controller-2
    name: node02
- name: Compute
  count: 100
  instances:
  - hostname: overcloud-compute-special
    name: node04
```

表8.2 instances および defaults パラメーター

パラメーター	値
hostname	ホスト名が **hostname_format** のパターンに適合する場合、他のプロパティーがこのホスト名に割り当てられたノードに適用されます。そうでない場合は、このノードにカスタムのホスト名を使用できます。
name	プロビジョニングするノードの名前
image	ノードにプロビジョニングするイメージの詳細。**image** パラメーターでサポートされるプロパティーについての詳しい情報は、表8.3「**image** パラメーター」 を参照してください。
capabilities	ノードのケイパビリティーを照合する際の選択基準
nics	要求された NIC を表すディクショナリーのリスト。**nics** パラメーターでサポートされるプロパティーについての詳しい情報は、表8.4「**nic** パラメーター」 を参照してください。
profile	高度なプロファイルマッチングを使用する際の選択基準

パラメーター	値
provisioned	このノードがプロビジョニングされているかどうかを判断するブール値。デフォルト値は **true** です。プロビジョニングされていないノードには **false** を使用します。詳細は、Scaling down bare metal nodes を参照してください。
resource_class	ノードのリソースクラスを照合する際の選択基準。デフォルト値は **baremetal** です。
root_size_gb	ルートパーティションのサイズ (GiB 単位)。デフォルト値は **49** です。
swap_size_mb	スワップパーティションのサイズ (MiB 単位)
traits	ノード特性を照合する際の選択基準としての特性のリスト

構文の例
以下の例では、すべてのコントローラーサーバーでカスタムのデフォルトオーバークラウドイメージ **overcloud-full-custom** が使用されます。コントローラーサーバー **overcloud-controller-0** は **node00** 上に予測的に配置され、カスタム設定のルートパーティションおよびスワップパーティションサイズを持ちます。他の 2 つのコントローラーサーバーは、利用可能なノードのプールから自動的に割り当てられるノード上にあり、デフォルト設定のルートパーティションおよびスワップパーティションサイズを持ちます。

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
  instances:
  - hostname: overcloud-controller-0
    name: node00
    root_size_gb: 140
    swap_size_mb: 600
```

表8.3 image パラメーター

パラメーター	値
href	glance イメージの参照、またはルートパーティションもしくは完全なディスクイメージの URL。サポートされる URL スキームは、**file://**、**http://**、および **https://** です。値が有効な URL ではない場合、この値は有効な glance イメージの参照でなければなりません。

パラメーター	値
checksum	href が URL の場合、この値はルートパーティションまたは完全なディスクイメージの SHA512 チェックサムでなければなりません。
kernel	glance イメージの参照またはカーネルイメージの URL。パーティションイメージに対してのみ、この属性を使用します。
ramdisk	glance イメージの参照または ramdisk イメージの URL。パーティションイメージに対してのみ、この属性を使用します。

構文の例
以下の例では、3 つのコントローラーサーバーは、すべて利用可能なノードのプールから自動的に割り当てられるノード上にあります。この環境の全コントローラーサーバーは、デフォルトのカスタムイメージ **overcloud-full-custom** を使用します。

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
      checksum: 1582054665
      kernel: file:///var/lib/ironic/images/overcloud-full-custom.vmlinuz
      ramdisk: file:///var/lib/ironic/images/overcloud-full-custom.initrd
```

表8.4 nic パラメーター

パラメーター	値
fixed_ip	この NIC に使用する特定の IP アドレス
network	この NIC のポートを作成する neutron ネットワーク
subnet	この NIC のポートを作成する neutron サブネット
port	新しいポートを作成する代わりに使用する既存の neutron ポート

構文の例
以下の例では、3 つのコントローラーサーバーは、すべて利用可能なノードのプールから自動的に割り当てられるノード上にあります。この環境の全コントローラーサーバーは、デフォルトのカスタムイメージ **overcloud-full-custom** を使用します。また、特定のネットワーク要件を持ちます。

```
- name: Controller
  count: 3
  defaults:
```

```
image:
  href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
  nics:
    network: custom-network
    subnet: custom-subnet
```

第9章 事前にプロビジョニングされたノードを使用した基本的なオーバークラウドの設定

本章では、事前にプロビジョニングされたノードを使用して Red Hat OpenStack Platform (RHOSP) 環境を設定するのに使用できる基本的な設定手順を説明します。以下のシナリオは、標準のオーバークラウド作成のシナリオとはさまざまな点で異なります。

- 外部ツールを使用してノードをプロビジョニングしてから、director でオーバークラウドの設定のみを制御することができます。
- director のプロビジョニングの方法に依存せずにノードを使用することができます。これは、電源管理制御を設定せずにオーバークラウドを作成する場合や、DHCP/PXE ブートの制限があるネットワークを使用する場合に便利です。
- director では、ノードを管理するのに OpenStack Compute (nova)、OpenStack Bare Metal (ironic)、または OpenStack Image (glance) を使用しません。
- 事前にプロビジョニングされたノードでは、QCOW2 overcloud-full イメージに依存しないカスタムパーティションレイアウトを使用することができます。

このシナリオには、カスタム機能を持たない基本的な設定のみが含まれています。ただし、オーバークラウドの高度なカスタマイズ に記載の手順に従って、この基本的なオーバークラウドに高度な設定オプションを追加し、仕様に合わせてカスタマイズすることができます。

重要

事前にプロビジョニングされたノードと director がプロビジョニングしたノードを組み合わせることはできません。

9.1. 事前にプロビジョニングされたノードの要件

事前にプロビジョニングされたノードを使用してオーバークラウドのデプロイメントを開始する前に、以下の項目が環境に存在していること確認してください。

- *4章 アンダークラウドへの director のインストール* で作成した director ノード
- ノードに使用するベアメタルマシンのセット。必要なノード数は、作成予定のオーバークラウドのタイプにより異なります。これらのマシンは、各ノード種別に設定された要件に従う必要があります。これらのノードには、ホストオペレーティングシステムとして Red Hat Enterprise Linux 8.2 をインストールする必要があります。Red Hat では、利用可能な最新バージョンの使用を推奨します。
- 事前にプロビジョニングされたノードを管理するためのネットワーク接続 1 つ。このシナリオでは、オーケストレーションエージェントの設定のために、ノードへの SSH アクセスが中断されないようにする必要があります。
- コントロールプレーンネットワーク用のネットワーク接続 1 つ。このネットワークには、主に 2 つのシナリオがあります。
  - プロビジョニングネットワークをコントロールプレーンとして使用するデフォルトのシナリオ:このネットワークは通常、事前にプロビジョニングされたノードから director へのレイヤー 3 (L3) を使用したルーティング可能なネットワーク接続です。このシナリオの例では、以下の IP アドレスの割り当てを使用します。

    表9.1 プロビジョニングネットワークの IP 割り当て

ノード名	IP アドレス
director	192.168.24.1
Controller 0	192.168.24.2
Compute 0	192.168.24.3

    - 別のネットワークを使用するシナリオ:director のプロビジョニングネットワークがプライベートのルーティング不可能なネットワークの場合には、サブネットからノードの IP アドレスを定義して、パブリック API エンドポイント経由で director と通信することができます。このシナリオの要件についての詳細は、「事前にプロビジョニングされたノードへの別ネットワークの使用」を参照してください。

- この例で使用するその他すべてのネットワーク種別も、OpenStack サービス用のコントロールプレーンネットワークを使用します。ただし、ネットワークトラフィックの他のタイプに追加でネットワークを作成することができます。

- いずれかのノードで Pacemaker リソースが使用される場合、サービスユーザー **hacluster** およびサービスグループ **haclient** の UID/GID は、**189** でなければなりません。これは CVE-2018-16877 に対応するためです。オペレーティングシステムと共に Pacemaker をインストールした場合、インストールによりこれらの ID が自動的に作成されます。ID の値が正しく設定されていない場合は、アーティクル OpenStack minor update / fast-forward upgrade can fail on the controller nodes at pacemaker step with "Could not evaluate: backup_cib" の手順に従って ID の値を変更します。

- 一部のサービスが誤った IP アドレスにバインドされてデプロイメントに失敗するのを防ぐために、**/etc/hosts** ファイルに **node-name=127.0.0.1** のマッピングが含まれないようにします。

9.2. 事前にプロビジョニングされたノードでのユーザーの作成

事前にプロビジョニングされたノードを使用してオーバークラウドを設定する場合、director はオーバークラウドノードに SSH アクセスする必要があります。事前にプロビジョニングされたノードで SSH 鍵認証のユーザーを作成し、そのユーザーに対してパスワード不要の sudo アクセスを設定する必要があります。事前にプロビジョニングされたノードでユーザーを作成したら、**openstack overcloud deploy** コマンドで **--overcloud-ssh-user** および **--overcloud-ssh-key** オプションを使用して、事前にプロビジョニングされたノードでオーバークラウドを作成することができます。

デフォルトでは、オーバークラウドの SSH ユーザーおよびオーバークラウドの SSH 鍵の値は、それぞれ **stack** ユーザーおよび **~/.ssh/id_rsa** です。**stack** ユーザーを作成するには、以下の手順を実施します。

手順

1. 各オーバークラウドノードで、**stack** ユーザーを作成して、それぞれにパスワードを設定します。コントローラーノードで、以下の例に示すコマンドを実行します。

```
[root@controller-0 ~]# useradd stack
[root@controller-0 ~]# passwd stack  # specify a password
```

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

```
[root@controller-0 ~]# echo "stack ALL=(root) NOPASSWD:ALL" | tee -a
/etc/sudoers.d/stack
[root@controller-0 ~]# chmod 0440 /etc/sudoers.d/stack
```

3. 事前にプロビジョニングされた全ノードで **stack** ユーザーの作成と設定を行った後に、director ノードから各オーバークラウドノードに **stack** ユーザーの公開 SSH 鍵をコピーします。director の公開 SSH 鍵をコントローラーノードにコピーするには、以下の例に示すコマンドを実行します。

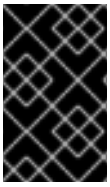
```
[stack@director ~]$ ssh-copy-id stack@192.168.24.2
```

重要

SSH 鍵をコピーするには、各オーバークラウドノードの SSH 設定で一時的に **PasswordAuthentication Yes** を設定しなければならない場合があります。SSH 鍵をコピーした後に **PasswordAuthentication No** を設定し、今後は SSH 鍵を使用して認証を行います。

9.3. 事前にプロビジョニングされたノードのオペレーティングシステムの登録

それぞれのノードには、Red Hat サブスクリプションへのアクセスが必要です。ノードを Red Hat コンテンツ配信ネットワークに登録するには、各ノードで以下の手順を実施します。



重要

記載されたリポジトリーのみを有効にします。追加のリポジトリーを使用すると、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。他のリポジトリーは有効にしないでください。

手順

1. 登録コマンドを実行して、プロンプトが表示されたらカスタマーポータルのユーザー名とパスワードを入力します。

```
[heat-admin@controller-0 ~]$ sudo subscription-manager register
```

2. Red Hat OpenStack Platform 16.1 のエンタイトルメントプールを検索します。

```
[heat-admin@controller-0 ~]$ sudo subscription-manager list --available --all --matches="Red
Hat OpenStack"
```

3. 上記のステップで特定したプール ID を使用して、Red Hat OpenStack Platform 16 のエンタイトルメントをアタッチします。

```
[heat-admin@controller-0 ~]$ sudo subscription-manager attach --pool=pool_id
```

4. デフォルトのリポジトリーをすべて無効にします。

```
[heat-admin@controller-0 ~]$ sudo subscription-manager repos --disable=*
```

5. 必要な Red Hat Enterprise Linux リポジトリーを有効にします。

   a. x86_64 システムの場合は、以下のコマンドを実行します。

   ```
   [heat-admin@controller-0 ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-eus-rpms --enable=rhel-8-for-x86_64-appstream-eus-rpms --enable=rhel-8-for-x86_64-highavailability-eus-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms --enable=openstack-16.1-for-rhel-8-x86_64-rpms --enable=fast-datapath-for-rhel-8-x86_64-rpms --enable=advanced-virt-for-rhel-8-x86_64-rpms
   ```

   b. POWER システムの場合は、以下のコマンドを実行します。

   ```
   [heat-admin@controller-0 ~]$ sudo subscription-manager repos --enable=rhel-8-for-ppc64le-baseos-rpms --enable=rhel-8-for-ppc64le-appstream-rpms --enable=rhel-8-for-ppc64le-highavailability-rpms --enable=ansible-2.8-for-rhel-8-ppc64le-rpms --enable=openstack-16-for-rhel-8-ppc64le-rpms --enable=fast-datapath-for-rhel-8-ppc64le-rpms
   ```

6. **container-tools** リポジトリーモジュールをバージョン **2.0** に設定します。

   ```
   [heat-admin@controller-0 ~]$ sudo dnf module disable -y container-tools:rhel8
   [heat-admin@controller-0 ~]$ sudo dnf module enable -y container-tools:2.0
   ```

7. オーバークラウドで Ceph Storage ノードを使用する場合は、該当する Ceph Storage リポジトリーを有効にします。

   ```
   [heat-admin@cephstorage-0 ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-rpms --enable=rhel-8-for-x86_64-appstream-rpms --enable=rhel-8-for-x86_64-highavailability-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms --enable=openstack-16.1-deployment-tools-for-rhel-8-x86_64-rpms
   ```

8. **dnf update** を実行する前に、各ノードを Red Hat Enterprise Linux 8.2 にロックします。

   ```
   [heat-admin@controller-0 ~]$ sudo subscription-manager release --set=8.2
   ```

9. システムを更新して、ベースシステムパッケージが最新の状態になるようにします。

   ```
   [heat-admin@controller-0 ~]$ sudo dnf update -y
   [heat-admin@controller-0 ~]$ sudo reboot
   ```

このノードをオーバークラウドに使用する準備ができました。

9.4. DIRECTOR への SSL/TLS アクセスの設定

director が SSL/TLS を使用する場合は、事前にプロビジョニングされたノードには、director の SSL/TLS 証明書の署名に使用する認証局ファイルが必要です。独自の認証局を使用する場合には、各オーバークラウドノード上で以下の操作を実施します。

手順

1. 事前にプロビジョニングされた各ノードの **/etc/pki/ca-trust/source/anchors/** ディレクトリーに認証局ファイルをコピーします。

2. 各オーバークラウドノード上で以下のコマンドを実行します。

```
[root@controller-0 ~]#  sudo update-ca-trust extract
```

この手順により、オーバークラウドノードが director のパブリック API に SSL/TLS 経由でアクセスできるようになります。

9.5. コントロールプレーンのネットワークの設定

事前にプロビジョニングされたオーバークラウドノードは、標準の HTTP 要求を使用して director からメタデータを取得します。これは、オーバークラウドノードでは以下のいずれかに対して L3 アクセスが必要であることを意味します。

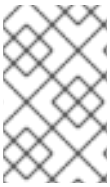
- director のコントロールプレーンネットワーク。これは、**undercloud.conf** ファイルの **network_cidr** パラメーターで定義するサブネットです。オーバークラウドノードには、このサブネットへの直接アクセスまたはルーティング可能なアクセスのいずれかが必要です。
- **undercloud.conf** ファイルの **undercloud_public_host** パラメーターで指定する director のパブリック API エンドポイント。コントロールプレーンへの L3 ルートがない場合や、SSL/TLS 通信を使用する場合に、このオプションを利用することができます。パブリック API エンドポイントを使用するオーバークラウドノードの設定についての詳細は、「事前にプロビジョニングされたノードへの別ネットワークの使用」を参照してください。

director は、コントロールプレーンネットワークを使用して標準のオーバークラウドを管理、設定します。事前にプロビジョニングされたノードを使用したオーバークラウドの場合には、director と事前にプロビジョニングされたノード間の通信に対応するために、ネットワーク設定を変更しなければならない場合があります。

ネットワーク分離の使用

ネットワーク分離を使用することで、サービスをグループ化してコントロールプレーンなど特定のネットワークを使用することができます。オーバークラウドの高度なカスタマイズ には、ネットワーク分離の設定が複数記載されています。コントロールプレーン上のノードに特定の IP アドレスを定義することも可能です。ネットワーク分離や予測可能なノード配置方法の策定に関する詳しい情報は、**オーバークラウドの高度なカスタマイズ**の以下のセクションを参照してください。

- 基本的なネットワーク分離
- ノード配置の制御



注記

ネットワーク分離を使用する場合には、NIC テンプレートに、アンダークラウドのアクセスに使用する NIC を含めないようにしてください。これらのテンプレートにより NIC が再設定され、デプロイメント時に接続性や設定の問題が発生する可能性があります。

IP アドレスの割り当て

ネットワーク分離を使用しない場合には、単一のコントロールプレーンを使用して全サービスを管理することができます。これには、各ノード上のコントロールプレーンの NIC を手動で設定して、コントロールプレーンネットワークの範囲内の IP アドレスを使用するようにする必要があります。director のプロビジョニングネットワークをコントロールプレーンとして使用する場合には、選択したオーバークラウドの IP アドレスが、プロビジョニング (**dhcp_start** および **dhcp_end**) とイントロスペクション (**inspection_iprange**) 両方の DHCP 範囲外になるようにしてください。

標準のオーバークラウド作成中には、director は OpenStack Networking (neutron) ポートを作成し、プロビジョニング/コントロールプレーンネットワークのオーバークラウドノードに IP アドレスを自動的に割り当てます。ただし、これにより、各ノードに手動で設定する IP アドレスとは異なるアドレスを director が割り当ててしまう可能性があります。このような場合には、予測可能な IP アドレス割り当て方法を使用して、director がコントロールプレーン上で事前にプロビジョニングされた IP の割り当てを強制的に使用するようにしてください。

たとえば、予測可能な IP アドレス設定を実装するために、以下の IP アドレスを割り当てた環境ファイル **ctlplane-assignments.yaml** を使用することができます。

```
resource_registry:
  OS::TripleO::DeployedServer::ControlPlanePort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml

parameter_defaults:
  DeployedServerPortMap:
    controller-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.24.2
      subnets:
        - cidr: 192.168.24.0/24
      network:
        tags:
          192.168.24.0/24
    compute-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.24.3
      subnets:
        - cidr: 192.168.24.0/24
      network:
        tags:
          - 192.168.24.0/24
```

この例では、**OS::TripleO::DeployedServer::ControlPlanePort** リソースはパラメーターセットを director に渡して、事前にプロビジョニングされたノードの IP 割り当てを定義します。**DeployedServerPortMap** パラメーターを使用して、各オーバークラウドノードに対応する IP アドレスおよびサブネット CIDR を定義します。マッピングにより、以下の属性が定義されます。

1. 割り当ての名前。形式は **<node_hostname>-<network>** です。ここで、**<node_hostname>** の値はノードの短縮ホスト名で、**<network>** はネットワークの小文字を使用した名前です。たとえば、**controller-0.example.com** であれば **controller-0-ctlplane** となり、**compute-0.example.com** の場合は **compute-0-ctlplane** となります。

2. 以下のパラメーターパターンを使用する IP 割り当て

   - **fixed_ips/ip_address**: コントロールプレーンの固定 IP アドレスを定義します。複数の IP アドレスを定義する場合には、複数の **ip_address** パラメーターをリストで指定してください。

   - **subnets/cidr**: サブネットの CIDR 値を定義します。

本章のこの後のセクションで、作成された環境ファイル (**ctlplane-assignments.yaml**) を **openstack overcloud deploy** コマンドの一部として使用します。

9.6. 事前にプロビジョニングされたノードへの別ネットワークの使用

デフォルトでは、director はオーバークラウドのコントロールプレーンとしてプロビジョニングネットワークを使用します。ただし、このネットワークが分離されてルーティング不可能な場合には、ノードは設定中に director の Internal API と通信することができません。このような状況では、ノードに別のネットワークを定義して、パブリック API 経由で director と通信するように設定しなければならない場合があります。

このシナリオには、いくつかの要件があります。

- オーバークラウドノードは、「コントロールプレーンのネットワークの設定」からの基本的なネットワーク設定に対応する必要があります。
- パブリック API エンドポイントを使用できるように director 上で SSL/TLS を有効化する必要があります。詳しい情報は、「director の設定パラメーター」 および *20章カスタム SSL/TLS 証明書の設定* を参照してください。
- director 向けにアクセス可能な完全修飾ドメイン名 (FQDN) を定義する必要があります。この FQDN は、director にルーティング可能な IP アドレスを解決する必要があります。**undercloud.conf** ファイルの **undercloud_public_host** パラメーターを使用して、この FQDN を設定します。

本項に記載する例では、主要なシナリオとは異なる IP アドレスの割り当てを使用します。

表9.2 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 アドレスまたは FQDN
director (Internal API)	192.168.24.1 (プロビジョニングネットワークおよびコントロールプレーン)
director (パブリック API)	10.1.1.1 / director.example.com
オーバークラウドの仮想 IP	192.168.100.1
Controller 0	192.168.100.2
Compute 0	192.168.100.3

以下の項では、オーバークラウドノードに別のネットワークが必要な場合の追加の設定について説明します。

IP アドレスの割り当て

IP の割り当て方法は、「コントロールプレーンのネットワークの設定」に記載の手順と類似しています。ただし、コントロールプレーンはデプロイしたサーバーからルーティング可能ではないので、**DeployedServerPortMap** パラメーターを使用して、コントロールプレーンにアクセスする仮想 IP アドレスを含め、選択したオーバークラウドノードのサブネットから IP アドレスを割り当てる必要があります。以下の例は、「コントロールプレーンのネットワークの設定」 からの **ctlplane-assignments.yaml** 環境ファイルを修正したバージョンで、このネットワークアーキテクチャーに対応します。

```
resource_registry:
  OS::TripleO::DeployedServer::ControlPlanePort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml
  OS::TripleO::Network::Ports::ControlPlaneVipPort: /usr/share/openstack-tripleo-heat-
```

```
templates/deployed-server/deployed-neutron-port.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
  OS::TripleO::Network::Ports::OVNDBsVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml (1)

parameter_defaults:
  NeutronPublicInterface: eth1
  DeployedServerPortMap:
    control_virtual_ip:
      fixed_ips:
        - ip_address: 192.168.100.1
      subnets:
        - cidr: 24
    controller-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.100.2
      subnets:
        - cidr: 24
    compute-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.100.3
      subnets:
        - cidr: 24
```

(1) **RedisVipPort** リソースおよび **OVNDBsVipPort** リソースは、**network/ports/noop.yaml** にマップされます。デフォルトの Redis および OVNDB の VIP アドレスはコントロールプレーンから取得されるため、このマッピングが必要です。このような場合には、**noop** を使用して、このコントロールプレーンマッピングを無効化します。

9.7. 事前にプロビジョニングされたノードのホスト名のマッピング

事前にプロビジョニングされたノードを設定する場合には、heat ベースのホスト名をそれらの実際のホスト名にマッピングして、**ansible-playbook** が解決されたホストに到達できるようにする必要があります。それらの値は、**HostnameMap** を使用してマッピングします。

手順

1. 環境ファイル (たとえば **hostname-map.yaml**) を作成して、**HostnameMap** パラメーターとホスト名のマッピングを指定します。以下の構文を使用してください。

```
parameter_defaults:
  HostnameMap:
    [HEAT HOSTNAME]: [ACTUAL HOSTNAME]
    [HEAT HOSTNAME]: [ACTUAL HOSTNAME]
```

[HEAT HOSTNAME] は通常 **[STACK NAME]-[ROLE]-[INDEX]** の表記法に従います。

```
parameter_defaults:
  HostnameMap:
    overcloud-controller-0: controller-00-rack01
    overcloud-controller-1: controller-01-rack02
    overcloud-controller-2: controller-02-rack03
```

```
overcloud-novacompute-0: compute-00-rack01
overcloud-novacompute-1: compute-01-rack01
overcloud-novacompute-2: compute-02-rack01
```

2. **hostname-map.yaml** ファイルを保存します。

9.8. ネットワークインターフェイスのエイリアスへのマッピング

Red Hat OpenStack Platform 16.1 の場合、事前にプロビジョニングされたノードでは、オーバークラウドネットワークインターフェイスのマッピングが自動的に行われません。代わりに、事前にプロビジョニングされた各ノードの **/etc/os-net-config/mapping.yaml** ファイルでマッピングを手動で定義する必要があります。

手順

1. 事前にプロビジョニングされた各ノードにログインします。
2. **/etc/os-net-config/mapping.yaml** ファイルを作成し、インターフェイスマッピングの詳細を含めます。

```
interface_mapping:
  nic1: em1
  nic2: em2
```

9.9. 事前にプロビジョニングされたノード向けの CEPH STORAGE の設定

すでにデプロイされているノードに **ceph-ansible** を設定するには、アンダークラウドホストで以下の手順を実施します。

手順

1. アンダークラウドホストで環境変数 **OVERCLOUD_HOSTS** を作成し、変数に Ceph クライアントとして使用するオーバークラウドホストの IP アドレスのスペース区切りリストを設定します。

```
export OVERCLOUD_HOSTS="192.168.1.8 192.168.1.42"
```

2. デフォルトのオーバークラウドプランの名前は **overcloud** です。別の名前を使用する場合は、環境変数 **OVERCLOUD_PLAN** を作成してカスタムの名前を保存します。

```
export OVERCLOUD_PLAN="<custom-stack-name>"
```

   - **<custom-stack-name>** を実際のスタックの名前に置き換えます。

3. **enable-ssh-admin.sh** スクリプトを実行して、Ansible が Ceph クライアントの設定に使用することのできるオーバークラウドノードのユーザーを設定します。

```
bash /usr/share/openstack-tripleo-heat-templates/deployed-server/scripts/enable-ssh-admin.sh
```

openstack overcloud deploy コマンドを実行すると、Ansible は **OVERCLOUD_HOSTS** 変数で Ceph クライアントとして定義したホストを設定します。

9.10. 事前にプロビジョニングされたノードを使用したオーバークラウドの作成

オーバークラウドのデプロイメントには、「デプロイメントコマンド」に記載された標準の CLI の方法を使用します。事前にプロビジョニングされたノードの場合は、デプロイメントコマンドに追加のオプションと、コア heat テンプレートコレクションからの環境ファイルが必要です。

- **--disable-validations**: このオプションを使用して、事前にプロビジョニングされたインフラストラクチャーで使用しないサービスに対する基本的な CLI 検証を無効にします。これらの検証を無効にしないと、デプロイメントに失敗します。

- **environments/deployed-server-environment.yaml**: 事前にプロビジョニングされたインフラストラクチャーを作成、設定するには、この環境ファイルを追加します。この環境ファイルは、**OS::Nova::Server** リソースを **OS::Heat::DeployedServer** リソースに置き換えます。

以下のコマンドは、事前にプロビジョニングされたアーキテクチャー固有の環境ファイルを使用したオーバークラウドデプロイメントコマンドの例です。

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy \
  --disable-validations \
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-environment.yaml \
  -e /home/stack/templates/hostname-map.yaml \
  --overcloud-ssh-user stack \
  --overcloud-ssh-key ~/.ssh/id_rsa \
  <OTHER OPTIONS>
```

--overcloud-ssh-user および **--overcloud-ssh-key** オプションは、設定ステージ中に各オーバークラウドノードに SSH 接続して、初期 **tripleo-admin** ユーザーを作成し、SSH キーを **/home/tripleo-admin/.ssh/authorized_keys** に挿入するのに使用します。SSH キーを挿入するには、**--overcloud-ssh-user** および **--overcloud-ssh-key** (**~/.ssh/id_rsa** がデフォルト) を使用して、初回 SSH 接続用の認証情報を指定します。**--overcloud-ssh-key** オプションで指定する秘密鍵の公開を制限するために、director は heat や Workflow サービス (mistral) などのどの API サービスにもこの鍵を渡さず、director の **openstack overcloud deploy** コマンドだけがこの鍵を使用して **tripleo-admin** ユーザーのアクセスを有効化します。

9.11. オーバークラウドデプロイメントの出力

オーバークラウドの作成が完了すると、オーバークラウドを設定するために実施された Ansible のプレイの概要が director により提示されます。

```
PLAY RECAP *********************************************************************
overcloud-compute-0     : ok=160  changed=67   unreachable=0    failed=0
overcloud-controller-0  : ok=210  changed=93   unreachable=0    failed=0
undercloud              : ok=10   changed=7    unreachable=0    failed=0

Tuesday 15 October 2018  18:30:57 +1000 (0:00:00.107) 1:06:37.514 ******
===============================================================================
```

director により、オーバークラウドへのアクセス情報も提供されます。

```
Ansible passed.
Overcloud configuration completed.
Overcloud Endpoint: http://192.168.24.113:5000
```

```
Overcloud Horizon Dashboard URL: http://192.168.24.113:80/dashboard
Overcloud rc file: /home/stack/overcloudrc
Overcloud Deployed
```

9.12. 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と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ます。director は、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。

```
(undercloud) $ source ~/overcloudrc
```

このコマンドにより、アンダークラウド CLI からオーバークラウドと対話するのに必要な環境変数が読み込まれます。コマンドプロンプトが変わり、オーバークラウドと対話していることが示されます。

```
(overcloud) $
```

アンダークラウドとの対話に戻るには、以下のコマンドを実行します。

```
(overcloud) $ source ~/stackrc
(undercloud) $
```

9.13. 事前にプロビジョニングされたノードのスケーリング

事前にプロビジョニングされたノードをスケーリングするプロセスは、16章*オーバークラウドノードのスケーリング*に記載する標準のスケーリング手順と類似しています。ただし、事前にプロビジョニングされたノードを新たに追加するプロセスは異なります。これは、事前にプロビジョニングされたノードが OpenStack Bare Metal (ironic) および OpenStack Compute (nova) からの標準の登録および管理プロセスを使用しないためです。

事前にプロビジョニングされたノードのスケールアップ

事前にプロビジョニングされたノードでオーバークラウドをスケールアップする際には、各ノードで director のノード数に対応するようにオーケストレーションエージェントを設定する必要があります。

オーバークラウドノードをスケールアップするには、以下の操作を実施します。

1. 「事前にプロビジョニングされたノードの要件」の説明に従って、事前にプロビジョニングされたノードを新たに準備します。
2. ノードをスケールアップします。詳細は、16章*オーバークラウドノードのスケーリング*を参照してください。
3. デプロイメントコマンドを実行した後に、director が新しいノードリソースを作成して設定を開始するまで待ちます。

事前にプロビジョニングされたノードのスケールダウン

事前にプロビジョニングされたノードを持つオーバークラウドをスケールダウンするには、16章*オーバークラウドノードのスケーリング*に記載するスケールダウン手順に従います。

スケールダウン操作では、OSP でプロビジョニングされたノードと事前にプロビジョニングされたノードの両方にホスト名を使用できます。OSP プロビジョニングされたノードに UUID を使用することもできます。ただし、事前にプロビジョニングされたノードには UUID がないため、常にホスト名を使

用します。ホスト名または UUID 値を **openstack overcloud node delete** コマンドに渡します。

手順

1. 削除するノードの名前を特定します。

```
$ openstack stack resource list overcloud -n5 --filter
type=OS::TripleO::ComputeDeployedServerServer
```

2. 対応するノード名を **stack_name** 列から **openstack overcloud node delete** コマンドに渡します。

```
$ openstack overcloud node delete --stack <overcloud> <stack>
```

   - **<overcloud>** は、オーバークラウドスタックの名前または UUID に置き換えてください。
   - **<stack_name>** を削除するノードの名前に置き換えます。**openstack overcloud node delete** コマンドに複数のノード名を含めることができます。

3. **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
```

   削除の操作が完了すると、**オーバークラウド** スタックのステータスは **UPDATE_COMPLETE** と表示されます。

スタックからオーバークラウドノードを削除したら、それらのノードの電源をオフにします。標準のデプロイメントでは、director のベアメタルサービスがこの機能を制御します。ただし、事前にプロビジョニングされたノードでは、これらのノードを手動でシャットダウンするか、物理システムごとに電源管理制御を使用する必要があります。スタックからノードを削除した後にノードの電源をオフにしないと、稼動状態が続き、オーバークラウド環境の一部として再接続されてしまう可能性があります。

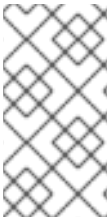
削除したノードの電源をオフにした後に、それらのノードをベースオペレーティングシステムの設定に再プロビジョニングし、意図せずにオーバークラウドに加わらないようにします。

注記

オーバークラウドから以前に削除したノードは、再プロビジョニングしてベースオペレーティングシステムを新規インストールするまでは、再利用しないようにしてください。スケールダウンのプロセスでは、オーバークラウドスタックからノードを削除するだけで、パッケージはアンインストールされません。

事前にプロビジョニングされたオーバークラウドの削除

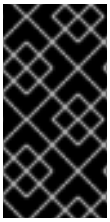
事前にプロビジョニングされたノードを使用するオーバークラウド全体を削除するには、「オーバークラウドの削除」 で標準のオーバークラウド削除手順を参照してください。オーバークラウドを削除した後に、全ノードの電源をオフにしてからベースオペレーティングシステムの設定に再プロビジョニングします。



注記

オーバークラウドから以前に削除したノードは、再プロビジョニングしてベースオペレーティングシステムを新規インストールするまでは、再利用しないようにしてください。削除のプロセスでは、オーバークラウドスタックを削除するだけで、パッケージはアンインストールされません。

第10章 複数のオーバークラウドのデプロイ



重要

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

1 つのアンダークラウドノードを使用して、複数のオーバークラウドをデプロイおよび管理することができます。それぞれのオーバークラウドは、スタックリソースを共有しない個別の heat スタックです。この設定は、アンダークラウドとオーバークラウドの比率が 1:1 の環境で、無視できない程度のオーバーヘッドが発生する場合に有用です。たとえば、エッジサイト、複数サイト、および複数製品にまたがる環境などです。

複数のオーバークラウドデプロイメントのオーバークラウド環境は完全に分離されており、**source** コマンドを使用して環境を切り替えることができます。各オーバークラウドには、デプロイプロセスによって作成される一意の認証情報ファイルがあります。オーバークラウドに関する操作を行うには、source コマンドで適切な認証情報ファイルを読み込む必要があります。

Bare Metal Provisioning サービス (ironic) をベアメタルのプロビジョニングに使用する場合は、すべてのオーバークラウドが同じプロビジョニングネットワーク上にある必要があります。同じプロビジョニングネットワークを使用することができない場合には、デプロイされたサーバー法を使用して、ルーティングされたネットワークで複数のオーバークラウドをデプロイすることができます。このシナリオでは、**HostnameMap** パラメーターの値が各オーバークラウドのスタック名と一致している必要があります。

1 つのアンダークラウドに複数のオーバークラウドをデプロイするには、以下のタスクを実行する必要があります。

1. アンダークラウドをデプロイします。詳細は、パート I. Director のインストールおよび設定 を参照してください。
2. 最初のオーバークラウドをデプロイします。詳細については、パート II基本的なオーバークラウドのデプロイメント を参照してください。
3. 新しいオーバークラウド用の環境ファイルの新しいセットを作成し、デプロイコマンドで新しい設定ファイルと新しい **stack** 名とともにコア Heat テンプレートを指定することにより、追加のオーバークラウドをデプロイします。

10.1. 追加のオーバークラウドのデプロイ

1 つのアンダークラウドに複数のオーバークラウドをデプロイできます。以下の手順は、既存のオーバークラウド **overcloud-one** を持つ既存の Red Hat OpenStack Platform (RHOSP) デプロイメントに新しいオーバークラウド **overcloud-two** を作成してデプロイする方法を示しています。

前提条件

- アンダークラウド。
- 1 つ以上のオーバークラウド。
- 追加のオーバークラウドで使用できるノード。

- それぞれのオーバークラウドが作成されるスタックで固有のネットワークを持つように、追加のオーバークラウド用のカスタムネットワーク

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

3. デプロイする追加のオーバークラウド用に、新たなディレクトリーを作成します。

```
(undercloud)$ mkdir ~/overcloud-two
```

4. **network_data.yaml** ファイルを既存のオーバークラウドから追加のオーバークラウド用の新しいディレクトリーにコピーします。

```
(undercloud)$ cp network_data.yaml ~/overcloud-two/network_data.yaml
```

5. **~/overcloud-two/network_data.yaml** ファイルを開き、**name_lower** を追加のオーバークラウドネットワークの一意の名前に更新します。

```
- name: InternalApi
  name_lower: internal_api_cloud_2
  ...
```

6. まだ存在しない場合は **service_net_map_replace** を追加し、値を追加のオーバークラウドネットワークのデフォルト値に設定します。

```
- name: InternalApi
  name_lower: internal_api_cloud_2
  service_net_map_replace: internal_api
```

7. 追加のオーバークラウドの各サブネットに VLAN ID を指定します。

```
- name: InternalApi
  ...
  vip: true
  vlan: 21
  ip_subnet: '172.21.0.0/24'
  allocation_pools: [{'start': '172.21.0.4', 'end': '172.21.0.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:2001::/64'
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:2001::10', 'end':
'fd00:fd00:fd00:2001:ffff:ffff:ffff:fffe'}]
  mtu: 1500
- name: Storage
  ...
```

8. **overcloud-two** 外部ネットワークのゲートウェイの IP アドレスを指定します。

```
- name: External
  ...
```

```
  gateway_ip: <ip_address>
  ...
```

- **<ip_address>** を **overcloud-two** 外部ネットワークのゲートウェイの IP アドレス (例: **10.0.10.1**) に置き換えます。

9. **/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml** ファイル (例: **network_overrides.yaml**) で提供されるデフォルトの隔離されたネットワーク設定をオーバーライドする、追加のオーバークラウド用のネットワーク設定ファイルを作成します。

10. **~/overcloud-two/network_overrides.yaml** ファイルを開き、**overcloud-two** DNS サーバーの IP アドレスを追加します。

```
parameter_defaults:
  ...
  DnsServers:
    - <ip_address>
  ...
```

- **<ip_address>** を **overcloud-two** DNS サーバーの IP アドレス (例: **10.0.10.2**) に置き換えます。

11. デプロイメントで予測可能な IP アドレスを使用する場合は、新しいネットワーク IP アドレスマッピングファイル **ips-from-pool-overcloud-two.yaml** で **overcloud-two** ノードの IP アドレスを設定します。

```
parameter_defaults:
  ControllerIPs:
  ...
    internal_api_cloud_2:
    - 192.168.1.10
    - 192.168.1.11
    - 192.168.1.12
    ...
    external_cloud_2:
    - 10.0.1.41
    ...
```

12. **overcloud-two** 環境ファイルを他の環境ファイルと一緒にスタックに追加し、追加のオーバークラウドをデプロイします。

```
(undercloud)$ openstack overcloud deploy --templates \
  --stack overcloud-two \
  -n ~/overcloud-two/network_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml
\
  -e ~/overcloud-two/network_overrides.yaml \
  -e [your environment files] \
  ...
```

デプロイメントプロセスにより、**overcloud-two** とやり取りして管理するための **overcloud-tworc** が作成されます。

13. 追加のオーバークラウドと対話するには、オーバークラウド認証情報ファイルを入手します。

```
$ source overcloud-tworc
```

10.2. 複数のオーバークラウドの管理

デプロイするそれぞれのオーバークラウドでは、同じコア heat テンプレートセット **/usr/share/openstack-tripleo-heat-templates** が使用されます。非標準のコアテンプレートセットを使用すると、更新およびアップグレード時に問題が発生する可能性があるので、Red Hat では、これらのテンプレートを変更したり複製したりしないことを推奨します。

その代わりに、複数のオーバークラウドをデプロイまたは維持する際の管理を容易にするため、各クラウドに固有の環境ファイル用ディレクトリーを個別に作成します。各クラウドのデプロイコマンドを実行する際に、コア heat テンプレートと共に個別に作成したクラウド固有の環境ファイルを含めます。たとえば、アンダークラウドおよび 2 つのオーバークラウド用に以下のディレクトリーを作成します。

~stack/undercloud

アンダークラウドに固有の環境ファイルを保管します。

~stack/overcloud-one

最初のオーバークラウドに固有の環境ファイルを保管します。

~stack/overcloud-two

2 番目のオーバークラウドに固有の環境ファイルを保管します。

overcloud-one または **overcloud-two** をデプロイまたは再デプロイする場合には、**--templates** オプションでデプロイコマンドにコア heat テンプレートを追加し、続いてクラウド固有の環境ファイルディレクトリーからの追加環境ファイルをすべて指定します。

あるいは、バージョン管理システムにリポジトリーを作成し、デプロイメントごとにブランチを使用します。詳しくは、**オーバークラウドの高度なカスタマイズ**の カスタムのコア Heat テンプレートの使用 セクションを参照してください。

利用可能なオーバークラウドプランのリストを表示するには、以下のコマンドを使用します。

```
$ openstack overcloud plan list
```

現在デプロイされているオーバークラウドのリストを表示するには、以下のコマンドを使用します。

```
$ openstack stack list
```

第11章 オーバークラウドのインストール後タスクの実施

本章では、オーバークラウドを作成したすぐ後に実施するタスクについて説明します。これらのタスクにより、オーバークラウドを使用可能な状態にすることができます。

11.1. オーバークラウドデプロイメントステータスの確認

オーバークラウドのデプロイメントステータスを確認するには、**openstack overcloud status** コマンドを使用します。このコマンドにより、すべてのデプロイメントステップの結果が返されます。

手順

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. デプロイメントステータスの確認コマンドを実行します。

```
$ openstack overcloud status
```

このコマンドの出力に、オーバークラウドのステータスが表示されます。

```
+-----------+---------------------+---------------------+-------------------+
| Plan Name |       Created       |       Updated       | Deployment Status |
+-----------+---------------------+---------------------+-------------------+
| overcloud | 2018-05-03 21:24:50 | 2018-05-03 21:27:59 |   DEPLOY_SUCCESS  |
+-----------+---------------------+---------------------+-------------------+
```

実際のオーバークラウドに別の名前が使用されている場合には、**--plan** 引数を使用してその名前のオーバークラウドを選択します。

```
$ openstack overcloud status --plan my-deployment
```

11.2. 基本的なオーバークラウドフレーバーの作成

本ガイドの検証ステップは、インストール環境にフレーバーが含まれていることを前提としてます。まだ1つのフレーバーも作成していない場合には、以下の手順を実施して、さまざまなストレージおよび処理能力に対応する基本的なデフォルトフレーバーセットを作成してください。

手順

1. source コマンドで **overcloudrc** ファイルを読み込みます。

```
$ source ~/overcloudrc
```

2. **openstack flavor create** コマンドを実行してフレーバーを作成します。以下のオプションを使用して、各フレーバーのハードウェア要件を指定します。

--disk

仮想マシンのボリュームのハードディスク容量を定義します。

--ram

仮想マシンに必要な RAM を定義します。

--vcpus

仮想マシンの仮想 CPU 数を定義します。

3. デフォルトのオーバークラウドフレーバー作成の例を以下に示します。

```
$ openstack flavor create m1.tiny --ram 512 --disk 0 --vcpus 1
$ openstack flavor create m1.smaller --ram 1024 --disk 0 --vcpus 1
$ openstack flavor create m1.small --ram 2048 --disk 10 --vcpus 1
$ openstack flavor create m1.medium --ram 3072 --disk 10 --vcpus 2
$ openstack flavor create m1.large --ram 8192 --disk 10 --vcpus 4
$ openstack flavor create m1.xlarge --ram 8192 --disk 10 --vcpus 8
```

注記

openstack flavor create コマンドについての詳しい情報は、**$ openstack flavor create --help** で確認してください。

11.3. デフォルトの TENANT ネットワークの作成

仮想マシンが内部で通信できるように、オーバークラウドにはデフォルトの Tenant ネットワークが必要です。

手順

1. source コマンドで **overcloudrc** ファイルを読み込みます。

```
$ source ~/overcloudrc
```

2. デフォルトの Tenant ネットワークを作成します。

```
(overcloud) $ openstack network create default
```

3. ネットワーク上にサブネットを作成します。

```
(overcloud) $ openstack subnet create default --network default --gateway 172.20.1.1 --
subnet-range 172.20.0.0/16
```

4. 作成したネットワークを確認します。

```
(overcloud) $ openstack network list
+----------------------+-------------+--------------------------------------+
| id                   | name        | subnets                              |
+----------------------+-------------+--------------------------------------+
| 95fadaa1-5dda-4777... | default     | 7e060813-35c5-462c-a56a-1c6f8f4f332f |
+----------------------+-------------+--------------------------------------+
```

これらのコマンドにより、**default** という名前の基本的な Networking サービス (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て、このネットワークから仮想マシンに IP アドレスを自動的に割り当てます。

11.4. デフォルトの FLOATING IP ネットワークの作成

オーバークラウドの外部から仮想マシンにアクセスするためには、仮想マシンに Floating IP アドレスを提供するExternal ネットワークを設定する必要があります。

ここでは、2 つの手順例を示します。実際の環境に最も適した例を使用してください。

- ネイティブ VLAN (フラットネットワーク)
- 非ネイティブ VLAN (VLAN ネットワーク)

これらの例の両方で、**public** という名前のネットワークを作成します。オーバークラウドでは、デフォルトの Floating IP プールにこの特定の名前が必要です。この名前は、「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。

デフォルトでは、OpenStack Networking (neutron) は、**datacentre** という物理ネットワーク名をホストノード上の **br-ex** ブリッジにマッピングします。**public** オーバークラウドネットワークを物理ネットワーク **datacentre** に接続し、これにより **br-ex** ブリッジを通じてゲートウェイが提供されます。

前提条件

- Floating IP ネットワーク向けの専用インターフェイスまたはネイティブ VLAN

手順

1. source コマンドで **overcloudrc** ファイルを読み込みます。

```
$ source ~/overcloudrc
```

2. **public** ネットワークを作成します。
   - ネイティブ VLAN 接続用に **flat** ネットワークを作成します。

```
(overcloud) $ openstack network create public --external --provider-network-type flat --provider-physical-network datacentre
```

   - 非ネイティブ VLAN 接続用に **vlan** ネットワークを作成します。

```
(overcloud) $ openstack network create public --external --provider-network-type vlan --provider-physical-network datacentre --provider-segment 201
```

     --provider-segment オプションを使用して、使用する VLAN を定義します。この例では、VLAN は **201** です。

3. Floating IP アドレスの割り当てプールを使用してサブネットを作成します。以下の例では、IP 範囲は **10.1.1.51** から **10.1.1.250** までです。

```
(overcloud) $ openstack subnet create public --network public --dhcp --allocation-pool start=10.1.1.51,end=10.1.1.250 --gateway 10.1.1.1 --subnet-range 10.1.1.0/24
```

   この範囲が、External ネットワークの他の IP アドレスと競合しないようにしてください。

11.5. デフォルト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別の種別の External ネットワーク接続で、トラフィックをプライベート Tenant ネットワークから External インフラストラクチャーネットワークにルーティングします。プロバイダーネットワークは Floating IP ネットワークと類似していますが、プライベートネットワークをプロバイダーネットワークに接続するのに、論理ルーターが使用されます。

ここでは、2 つの手順例を示します。実際の環境に最も適した例を使用してください。

- ネイティブ VLAN (フラットネットワーク)
- 非ネイティブ VLAN (VLAN ネットワーク)

デフォルトでは、OpenStack Networking (neutron) は、**datacentre** という物理ネットワーク名をホストノード上の **br-ex** ブリッジにマッピングします。**public** オーバークラウドネットワークを物理ネットワーク **datacentre** に接続し、これにより **br-ex** ブリッジを通じてゲートウェイが提供されます。

手順

1. source コマンドで **overcloudrc** ファイルを読み込みます。

```
$ source ~/overcloudrc
```

2. **provider** ネットワークを作成します。
   - ネイティブ VLAN 接続用に **flat** ネットワークを作成します。

```
(overcloud) $ openstack network create provider --external --provider-network-type flat --provider-physical-network datacentre --share
```

   - 非ネイティブ VLAN 接続用に **vlan** ネットワークを作成します。

```
(overcloud) $ openstack network create provider --external --provider-network-type vlan --provider-physical-network datacentre --provider-segment 201 --share
```

     --provider-segment オプションを使用して、使用する VLAN を定義します。この例では、VLAN は **201** です。

   例に示すこれらのコマンドにより、共有ネットワークが作成されます。テナントだけが新しいネットワークにアクセスするように、**--share** を指定する代わりにテナントを指定することも可能です。

   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。

3. **provider** ネットワークにサブネットを追加して、DHCP サービスを提供します。

```
(overcloud) $ openstack subnet create provider-subnet --network  provider --dhcp --allocation-pool start=10.9.101.50,end=10.9.101.100 --gateway 10.9.101.254 --subnet-range 10.9.101.0/24
```

4. 他のネットワークがプロバイダーネットワークを通じてトラフィックをルーティングできるように、ルーターを作成します。

```
(overcloud) $ openstack router create external
```

5. ルーターの外部ゲートウェイを **provider** ネットワークに設定します。

```
(overcloud) $ openstack router set --external-gateway provider external
```

6. このルーターに他のネットワークを接続します。たとえば、以下のコマンドを実行してサブネット **subnet1** をルーターに割り当てます。

```
(overcloud) $ openstack router add subnet external subnet1
```

このコマンドにより、**subnet1** がルーティングテーブルに追加され、**subnet1** を使用する仮想マシンからのトラフィックをプロバイダーネットワークにルーティングできるようになります。

11.6. 新たなブリッジマッピングの作成

デプロイメント時に追加のブリッジをマッピングすれば、Floating IP ネットワークは **br-ex** だけでなく任意のブリッジを使用することができます。

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、環境ファイルに **NeutronBridgeMappings** パラメーターを追加します。

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,floating:br-floating"
```

この手法により、オーバークラウドの作成後に独立した External ネットワークを作成することができます。たとえば、**floating** 物理ネットワークにマッピングする Floating IP ネットワークを作成するには、以下のコマンドを実行します。

```
$ source ~/overcloudrc
(overcloud) $ openstack network create public --external --provider-physical-network floating --
provider-network-type vlan --provider-segment 105
(overcloud) $ openstack subnet create public --network public --dhcp --allocation-pool
start=10.1.2.51,end=10.1.2.250 --gateway 10.1.2.1 --subnet-range 10.1.2.0/24
```

11.7. オーバークラウドの検証

オーバークラウドは、OpenStack Integration Test Suite (tempest) ツールセットを使用して、一連の統合テストを行います。本項では、統合テストを実施するための準備について説明します。OpenStack Integration Test Suite の使用方法についての詳しい説明は、OpenStack Integration Test Suite Guide を参照してください。

Integration Test Suite では、テストを成功させるために、いくつかのインストール後手順が必要になります。

手順

1. 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 Internal 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 Internal 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的な VLAN を追加します。

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

2. OpenStack Integration Test Suite Guide の説明に従って、統合テストを実施します。

3. 検証が完了したら、オーバークラウドの Internal 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
(undercloud) $ sudo ovs-vsctl del-port vlan201
```

11.8. オーバークラウドの削除防止

オーバークラウドが削除されないように、heat のカスタムポリシーを設定します。

手順

1. **prevent-stack-delete.yaml** という名前の環境ファイルを作成します。

2. **HeatApiPolicies** パラメーターを設定します。

```
parameter_defaults:
  HeatApiPolicies:
    heat-deny-action:
      key: 'actions:action'
      value: 'rule:deny_everybody'
    heat-protect-overcloud:
      key: 'stacks:delete'
      value: 'rule:deny_everybody'
```

重要

heat-deny-action は、アンダークラウドのインストールに追加する必要のあるデフォルトポリシーです。

3. **undercloud.conf** ファイルの **custom_env_files** パラメーターに、**prevent-stack-delete.yaml** 環境ファイルを追加します。

```
custom_env_files = prevent-stack-delete.yaml
```

4. アンダークラウドのインストールコマンドを実行して設定をリフレッシュします。

```
$ openstack undercloud install
```

この環境ファイルにより、オーバークラウド内のスタックを削除することができなくなります。したがって、以下の操作を実施することはできません。

- オーバークラウドの削除
- 個々のコンピュートノードまたは Ceph Storage ノードの削除
- コントローラーノードの置き換え

スタックの削除を有効にするには、**custom_env_files** パラメーターから **prevent-stack-delete.yaml** ファイルを削除し、**openstack undercloud install** コマンドを実行します。

第12章 基本的なオーバークラウド管理タスクの実施

本章では、オーバークラウドのライフサイクル期間中に実行しなければならない可能性がある、基本的なタスクについて説明します。

12.1. SSH を使用したオーバークラウドノードへのアクセス

SSH プロトコルを使用して、各オーバークラウドノードにアクセスすることができます。

- 各オーバークラウドノードには **heat-admin** ユーザーが含まれます。
- アンダークラウドの **stack** ユーザーは、各オーバークラウドノードの **heat-admin** ユーザーに鍵ベースの SSH アクセスを行うことができます。
- すべてのオーバークラウドノードは短縮ホスト名を持ち、アンダークラウドはこのホスト名をコントロールプレーンネットワーク上の IP アドレスに解決します。それぞれの短縮ホスト名には、**.ctlplane** 接尾辞が使用されます。たとえば、**overcloud-controller-0** の短縮名は **overcloud-controller-0.ctlplane** です。

前提条件

- 稼動状態にあるコントロールプレーンネットワークと共にデプロイされたオーバークラウド

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. source コマンドで **overcloudrc** ファイルを読み込みます。

```
$ source ~/stackrc
```

3. アクセスするノードの名前を確認します。

```
(undercloud) $ openstack server list
```

4. ノードに **heat-admin** ユーザーとして接続し、ノードの短縮ホスト名を使用します。

```
(undercloud) $ ssh heat-admin@overcloud-controller-0.ctlplane
```

12.2. コンテナー化されたサービスの管理

Red Hat OpenStack Platform (RHOSP) では、アンダークラウドおよびオーバークラウドノード上のコンテナー内でサービスが実行されます。特定の状況では、1 つのホスト上で個別のサービスを制御する必要がある場合があります。本項では、コンテナー化されたサービスを管理するためにノード上で実行することのできる、一般的なコマンドについて説明します。

コンテナーとイメージのリスト表示

実行中のコンテナーをリスト表示するには、以下のコマンドを実行します。

```
$ sudo podman ps
```

コマンド出力に停止中またはエラーの発生したコンテナーを含めるには、コマンドに **--all** オプションを追加します。

```
$ sudo podman ps --all
```

コンテナーイメージをリスト表示するには、以下のコマンドを実行します。

```
$ sudo podman images
```

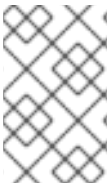
コンテナーの属性の確認

コンテナーまたはコンテナーイメージのプロパティーを表示するには、**podman inspect** コマンドを使用します。たとえば、**keystone** コンテナーを検査するには、以下のコマンドを実行します。

```
$ sudo podman inspect keystone
```

Systemd サービスを使用したコンテナーの管理

以前のバージョンの OpenStack Platform では、コンテナーは Docker およびそのデーモンで管理されていました。OpenStack Platform 16 では、Systemd サービスインターフェイスでコンテナーのライフサイクルが管理されます。それぞれのコンテナーはサービスであり、Systemd コマンドを実行して各コンテナーに関する特定の操作を実施します。



注記

Systemd は再起動ポリシーを適用するため、Podman CLI を使用してコンテナーを停止、起動、および再起動することは推奨されません。その代わりに、Systemd サービスコマンドを使用してください。

コンテナーのステータスを確認するには、**systemctl status** コマンドを実行します。

```
$ sudo systemctl status tripleo_keystone
● tripleo_keystone.service - keystone container
   Loaded: loaded (/etc/systemd/system/tripleo_keystone.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
 Main PID: 29012 (podman)
   CGroup: /system.slice/tripleo_keystone.service
           └─29012 /usr/bin/podman start -a keystone
```

コンテナーを停止するには、**systemctl stop** コマンドを実行します。

```
$ sudo systemctl stop tripleo_keystone
```

コンテナーを起動するには、**systemctl start** コマンドを実行します。

```
$ sudo systemctl start tripleo_keystone
```

コンテナーを再起動するには、**systemctl restart** コマンドを実行します。

```
$ sudo systemctl restart tripleo_keystone
```

コンテナーステータスを監視するデーモンはないので、以下の状況では Systemd はほとんどのコンテナーを自動的に再起動します。

- **podman stop** コマンドの実行など、明瞭な終了コードまたはシグナル
- 起動後に podman コンテナーがクラッシュするなど、不明瞭な終了コード
- 不明瞭なシグナル
- コンテナーの起動に 1 分 30 秒以上かかった場合のタイムアウト

Systemd サービスに関する詳しい情報は、**systemd.service** のドキュメント を参照してください。

注記

コンテナー内のサービス設定ファイルに加えた変更は、コンテナーの再起動後には元に戻ります。これは、コンテナーがノードのローカルファイルシステム上の **/var/lib/config-data/puppet-generated/** にあるファイルに基づいてサービス設定を再生成するためです。たとえば、**keystone** コンテナー内の **/etc/keystone/keystone.conf** を編集してコンテナーを再起動すると、そのコンテナーはノードのローカルシステム上にある **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf** を使用して設定を再生成します。再起動前にコンテナー内で加えられた変更は、この設定によって上書きされます。

Systemd タイマーを使用した podman コンテナーの監視

Systemd タイマーインターフェイスは、コンテナーのヘルスチェックを管理します。各コンテナーのタイマーがサービスユニットを実行し、そのユニットがヘルスチェックスクリプトを実行します。

すべての OpenStack Platform コンテナーのタイマーをリスト表示するには、**systemctl list-timers** コマンドを実行し、出力を **tripleo** が含まれる行に限定します。

```
$ sudo systemctl list-timers | grep tripleo
Mon 2019-02-18 20:18:30 UTC  1s left       Mon 2019-02-18 20:17:26 UTC  1min 2s ago
tripleo_nova_metadata_healthcheck.timer           tripleo_nova_metadata_healthcheck.service
Mon 2019-02-18 20:18:33 UTC  4s left       Mon 2019-02-18 20:17:03 UTC  1min 25s ago
tripleo_mistral_engine_healthcheck.timer          tripleo_mistral_engine_healthcheck.service
Mon 2019-02-18 20:18:34 UTC  5s left       Mon 2019-02-18 20:17:23 UTC  1min 5s ago
tripleo_keystone_healthcheck.timer                tripleo_keystone_healthcheck.service
Mon 2019-02-18 20:18:35 UTC  6s left       Mon 2019-02-18 20:17:13 UTC  1min 15s ago
tripleo_memcached_healthcheck.timer               tripleo_memcached_healthcheck.service
(...)
```

特定のコンテナータイマーのステータスを確認するには、healthcheck サービスに対して **systemctl status** コマンドを実行します。

```
$ sudo systemctl status tripleo_keystone_healthcheck.service
● tripleo_keystone_healthcheck.service - keystone healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.service; disabled; vendor
preset: disabled)
   Active: inactive (dead) since Mon 2019-02-18 20:22:46 UTC; 22s ago
  Process: 115581 ExecStart=/usr/bin/podman exec keystone /openstack/healthcheck (code=exited,
status=0/SUCCESS)
 Main PID: 115581 (code=exited, status=0/SUCCESS)

Feb 18 20:22:46 undercloud.localdomain systemd[1]: Starting keystone healthcheck...
Feb 18 20:22:46 undercloud.localdomain podman[115581]: {"versions": {"values": [{"status": "stable",
```

```
"updated": "2019-01-22T00:00:00Z", "..."}]}]}}
Feb 18 20:22:46 undercloud.localdomain podman[115581]: 300 192.168.24.1:35357 0.012 seconds
Feb 18 20:22:46 undercloud.localdomain systemd[1]: Started keystone healthcheck.
```

コンテナータイマーを停止、起動、再起動、およびコンテナータイマーのステータスを表示するには、**.timer** Systemd リソースに対して該当する **systemctl** コマンドを実行します。たとえば、**tripleo_keystone_healthcheck.timer** リソースのステータスを確認するには、以下のコマンドを実行します。

```
$ sudo systemctl status tripleo_keystone_healthcheck.timer
● tripleo_keystone_healthcheck.timer - keystone container healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.timer; enabled; vendor preset:
disabled)
   Active: active (waiting) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
```

healthcheck サービスは無効だが、そのサービスのタイマーが存在し有効になっている場合には、チェックは現在タイムアウトしているが、タイマーに従って実行されることを意味します。チェックを手動で開始することもできます。

注記

podman ps コマンドは、コンテナーのヘルスステータスを表示しません。

コンテナーログの確認

OpenStack Platform 16 では、新たなロギングディレクトリー **/var/log/containers/stdout** が導入されています。ここには、すべてのコンテナーの標準出力 (stdout) と標準エラー (stderr) が、コンテナーごとに 1 つのファイルに統合されて保存されます。

paunch および **container-puppet.py** スクリプトは、出力を **/var/log/containers/stdout** ディレクトリーにプッシュするように podman コンテナーを設定します。これにより、**container-puppet-*** コンテナー等の削除されたコンテナーを含め、すべてのログのコレクションが作成されます。

また、ホストはこのディレクトリーにログローテーションを適用し、大きな容量のファイルがディスク容量を消費する問題を防ぎます。

コンテナーが置き換えられた場合には、新しいコンテナーは同じログファイルにログを出力します。**podman** はコンテナー ID ではなくコンテナー名を使用するためです。

podman logs コマンドを使用して、コンテナー化されたサービスのログを確認することもできます。たとえば、**keystone** コンテナーのログを確認するには、以下のコマンドを実行します。

```
$ sudo podman logs keystone
```

コンテナーへのアクセス

コンテナー化されたサービスのシェルに入るには、**podman exec** コマンドを使用して **/bin/bash** を起動します。たとえば、**keystone** コンテナーのシェルに入るには、以下のコマンドを実行します。

```
$ sudo podman exec -it keystone /bin/bash
```

root ユーザーとして **keystone** コンテナーのシェルに入るには、以下のコマンドを実行します。

```
$ sudo podman exec --user 0 -it <NAME OR ID> /bin/bash
```

コンテナーから出るには、以下のコマンドを実行します。

```
# exit
```

12.3. 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、新たな機能を追加したり、既存の操作を変更したりすることができます。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、「デプロイメントコマンド」に記載の手順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。

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e ~/templates/node-info.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  --ntp-server pool.ntp.org
```

director は heat 内の **overcloud** スタックを確認してから、環境ファイルと heat テンプレートのあるスタックで各アイテムを更新します。director はオーバークラウドを再度作成せずに、既存のオーバークラウドに変更を加えます。

重要

カスタム環境ファイルからパラメーターを削除しても、パラメーター値はデフォルト設定に戻りません。**/usr/share/openstack-tripleo-heat-templates** のコア heat テンプレートコレクションからデフォルト値を特定し、カスタム環境ファイルでその値を手動で設定する必要があります。

新規環境ファイルを追加する場合には、`-e` オプションを使用して **openstack overcloud deploy** コマンドにそのファイルを追加します。以下に例を示します。

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e ~/templates/new-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  -e ~/templates/node-info.yaml \
  --ntp-server pool.ntp.org
```

このコマンドにより、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

オーバークラウドの設定に手動で変更を加えることは推奨されません。director によりこれらの変更が後で上書きされてしまう可能性があるためです。

12.4. 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 環境からご自分の Red Hat OpenStack Platform (RHOSP) 環境に仮想マシンを移行することができます。

手順

1. 既存の OpenStack 環境において、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。

```
$ openstack server image create instance_name --name image_name
$ openstack image save image_name --file exported_vm.qcow2
```

2. エクスポートしたイメージをアンダークラウドノードにコピーします。

```
$ scp exported_vm.qcow2 stack@192.168.0.2:~/.
```

3. アンダークラウドに **stack** ユーザーとしてログインします。
4. source コマンドで **overcloudrc** ファイルを読み込みます。

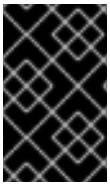
```
$ source ~/overcloudrc
```

5. 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ます。

```
(overcloud) $ openstack image create imported_image --file exported_vm.qcow2 --disk-format qcow2 --container-format bare
```

6. 新規インスタンスを起動します。

```
(overcloud) $ openstack server create  imported_instance --key-name default --flavor m1.demo --image imported_image --nic net-id=net_id
```



重要

これらのコマンドにより、各仮想マシンのディスクが既存の OpenStack 環境から新たな Red Hat OpenStack Platform にコピーされます。QCOW 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。

このプロファイルにより、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ます。インスタンスのダウンタイムなしにノードでメンテナンスを実行できるようになります。コンピュートノードを有効な状態に戻すには、以下のコマンドを実行します。

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

12.5. 動的インベントリースクリプトの実行

director を使用すると、Ansible ベースの自動化をご自分の Red Hat OpenStack Platform (RHOSP) 環境で実行することができます。director は、**tripleo-ansible-inventory** コマンドを使用して、環境内にノードの動的インベントリーを生成します。

手順

1. ノードの動的インベントリーを表示するには、**stackrc** を読み込んだ後に **tripleo-ansible-inventory** コマンドを実行します。

```
$ source ~/stackrc
(undercloud) $ tripleo-ansible-inventory --list
```

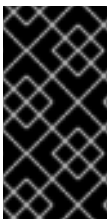
--list オプションを使用すると、全ホストの詳細が返されます。このコマンドにより、動的インベントリーが JSON 形式で出力されます。

```
{"overcloud": {"children": ["controller", "compute"], "vars": {"ansible_ssh_user": "heat-admin"}},
"controller": ["192.168.24.2"], "undercloud": {"hosts": ["localhost"], "vars":
{"overcloud_horizon_url": "http://192.168.24.4:80/dashboard", "overcloud_admin_password":
"abcdefghijklm12345678", "ansible_connection": "local"}}, "compute": ["192.168.24.3"]}
```

2. お使いの環境で Ansible のPlaybookを実行するには、**ansible** コマンドを実行し、**-i** オプションを使用して動的インベントリーツールの完全パスを追加します。以下に例を示します。

```
(undercloud) $ ansible [HOSTS] -i /bin/tripleo-ansible-inventory [OTHER OPTIONS]
```

- **[HOSTS]** を使用するホストの種別に置き換えてください。
  - 全コントローラーノードの場合には **controller**
  - 全コンピュートノードの場合には **compute**
  - オーバークラウドの全子ノードの場合には **overcloud**(たとえば、**コントローラー** ノードおよび **コンピュート** ノードの場合)
  - アンダークラウドの場合には **undercloud**
  - 全ノードの場合には **"*"**
- **[OTHER OPTIONS]** を追加の Ansible オプションに置き換えてください。
  - ホストキーの確認を省略するには、**--ssh-extra-args='-o StrictHostKeyChecking=no'** オプションを使用します。
  - Ansible の自動化を実行する SSH ユーザーを変更するには、**-u [USER]** オプションを使用します。オーバークラウドのデフォルトの SSH ユーザーは、動的インベントリーの **ansible_ssh_user** パラメーターで自動的に定義されます。**-u** オプションは、このパラメーターより優先されます。
  - 特定の Ansible モジュールを使用するには、**-m [MODULE]** オプションを使用します。デフォルトは **command** で Linux コマンドを実行します。
  - 選択したモジュールの引数を定義するには、**-a [MODULE_ARGS]** オプションを使用します。



重要

オーバークラウドのカスタム Ansible 自動化は、標準のオーバークラウドスタックの一部ではありません。この後に **openstack overcloud deploy** コマンドを実行すると、オーバークラウドノード上の OpenStack Platform サービスに対する Ansible ベースの設定を上書きする可能性があります。

12.6. オーバークラウドの削除

オーバークラウドを削除するには、**openstack overcloud delete** コマンドを実行します。

手順

1. 既存のオーバークラウドを削除します。

```
$ source ~/stackrc
(undercloud) $ openstack overcloud delete overcloud
```

2. **openstack stack list** コマンドの出力にオーバークラウドが表示されなくなったことを確認します。

```
(undercloud) $ openstack stack list
```

削除には、数分かかります。

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

第13章 ANSIBLE を使用したオーバークラウドの設定

Ansible は、オーバークラウドの設定を適用する主要な方法です。本章では、オーバークラウドの Ansible 設定を操作する方法について説明します。

director は Ansible Playbook を自動生成しますが、Ansible の構文を十分に理解しておくと役立ちます。Ansible の使用についての詳細は、Ansible のドキュメント を参照してください。

注記

Ansible でもロールの概念を使用します。これは、OpenStack Platform director のロールとは異なります。**Ansible のロール** は再利用可能な Playbook のコンポーネントを形成しますが、director のロールには OpenStack サービスのノード種別へのマッピングが含まれます。

13.1. ANSIBLE ベースのオーバークラウド設定 (CONFIG-DOWNLOAD)

director は、**config-download** 機能を使用してオーバークラウドを設定します。director は、OpenStack Orchestration サービス (heat) および OpenStack Workflow サービス (mistral) と共に **config-download** を使用してソフトウェア設定を生成し、その設定を各オーバークラウドノードに適用します。heat は **SoftwareDeployment** リソースから全デプロイメントデータを作成して、オーバークラウドのインストールと設定を行いますが、設定の適用は一切行いません。heat は、heat API から設定データの提供のみを行います。director がスタックを作成する場合には、mistral ワークフローが heat API に対して設定データ取得のクエリーを実行し、Ansible Playbook のセットを生成してオーバークラウドに適用します。

結果として、**openstack overcloud deploy** コマンドを実行すると、以下のプロセスが実行されます。

- director は **openstack-tripleo-heat-templates** を元に新たなデプロイメントプランを作成し、プランをカスタマイズするための環境ファイルおよびパラメーターをすべて追加します。
- director は heat を使用してデプロイメントプランを翻訳し、オーバークラウドスタックとすべての子リソースを作成します。これには、OpenStack Bare Metal サービス (ironic) を使用したノードのプロビジョニングも含まれます。
- heat はデプロイメントプランからソフトウェア設定も作成します。director はこのソフトウェア設定から Ansible Playbook をコンパイルします。
- director は、特に Ansible SSH アクセス用としてオーバークラウドノードに一時ユーザー (**tripleo-admin**) を生成します。
- director は heat ソフトウェア設定をダウンロードし、heat の出力を使用して Ansible Playbook のセットを生成します。
- director は、**ansible-playbook** を使用してオーバークラウドノードに Ansible Playbook を適用します。

13.2. CONFIG-DOWNLOAD の作業ディレクトリー

director により、**config-download** プロセス用に Ansible Playbook のセットが生成されます。これらの Playbook は **/var/lib/mistral/** 内の作業ディレクトリーに保管されます。このディレクトリーには、オーバークラウドの名前が付けられます。したがって、デフォルトでは **overcloud** です。

作業ディレクトリーには、各オーバークラウドロールの名前が付けられた複数のサブディレクトリーが存在します。これらのサブディレクトリーには、オーバークラウドロールのノードの設定に関連するす

べてのタスクが含まれます。さらに、これらのサブディレクトリーには、特定のノードの名前が付けられたサブディレクトリーが存在します。これらのサブディレクトリーには、オーバークラウドロールのタスクに適用するノード固有の変数が含まれます。したがって、作業ディレクトリー内のオーバークラウドロールは、以下のような設定になります。

```
─ /var/lib/mistral/overcloud
  │
  ├── Controller
  │   ├── overcloud-controller-0
  │   ├── overcloud-controller-1
  │   └── overcloud-controller-2
  ├── Compute
  │   ├── overcloud-compute-0
  │   ├── overcloud-compute-1
  │   └── overcloud-compute-2
  ...
```

それぞれの作業ディレクトリーは、各デプロイメント操作後の変更を記録するローカルの Git リポジトリーです。ローカル Git リポジトリーを使用して、各デプロイメント間の設定変更を追跡します。

13.3. CONFIG-DOWNLOAD の作業ディレクトリーへのアクセスの有効化

/var/lib/mistral/ にある作業ディレクトリー内の全ファイルの所有者は、OpenStack Workflow サービス (mistral) コンテナーの **mistral** ユーザーです。アンダークラウドの **stack** ユーザーに、このディレクトリー内の全ファイルへのアクセス権限を付与することができます。この設定は、ディレクトリー内の特定操作を実施するのに役立ちます。

手順

1. アンダークラウドの **stack** ユーザーに **/var/lib/mistral** ディレクトリーのファイルへのアクセス権限を付与するには、**setfacl** コマンドを使用します。

```
$ sudo setfacl -R -m u:stack:rwx /var/lib/mistral
$ sudo chmod -R og-rwx /var/lib/mistral/.ssh
```

このコマンドを実行しても、**mistral** ユーザーのディレクトリーへのアクセス権限は維持されます。

13.4. CONFIG-DOWNLOAD ログの確認

config-download プロセス中、Ansible によりアンダークラウド内の **config-download** の作業ディレクトリーにログファイルが作成されます。

手順

1. **less** コマンドを使用して、**config-download** の作業ディレクトリー内のログを表示します。以下の例では、**overcloud** 作業ディレクトリーが使われています。

```
$ less /var/lib/mistral/overcloud/ansible.log
```

13.5. 作業ディレクトリーでの GIT 操作の実施

config-download の作業ディレクトリーは、ローカルの Git リポジトリーです。デプロイメント操作を

実行するたびに、director は該当する変更に関する Git コミットを作業ディレクトリーに追加します。Git 操作を実施して、さまざまなステージでのデプロイメント設定を表示したり、異なるデプロイメント間で設定を比較したりすることができます。

作業ディレクトリーには制限がある点に注意してください。たとえば、Git を使用して **config-download** の作業ディレクトリーを前のバージョンに戻しても、この操作は作業ディレクトリー内の設定にしか影響を及ぼしません。したがって、以下の設定は影響を受けません。

- **オーバークラウドデータスキーマ:** 作業ディレクトリーのソフトウェア設定の前のバージョンを適用しても、データ移行およびスキーマ変更は取り消されません。
- **オーバークラウドのハードウェアレイアウト:** 以前のソフトウェア設定に戻しても、スケールアップ/ダウン等のオーバークラウドハードウェアに関する変更は取り消されません。
- **heat スタック:** 作業ディレクトリーを前のバージョンに戻しても、heat スタックに保管された設定は影響を受けません。heat スタックは新たなバージョンのソフトウェア設定を作成し、それがオーバークラウドに適用されます。オーバークラウドに永続的な変更を加えるには、**openstack overcloud deploy** コマンドを再度実行する前に、オーバークラウドスタックに適用する環境ファイルを変更します。

config-download の作業ディレクトリー内の異なるコミットを比較するには、以下の手順を実施します。

手順

1. オーバークラウドに関する **config-download** の作業ディレクトリーに移動します。この例の作業ディレクトリーは、**overcloud** という名前のオーバークラウド用です。

```
$ cd /var/lib/mistral/overcloud
```

2. **git log** コマンドを実行して、作業ディレクトリー内のコミットのリストを表示します。ログの出力に日付が表示されるようにフォーマットを設定することもできます。

```
$ git log --format=format:"%h%x09%cd%x09"
a7e9063 Mon Oct 8 21:17:52 2018 +1000
dfb9d12 Fri Oct 5 20:23:44 2018 +1000
d0a910b Wed Oct 3 19:30:16 2018 +1000
...
```

   デフォルトでは、最新のコミットから順に表示されます。

3. 2 つのコミットのハッシュに対して **git diff** コマンドを実行し、デプロイメント間の違いをすべて表示します。

```
$ git diff a7e9063 dfb9d12
```

13.6. CONFIG-DOWNLOAD を使用するデプロイメント方式

オーバークラウドのデプロイメントに関して、**config-download** を使用する方式は以下の 4 つに大別されます。

標準のデプロイメント

openstack overcloud deploy コマンドを実行して、プロビジョニングステージの後に設定ステージを自動的に実行します。これは、**openstack overcloud deploy** コマンドを実行する際のデフォルトの方式です。

プロビジョニングと設定の分離

特定のオプションを指定して **openstack overcloud deploy** コマンドを実行し、プロビジョニングステージと設定ステージを分離します。

デプロイメント後の ansible-playbook-command.sh スクリプトの実行

プロビジョニングステージと設定ステージを分離または組み合わせて **openstack overcloud deploy** コマンドを実行し、続いて **config-download** の作業ディレクトリーに用意されている **ansible-playbook-command.sh** スクリプトを実行し、設定ステージを再度適用します。

ノードのプロビジョニング、config-download の手動作成、および Ansible の実行

特定のオプションを指定して **openstack overcloud deploy** コマンドを実行し、ノードをプロビジョニングしてから、**deploy_steps_playbook.yaml** を指定して **ansible-playbook** コマンドを実行します。

13.7. 標準デプロイメントでの CONFIG-DOWNLOAD の実行

config-download を実行するためのデフォルトの方法は、**openstack overcloud deploy** コマンドを実行することです。この方式は、ほとんどの環境に適します。

前提条件

- アンダークラウドの正常なインストール。
- デプロイ可能なオーバークラウドノード
- 実際のオーバークラウドカスタマイズに該当する Heat 環境ファイル

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. デプロイメントコマンドを実行します。オーバークラウドに必要なすべての環境ファイルを追加します。

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
```

4. デプロイメントプロセスが完了するまで待ちます。

デプロイメントプロセス中に、director は **/var/lib/mistral/** の作業ディレクトリーに **config-download** ファイルを生成します。デプロイメントプロセスが終了したら、作業ディレクトリーの Ansible Playbooks を表示して、オーバークラウドを設定するために director が実行したタスクを確認します。

13.8. プロビジョニングと設定を分離した CONFIG-DOWNLOAD の実行

openstack overcloud deploy コマンドは、heat ベースのプロビジョニングプロセスの後に、**config-download** 設定プロセスを実行します。各プロセスを個別に実施するように、デプロイメントコマンドを実行することもできます。独立したプロセスとしてオーバークラウドノードをプロビジョニングするには、この方式を使用します。これにより、オーバークラウドの設定プロセスを実施する前に、ノードで手動の事前設定タスクを実行することができます。

前提条件

- アンダークラウドの正常なインストール。
- デプロイ可能なオーバークラウドノード
- 実際のオーバークラウドカスタマイズに該当する Heat 環境ファイル

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. **--stack-only** オプションを指定してデプロイメントコマンドを実行します。オーバークラウドに必要なすべての環境ファイルを追加します。

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --stack-only
```

4. プロビジョニングプロセスが完了するまで待ちます。
5. **tripleo-admin** ユーザーによるアンダークラウドからオーバークラウドへの SSH アクセスを有効にします。**config-download** プロセスでは、**tripleo-admin** ユーザーを使用して Ansible ベースの設定を実施します。

```
$ openstack overcloud admin authorize
```

6. ノードで手動の事前設定タスクを実行します。設定に Ansible を使用する場合は、**tripleo-admin** ユーザーを使用してノードにアクセスします。
7. **--config-download-only** オプションを指定してデプロイメントコマンドを実行します。オーバークラウドに必要なすべての環境ファイルを追加します。

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --config-download-only
```

8. 設定プロセスが完了するまで待ちます。

設定ステージ中に、director は **/var/lib/mistral/** の作業ディレクトリーに **config-download** ファイルを生成します。デプロイメントプロセスが終了したら、作業ディレクトリーの Ansible Playbooks を表示して、オーバークラウドを設定するために director が実行したタスクを確認します。

13.9. ANSIBLE-PLAYBOOK-COMMAND.SH スクリプトを使用した CONFIG-DOWNLOAD の実行

標準の方式または個別のプロビジョニングおよび設定プロセスを使用してオーバークラウドをデプロイすると、director は **/var/lib/mistral/** に作業ディレクトリーを生成します。このディレクトリーには、設定プロセスを再度実行するのに必要な Playbook およびスクリプトが含まれています。

前提条件

- 以下の方式のいずれかでデプロイされたオーバークラウド
  - プロビジョニングプロセスと設定プロセスをまとめて実施する標準の方式
  - プロビジョニングプロセスと設定プロセスを分離する方式

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. Ansible Playbook のディレクトリーに移動します。

```
$ cd /var/lib/mistral/overcloud/
```

3. **/var/lib/mistral/.ssh** ディレクトリーの所有者を **stack** ユーザーに変更します。

```
$ sudo chown stack. -R /var/lib/mistral/.ssh/
```

4. **ansible-playbook-command.sh** コマンドを実行して、オーバークラウドの設定を実行します。

```
$ sudo ./ansible-playbook-command.sh
```

5. **/var/lib/mistral/.ssh** ディレクトリーの所有者を **mistral** ユーザーに変更します。これは、**mistral_executor** コンテナー内で実行されている ansible-playbook コマンドが正常に実行されるようにするために必要です。

```
$ sudo chown 42430:42430 -R /var/lib/mistral/.ssh/
```

6. **mistral** ユーザーとしてスクリプトを再実行します。
このスクリプトには追加の Ansible 引数を渡すことができ、それらの引数は、そのまま **ansible-playbook** コマンドに渡されます。つまり、チェックモード (**--check**)、ホストの限定 (**--limit**)、変数のオーバーライド (**-e**) など、他の Ansible 機能を使用することができます。以下に例を示します。

```
$ ./ansible-playbook-command.sh --limit Controller
```

警告

--limit を使用して大規模にデプロイする場合、実行に含まれるホストのみがノード全体の SSH **known_hosts** ファイルに追加されます。したがって、ライブマイグレーションなどの一部の操作は、**known_hosts** ファイルにないノード間では機能しない場合があります。

7. 設定プロセスが完了するまで待ちます。

関連情報

- 作業ディレクトリーには、オーバークラウドの設定タスクを管理する **deploy_steps_playbook.yaml** という名前の Playbook が含まれています。この Playbook を表示するには、以下のコマンドを実行します。

```
$ less deploy_steps_playbook.yaml
```

  Playbook は、作業ディレクトリーに含まれているさまざまなタスクファイルを使用します。タスクファイルには、OpenStack Platform の全ロールに共通するものと、特定の OpenStack Platform ロールおよびサーバー固有のものがあります。

- 作業ディレクトリーには、オーバークラウドの **roles_data** ファイルで定義する各ロールに対応するサブディレクトリーも含まれます。以下に例を示します。

```
$ ls Controller/
```

  各 OpenStack Platform ロールにディレクトリーには、そのロール種別の個々のサーバー用のサブディレクトリーも含まれます。これらのディレクトリーには、コンポーザブルロールのホスト名の形式を使用します。

```
$ ls Controller/overcloud-controller-0
```

- **deploy_steps_playbook.yaml** の Ansible タスクはタグ付けされます。タグの全リストを確認するには、**ansible-playbook** で CLI オプション **--list-tags** を使用します。

```
$ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags deploy_steps_playbook.yaml
```

  次に、**ansible-playbook-command.sh** スクリプトで **--tags**、**--skip-tags**、**--start-at-task** のいずれかを使用して、タグ付けした設定を適用します。

```
$ ./ansible-playbook-command.sh --tags overcloud
```

1. オーバークラウドに対して **config-download** Playbook を実行すると、それぞれのホストの SSH フィンガープリントに関するメッセージが表示される場合があります。これらのメッセージを回避するには、**ansible-playbook-command.sh** スクリプトの実行時に、**--ssh-common-args="-o StrictHostKeyChecking=no"** を追加します。

```
$ ./ansible-playbook-command.sh --tags overcloud --ssh-common-args="-o
StrictHostKeyChecking=no"
```

13.10. 手動で作成した PLAYBOOK を使用した CONFIG-DOWNLOAD の実行

標準のワークフローとは別に、専用の **config-download** ファイルを作成することができます。たとえば、**--stack-only** オプションを指定して **openstack overcloud deploy** コマンドを実行し、ノードをプロビジョニングしてから、別途 Ansible 設定を手動で適用することができます。

前提条件

- アンダークラウドの正常なインストール。
- デプロイ可能なオーバークラウドノード
- 実際のオーバークラウドカスタマイズに該当する Heat 環境ファイル

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. **--stack-only** オプションを指定してデプロイメントコマンドを実行します。オーバークラウドに必要なすべての環境ファイルを追加します。

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --stack-only
```

4. プロビジョニングプロセスが完了するまで待ちます。
5. **tripleo-admin** ユーザーによるアンダークラウドからオーバークラウドへの SSH アクセスを有効にします。**config-download** プロセスでは、**tripleo-admin** ユーザーを使用して Ansible ベースの設定を実施します。

```
$ openstack overcloud admin authorize
```

6. **config-download** ファイルを生成します。

```
$ openstack overcloud config download \
  --name overcloud \
  --config-dir ~/config-download
```

   - **--name** は、Ansible ファイルのエクスポートに使用するオーバークラウドの名前です。
   - **--config-dir** は、**config-download** ファイルを保存する場所です。

7. **config-download** ファイルが含まれるディレクトリーに移動します。

```
$ cd ~/config-download
```

8. 静的なインベントリーファイルを生成します。

```
$ tripleo-ansible-inventory \
  --stack <overcloud> \
  --ansible_ssh_user heat-admin \
  --static-yaml-inventory inventory.yaml
```

   - **<overcloud>** を実際のオーバークラウドの名前に置き換えてください。

9. **config-download** ファイルおよび静的なインベントリーファイルを使用して、設定を実施します。デプロイメント用の Playbook を実行するには、**ansible-playbook** コマンドを実行します。

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  ~/config-download/deploy_steps_playbook.yaml
```

10. 設定プロセスが完了するまで待ちます。

11. この設定から手動で **overcloudrc** ファイルを生成するには、以下のコマンドを実行します。

```
$ openstack action execution run \
  --save-result \
  --run-sync \
  tripleo.deployment.overcloudrc \
  '{"container":"overcloud"}' \
  | jq -r '.["result"]["overcloudrc.v3"]' > overcloudrc.v3
```

12. デプロイメントステータスを手動で success に設定します。

```
$ openstack workflow execution create
tripleo.deployment.v1.set_deployment_status_success '{"plan": "<OVERCLOUD>"}'
```

   - **<OVERCLOUD>** を実際のオーバークラウドの名前に置き換えてください。

関連情報

- **config-download** ディレクトリーには、オーバークラウドの設定を実行する **deploy_steps_playbook.yaml** という名前の Playbook が含まれています。この Playbook を表示するには、以下のコマンドを実行します。

```
$ less deploy_steps_playbook.yaml
```

  Playbook は、作業ディレクトリーに含まれているさまざまなタスクファイルを使用します。タスクファイルには、OpenStack Platform の全ロールに共通するものと、特定の OpenStack Platform ロールおよびサーバー固有のものがあります。

- **config-download** ディレクトリーには、オーバークラウドの **roles_data** ファイルで定義する各ロールに対応するサブディレクトリーも含まれます。以下に例を示します。

```
$ ls Controller/
```

各 OpenStack Platform ロールにディレクトリーには、そのロール種別の個々のサーバー用のサブディレクトリーも含まれます。これらのディレクトリーには、コンポーザブルロールのホスト名の形式を使用します。

```
$ ls Controller/overcloud-controller-0
```

- **deploy_steps_playbook.yaml** の Ansible タスクはタグ付けされます。タグの全リストを確認するには、**ansible-playbook** で CLI オプション **--list-tags** を使用します。

```
$ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags deploy_steps_playbook.yaml
```

次に、**ansible-playbook-command.sh** スクリプトで **--tags**、**--skip-tags**、**--start-at-task** のいずれかを使用して、タグ付けした設定を適用します。

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  --tags overcloud \
  ~/config-download/deploy_steps_playbook.yaml
```

1. オーバークラウドに対して **config-download** Playbook を実行すると、それぞれのホストの SSH フィンガープリントに関するメッセージが表示される場合があります。これらのメッセージを回避するには、**--ssh-common-args="-o StrictHostKeyChecking=no"** を **ansible-playbook** コマンドに追加します。

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --ssh-common-args="-o StrictHostKeyChecking=no" \
  --become \
  --tags overcloud \
  ~/config-download/deploy_steps_playbook.yaml
```

13.11. CONFIG-DOWNLOAD の制限事項

config-download 機能にはいくつかの制限があります。

- **--tags**、**--skip-tags**、**--start-at-task** などの ansible-playbook CLI 引数を使用する場合には、デプロイメントの設定は、間違った順序で実行したり適用したりしないでください。これらの CLI 引数は、以前に失敗したタスクを再度実行する場合や、初回のデプロイメントを繰り返す場合に便利な方法です。ただし、デプロイメントの一貫性を保証するには、**deploy_steps_playbook.yaml** の全タスクを順番どおりに実行する必要があります。

- タスク名に変数を使用する特定のタスクに **--start-at-task** 引数を使用することはできません。たとえば、**--start-at-task** 引数は、以下の Ansible タスクでは機能しません。

```
- name: Run puppet host configuration for step {{ step }}
```

- オーバークラウドのデプロイメントに director でデプロイされた Ceph Storage クラスターが含まれる場合、**external_deploy_steps** のタスクも省略しない限り、**--check** オプションを使用する際に **step1** のタスクを省略することはできません。

- **--forks** オプションを使用して、同時に実施する Ansible タスクの数を設定することができます。ただし、同時タスクが 25 を超えると、**config-download** 操作のパフォーマンスが低下します。このため、**--forks** オプションに 25 を超える値を設定しないでください。

13.12. CONFIG-DOWNLOAD の主要ファイル

config-download の作業ディレクトリー内の主要なファイルを以下に示します。

Ansible の設定および実行

config-download の作業ディレクトリー内の以下のファイルは、Ansible を設定/実行するための専用ファイルです。

ansible.cfg
: **ansible-playbook** 実行時に使用する設定ファイル

ansible.log
: 最後に実行した **ansible-playbook** に関するログファイル

ansible-errors.json
: デプロイメントエラーが含まれる JSON 構造のファイル

ansible-playbook-command.sh
: 最後のデプロイメント操作の **ansible-playbook** コマンドを再実行するための実行可能スクリプト

ssh_private_key
: Ansible がオーバークラウドノードにアクセスする際に使用する SSH 秘密鍵

tripleo-ansible-inventory.yaml
: すべてのオーバークラウドノードのホストおよび変数が含まれる Ansible インベントリーファイル

overcloud-config.tar.gz
: 作業ディレクトリーのアーカイブ

Playbook

以下のファイルは、**config-download** の作業ディレクトリー内の Playbook です。

deploy_steps_playbook.yaml
: デプロイメントのメインステップ。この Playbook により、オーバークラウド設定の主要な操作が実施されます。

pre_upgrade_rolling_steps_playbook.yaml
: メジャーアップグレードのための事前アップグレードステップ

upgrade_steps_playbook.yaml
: メジャーアップグレードのステップ

post_upgrade_steps_playbook.yaml
: メジャーアップグレードに関するアップグレード後ステップ

update_steps_playbook.yaml

マイナー更新のステップ

fast_forward_upgrade_playbook.yaml

Fast Forward Upgrade のタスク。Red Hat OpenStack Platform のロングライフバージョンから次のロングライフバージョンにアップグレードする場合にのみ、この Playbook 使用します。

13.13. CONFIG-DOWNLOAD のタグ

Playbook では、オーバークラウドに適用されるタスクを管理するのにタグ付けされたタスクを使用します。**ansible-playbook** CLI の引数 **--tags** または **--skip-tags** でタグを使用して、実行するタスクを管理します。デフォルトで有効なタグに関する情報を、以下のリストに示します。

facts

ファクト収集操作

common_roles

すべてのノードに共通な Ansible ロール

overcloud

オーバークラウドデプロイメント用のすべてのプレイ

pre_deploy_steps

deploy_steps の操作の前に実施されるデプロイメント

host_prep_steps

ホスト準備のステップ

deploy_steps

デプロイメントのステップ

post_deploy_steps

deploy_steps の操作の後に実施される手順

external

すべての外部デプロイメントタスク

external_deploy_steps

アンダークラウドでのみ実行される外部デプロイメントタスク

13.14. CONFIG-DOWNLOAD のデプロイメントステップ

deploy_steps_playbook.yaml Playbook により、オーバークラウドが設定されます。この Playbook により、オーバークラウドデプロイメントプランに基づき完全なオーバークラウドをデプロイするのに必要なすべてのソフトウェア設定が適用されます。

本項では、この Playbook で使用されるさまざまな Ansible プレイの概要について説明します。本項のプレイと同じ名前が、Playbook 内で使用され **ansible-playbook** の出力にも表示されます。本項では、それぞれのプレイに設定される Ansible タグについても説明します。

Gather facts from undercloud

アンダークラウドノードからファクトを収集します。
タグ: facts

Gather facts from overcloud

オーバークラウドノードからファクトを収集します。
タグ: facts

Load global variables

global_vars.yaml からすべての変数を読み込みます。
タグ: **always**

Common roles for TripleO servers

共通の Ansible ロールをすべてのオーバークラウドノードに適用します。これには、ブートストラップパッケージをインストールする tripleo-bootstrap および ssh の既知のホストを設定する tripleo-ssh-known-hosts が含まれます。
タグ: **common_roles**

Overcloud deploy step tasks for step 0

deploy_steps_tasks テンプレートインターフェイスからのタスクを適用します。
タグ: **overcloud**、**deploy_steps**

Server deployments

ネットワーク設定や hieradata 等の設定に、サーバー固有の heat デプロイメントを適用します。これには、NetworkDeployment、<Role>Deployment、<Role>AllNodesDeployment 等が含まれます。
タグ: **overcloud**、**pre_deploy_steps**

Host prep steps

host_prep_steps テンプレートインターフェイスからのタスクを適用します。
タグ: **overcloud**、**host_prep_steps**

External deployment step [1,2,3,4,5]

external_deploy_steps_tasks テンプレートインターフェイスからのタスクを適用します。Ansible は、アンダークラウドノードに対してのみこれらのタスクを実行します。
タグ: **external**、**external_deploy_steps**

Overcloud deploy step tasks for [1,2,3,4,5]

deploy_steps_tasks テンプレートインターフェイスからのタスクを適用します。
タグ: **overcloud**、**deploy_steps**

Overcloud common deploy step tasks [1,2,3,4,5]

各ステップで実施される共通タスクを適用します。これには、puppet ホストの設定、**container-puppet.py**、および paunch (コンテナー設定) が含まれます。
タグ: **overcloud**、**deploy_steps**

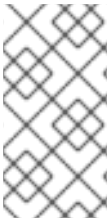
Server Post Deployments

5 ステップのデプロイメントプロセス後に実施される設定に、サーバー固有の heat デプロイメントを適用します。
タグ: **overcloud**、**post_deploy_steps**

External deployment Post Deploy tasks

external_post_deploy_steps_tasks テンプレートインターフェイスからのタスクを適用します。Ansible は、アンダークラウドノードに対してのみこれらのタスクを実行します。
タグ: **external**、**external_deploy_steps**

第14章 ANSIBLE を使用したコンテナーの管理



注記

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

Red Hat OpenStack Platform 16.1 では、Paunch を使用してコンテナーを管理します。ただし、Ansible ロール **tripleo-container-manage** を使用してコンテナーの管理操作を実施することもできます。**tripleo-container-manage** ロールを使用する場合は、初めに Paunch を無効にする必要があります。Paunch が無効になっていると、director は自動的に Ansible ロールを使用します。カスタム Playbook を作成して、特定のコンテナー管理操作を実施することもできます。

- heat が生成するコンテナー設定データを収集する。**tripleo-container-manage** ロールは、このデータを使用してコンテナーのデプロイメントをオーケストレーションします。
- コンテナーを起動する。
- コンテナーを停止する。
- コンテナーを更新する。
- コンテナーを削除する。
- 特定の設定でコンテナーを実行する。

director はコンテナー管理を自動的に実施しますが、コンテナー設定をカスタマイズしなければならない場合や、オーバークラウドを再デプロイせずにコンテナーにホットフィックスを適用しなければならない場合があります。

注記

このロールがサポートするのは Podman コンテナー管理だけです。

前提条件

- アンダークラウドの正常なインストール。詳細は、「director のインストール」 を参照してください。

14.1. アンダークラウドでの TRIPLEO-CONTAINER-MANAGE ANSIBLE ロールの有効化

注記

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

Red Hat OpenStack Platform 16.1 では、Paunch がデフォルトのコンテナー管理メカニズムです。ただし、**tripleo-container-manage** Ansible ロールを使用することもできます。このロールを使用する場合は、Paunch を無効にする必要があります。

前提条件

- ベースオペレーティングシステムおよび **python3-tripleoclient** パッケージがインストールされたホストマシン。詳細は、*3章director インストールの準備* を参照してください。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **undercloud.conf** ファイルで **undercloud_enable_paunch** パラメーターを **false** に設定します。

```
undercloud_enable_paunch: false
```

3. **openstack undercloud install** コマンドを実行します。

```
$ openstack undercloud install
```

14.2. オーバークラウドでの TRIPLEO-CONTAINER-MANAGE ANSIBLE ロールの有効化

注記

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

Red Hat OpenStack Platform 16.1 では、Paunch がデフォルトのコンテナー管理メカニズムです。ただし、**tripleo-container-manage** Ansible ロールを使用することもできます。このロールを使用する場合は、Paunch を無効にする必要があります。

前提条件

- アンダークラウドの正常なインストール。詳細は、*4章 アンダークラウドへの director のインストール* を参照してください。

手順

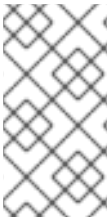
1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. source コマンドで **stackrc** 認証情報ファイルを読み込みます。

```
$ source ~/stackrc
```

3. デプロイメントに該当するその他の環境ファイルと共に、オーバークラウドデプロイメントコマンドに **/usr/share/openstack-tripleo-heat-templates/environments/disable-paunch.yaml** ファイルを追加します。

```
(undercloud) [stack@director ~]$ openstack overcloud deploy --templates \
  -e /usr/share/openstack-tripleo-heat-templates/environments/disable-paunch.yaml
  -e <other_environment_files>
  ...
```

14.3. 単一コンテナーでの操作の実施



注記

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

tripleo-container-manage ロールを使用して、すべてのコンテナーまたは特定のコンテナーを管理することができます。特定のコンテナーを管理する場合は、特定のコンテナーをカスタム Ansible Playbook の対象にできるように、コンテナーデプロイメントステップおよびコンテナー設定 JSON ファイルの名前を特定する必要があります。

前提条件

- アンダークラウドの正常なインストール。詳細は、*4章 アンダークラウドへの director のインストール* を参照してください。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. source コマンドで **overcloudrc** 認証情報ファイルを読み込みます。

```
$ source ~/overcloudrc
```

3. コンテナーデプロイメントステップを特定します。各ステップのコンテナー設定は、**/var/lib/tripleo-config/container-startup-config/step_{1,2,3,4,5,6}** ディレクトリーに保存されています。
4. コンテナーの JSON 設定ファイルを特定します。コンテナー設定ファイルは、該当する **step_*** ディレクトリーに保存されています。たとえば、ステップ 1 の HAProxy コンテナーの設定ファイルは、**/var/lib/tripleo-config/container-startup-config/step_1/haproxy.json** です。
5. 適切な Ansible Playbook を作成します。たとえば、HAProxy コンテナーイメージを置き換えるには、以下の例に示す Playbook を使用します。

```
- hosts: localhost
  become: true
  tasks:
    - name: Manage step_1 containers using tripleo-ansible
      block:
        - name: "Manage HAproxy container at step 1 with tripleo-ansible"
          include_role:
            name: tripleo-container-manage
          vars:
            tripleo_container_manage_systemd_order: true
```

```
        tripleo_container_manage_config_patterns: 'haproxy.json'
        tripleo_container_manage_config: "/var/lib/tripleo-config/container-startup-
config/step_1"
        tripleo_container_manage_config_id: "tripleo_step1"
        tripleo_container_manage_config_overrides:
          haproxy:
            image: registry.redhat.io/tripleomaster/<HAProxy-container>:hotfix
```

tripleo-container-manage ロールで使用することのできる変数についての詳細は、「tripleo-container-manage ロールの変数」 を参照してください。

6. Playbook を実行します。

```
(overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml
```

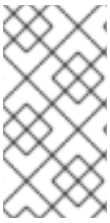
変更を適用せずに Playbook を実行する場合は、**ansible-playbook** コマンドに **--check** オプションを追加します。

```
(overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml --check
```

変更を適用せずに Playbook がコンテナーに加える変更を把握する場合は、**ansible-playbook** コマンドに **--check** および **--diff** オプションを追加します。

```
(overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml --check --diff
```

14.4. TRIPLEO-CONTAINER-MANAGE ロールの変数



注記

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

tripleo-container-manage Ansible ロールには、以下の変数が含まれます。

表14.1 ロール変数

名前	デフォルト値	説明

名前	デフォルト値	説明
tripleo_container_manage_check_puppet_config	false	Ansible で Puppet コンテナー設定を確認する場合は、この変数を使用します。Ansible は、設定ハッシュを使用して更新されたコンテナー設定を識別することができます。コンテナーに Puppet からの新規設定がある場合は、この変数を **true** に設定します。これにより、Ansible は新規設定を検出し、Ansible が再起動しなければならないコンテナーリストにコンテナーを追加することができます。
tripleo_container_manage_cli	podman	この変数を使用して、コンテナーを管理するのに使用するコマンドラインインターフェイスを設定します。**tripleo-container-manage** ロールがサポートするのは Podman だけです。
tripleo_container_manage_concurrency	1	この変数を使用して、同時に管理するコンテナーの数を設定します。
tripleo_container_manage_config	/var/lib/tripleo-config/	この変数を使用して、コンテナー設定ディレクトリーへのパスを設定します。
tripleo_container_manage_config_id	tripleo	この変数を使用して、特定の設定ステップの ID を設定します。たとえば、デプロイメントのステップ 2 のコンテナーを管理するには、この値を **tripleo_step2** に設定します。
tripleo_container_manage_config_patterns	*.json	この変数を使用して、コンテナー設定ディレクトリーの設定ファイルを識別する bash 正規表現を設定します。

名前	デフォルト値	説明
tripleo_container_manage_debug	false	この変数を使用して、デバッグモードを有効または無効にします。特定の一度限りの設定でコンテナーを実行する場合、コンテナーのライフサイクルを管理するコンテナーコマンドを出力する場合、またはテストおよび検証目的で操作を行わずにコンテナー管理を実施する場合に、**tripleo-container-manage** ロールをデバッグモードで実行します。
tripleo_container_manage_healthcheck_disable	false	この変数を使用して、ヘルスチェックを有効または無効にします。
tripleo_container_manage_log_path	/var/log/containers/stdouts	この変数を使用して、コンテナーの stdout ログパスを設定します。
tripleo_container_manage_systemd_order	false	この変数を使用して、Ansible による systemd のシャットダウン順序を有効または無効にします。
tripleo_container_manage_systemd_teardown	true	この変数を使用して、使用されなくなったコンテナーのクリーンアップをトリガーします。
tripleo_container_manage_config_overrides	{}	この変数を使用して、コンテナー設定をオーバーライドします。この変数では、値にディクショナリー形式が使用されます。ここで、それぞれのキーはコンテナー名およびオーバーライドするパラメーター (例: コンテナーイメージ、ユーザー) です。この変数は、JSON コンテナー設定ファイルにカスタムオーバーライドを書き込みません。したがって、コンテナーが新たにデプロイ、更新、またはアップグレードされると、JSON 設定ファイルの内容に戻ります。
tripleo_container_manage_valid_exit_code	[]	この変数を使用して、コンテナーが終了コードを返すかどうかを確認します。この値はリストにする必要があります (例: **[0,3]**)。

第15章 検証フレームワークの使用

Red Hat OpenStack Platform には検証フレームワークが含まれており、アンダークラウドおよびオーバークラウドの要件と機能の検証に使用することができます。フレームワークには、以下に示す 2 つの検証種別が含まれます。

- Ansible ベースの手動検証: **openstack tripleo validator** コマンドセットを使用して実行します。
- インフライトの自動検証: デプロイメントプロセス中に実行されます。

実行する検証を理解し、環境に関係のない検証をスキップする必要があります。たとえば、事前デプロイメントの検証には、どこでも TLS のテストが含まれます。環境を TLS 用に設定する予定がない場合、どこでも、このテストは失敗します。**openstack tripleo validator run** コマンドの **--validation** オプションを使用して、環境に応じて検証を調整します。

15.1. ANSIBLE ベースの検証

Red Hat OpenStack Platform director のインストール時に、director は **openstack-tripleo-validations** パッケージから Playbook のセットもインストールします。それぞれの Playbook には、特定のシステム要件のテストおよびグループが含まれます。このグループを使用して、OpenStack Platform のライフサイクル中の特定のステージにタグ付けされた一連のテストを実施することができます。

no-op

no-op (操作なし) タスクを実行してワークフローが正しく機能していることを確かめる検証。これらの検証は、アンダークラウドとオーバークラウドの両方で実行されます。

prep

アンダークラウドノードのハードウェア設定を確認する検証。**openstack undercloud install** コマンドを実行する前に、これらの検証を実行します。

openshift-on-openstack

環境が要件を満たし OpenShift on OpenStack をデプロイできることを確認する検証

pre-introspection

Ironic Inspector を使用するノードのイントロスペクション前に実行する検証

pre-deployment

openstack overcloud deploy コマンドの前に実行する検証

post-deployment

オーバークラウドのデプロイメントが完了した後に実行する検証

pre-upgrade

アップグレード前の OpenStack デプロイメントに対する検証

post-upgrade

アップグレード後の OpenStack デプロイメントに対する検証

15.2. 検証のリスト表示

利用可能なさまざまな種別の検証をリスト表示するには、**openstack tripleo validator list** コマンドを実行します。

手順

1. source コマンドで **stackrc** ファイルを読み込みます。

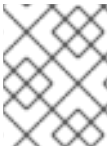
```
$ source ~/stackrc
```

2. **openstack tripleo validator list** コマンドを実行します。
   - すべての検証をリスト表示するには、オプションを設定せずにコマンドを実行します。

```
$ openstack tripleo validator list
```

   - グループの検証をリスト表示するには、**--group** オプションを指定してコマンドを実行します。

```
$ openstack tripleo validator list --group prep
```



注記

オプションの全リストについては、**openstack tripleo validator list --help** を実行してください。

15.3. 検証の実行

検証または検証グループを実行するには、**openstack tripleo validator run** コマンドを使用します。オプションの全リストを表示するには、**openstack tripleo validator run --help** コマンドを使用します。

手順

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. **inventory.yaml** という静的インベントリーファイルを作成して検証します。

```
$ tripleo-ansible-inventory --static-yaml-inventory inventory.yaml
$ openstack tripleo validator run  --group pre-introspection -i inventory.yaml
```

3. **openstack tripleo validator run** コマンドを入力します。
   - 単一の検証を実行するには、**--validation** オプションで検証の名前を指定してコマンドを入力します。たとえば、アンダークラウドのメモリー要件を確認するには、**--validation check-ram** と入力します。

```
$ openstack tripleo validator run --validation check-ram
```

     オーバークラウドにデフォルトのプラン名 **overcloud** 以外の名前を使用する場合は、**--plan** オプションでプラン名を設定します。

```
$ openstack tripleo validator run --validation check-ram --plan myovercloud
```

     複数の特定の検証を実行するには、実行する検証のコンマ区切りリストとともに **--validation** オプションを使用します。使用可能な検証のリストを表示する方法の詳細については、Listing validations を参照してください。

- グループのすべての検証を実行するには、**--group** オプションを指定してコマンドを入力します。

```
$ openstack tripleo validator run --group prep
```

特定の検証からの詳細出力を表示するには、レポートからの特定検証の UUID を指定して **openstack tripleo validator show run --full** コマンドを実行します。

```
$ openstack tripleo validator show run --full <UUID>
```

15.4. 検証履歴の表示

検証または検証グループの実行後に、director は各検証の結果を保存します。**openstack tripleo validator show history** コマンドを使用して、過去の検証結果を表示します。

前提条件

- 検証または検証グループを実行している。

手順

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. すべての検証のリストを表示します。

```
$ openstack tripleo validator show history
```

特定の検証種別の履歴を表示するには、**--validation** オプションを指定して同じコマンドを実行します。

```
$ openstack tripleo validator show history --validation ntp
```

3. **openstack tripleo validator show run --full** コマンドを使用して、特定の検証 UUID のログを表示します。

```
$ openstack tripleo validator show run --full 7380fed4-2ea1-44a1-ab71-aab561b44395
```

15.5. 検証フレームワークのログ形式

検証または検証グループの実行後に、director は各検証の JSON 形式のログを **/var/logs/validations** ディレクトリーに保存します。ファイルを手動で表示するか、**openstack tripleo validator show run --full** コマンドを使用して特定の検証 UUID のログを表示することができます。

それぞれの検証ログファイルは、特定の形式に従います。

- **<UUID>_<Name>_<Time>**

  UUID

  検証用の Ansible UUID

  名前

検証の Ansible 名

Time

検証を実行した時の開始日時

それぞれの検証ログには、3 つの主要部分が含まれます。

- plays
- stats
- validation_output

plays

plays セクションには、検証の一部として director が実行したタスクに関する情報が含まれます。

play

プレイはタスクのグループです。それぞれの **play** セクションには、開始/終了時刻、期間、プレイのホストグループ、ならびに検証 ID およびパス等の、特定のタスクグループに関する情報が含まれます。

tasks

検証を行うために director が実行する個別の Ansible タスク。それぞれの **tasks** セクションには **hosts** セクションが含まれ、それぞれの個別ホストで生じたアクションおよびアクションの実行結果が含まれます。**tasks** セクションには **task** セクションも含まれ、これにはタスクの期間が含まれます。

stats

stats セクションには、各ホストで実施した全タスクの結果の基本概要 (成功したタスクおよび失敗したタスク等) が含まれます。

validation_output

検証中にいずれかのタスクが失敗したか、警告メッセージが発行された場合、**validation_output** にその失敗または警告の出力が含まれます。

15.6. インフライト検証

Red Hat OpenStack Platform では、コンポーザブルサービスのテンプレートに、インフライト検証が含まれています。インフライト検証により、オーバークラウドのデプロイメントプロセスの主要ステップにおけるサービスの動作ステータスを確認します。

インフライト検証は、デプロイメントプロセスの一部として自動的に実行されます。一部のインフライト検証は、**openstack-tripleo-validations** パッケージからのロールも使用します。

第16章 オーバークラウドノードのスケーリング

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場合は、オーバークラウドを更新する必要があります。

警告

オーバークラウドからノードを削除する場合は、**openstack server delete** を使用しないでください。本項の手順に従って、適切にノードの削除/置き換えを行ってください。

注記

オーバークラウドノードのスケールアウトまたは削除を開始する前に、ベアメタルノードがメンテナンスモードに設定されていないことを確認してください。

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

表16.1 各ノード種別のスケーリングサポート

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	非対応	非対応	*17章 コントローラーノードの置き換え* に記載の手順を使用して、コントローラーノードを置き換えることができます。
Compute	対応	対応	
Ceph Storage ノード	対応	非対応	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。
オブジェクトストレージノード	対応	対応	

重要

オーバークラウドをスケーリングする前に、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。

16.1. オーバークラウドへのノード追加

director のノードプールにさらにノードを追加するには、以下の手順を実施します。

注記

Red Hat OpenStack Platform の新規インストールには、セキュリティーエラータやバグ修正などの特定の更新が含まれていません。その結果、Red Hat Customer Portal または Red Hat Satellite Server を使用する接続環境をスケールアップすると、RPM 更新は新しいノードに適用されません。最新の更新をオーバークラウドノードに適用するには、以下のいずれかを実行する必要があります。

- スケールアウト操作後にノードのオーバークラウド更新を完了します。
- **virt-customize** ツールを使用して、スケールアウト操作の前にパッケージをベースのオーバークラウドイメージに変更します。詳細は、Red Hat ナレッジベースで Modifying the Red Hat Linux OpenStack Platform Overcloud Image with virt-customize のソリューションを参照してください。

手順

1. 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 (**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":"ipmi",
        "pm_user":"admin",
        "pm_password":"p@55w0rd!",
        "pm_addr":"192.168.24.207"
    },
    {
        "mac":[
            "ee:ee:ee:ee:ee:ee"
        ],
        "cpu":"4",
        "memory":"6144",
        "disk":"40",
        "arch":"x86_64",
        "pm_type":"ipmi",
        "pm_user":"admin",
        "pm_password":"p@55w0rd!",
        "pm_addr":"192.168.24.208"
    }
  ]
}
```

2. 新しいノードを登録します。

```
$ source ~/stackrc
$ openstack overcloud node import newnodes.json
```

3. 新しいノードを登録したら、新しいノードごとにイントロスペクションプロセスを起動します。

```
$ openstack overcloud node introspect <node_UUID> --provide
```

- **<node_UUID>** を、追加するノードの UUID に置き換えます。このプロセスにより、ノードのハードウェア属性の検出とベンチマークが実行されます。

4. ノードのイメージプロパティーを設定します。

```
$ openstack overcloud node configure <node_UUID>
```

16.2. ロールのノード数の追加

特定ロールのオーバークラウドノード (たとえばコンピュートノード) をスケーリングするには、以下の手順を実施します。

手順

1. それぞれの新規ノードを希望するロールにタグ付けします。たとえば、ノードをコンピュートロールにタグ付けするには、以下のコマンドを実行します。

```
$ openstack baremetal node set --property capabilities='profile:compute,boot_option:local'
<node_UUID>
```

- **<node_UUID>** をタグ付けするノードの UUID に置き換えます。

2. オーバークラウドをスケーリングするには、ノード数が含まれる環境ファイルを編集してオーバークラウドを再デプロイする必要があります。たとえば、オーバークラウドをコンピュートノード 5 台にスケーリングするには、**ComputeCount** パラメーターを編集します。

```
parameter_defaults:
  ...
  ComputeCount: 5
  ...
```

3. 更新したファイルを使用して、デプロイメントコマンドを再度実行します。以下の例では、このファイルは **node-info.yaml** という名前です。

```
$ openstack overcloud deploy --templates \
  -e /home/stack/templates/node-info.yaml \
  -e [..]
```

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。これには、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が含まれます。

4. デプロイメント操作が完了するまで待ちます。

16.3. コンピュートノードの削除または置き換え

状況によっては、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ります。たとえば、クラウドをスケールダウンするには、問題のあるコンピュートノードを置き換えるか、コンピュートノードのグループを削除する必要がある場合があります。コンピュートノードを削除すると、スケールアウト操作中にインデックスが再利用されないように、ノードのインデックスがデフォルトで拒否リストに追加されます。

オーバークラウドデプロイメントからノードを削除した後、削除されたコンピュートノードを置き換えることができます。

前提条件

- 削除するノードでは、ノードが新しいインスタンスをスケジュールできないようにするために、コンピュートサービスが無効になっています。Compute サービスが無効になっていることを確認するには、以下のコマンドを使用して Compute サービスを一覧表示します。

  ```
  (overcloud)$ openstack compute service list
  ```

  Compute サービスが無効になっていない場合は、Compture サービスを無効にします。

  ```
  (overcloud)$ openstack compute service set <hostname> nova-compute --disable
  ```

- **<hostname>** を、無効にするコンピュートノードのホスト名に置き換えます。

  ヒント

  --disable-reason オプションを使用して、サービスを無効にする理由についての簡単な説明を追加します。これは、コンピュートサービスを再デプロイする場合に役立ちます。

- コンピュートノードのワークロードは、他のコンピュートノードに移行されました。詳しくは、Migrating virtual machine instances between Compute nodes を参照してください。
- インスタンス HA が有効になっている場合は、次のいずれかのオプションを選択します。
  - Compute ノードにアクセスできる場合は、**root** ユーザーとして Compute ノードにログインし、**shutdown -h now** コマンドを使用してクリーンシャットダウンを実行します。
  - Compute ノードにアクセスできない場合は、**root** ユーザーとしてコントローラーノードにログインし、Compute ノードの STONITH デバイスを無効にして、ベアメタルノードをシャットダウンします。

    ```
    $ sudo pcs stonith disable <compute_UUID>
    ```

  - source コマンドで **stackrc** アンダークラウド認証情報ファイルを読み込み、ベアメタルノードの電源をオフにします。

    ```
    $ source ~/stackrc
    (undercloud)$ openstack baremetal node power off <compute_UUID>
    ```

- **<compute_UUID>** を、削除するコンピュートノードの UUID に置き換えます。

手順

1. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

2. オーバークラウドスタックの名前を特定します。

```
(undercloud)$ openstack stack list
```

3. 削除するコンピュートノードの UUID またはホスト名を特定します。

```
(undercloud)$ openstack server list
```

4. (オプション) **--update-plan-only** オプションを指定して **overcloud deploy** コマンドを実行し、テンプレートからの最新の設定でプランを更新します。これにより、コンピュートノードを削除する前に、オーバークラウドの設定が最新の状態になります。

```
(undercloud)$ openstack overcloud deploy --stack <overcloud> --update-plan-only \
  --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 \
  -e /home/stack/templates/rhel-registration/environment-rhel-registration.yaml \
  -e [...]
```

&lt **;overcloud&gt**; をオーバークラウドスタックの名前に置き換えます。

注記

オーバークラウドノードのブロックリストを更新した場合は、オーバークラウドプランを更新する必要があります。オーバークラウドノードを拒否リストに追加する方法は、ノードのブロックリスト を参照してください。

5. スタックからコンピュートノードを削除します。

```
(undercloud)$ openstack overcloud node delete --stack <overcloud> \
 <node_1> ... [node_n]
```

- &lt **;overcloud&gt**; をオーバークラウドスタックの名前に置き換えます。
- **<node_1>** (およびオプションとして **[node_n]** までのすべてのノード) を、削除するコンピュートノードのコンピュートサービスのホスト名または UUID に置き換えます。UUID とホスト名を混在させて使用しないでください。UUID だけ、またはホスト名だけを使用します。

注記

ノードの電源がすでにオフになっている場合、このコマンドは **WARNING** メッセージを返します。

```
Ansible failed, check log at /var/lib/mistral/overcloud/ansible.log
WARNING: Scale-down configuration error. Manual cleanup of some
actions may be necessary. Continuing with node removal.
```

電源がオフになっているノード（ノード）による問題を手動で対処するには、到達不可能なコンピュートノードの削除の完了の ステップ 1 から 8 を完了してから、この手順の次のステップに進みます。

6. コンピュートノードが削除されるまで待ちます。

7. 削除した各ノードのネットワークエージェントを削除します。

```
(undercloud)$ source ~/overcloudrc
(overcloud)$ for AGENT in $(openstack network agent list \
  --host <scaled_down_node> -c ID -f value) ; \
  do openstack network agent delete $AGENT ; done
```

&lt **;scaled down node&gt**; を、削除したノードのホスト名に置き換えます。

8. コマンドの出力を確認します。RHOSP 16.1.7 以前のバグにより、エージェントを削除できなかったことを示すメッセージが表示される可能性があります。

```
Bad agent request: OVN agents cannot be deleted.
```

ボー **エージェント要求メッセージが表示され** ない場合は、次の手順に進みます。

バッド **エージェント要求メッセージが表示** された場合は、ネットワークエージェントの削除（バグの回避策）を参照し てください。回避策の手順が完了したら、ここに戻り、次のステップに進みます。

9. ノードの削除が完了したら、オーバークラウドスタックのステータスを確認します。

```
(overcloud)$ source ~/stackrc
(undercloud)$ openstack stack list
```

表16.2 結果

ステータス	説明
UPDATE_COMPLETE	コンピュートノードの削除が正常に完了しました。次のステップに進みます。

ステータス	説明
UPDATE_FAILED	コンピュートノードの削除に失敗しました。 コンピュートノードの削除に失敗する一般的な理由は、削除するノードの IPMI インターフェイスに到達できないことです。 削除に失敗した場合は、手動でプロセスを完了する必要があります。到達不能なコンピュートノードの削除が完了したら、コンピュートノード の削除を完了します。

10. インスタンス HA が有効な場合は、以下の操作を実行します。

    a. コンピュートノードの Pacemaker リソースをクリーンアップします。

    ```
    $ sudo pcs resource delete <compute_UUID>
    $ sudo cibadmin -o nodes --delete --xml-text '<node id="<compute_UUID>"/>'
    $ sudo cibadmin -o fencing-topology --delete --xml-text '<fencing-level target="
    <compute_UUID>"/>'
    $ sudo cibadmin -o status --delete --xml-text '<node_state id="<compute_UUID>"/>'
    $ sudo cibadmin -o status --delete-all --xml-text '<node id="<compute_UUID>"/>' --force
    ```

    b. ノードの STONITH デバイスを削除します。

    ```
    $ sudo pcs stonith delete <compute_UUID>
    ```

11. オーバークラウドで削除されたコンピュートノードを置き換えない場合は、ノード数を含む環境ファイルの **ComputeCount** パラメーターを減らします。このファイルは、通常 **node-info.yaml** という名前です。たとえば、ノードを 1 つ削除する場合、ノード数を 4 から 3 に減らします。

    ```
    parameter_defaults:
      ...
      ComputeCount: 3
    ```

    ノード数を減らすと、**openstack overcloud deploy** を実行しても director は新規ノードをプロビジョニングしません。

    オーバークラウドデプロイメントで削除されたコンピュートノードを置き換える場合は、Replacing a removed Compute node を参照してください。

16.3.1. 到達不能なコンピュートノードの削除の完了

openstack overcloud node delete コマンドが到達不能なノードのために失敗した場合は、オーバークラウドからのコンピュートノードの削除を手動で完了する必要があります。

前提条件

- コンピュートノードの削除または置換 手順を実行すると、ステータス **UPDATE_FAILED** が返されました。

手順

1. オーバークラウドスタックの UUID を特定します。

```
(undercloud)$ openstack stack list
```

2. 手動で削除するノードの UUID を特定します。

```
(undercloud)$ openstack baremetal node list
```

3. 削除するノードをメンテナンスモードに設定します。

```
(undercloud)$ openstack baremetal node maintenance set <UUID>
```

- **<UUID>** を、メンテナンスモードにするノードの UUID に置き換えます。

4. コンピュートサービスがその状態をベアメタルサービスと同期するのを待ちます。これには最大 4 分かかる場合があります。

5. source コマンドでオーバークラウド設定を読み込みます。

```
(undercloud)$ source ~/overcloudrc
```

6. ノードが新しいインスタンスをスケジュールできないように、オーバークラウド上の削除されたノードでコンピュートサービスが無効になっていることを確認します。

```
(overcloud)$ openstack compute service list
```

コンピュートサービスが無効になっていない場合は、無効にします。

```
(overcloud)$ openstack compute service set <hostname> nova-compute --disable
```

- **<hostname>** は、コンピュートノードのホスト名に置き換えます。

ヒント

--disable-reason オプションを使用して、サービスを無効にする理由についての簡単な説明を追加します。これは、コンピュートサービスを再デプロイする場合に役立ちます。

7. 削除したコンピュートノードから Compute サービスを削除します。

```
(overcloud)$ openstack compute service delete <service_id>
```

- &lt **;service_id&gt**; を、削除済みのノードで実行していた Compute サービスの ID に置き換えます。

8. 削除した Compute サービスを、Placement サービスのリソースプロバイダーから除外します。

```
(overcloud)$ openstack resource provider list
(overcloud)$ openstack resource provider delete <UUID>
```

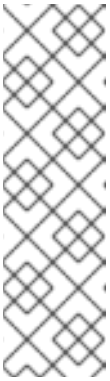
9. source コマンドでアンダークラウド設定を読み込みます。

```
(overcloud)$ source ~/stackrc
```

10. スタックからコンピュートノードを削除します。

```
(undercloud)$ openstack overcloud node delete --stack <overcloud> <node>
```

- **<overcloud>** は、オーバークラウドスタックの名前または UUID に置き換えてください。
- **<node>** を、削除するコンピュートノードのコンピュートサービスホスト名または UUID に置き換えます。



注記

ノードの電源がすでにオフになっている場合、このコマンドは **WARNING** メッセージを返します。

```
Ansible failed, check log at `/var/lib/mistral/overcloud/ansible.log`
WARNING: Scale-down configuration error. Manual cleanup of some
actions may be necessary. Continuing with node removal.
```

このメッセージは無視しても問題ありません。

11. オーバークラウドノードが削除されるまで待ちます。

12. source コマンドでオーバークラウド設定を読み込みます。

```
(undercloud)$ source ~/overcloudrc
```

13. 削除したノードのネットワークエージェントを削除します。

```
(overcloud)$ for AGENT in $(openstack network agent list \
  --host <scaled_down_node> -c ID -f value) ; \
  do openstack network agent delete $AGENT ; done
```

- &lt **;scaled_down_node&gt**; を削除したノードの名前に置き換えます。

14. コマンドの出力を確認します。RHOSP 16.1.7 以前のバグにより、エージェントを削除できなかったことを示すメッセージが表示される可能性があります。

```
Bad agent request: OVN agents cannot be deleted.
```

このメッセージが表示されない場合は、次の手順に進みます。

このメッセージが表示される場合は、ネットワークエージェントの削除：バグの回避策 の 手順を実行してください。回避策の手順が完了したら、ここに戻り、次のステップに進みます。

15. source コマンドでアンダークラウド設定を読み込みます。

```
(overcloud)$ source ~/stackrc
```

16. ノードの削除が完了したら、オーバークラウドスタックのステータスを確認します。

```
(undercloud)$ openstack stack list
```

表16.3 結果

ステータス	説明
UPDATE_COMPLETE	コンピュートノードの削除が正常に完了しました。次のステップに進みます。
UPDATE_FAILED	コンピュートノードの削除に失敗しました。 ノードがメンテナンスモードの際にコンピュートノードの削除に失敗した場合には、問題がハードウェアにある可能性があります。ハードウェアを確認します。

17. インスタンス HA が有効な場合は、以下の操作を実行します。

    a. ノードの Pacemaker リソースをクリーンアップします。

```
$ sudo pcs resource delete <scaled_down_node>
$ sudo cibadmin -o nodes --delete --xml-text '<node id="<scaled_down_node>"/>'
$ sudo cibadmin -o fencing-topology --delete --xml-text '<fencing-level target="
<scaled_down_node>"/>'
$ sudo cibadmin -o status --delete --xml-text '<node_state id="<scaled_down_node>"/>'
$ sudo cibadmin -o status --delete-all --xml-text '<node id="<scaled_down_node>"/>' --
force
```

    b. ノードの STONITH デバイスを削除します。

```
$ sudo pcs stonith delete <device-name>
```

18. オーバークラウドで削除されたコンピュートノードを置き換えない場合は、ノード数を含む環境ファイルの **ComputeCount** パラメーターを減らします。このファイルは、通常 **node-info.yaml** という名前です。たとえば、ノードを 1 つ削除する場合、ノード数を 4 から 3 に減らします。

```
parameter_defaults:
  ...
  ComputeCount: 3
  ...
```

ノード数を減らすと、**openstack overcloud deploy** を実行しても director は新規ノードをプロビジョニングしません。

オーバークラウドデプロイメントで削除されたコンピュートノードを置き換える場合は、Replacing a removed Compute node を参照してください。

16.3.2. ネットワークエージェントの削除：バグの回避策

コンピュートノードを削除したら、関連付けられたネットワークエージェントを削除する必要があります。デプロイメントで RHOSP 16.1.7 以前の場合、バグにより、ネットワークエージェントを予想通りに削除できなくなります。BZ1788336-ovn-controllers are listed as agents but cannot be removed を参照してください。

このバグにより、指示に従ってエージェントを削除しようとすると、Networking サービスに以下のエラーメッセージが表示されます。

```
Bad agent request: OVN agents cannot be deleted.
```

そのエラーメッセージが表示された場合は、以下の手順を実行してエージェントを削除します。

前提条件

- 以下のエラーメッセージで示されるように、コンピュートノードの削除に失敗した後にネットワークエージェントを削除しようとしました。

  ```
  Bad agent request: OVN agents cannot be deleted.
  ```

手順

1. オーバークラウドノードを一覧表示します。

   ```
   (undercloud)$ openstack server list
   ```

2. root 権限を持つユーザーとしてコントローラーノードにログインします。

   ```
   $ ssh heat-admin@controller-0.ctlplane
   ```

3. ovn_controller コンテナーの ovn-sbctl コマンドへのアクセスを簡素化するコマンドエイリアスを設定していない場合は、コマンドエイリアスを設定します。詳細については、OVN トラブルシューティングコマンドのエイリアスの作成 を参照してください。

4. **ovn-controller.log** ファイルから IP アドレスを取得します。

   ```
   $ sudo less /var/log/containers/openvswitch/ovn-controller.log
   ```

   ovn-controller.log が空の場合は、**ovn-controller.log.1** を試してみてください。

5. IP アドレスが正しいことを確認します。

   ```
   $ ovn-sbctl list encap |grep -a3 <IP_address_from_ovn-controller.log>
   ```

   <IP_address_from_ovn-controller.log> をコントローラーログファイルの IP アドレスに置き換えます。

6. IP アドレスを含むシャーシを削除します。

   ```
   $ ovn-sbctl chassis-del <chassis-name>
   ```

<chassis-id> を、前の手順の **ovn-sbctl list encap** コマンドの出力からの **chassis_name** 値に置き換えます。

7. Chassis_Private テーブルをチェックして、シャーシが削除されたことを確認します。

```
$ ovn-sbctl find Chassis_private chassis="[]"
```

8. chasis が表示される場合は、以下のコマンドでそれぞれを削除します。

```
$ ovn-sbctl destroy Chassis_Private <listed_name>
```

&lt **;listed_name&gt**; を、削除するシャーシの名前に置き換えます。

9. 手順に戻り、コンピュートノードの削除を完了します。

16.3.3. 削除されたコンピュートノードの交換

オーバークラウドデプロイメントで削除されたコンピュートノードを置き換えるには、新しいコンピュートノードを登録して検査するか、削除したコンピュートノードを再度追加します。また、ノードをプロビジョニングするようにオーバークラウドを設定する必要があります。

手順

1. オプション: 削除されたコンピュートノードのインデックスを再利用するには、コンピュートノードが削除されたときに拒否リストを置き換えるロールの **RemovalPoliciesMode** パラメーターと **RemovalPolicies** パラメーターを設定します。

```
parameter_defaults:
  <RoleName>RemovalPoliciesMode: update
  <RoleName>RemovalPolicies: [{'resource_list': []}]
```

2. 削除されたコンピュートノードを置き換えます。

   - 新しいコンピュートノードを追加するには、新しいノードを登録、検査、およびタグ付けして、プロビジョニングの準備をします。詳細については、Configuring a basic overcloud を参照してください。

   - 手動で削除したコンピュートノードを再度追加するには、ノードをメンテナンスモードから削除します。

```
$ openstack baremetal node maintenance unset <node_uuid>
```

3. 既存のオーバークラウドのデプロイに使用した **openstack overcloud deploy** コマンドを再実行します。

4. デプロイメントプロセスが完了するまで待ちます。

5. ディレクターが新しいコンピュートノードを正常に登録したことを確認します。

```
$ openstack baremetal node list
```

6. 手順 1 を実行して、**update** 用のロールの **RemovalPoliciesMode** を設定した場合は、ロールの **RemovalPoliciesMode** をデフォルト値の **append** にリセットして、コンピュートノードが削除されたときにコンピュートノードインデックスを現在の拒否リストに追加する必要がありま

す。

```
parameter_defaults:
  <RoleName>RemovalPoliciesMode: append
```

7. 既存のオーバークラウドのデプロイに使用した **openstack overcloud deploy** コマンドを再実行します。

16.4. 予測可能な IP アドレスと HOSTNAMEMAP を使用するノードを置き換えるときにホスト名を保持する

予測可能な IP アドレスを使用するようにオーバークラウドを設定し、ヒートベースのホスト名を事前プロビジョニングされたノードのホスト名にマップするように **HostNameMap** を設定した場合は、新しい置換ノードインデックスを IP アドレスとホスト名にマップするようにオーバークラウドを設定する必要があります。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. 置き換えるリソースの **physical_resource_id** と **removed_rsrc_list** を取得します。

```
$ openstack stack resource show <stack> <role>
```

- **<stack>** を、リソースが属するスタックの名前 (**overcloud** など) に置き換えます。
- **<role>** を、ノードを置き換えるロールの名前 (**Compute** など) に置き換えます。
出力例:

```
+-----------------------+--------------------------------------------------------+
| Field                 | Value                                                  |
+-----------------------+--------------------------------------------------------+
| attributes            | {u'attributes': None, u'refs': None, u'refs_map': None,   |
|                       | u'removed_rsrc_list': [u'2', u'3']}          | 1
| creation_time         | 2017-09-05T09:10:42Z                                   |
| description           |                                                        |
| links                 | [{u'href': u'http://192.168.24.1:8004/v1/bd9e6da805594de9 |
|                       | 8d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e-          |
|                       | 4d61-a5d8-9f964915d503/resources/Compute', u'rel':        |
|                       | u'self'}, {u'href': u'http://192.168.24.1:8004/v1/bd9e6da |
|                       | 805594de98d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e- |
|                       | 4d61-a5d8-9f964915d503', u'rel': u'stack'}, {u'href': u'h |
|                       | ttp://192.168.24.1:8004/v1/bd9e6da805594de98d4a1d3a3ee874 |
|                       | dd/stacks/overcloud-Compute-zkjccox63svg/7632fb0b-        |
|                       | 80b1-42b3-9ea7-6114c89adc29', u'rel': u'nested'}]         |
| logical_resource_id   | Compute                                                |
| physical_resource_id  | 7632fb0b-80b1-42b3-9ea7-6114c89adc29                   |
| required_by           | [u'AllNodesDeploySteps',                               |
|                       | u'ComputeAllNodesValidationDeployment',                |
```

```
|                       | u'AllNodesExtraConfig', u'ComputeIpListMap',              |
|                       | u'ComputeHostsDeployment', u'UpdateWorkflow',             |
|                       | u'ComputeSshKnownHostsDeployment', u'hostsConfig',        |
|                       | u'SshKnownHostsConfig', u'ComputeAllNodesDeployment']     |
| resource_name         | Compute                                                   |
| resource_status       | CREATE_COMPLETE                                           |
| resource_status_reason | state changed                                            |
| resource_type         | OS::Heat::ResourceGroup                                   |
| updated_time          | 2017-09-05T09:10:42Z                                      |
+-----------------------+-----------------------------------------------------------+
```

1. **removed_rsrc_list** は、リソースからすでに削除されているノードのインデックスをリスト表示します。

4. **resource_name** を取得して、このリソースのノードに heat が適用した最大インデックスを決定します。

```
$ openstack stack resource list <physical_resource_id>
```

   - **<physical_resource_id>** を手順 2 で取得した ID に置き換えます。

5. **resource_name** と **removed_rsrc_list** を使用して、heat が新しいノードに適用する次のインデックスを決定します。

   - **removed_rsrc_list** が空の場合、次のインデックスは (current_maximum_index) + 1 になります。
   - **removed_rsrc_list** に値 (current_maximum_index) + 1 が含まれている場合、次のインデックスは次に使用可能なインデックスになります。

6. 置換用のベアメタルノードの ID を取得します。

```
$ openstack baremetal node list
```

7. 置換ノードの機能を新しいインデックスで更新します。

```
$ openstack baremetal node set --property capabilities='node:<role>-
<index>,boot_option:local' <node>
```

   - **<role>** を、ノードを置き換えるロールの名前 (**compute** など) に置き換えます。
   - **<index>** を手順 5 で計算したインデックスに置き換えます。
   - **<node>** をベアメタルノードの ID に置き換えてください。

   コンピュートスケジューラーは、ノード機能を使用して、デプロイメント時にノードを照合します。

8. **HostnameMap** 設定にインデックスを追加して、新しいノードにホスト名を割り当てます。次に例を示します。

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

```
    'capabilities:node': 'compute-%index%'
  HostnameMap:
    overcloud-controller-0: overcloud-controller-prod-123-0
    overcloud-controller-1: overcloud-controller-prod-456-0 (1)
    overcloud-controller-2: overcloud-controller-prod-789-0
    overcloud-controller-3: overcloud-controller-prod-456-0 (2)
    overcloud-compute-0: overcloud-compute-prod-abc-0
    overcloud-compute-3: overcloud-compute-prod-abc-3 (3)
    overcloud-compute-8: overcloud-compute-prod-abc-3 (4)
    ....
```

1. 削除して新しいノードに置き換えるノード。
2. 新しいノード。
3. 削除して新しいノードに置き換えるノード。
4. 新しいノード。

注記

削除されたノードのマッピングを **HostnameMap** から削除しないでください。

9. 置換ノードの IP アドレスを、ネットワーク IP アドレスマッピングファイル **ips-from-pool-all.yaml** の各ネットワーク IP アドレスリストの最後に追加します。次の例では、新しいインデックス **overcloud-controller-3** の IP アドレスが、各 **ControllerIPs** ネットワークの IP アドレスリストの最後に追加され、**overcloud-controller-1** を置き換えるため、**overcloud-controller-1** と同じ IP アドレスが割り当てられます。新しいインデックスの IP アドレス **overcloud-compute-8** も、各 **ComputeIPs** ネットワークの IP アドレスリストの最後に追加され、置き換えられるインデックス **overcloud-compute-3** と同じ IP アドレスが割り当てられます。

```
parameter_defaults:
  ControllerIPs:
    ...
    internal_api:
      - 192.168.1.10 (1)
      - 192.168.1.11 (2)
      - 192.168.1.12 (3)
      - 192.168.1.11 (4)
    ...
    storage:
      - 192.168.2.10
      - 192.168.2.11
      - 192.168.2.12
      - 192.168.2.11
    ...

  ComputeIPs:
    ...
    internal_api:
      - 172.17.0.10 (5)
      - 172.17.0.11 (6)
```

```
    - 172.17.0.11 7
...
storage:
  - 172.17.0.10
  - 172.17.0.11
  - 172.17.0.11
...
```

1. インデックス 0 に割り当てられた IP アドレス、ホスト名 **overcloud-controller-prod-123-0**。
2. インデックス 1 に割り当てられた IP アドレス、ホスト名 **overcloud-controller-prod-456-0**。このノードはインデックス 3 に置き換えられます。**このエントリーは削除しないでください。**
3. インデックス 2 に割り当てられた IP アドレス、ホスト名 **overcloud-controller-prod-789-0**。
4. インデックス 3 に割り当てられた IP アドレス、ホスト名 **overcloud-controller-prod-456-0**。これは、インデックス 1 を置き換える新しいノードです。
5. インデックス 0 に割り当てられた IP アドレス、ホスト名 **overcloud-compute-0**。
6. インデックス 1 に割り当てられた IP アドレス、ホスト名 **overcloud-compute-3**。このノードはインデックス 2 に置き換えられます。**このエントリーは削除しないでください。**
7. インデックス 2 に割り当てられた IP アドレス、ホスト名 **overcloud-compute-8**。これは、インデックス 1 を置き換える新しいノードです。

16.5. CEPH STORAGE ノードの置き換え

director を使用して、director で作成したクラスター内の Ceph Storage ノードを置き換えることができます。詳しい情報は、コンテナー化された Red Hat Ceph を持つオーバークラウドのデプロイ を参照してください。

16.6. オブジェクトストレージノードの置き換え

本項の手順で、クラスターの整合性に影響を与えずにオブジェクトストレージノードを置き換える方法を説明します。以下の例では、3 台のノードで設定されるオブジェクトストレージクラスターで、**overcloud-objectstorage-1** ノードを置き換えます。この手順の目的は、ノードを 1 台追加し、その後 **overcloud-objectstorage-1** ノードを削除することです。**overcloud-objectstorage-1** ノードが新しいノードに置き換えられます。

手順

1. **ObjectStorageCount** パラメーターを使用してオブジェクトストレージ数を増やします。このパラメーターは、通常ノード数を指定する環境ファイルの **node-info.yaml** に含まれています。

```
parameter_defaults:
  ObjectStorageCount: 4
```

ObjectStorageCount パラメーターで、環境内のオブジェクトストレージノードの数を定義します。今回の例では、オブジェクトストレージノードの数を **3** から **4** にスケーリングします。

2. 更新した **ObjectStorageCount** パラメーターを使用して、デプロイメントコマンドを実行します。

```
$ source ~/stackrc
$ openstack overcloud deploy --templates -e node-info.yaml <environment_files>
```

デプロイメントコマンドの実行が完了すると、オーバークラウドには追加のオブジェクトストレージノードが含まれるようになります。

3. 新しいノードにデータを複製します。ノードを削除する前に (ここでは **overcloud-objectstorage-1**)、複製のパスが新規ノードで完了するのを待ちます。**/var/log/swift/swift.log** ファイルで複製パスの進捗を確認することができます。パスが完了すると、Object Storage サービスは以下の例のようなエントリーをログに残します。

```
Mar 29 08:49:05 localhost *object-server: Object replication complete.*
Mar 29 08:49:11 localhost *container-server: Replication run OVER*
Mar 29 08:49:13 localhost *account-server: Replication run OVER*
```

4. リングから不要になったノードを削除するには、**ObjectStorageCount** パラメーターの数を減らして不要になったノードを削除します。この例では、**ObjectStorageCount** パラメーターを **3** に減らします。

```
parameter_defaults:
  ObjectStorageCount: 3
```

5. 新規環境ファイル (**remove-object-node.yaml**) を作成します。このファイルでオブジェクトストレージノードを特定し、そのノードを削除します。以下の内容では **overcloud-objectstorage-1** の削除を指定します。

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

6. デプロイメントコマンドに **node-info.yaml** ファイルと **remove-object-node.yaml** ファイルの両方を含めます。

```
$ openstack overcloud deploy --templates -e node-info.yaml <environment_files> -e remove-object-node.yaml
```

director は、オーバークラウドからオブジェクトストレージノードを削除して、オーバークラウド上の残りのノードを更新し、ノードの削除に対応します。

重要

オーバークラウドの初回作成時に使用した環境ファイルおよびオプションをすべて含めます。これには、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が含まれます。

16.7. スキップデプロイ識別子の使用

スタック更新操作中に、puppet はデフォルトで、すべてのマニフェストを再適用します。その結果、必要のない操作に時間がかかることがあります。

デフォルトの操作をオーバーライドするには、**skip-deploy-identifier** オプションを使用します。

```
openstack overcloud deploy --skip-deploy-identifier
```

デプロイメントコマンドで **DeployIdentifier** パラメーターの一意の ID を生成するのを希望しない場合は、このオプションを使用します。ソフトウェア設定のデプロイメントステップは、実際に設定が変更された場合にしか実行されません。このオプションの使用には注意が必要です。特定のロールをスケールアウトする時など、ソフトウェア設定の実行が明らかに不要な場合にしか使用しないでください。

注記

puppet マニフェストまたは hierdata に変更がある場合、**--skip-deploy-identifier** が指定されている場合でも、puppet はすべてのマニフェストを再適用します。

16.8. ノードのブロックリスト

オーバークラウドノードがデプロイメントの更新を受け取らないように除外することができます。これは、新規ノードをスケーリングする場合に、既存のノードがコア heat テンプレートコレクションから更新されたパラメーターセットやリソースを受け取らないように除外するのに役立ちます。つまり、ブロックリストに登録されているノードは、スタック操作の影響を受けなくなります。

ブロックリストを作成するには、環境ファイルの **DeploymentServerBlacklist** パラメーターを使います。

ブロックリストの設定

DeploymentServerBlacklist パラメーターは、サーバー名のリストです。新たな環境ファイルを作成するか、既存のカスタム環境ファイルにパラメーター値を追加して、ファイルをデプロイメントコマンドに渡します。

```
parameter_defaults:
  DeploymentServerBlacklist:
    - overcloud-compute-0
    - overcloud-compute-1
    - overcloud-compute-2
```

注記

パラメーター値のサーバー名には、実際のサーバーホスト名ではなく、OpenStack Orchestation (heat) で定義されている名前を使用します。

openstack overcloud deploy コマンドで、この環境ファイルを指定します。

```
$ source ~/stackrc
$ openstack overcloud deploy --templates \
  -e server-blocklist.yaml \
  -e [...]
```

heat はリスト内のサーバーをすべてブロックリストし、heat デプロイメントの更新を受け取らないようにします。スタック操作が完了した後には、ブロックリストに登録されたサーバーは以前の状態のままとなります。操作中に **os-collect-config** エージェントの電源をオフにしたり、停止したりすることもできます。

警告

- ノードをブロックリストに追加する場合は、注意が必要です。ブロックリストを有効にした状態で要求された変更を適用する方法を十分に理解していない限り、ブラックリストは使用しないでください。ブロックリスト機能を使用すると、スタックがハングアップしたり、オーバークラウドが誤って設定されたりする場合があります。たとえば、クラスター設定の変更が Pacemaker クラスターの全メンバーに適用される場合、この変更の間に Pacemaker クラスターのメンバーをブロックリストに登録すると、クラスターが機能しなくなることがあります。

- 更新またはアップグレードの操作中にブロックリストを使わないでください。これらの操作には、特定のサーバーに対する変更を分離するための独自の方法があります。

- サーバーをブロックリストに追加すると、そのサーバーをブロックリストから削除するまでは、それらのノードにはさらなる変更は適用されません。これには、更新、アップグレード、スケールアップ、スケールダウン、およびノードの置き換えが含まれます。たとえば、新規コンピュートノードを使用してオーバークラウドをスケールアウトする際に既存のコンピュートノードをブロックリストに登録すると、ブロックリストに登録したノードには **/etc/hosts** および **/etc/ssh/ssh_known_hosts** に加えられた情報が反映されません。これにより、移行先ホストによっては、ライブマイグレーションが失敗する可能性があります。コンピュートノードのブロックリスト登録が解除される次回のオーバークラウドデプロイメント時に、これらのノードは **/etc/hosts** および **/etc/ssh/ssh_known_hosts** に加えられた情報で更新されます。**/etc/hosts** ファイルおよび **/etc/ssh/ssh_known_hosts** ファイルは手動で変更しないでください。**/etc/hosts** ファイルおよび **/etc/ssh/ssh_known_hosts** ファイルを変更するには、Clearing the Blocklist セクションで説明するように、オーバークラウドのデプロイコマンドを実行します。

ブロックリストの消去

その後のスタック操作のためにブロックリストをクリアするには、**DeploymentServerBlacklist** を編集して空の配列を使用します。

```
parameter_defaults:
  DeploymentServerBlacklist: []
```

警告

DeploymentServerBlacklist パラメーターを削除しないでください。パラメーターを削除すると、オーバークラウドのデプロイメントには、前回保存された値が使用されます。

第17章 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ます。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。

以下のシナリオの手順を実施して、コントローラーノードを置き換えます。コントローラーノードを置き換えるプロセスでは、**openstack overcloud deploy** コマンドを実行し、コントローラーノードを置き換えるリクエストでオーバークラウドを更新します。

重要

以下の手順は、高可用性環境にのみ適用されます。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。

17.1. コントローラー置き換えの準備

オーバークラウドのコントローラーノードを置き換える前に、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。このチェックをしておくと、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。

手順

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
(undercloud)$ openstack stack list --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. データベースクライアントツールをインストールします。

```
(undercloud)$ sudo dnf -y install mariadb
```

3. root ユーザーのデータベースへのアクセス権限を設定します。

```
(undercloud)$ sudo cp /var/lib/config-data/puppet-generated/mysql/root/.my.cnf /root/.
```

4. 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。

```
(undercloud)$ mkdir /home/stack/backup
(undercloud)$ sudo mysqldump --all-databases --quick --single-transaction | gzip >
/home/stack/backup/dump_db_undercloud.sql.gz
```

5. アンダークラウドに、新規ノードプロビジョニング時の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ことを確認します。

```
(undercloud)$ df -h
```

6. 新規コントローラーノード用に IP アドレスを再利用する場合は、古いコントローラーが使用したポートを削除するようにしてください。

```
(undercloud)$ openstack port delete <port>
```

7. 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 のステータスを表示します。

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

この出力には、既存のノードで動作中のサービスおよび障害の発生しているノードで停止しているサービスがすべて表示されます。

8. オーバークラウド 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。以下の例では、コントローラーノードの IP アドレスは、それぞれ 192.168.0.47 と 192.168.0.46 です。

```
(undercloud)$ for i in 192.168.0.46 192.168.0.47 ; do echo "*** $i ***" ; ssh heat-admin@$i "sudo podman exec \$(sudo podman ps --filter name=galera-bundle -q) mysql -e \"SHOW STATUS LIKE 'wsrep_local_state_comment'; SHOW STATUS LIKE 'wsrep_cluster_size';\""; done
```

9. 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、以下のコマンドで RabbitMQ のステータスを表示します。

```
(undercloud)$ ssh heat-admin@192.168.0.47 "sudo podman exec \$(sudo podman ps -f name=rabbitmq-bundle -q) 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。

10. フェンシングが有効な場合は、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のステータスを確認します。

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

フェンシングを無効にするには、以下のコマンドを実行します。

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

11. director ノードで Compute サービスがアクティブであることを確認します。

```
(undercloud)$ openstack 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。

12. アンダークラウドコンテナーがすべて実行中であることを確認します。

```
(undercloud)$ sudo podman ps
```

13. 障害が発生したコントローラーノードで実行されているすべての **nova_*** コンテナーを停止します。

```
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_api.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_api_cron.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_compute.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_conductor.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_metadata.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_placement.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_scheduler.service
```

14. オプション: virt ドライバーとして Bare Metal サービス (ironic) を使用している場合は、削除するコントローラーに **instances.host** が設定されているベアメタルインスタンスのセルデータベースのサービスエントリーを手動で更新する必要があります。レッドハットサポートにお問い合わせください。

注記

Bare Metal サービス (ironic) を virt ドライバーとして使用する場合のセルデータベースのこの手動更新は、BZ2017980 が完了するまで、ノードのリバランスを確保するための一時的な回避策です。

17.2. CEPH MONITOR デーモンの削除

コントローラーノードが Ceph monitor サービスを実行している場合には、以下のステップを完了して、**ceph-mon** デーモンを削除してください。

注記

クラスターに新しいコントローラーノードを追加すると、新しい Ceph monitor デーモンも自動的に追加されます。

手順

1. 置き換えるコントローラーノードに接続して、root ユーザーになります。

```
# ssh heat-admin@192.168.0.47
# sudo su -
```

注記

コントローラーノードにアクセスすることができない場合、ステップ 1 と 2 をスキップして、稼働している任意のコントローラーノードでステップ 3 から手順を続行してください。

2. monitor を停止します。

```
# systemctl stop ceph-mon@<monitor_hostname>
```

以下に例を示します。

```
# systemctl stop ceph-mon@overcloud-controller-1
```

3. 置き換えるコントローラーノードとの接続を終了します。

4. 既存のコントローラーノードのいずれかに接続します。

```
# ssh heat-admin@192.168.0.46
# sudo su -
```

5. クラスターから monitor を削除します。

```
# sudo podman exec -it ceph-mon-controller-0 ceph mon remove overcloud-controller-1
```

6. すべてのコントローラーノード上で、**/etc/ceph/ceph.conf** から v1 および v2 monitor のエントリーを削除します。たとえば、controller-1 を削除する場合には、controller-1 の IP アドレスとホスト名を削除します。
編集前:

```
mon host = [v2:172.18.0.21:3300,v1:172.18.0.21:6789],
[v2:172.18.0.22:3300,v1:172.18.0.22:6789],[v2:172.18.0.24:3300,v1:172.18.0.24:6789]
mon initial members = overcloud-controller-2,overcloud-controller-1,overcloud-controller-0
```

変更後:

```
mon host = [v2:172.18.0.21:3300,v1:172.18.0.21:6789],
[v2:172.18.0.24:3300,v1:172.18.0.24:6789]
mon initial members = overcloud-controller-2,overcloud-controller-0
```

注記

置き換え用のコントローラーノードを追加すると、director によって該当するオーバークラウドノード上の **ceph.conf** ファイルが更新されます。通常は、director がこの設定ファイルを管理するだけで、手動でファイルを編集する必要はありません。ただし、新規ノードが追加する前に他のノードが再起動してしまった場合に一貫性を保つために、手動でファイルを編集することができます。

7. (オプション) monitor データをアーカイブし、アーカイブを別のサーバーに保存します。

```
# mv /var/lib/ceph/mon/<cluster>-<daemon_id> /var/lib/ceph/mon/removed-<cluster>-
<daemon_id>
```

17.3. コントローラーノードを置き換えるためのクラスター準備

古いノードを置き換える前に、Pacemaker がノード上で実行されていないことを確認してからそのノードを Pacemaker クラスターから削除する必要があります。

手順

1. コントローラーノードの IP アドレスリストを表示するには、以下のコマンドを実行します。

```
(undercloud) $ openstack server list -c Name -c Networks
+------------------------+-----------------------+
| Name                   | Networks              |
+------------------------+-----------------------+
| overcloud-compute-0    | ctlplane=192.168.0.44 |
| overcloud-controller-0 | ctlplane=192.168.0.47 |
| overcloud-controller-1 | ctlplane=192.168.0.45 |
| overcloud-controller-2 | ctlplane=192.168.0.46 |
+------------------------+-----------------------+
```

2. まだ古いノードにアクセスできる場合は、残りのノードのいずれかにログインし、古いノード上の Pacemaker を停止します。以下の例では、overcloud-controller-1 上の Pacemaker を停止しています。

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs status | grep -w Online | grep -w overcloud-controller-1"
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs cluster stop overcloud-controller-1"
```

注記

古いノードが物理的に利用できない、または停止している場合には、そのノードでは Pacemaker はすでに停止しているので、この操作を実施する必要はありません。

3. 古いノード上の Pacemaker を停止したら、Pacemaker クラスターから古いノードを削除します。以下に示すコマンドの例では、**overcloud-controller-0** にログインし、**overcloud-controller-1** を削除しています。

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs cluster node remove overcloud-controller-1"
```

置き換えるノードにアクセスすることができない場合には (ハードウェア障害などの理由により)、**pcs** コマンドを実行する際にさらに **--skip-offline** および **--force** オプションを指定して、ノードをクラスターから強制的に削除します。

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs cluster node remove overcloud-controller-1 --skip-offline --force"
```

4. Pacemaker クラスターから古いノードを削除したら、Pacemaker の既知のホストリストからノードを削除します。

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs host deauth overcloud-controller-1"
```

ノードにアクセス可能かどうかにかかわらず、このコマンドを実行することができます。

5. 置き換え後に新しいコントローラーノードで正しい STONITH フェンシングデバイスが使用されるようにするには、以下のコマンドを入力してノードから古いデバイスを削除します。

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs stonith delete <stonith_resource_name>"
```

- **<stonith_resource_name>** を、古いノードに対応する STONITH リソースの名前に置き換えます。リソース名には **<resource_agent>-<host_mac>** 形式が使用されます。リソース

エージェントおよびホストの MAC アドレスは、**fencing.yaml** ファイルの **FencingConfig** セクションに記載されています。

6. オーバークラウドデータベースは、置き換え手順の実行中に稼働し続ける必要があります。この手順の実行中に Pacemaker が Galera を停止しないようにするには、実行中のコントローラーノードを選択して、そのコントローラーノードの IP アドレスを使用して、アンダークラウドで以下のコマンドを実行します。

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs resource unmanage galera-bundle"
```

17.4. コントローラーノードの置き換え

コントローラーノードを置き換えるには、置き換えるノードのインデックスを特定します。

- ノードが仮想ノードの場合には、障害の発生したディスクが含まれるノードを特定し、バックアップからそのディスクをリストアします。障害の発生したサーバー上での PXE ブートに使用する NIC の MAC アドレスは、ディスク置き換え後も同じアドレスにしてください。
- ノードがベアメタルノードの場合には、ディスクを置き換え、オーバークラウド設定で新しいディスクを準備し、新しいハードウェア上でノードのイントロスペクションを実施します。
- ノードがフェンシングの設定された高可用性クラスターの一部である場合、Galera ノードを個別に復元しなければならない場合があります。詳しくは、アーティクル How Galera works and how to rescue Galera clusters in the context of Red Hat OpenStack Platform を参照してください。

overcloud-controller-1 ノードを **overcloud-controller-3** ノードに置き換えるには、以下の手順例を実施します。**overcloud-controller-3** ノードの ID は **75b25e9a-948d-424a-9b3b-f0ef70a6eacf** です。

重要

ノードを既存のベアメタルノードに置き換えるには、director がノードを自動的に再プロビジョニングしないように、削除するノードをメンテナンスモードに切り替えます。

手順

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. **overcloud-controller-1** ノードのインデックスを特定します。

```
$ INSTANCE=$(openstack server list --name overcloud-controller-1 -f value -c ID)
```

3. インスタンスに関連付けられたベアメタルノードを特定します。

```
$ NODE=$(openstack baremetal node list -f csv --quote minimal | grep $INSTANCE | cut -f1 -d,)
```

4. ノードをメンテナンスモードに切り替えます。

```
$ openstack baremetal node maintenance set $NODE
```

5. コントローラーノードが仮想ノードの場合には、コントローラーホストで以下のコマンドを実行し、障害の発生した仮想ディスクをバックアップからの仮想ディスクに置き換えます。

```
$ cp <VIRTUAL_DISK_BACKUP> /var/lib/libvirt/images/<VIRTUAL_DISK>
```

- **<VIRTUAL_DISK_BACKUP>** を障害の発生した仮想ディスクのバックアップへのパスに、**<VIRTUAL_DISK>** を置き換える仮想ディスクの名前にそれぞれ置き換えます。
削除するノードのバックアップがない場合には、新しい仮想ノードを使用する必要があります。

  コントローラーノードがベアメタルノードの場合には、以下の手順を実施してディスクを新しいベアメタルディスクに置き換えます。

  a. 物理ハードドライブまたはソリッドステートドライブを置き換えます。

  b. 障害が発生したノードと同じ設定のノードを準備します。

6. 関連付けられていないノードのリストを表示し、新規ノードの ID を特定します。

```
$ openstack baremetal node list --unassociated
```

7. 新規ノードを **control** プロファイルにタグ付けします。

```
(undercloud) $ openstack baremetal node set --property
capabilities='profile:control,boot_option:local' 75b25e9a-948d-424a-9b3b-f0ef70a6eacf
```

17.5. ブートストラップコントローラーノードの置き換え

ブートストラップ操作に使用するコントローラーノードを置き換え、ノード名を維持するには、以下の手順を実施して、置き換えプロセス後にブートストラップコントローラーノードの名前を設定します。

手順

1. 以下のコマンドを実行して、ブートストラップコントローラーノードの名前を確認します。

```
ssh tripleo-admin@CONTROLLER_IP "sudo hiera -c /etc/puppet/hiera.yaml
pacemaker_short_bootstrap_node_name"
```

- **CONTROLLER_IP** を、任意のアクティブなコントローラーノードの IP アドレスに置き換えます。

2. 環境ファイルに **ExtraConfig** セクションが含まれているかどうかを確認します。**ExtraConfig** パラメーターが存在しない場合は、以下の環境ファイル **~/templates/bootstrap-controller.yaml** を作成し、以下の内容を追加します。

```
parameter_defaults:
  ExtraConfig:
    pacemaker_short_bootstrap_node_name: NODE_NAME
    mysql_short_bootstrap_node_name: NODE_NAME
```

- **NODE_NAME** を、置き換えプロセス後にブートストラップ操作に使用する既存のコントローラーノードの名前に置き換えます。

お使いの環境ファイルに **ExtraConfig** パラメーターがすでに含まれている場合は、**pacemaker_short_bootstrap_node_name** および **mysql_short_bootstrap_node_name** パラメーターを設定する行だけを追加します。

3. コントローラーノード置き換えのトリガーとなる手順に従って、**overcloud deploy command** に環境ファイルを追加します。詳細は、Triggering the Controller node replacement を参照してください。

ブートストラップコントローラーノード置き換えのトラブルシューティングに関する情報は、アーティクル Replacement of the first controller node fails at step 1 if the same hostname is used for a new node を参照してください。

17.6. 予測可能な IP アドレスと HOSTNAMEMAP を使用するノードを置き換えるときにホスト名を保持する

予測可能な IP アドレスを使用するようにオーバークラウドを設定し、ヒートベースのホスト名を事前プロビジョニングされたノードのホスト名にマップするように **HostNameMap** を設定した場合は、新しい置換ノードインデックスを IP アドレスとホスト名にマップするようにオーバークラウドを設定する必要があります。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. 置き換えるリソースの **physical_resource_id** と **removed_rsrc_list** を取得します。

```
$ openstack stack resource show <stack> <role>
```

- **<stack>** を、リソースが属するスタックの名前 (**overcloud** など) に置き換えます。
- **<role>** を、ノードを置き換えるロールの名前 (**Compute** など) に置き換えます。
出力例:

```
+------------------------+--------------------------------------------------------------+
| Field                  | Value                                                        |
+------------------------+--------------------------------------------------------------+
| attributes             | {u'attributes': None, u'refs': None, u'refs_map': None,      |
|                        | u'removed_rsrc_list': [u'2', u'3']}                          | 1
| creation_time          | 2017-09-05T09:10:42Z                                         |
| description            |                                                              |
| links                  | [{u'href': u'http://192.168.24.1:8004/v1/bd9e6da805594de9     |
|                        | 8d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e-             |
|                        | 4d61-a5d8-9f964915d503/resources/Compute', u'rel':           |
|                        | u'self'}, {u'href': u'http://192.168.24.1:8004/v1/bd9e6da     |
|                        | 805594de98d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e-    |
|                        | 4d61-a5d8-9f964915d503', u'rel': u'stack'}, {u'href': u'h     |
|                        | ttp://192.168.24.1:8004/v1/bd9e6da805594de98d4a1d3a3ee874     |
|                        | dd/stacks/overcloud-Compute-zkjccox63svg/7632fb0b-           |
|                        | 80b1-42b3-9ea7-6114c89adc29', u'rel': u'nested'}]            |
| logical_resource_id    | Compute                                                      |
```

```
| physical_resource_id   | 7632fb0b-80b1-42b3-9ea7-6114c89adc29                    |
| required_by            | [u'AllNodesDeploySteps',                                |
|                        | u'ComputeAllNodesValidationDeployment',                 |
|                        | u'AllNodesExtraConfig', u'ComputeIpListMap',            |
|                        | u'ComputeHostsDeployment', u'UpdateWorkflow',           |
|                        | u'ComputeSshKnownHostsDeployment', u'hostsConfig',      |
|                        | u'SshKnownHostsConfig', u'ComputeAllNodesDeployment']   |
| resource_name          | Compute                                                 |
| resource_status        | CREATE_COMPLETE                                         |
| resource_status_reason | state changed                                           |
| resource_type          | OS::Heat::ResourceGroup                                 |
| updated_time           | 2017-09-05T09:10:42Z                                    |
+------------------------+---------------------------------------------------------+
```

1 **removed_rsrc_list** は、リソースからすでに削除されているノードのインデックスをリスト表示します。

4. **resource_name** を取得して、このリソースのノードに heat が適用した最大インデックスを決定します。

```
$ openstack stack resource list <physical_resource_id>
```

- **<physical_resource_id>** を手順 2 で取得した ID に置き換えます。

5. **resource_name** と **removed_rsrc_list** を使用して、heat が新しいノードに適用する次のインデックスを決定します。

- **removed_rsrc_list** が空の場合、次のインデックスは (current_maximum_index) + 1 になります。
- **removed_rsrc_list** に値 (current_maximum_index) + 1 が含まれている場合、次のインデックスは次に使用可能なインデックスになります。

6. 置換用のベアメタルノードの ID を取得します。

```
$ openstack baremetal node list
```

7. 置換ノードの機能を新しいインデックスで更新します。

```
$ openstack baremetal node set --property capabilities='node:<role>-
<index>,boot_option:local' <node>
```

- **<role>** を、ノードを置き換えるロールの名前 (**compute** など) に置き換えます。
- **<index>** を手順 5 で計算したインデックスに置き換えます。
- **<node>** をベアメタルノードの ID に置き換えてください。

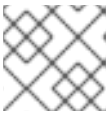
コンピュートスケジューラーは、ノード機能を使用して、デプロイメント時にノードを照合します。

8. **HostnameMap** 設定にインデックスを追加して、新しいノードにホスト名を割り当てます。次に例を示します。

```
parameter_defaults:
```

```
ControllerSchedulerHints:
  'capabilities:node': 'controller-%index%'
ComputeSchedulerHints:
  'capabilities:node': 'compute-%index%'
HostnameMap:
  overcloud-controller-0: overcloud-controller-prod-123-0
  overcloud-controller-1: overcloud-controller-prod-456-0 (1)
  overcloud-controller-2: overcloud-controller-prod-789-0
  overcloud-controller-3: overcloud-controller-prod-456-0 (2)
  overcloud-compute-0: overcloud-compute-prod-abc-0
  overcloud-compute-3: overcloud-compute-prod-abc-3 (3)
  overcloud-compute-8: overcloud-compute-prod-abc-3 (4)
  ....
```

1. 削除して新しいノードに置き換えるノード。
2. 新しいノード。
3. 削除して新しいノードに置き換えるノード。
4. 新しいノード。



注記

削除されたノードのマッピングを **HostnameMap** から削除しないでください。

9. 置換ノードの IP アドレスを、ネットワーク IP アドレスマッピングファイル **ips-from-pool-all.yaml** の各ネットワーク IP アドレスリストの最後に追加します。次の例では、新しいインデックス **overcloud-controller-3** の IP アドレスが、各 **ControllerIPs** ネットワークの IP アドレスリストの最後に追加され、**overcloud-controller-1** を置き換えるため、**overcloud-controller-1** と同じ IP アドレスが割り当てられます。新しいインデックスの IP アドレス **overcloud-compute-8** も、各 **ComputeIPs** ネットワークの IP アドレスリストの最後に追加され、置き換えられるインデックス **overcloud-compute-3** と同じ IP アドレスが割り当てられます。

```
parameter_defaults:
  ControllerIPs:
    ...
    internal_api:
      - 192.168.1.10 (1)
      - 192.168.1.11 (2)
      - 192.168.1.12 (3)
      - 192.168.1.11 (4)
    ...
    storage:
      - 192.168.2.10
      - 192.168.2.11
      - 192.168.2.12
      - 192.168.2.11
    ...

  ComputeIPs:
    ...
```

```
  internal_api:
    - 172.17.0.10 5
    - 172.17.0.11 6
    - 172.17.0.11 7
  ...
  storage:
    - 172.17.0.10
    - 172.17.0.11
    - 172.17.0.11
  ...
```

1. インデックス 0 に割り当てられた IP アドレス、ホスト名 **overcloud-controller-prod-123-0**。
2. インデックス 1 に割り当てられた IP アドレス、ホスト名 **overcloud-controller-prod-456-0**。このノードはインデックス 3 に置き換えられます。**このエントリーは削除しないでください。**
3. インデックス 2 に割り当てられた IP アドレス、ホスト名 **overcloud-controller-prod-789-0**。
4. インデックス 3 に割り当てられた IP アドレス、ホスト名 **overcloud-controller-prod-456-0**。これは、インデックス 1 を置き換える新しいノードです。
5. インデックス 0 に割り当てられた IP アドレス、ホスト名 **overcloud-compute-0**。
6. インデックス 1 に割り当てられた IP アドレス、ホスト名 **overcloud-compute-3**。このノードはインデックス 2 に置き換えられます。**このエントリーは削除しないでください。**
7. インデックス 2 に割り当てられた IP アドレス、ホスト名 **overcloud-compute-8**。これは、インデックス 1 を置き換える新しいノードです。

17.7. コントローラーノード置き換えのトリガー

古いコントローラーノードを削除して新規コントローラーノードに置き換えるには、以下の手順を実施します。

手順

1. 削除するコントローラーノードの UUID を決定し、それを **<NODEID>** 変数に格納します。**<node_name>** を、削除するノードの名前に置き換えてください。

```
(undercloud)[stack@director ~]$ NODEID=$(openstack server list -f value -c ID --name <node_name>)
```

2. Heat リソース ID を特定するには、以下のコマンドを入力します。

```
(undercloud)[stack@director ~]$ openstack stack resource show overcloud ControllerServers -f json -c attributes | jq --arg NODEID "$NODEID" -c '.attributes.value | keys[] as $k | if .[$k] == $NODEID then "Node index \($k) for \(.[$k])" else empty end'
```

3. 以下の内容で環境ファイル **~/templates/remove-controller.yaml** を作成し、削除するコントローラーノードのノードインデックスを含めます。

```
parameters:
  ControllerRemovalPolicies:
    [{'resource_list': ['<node_index>']}]
```

4. お使いの環境に該当するその他の環境ファイルと共に **remove-controller.yaml** 環境ファイルを指定して、オーバークラウドデプロイメントコマンドを入力します。

```
(undercloud) $ openstack overcloud deploy --templates \
    -e /home/stack/templates/remove-controller.yaml \
    [OTHER OPTIONS]
```

注記

- **-e ~/templates/remove-controller.yaml** は、デプロイメントコマンドのこのインスタンスに対してのみ指定します。これ以降のデプロイメント操作からは、この環境ファイルを削除してください。
- ブートストラップコントローラーノードを置き換え、ノード名を維持する場合は **~/templates/bootstrap-controller.yaml** を追加します。詳細は、ブートストラップコントローラーノードの交換 を参照してください。

5. 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。

```
(undercloud)$ openstack stack list --nested
```

6. デプロイコマンドが完了したら、古いノードが新しいノードに置き換えられていることを確認します。

```
(undercloud) $ openstack server list -c Name -c Networks
+------------------------+-----------------------+
| 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 |
+------------------------+-----------------------+
```

これで、新規ノードが稼動状態のコントロールプレーンサービスをホストするようになります。

17.8. コントローラーノード置き換え後のクリーンアップ

ノードの置き換えが完了したら、以下の手順を実施してコントローラークラスターの最終処理を行います。

手順

1. コントローラーノードにログインします。
2. Galera クラスターの Pacemaker 管理を有効にし、新規ノード上で Galera を起動します。

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource refresh galera-bundle
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource manage galera-bundle
```

3. 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 refresh** コマンドを使用して問題を解決し、そのサービスを再起動します。

4. director を終了します。

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

5. 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。

```
$ source ~/overcloudrc
```

6. オーバークラウド環境のネットワークエージェントを確認します。

```
(overcloud) $ openstack network agent list
```

7. 古いノードにエージェントが表示される場合には、そのエージェントを削除します。

```
(overcloud) $ for AGENT in $(openstack network agent list --host overcloud-controller-
1.localdomain -c ID -f value) ; do openstack network agent delete $AGENT ; done
```

8. 必要に応じて、新規ノード上の L3 エージェントホストにルーターを追加します。以下のコマンド例では、UUID に 2d1c1dc1-d9d4-4fa9-b2c8-f29cd1a649d4 を使用して L3 エージェントに **r1** という名称のルーターを追加しています。

```
(overcloud) $ openstack network agent add router --l3 2d1c1dc1-d9d4-4fa9-b2c8-
f29cd1a649d4 r1
```

9. cinder サービスを掃除します。

   a. cinder サービスをリスト表示します。

   ```
   (overcloud) $ openstack volume service list
   ```

   b. コントローラーノードにログインし、**cinder-api** コンテナーに接続し、**cinder-manage service remove** コマンドを使用して、残っているサービスを削除します。

   ```
   [heat-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-manage
   service remove cinder-backup <host>
   [heat-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-manage
   service remove cinder-scheduler <host>
   ```

10. RabbitMQ クラスターをクリーンアップします。

    a. コントローラーノードにログインします。

    b. **podman exec** コマンドを使用して bash を起動し、RabbitMQ クラスターのステータスを確認します。

    ```
    [heat-admin@overcloud-controller-0 ~]$ podman exec -it rabbitmq-bundle-podman-0
    bash
    [heat-admin@overcloud-controller-0 ~]$ rabbitmqctl cluster_status
    ```

    c. 置き換えられたコントローラーノードをクリアするには、**rabbitmqctl** コマンドを使用します。

    ```
    [heat-admin@overcloud-controller-0 ~]$ rabbitmqctl forget_cluster_node <node_name>
    ```

11. ブートストラップコントローラーノードを置き換えた場合は、置き換えプロセス後に環境ファイル **~/templates/bootstrap-controller.yaml** を削除するか、既存の環境ファイルから **pacemaker_short_bootstrap_node_name** および **mysql_short_bootstrap_node_name** パラメーターを削除する必要があります。このステップにより、これ以降の置き換えで director がコントローラーノード名をオーバーライドしようとするのを防ぎます。詳細は、ブートストラップコントローラーノードの置き換え を参照してください。

第18章 ノードの再起動

アンダークラウドおよびオーバークラウドで、ノードをリブートしなければならない場合があります。以下の手順を使用して、さまざまなノード種別をリブートする方法を説明します。

- 1 つのロールで全ノードをリブートする場合には、各ノードを個別にリブートすることを推奨しています。ロールの全ノードを同時に再起動すると、その操作中サービスにダウンタイムが生じる場合があります。
- OpenStack Platform 環境の全ノードをリブートする場合には、以下の順序でノードをリブートします。

推奨されるノードリブート順

1. アンダークラウドノードのリブート
2. コントローラーノードおよびその他のコンポーザブルノードのリブート
3. スタンドアロンの Ceph MON ノードのリブート
4. Ceph Storage ノードのリブート
5. Object Storage サービス (swift) ノードを再起動します。
6. コンピュートノードのリブート

18.1. アンダークラウドノードのリブート

アンダークラウドノードをリブートするには、以下の手順を実施します。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. アンダークラウドをリブートします。

```
$ sudo reboot
```

3. ノードがブートするまで待ちます。

18.2. コントローラーノードおよびコンポーザブルノードの再起動

設定可能なロールに基づいてコントローラーノードとスタンドアロンノードを再起動し、コンピュートノードと Ceph ストレージノードを除外します。

手順

1. リブートするノードにログインします。
2. オプション: ノードが Pacemaker リソースを使用している場合は、クラスターを停止します。

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

3. ノードをリブートします。

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

4. ノードがブートするまで待ちます。

検証

1. サービスが有効になっていることを確認します。

   a. ノードが Pacemaker サービスを使用している場合は、ノードがクラスターに再度加わったか確認します。

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

   b. ノードが Systemd サービスを使用している場合は、すべてのサービスが有効化されていることを確認します。

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

   c. ノードがコンテナー化されたサービスを使用している場合には、ノード上の全コンテナーがアクティブであることを確認します。

   ```
   [heat-admin@overcloud-controller-0 ~]$ sudo podman ps
   ```

18.3. スタンドアロンの CEPH MON ノードのリブート

スタンドアロンの Ceph MON ノードをリブートするには、以下の手順を実施します。

手順

1. Ceph MON ノードにログインします。
2. ノードをリブートします。

   ```
   $ sudo reboot
   ```

3. ノードがブートして MON クラスターに再度加わるまで待ちます。

クラスター内の各 MON ノードで、この手順を繰り返します。

18.4. CEPH STORAGE (OSD) クラスターのリブート

Ceph Storage (OSD) ノードのクラスターをリブートするには、以下の手順を実施します。

手順

1. Ceph MON またはコントローラーノードにログインして、Ceph Storage Cluster のリバランスを一時的に無効にします。

   ```
   $ sudo podman exec -it ceph-mon-controller-0 ceph osd set noout
   $ sudo podman exec -it ceph-mon-controller-0 ceph osd set norebalance
   ```

注記

マルチスタックまたは分散コンピュートノード (DCN) アーキテクチャーを使用している場合は、 **noout**フラグと**norebalance**フラグの設定時にクラスター名を指定する必要があります。例: **sudo podman exec -it ceph-mon-controller-0 ceph osd set noout --cluster <cluster_name>**

2. 再起動する最初の Ceph Storage ノードを選択し、そのノードにログインします。
3. ノードをリブートします。

```
$ sudo reboot
```

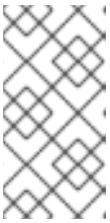
4. ノードがブートするまで待ちます。
5. ノードにログインして、クラスターのステータスを確認します。

```
$ sudo podman exec -it ceph-mon-controller-0 ceph status
```

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

6. ノードからログアウトして、次のノードを再起動し、ステータスを確認します。全 Ceph Storage ノードが再起動されるまで、このプロセスを繰り返します。
7. 完了したら、Ceph MON またはコントローラーノードにログインして、クラスターのリバランスを再度有効にします。

```
$ sudo podman exec -it ceph-mon-controller-0 ceph osd unset noout
$ sudo podman exec -it ceph-mon-controller-0 ceph osd unset norebalance
```



注記

マルチスタックまたは分散コンピュートノード (DCN) アーキテクチャーを使用している場合は、 **noout**フラグと**norebalance**フラグの設定解除時にクラスター名を指定する必要があります。例: **sudo podman exec -it ceph-mon-controller-0 ceph osd set noout --cluster <cluster_name>**

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** を報告していることを確認します。

```
$ sudo podman exec -it ceph-mon-controller-0 ceph status
```

18.5. OBJECT STORAGE サービス (SWIFT) ノードの再起動

次の手順では、Object Storage サービス (swift) ノードを再起動します。クラスター内のすべてのオブジェクトストレージノードに対して、次の手順を実行します。

手順

1. オブジェクトストレージノードにログインします。
2. ノードをリブートします。

```
$ sudo reboot
```

3. ノードがブートするまで待ちます。

4. クラスター内のオブジェクトストレージノードごとに再起動を繰り返します。

18.6. コンピュートノードの再起動

コンピュートノードをリブートするには、以下の手順を実施します。Red Hat OpenStack Platform 環境内のインスタンスのダウンタイムを最小限に抑えるために、この手順には、リブートするコンピュートノードからインスタンスを移行するステップも含まれています。これは、以下のワークフローを伴います。

- コンピュートノードをリブートする前に、インスタンスを別のノードに移行するかどうかを決定する。
- リブートするコンピュートノードを選択して無効にし、新規インスタンスをプロビジョニングしないようにする。
- インスタンスを別のコンピュートノードに移行する。
- 空のコンピュートノードを再起動します。
- 空のコンピュートノードを有効にします。

前提条件

コンピュートノードをリブートする前に、ノードをリブートする間インスタンスを別のコンピュートノードに移行するかどうかを決定する必要があります。

コンピュートノード間で仮想マシンインスタンスを移行する際に受ける可能性のある移行の制約のリストを確認してください。詳細は、**Configuring the Compute Service for Instance Creation**の Migration constraints を参照してください。

インスタンスを移行できない場合は、以下のコアテンプレートパラメーターを設定して、コンピュートノード再起動後のインスタンスの状態を制御する。

NovaResumeGuestsStateOnHostBoot

リブート後のコンピュートノードで、インスタンスを同じ状態に戻すか定義します。**False** に設定すると、インスタンスは停止した状態を維持し、手動で起動する必要があります。デフォルト値は **False** です。

NovaResumeGuestsShutdownTimeout

再起動する前に、インスタンスのシャットダウンを待つ秒数。この値を **0** に設定することは推奨されません。デフォルト値は 300 です。

オーバークラウドパラメーターおよびその使用方法についての詳細は、オーバークラウドのパラメーター を参照してください。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。

2. 全コンピュートノードとその UUID をリスト表示します。

```
$ source ~/stackrc
(undercloud) $ openstack server list --name compute
```

リブートするコンピュートノードの UUID を特定します。

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

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service list
(overcloud) $ openstack compute service set <hostname> nova-compute --disable
```

4. コンピュートノード上の全インスタンスをリスト表示します。

```
(overcloud) $ openstack server list --host <hostname> --all-projects
```

5. インスタンスを移行しない場合は、このステップ に進みます。
6. インスタンスを別のコンピュートノードに移行する場合には、以下のコマンドのいずれかを使用します。
   - インスタンスを別のホストに移行する。

     ```
     (overcloud) $ openstack server migrate <instance_id> --live <target_host> --wait
     ```

   - **nova-scheduler** がターゲットホストを自動的に選択できるようにします。

     ```
     (overcloud) $ nova live-migration <instance_id>
     ```

   - すべてのインスタンスを一度にライブマイグレーションします。

     ```
     $ nova host-evacuate-live <hostname>
     ```

     注記

     nova コマンドで非推奨の警告が表示される可能性がありますが、無視しても問題ありません。

7. 移行が完了するまで待ちます。
8. 移行が正常に完了したことを確認します。

```
(overcloud) $ openstack server list --host <hostname> --all-projects
```

9. 選択したコンピュートノードのインスタンスがなくなるまで、移行を続けます。
10. コンピュートノードにログインして、ノードをリブートします。

```
[heat-admin@overcloud-compute-0 ~]$ sudo reboot
```

11. ノードがブートするまで待ちます。
12. コンピュートノードを再度有効にします。

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service set <hostname>  nova-compute --enable
```

13. コンピュートノードが有効であることを確認します。

```
(overcloud) $ openstack compute service list
```

第19章 アンダークラウドおよびオーバークラウドのシャットダウンおよび起動

アンダークラウドおよびオーバークラウドでメンテナンスを実施する必要がある場合は、オーバークラウド起動時の問題を最小限に抑えるために、アンダークラウドおよびオーバークラウドノードを特定の順序でシャットダウンして起動する必要があります。

前提条件

- 動作中のアンダークラウドおよびオーバークラウド

19.1. アンダークラウドおよびオーバークラウドのシャットダウン順序

Red Hat OpenStack Platform 環境をシャットダウンするには、オーバークラウドおよびアンダークラウドを以下の順序でシャットダウンする必要があります。

1. オーバークラウドコンピュートノード上のインスタンスをシャットダウンします。
2. コンピュートノードをシャットダウンします。
3. コントローラーノードの高可用性サービスおよび OpenStack Platform のサービスをすべて停止します。
4. Ceph Storage ノードをシャットダウンします。
5. コントローラーノードをシャットダウンします。
6. アンダークラウドをシャットダウンします。

19.2. オーバークラウドコンピュートノード上のインスタンスのシャットダウン

Red Hat OpenStack Platform 環境のシャットダウンのサブタスクとして、コンピュートノードをシャットダウンする前にコンピュートノード上のインスタンスをすべてシャットダウンします。

前提条件

- Compute サービスがアクティブなオーバークラウド

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. source コマンドでオーバークラウドの認証情報ファイルを読み込みます。

```
$ source ~/overcloudrc
```

3. オーバークラウドで実行中のインスタンスを表示します。

```
$ openstack server list --all-projects
```

4. オーバークラウドのそれぞれのインスタンスを停止します。

```
$ openstack server stop <INSTANCE>
```

オーバークラウド内のすべてのインスタンスを停止するまで、それぞれのインスタンスでこのステップを繰り返します。

19.3. コンピュートノードのシャットダウン

Red Hat OpenStack Platform 環境をシャットダウンする際のサブタスクとして、それぞれのコンピュートノードにログインしてシャットダウンします。

前提条件

- コンピュートノード上のすべてのインスタンスがシャットダウンされている。

手順

1. コンピュートノードに **root** ユーザーとしてログインします。
2. ノードをシャットダウンします。

```
# shutdown -h now
```

3. すべてのコンピュートノードをシャットダウンするまで、それぞれのコンピュートノードでこの手順を実施します。

19.4. コントローラーノードのサービスの停止

Red Hat OpenStack Platform 環境のシャットダウンする際のサブタスクとして、コントローラーノードをシャットダウンする前にノードのサービスを停止します。これには Pacemaker サービスおよび systemd サービスが含まれます。

前提条件

- Pacemaker サービスがアクティブなオーバークラウド

手順

1. コントローラーノードに **root** ユーザーとしてログインします。
2. Pacemaker クラスターを停止します。

```
# pcs cluster stop --all
```

このコマンドにより、すべてのノード上のクラスターが停止します。

3. Pacemaker サービスが停止するまで待ち、サービスが停止したことを確認します。
   a. Pacemaker のステータスを確認します。

```
# pcs status
```

   b. Pacemaker サービスが実行されていないことを Podman で確認します。

```
# podman ps --filter "name=.*-bundle.*"
```

4. Red Hat OpenStack Platform のサービスを停止します。

```
# systemctl stop 'tripleo_*'
```

5. サービスが停止するまで待ち、サービスが実行されなくなったことを Podman で確認します。

```
# podman ps
```

19.5. CEPH STORAGE ノードのシャットダウン

Red Hat OpenStack Platform 環境のシャットダウンのサブタスクとして、Ceph Storage サービスを無効にし、続いてそれぞれの Ceph Storage ノードにログインしてシャットダウンします。

前提条件

- 正常な Ceph Storage クラスター。
- Ceph MON サービスがスタンドアロンの Ceph MON ノードまたはコントローラーノードで動作している。

手順

1. Ceph MON サービスを実行するノード (例: コントローラーノードまたはスタンドアロンの Ceph MON ノード) に **root** ユーザーとしてログインします。
2. クラスターが正常であることを確認します。以下の例の **podman** コマンドにより、コントローラーノード上の Ceph MON コンテナー内でステータス確認が実施されます。

```
# sudo podman exec -it ceph-mon-controller-0 ceph status
```

   ステータスが **HEALTH_OK** であることを確認します。

3. クラスターの **noout**、**norecover**、**norebalance**、**nobackfill**、**nodown**、および **pause** フラグを設定します。以下の例の **podman** コマンドにより、コントローラーノード上の Ceph MON コンテナーを通じてこれらのフラグが設定されます。

```
# sudo podman exec -it ceph-mon-controller-0 ceph osd set noout
# sudo podman exec -it ceph-mon-controller-0 ceph osd set norecover
# sudo podman exec -it ceph-mon-controller-0 ceph osd set norebalance
# sudo podman exec -it ceph-mon-controller-0 ceph osd set nobackfill
# sudo podman exec -it ceph-mon-controller-0 ceph osd set nodown
# sudo podman exec -it ceph-mon-controller-0 ceph osd set pause
```

4. それぞれの Ceph Storage ノードをシャットダウンします。
   a. Ceph Storage ノードに **root** ユーザーとしてログインします。
   b. ノードをシャットダウンします。

```
# shutdown -h now
```

c. すべての Ceph Storage ノードをシャットダウンするまで、それぞれの Ceph Storage ノードでこの手順を実施します。

5. スタンドアロンの Ceph MON ノードをすべてシャットダウンします。

a. スタンドアロンの Ceph MON ノードに **root** ユーザーとしてログインします。

b. ノードをシャットダウンします。

```
# shutdown -h now
```

c. スタンドアロンの Ceph MON ノードをすべてシャットダウンするまで、それぞれのスタンドアロンの Ceph MON ノードでこの手順を実施します。

関連情報

- What is the procedure to shutdown and bring up the entire ceph cluster?

19.6. コントローラーノードのシャットダウン

Red Hat OpenStack Platform 環境のシャットダウンのサブタスクとして、それぞれのコントローラーノードにログインしてシャットダウンします。

前提条件

- Pacemaker クラスターが停止している。
- コントローラーノードのすべての Red Hat OpenStack Platform サービスが停止している。

手順

1. コントローラーノードに **root** ユーザーとしてログインします。
2. ノードをシャットダウンします。

```
# shutdown -h now
```

3. すべてのコントローラーノードをシャットダウンするまで、それぞれのコントローラーノードでこの手順を実施します。

19.7. アンダークラウドのシャットダウン

Red Hat OpenStack Platform 環境のシャットダウンのサブタスクとして、アンダークラウドノードにログインしてアンダーグラウンドをシャットダウンします。

前提条件

- 動作中のアンダークラウド

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. アンダークラウドをシャットダウンします。

```
$ sudo shutdown -h now
```

19.8. システムメンテナンスの実施

アンダークラウドおよびオーバークラウドを完全にシャットダウンしたら、環境内のシステムに対するメンテナンスを実施し、続いてアンダークラウドおよびオーバークラウドを起動します。

19.9. アンダークラウドおよびオーバークラウドの起動順序

Red Hat OpenStack Platform 環境を起動するには、アンダークラウドおよびオーバークラウドを以下の順序で起動する必要があります。

1. アンダークラウドを起動する
2. コントローラーノードを起動する
3. Ceph Storage ノードを起動する
4. コンピュートノードを起動する
5. オーバークラウドコンピュートノード上のインスタンスを起動する

19.10. アンダークラウドの起動

Red Hat OpenStack Platform 環境の起動のサブタスクとして、アンダークラウドノードの電源をオンにし、アンダークラウドにログインし、アンダークラウドのサービスを確認します。

前提条件

- 電源がオフのアンダークラウド

手順

1. アンダークラウドの電源をオンにし、アンダークラウドがブートするまで待ちます。

検証

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. アンダークラウドのサービスを確認します。

```
$ systemctl list-units 'tripleo_*'
```

3. source コマンドでアンダークラウドの認証情報ファイルを読み込み、検証コマンドを実行して、すべてのサービスおよびコンテナーがアクティブで正常であることを確認します。

```
$ source stackrc
$ openstack tripleo validator run --validation service-status --limit undercloud
```

関連情報

- 検証フレームワークの使用

19.11. コントローラーノードの起動

Red Hat OpenStack Platform 環境の起動のサブタスクとして、それぞれのコントローラーノードの電源をオンにし、そのノードの Pacemaker 以外のサービスを確認します。

前提条件

- 電源がオフのコントローラーノード

手順

1. それぞれのコントローラーノードの電源をオンにします。

検証

1. **root** ユーザーとして各コントローラーノードにログインします。
2. コントローラーノードのサービスを確認します。

```
$ systemctl -t service
```

Pacemaker ベース以外のサービスだけが動作中です。

3. Pacemaker サービスが起動するまで待ち、サービスが起動したことを確認します。

```
$ pcs status
```

注記

環境でインスタンス HA を使用している場合、Pacemaker リソースは、コンピュートノードを起動するか、**pcs stonith confirm <compute_node>** コマンドを使用して手動でフェンスを解除する操作を実行するまで起動しません。このコマンドは、インスタンス HA を使用する各コンピュートノードで実行する必要があります。

19.12. CEPH STORAGE ノードの起動

Red Hat OpenStack Platform 環境の起動のサブタスクとして、Ceph MON ノードおよび Ceph Storage ノードの電源をオンにし、Ceph Storage サービスを有効にします。

前提条件

- 電源がオフの Ceph Storage クラスター。
- 電源がオフのスタンドアロンの Ceph MON ノードまたは電源がオンのコントローラーノードで、Ceph MON サービスが有効になっている。

手順

1. 使用している環境にスタンドアロンの Ceph MON ノードがある場合、それぞれの Ceph MON ノードの電源をオンにします。
2. それぞれの Ceph Storage ノードの電源をオンにします。

3. Ceph MON サービスを実行するノード (例: コントローラーノードまたはスタンドアロンの Ceph MON ノード) に **root** ユーザーとしてログインします。

4. クラスターノードのステータスを確認します。以下の例の **podman** コマンドにより、コントローラーノード上の Ceph MON コンテナー内でステータス確認が実施されます。

```
# sudo podman exec -it ceph-mon-controller-0 ceph status
```

それぞれのノードの電源がオンで、接続された状態であることを確認します。

5. クラスターの **noout**、**norecover**、**norebalance**、**nobackfill**、**nodown**、および **pause** フラグの設定を解除します。以下の例の **podman** コマンドにより、コントローラーノード上の Ceph MON コンテナーを通じてこれらのフラグの設定が解除されます。

```
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset noout
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset norecover
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset norebalance
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset nobackfill
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset nodown
# sudo podman exec -it ceph-mon-controller-0 ceph osd unset pause
```

検証

1. クラスターが正常であることを確認します。以下の例の **podman** コマンドにより、コントローラーノード上の Ceph MON コンテナー内でステータス確認が実施されます。

```
# sudo podman exec -it ceph-mon-controller-0 ceph status
```

ステータスが **HEALTH_OK** であることを確認します。

関連情報

- What is the procedure to shutdown and bring up the entire ceph cluster?

19.13. コンピュートノードの起動

Red Hat OpenStack Platform 環境の起動のサブタスクとして、それぞれのコンピュートノードの電源をオンにし、そのノードのサービスを確認します。

前提条件

- 電源がオフのコンピュートノード

手順

1. それぞれのコンピュートノードの電源をオンにします。

検証

1. 各コンピュートに **root** ユーザーとしてログインします。

2. コンピュートノードのサービスを確認します。

```
$ systemctl -t service
```

19.14. オーバークラウドコンピュートノード上のインスタンスの起動

Red Hat OpenStack Platform 環境の起動のサブタスクとして、コンピュートノード上のインスタンスを起動します。

前提条件

- アクティブなノードを持つアクティブなオーバークラウド

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. source コマンドでオーバークラウドの認証情報ファイルを読み込みます。

```
$ source ~/overcloudrc
```

3. オーバークラウドで実行中のインスタンスを表示します。

```
$ openstack server list --all-projects
```

4. オーバークラウド内のインスタンスを起動します。

```
$ openstack server start <INSTANCE>
```

第20章 カスタム SSL/TLS 証明書の設定

アンダークラウドがパブリックエンドポイントの通信に SSL/TLS を使用するように手動で設定できます。SSL/TLS を使用してアンダークラウドエンドポイントを手動で設定すると、概念実証としてセキュアなエンドポイントが作成されます。Red Hat は、認証局ソリューションを使用することを推奨します。

認証局 (CA) ソリューションを使用すると、証明書の更新、証明書失効リスト (CRL)、業界で受け入れられている暗号化など、運用に対応したソリューションが得られます。Red Hat Identity Manager (IdM) を CA として使用する方法は、Ansible を使用した TLS-e の実装 を参照してください。

独自の認証局で発行した SSL 証明書を使用する場合は、以下の設定手順を実施する必要があります。

20.1. 署名ホストの初期化

署名ホストとは、認証局を使用して新規証明書を生成し署名するホストです。選択した署名ホスト上で SSL 証明書を作成したことがない場合には、ホストを初期化して新規証明書に署名できるようにする必要がある可能性があります。

手順

1. すべての署名済み証明書の記録は、**/etc/pki/CA/index.txt** ファイルに含まれます。ファイルシステムパスと **index.txt** ファイルが存在することを確認します。

```
$ sudo mkdir -p /etc/pki/CA
$ sudo touch /etc/pki/CA/index.txt
```

2. **/etc/pki/CA/serial** ファイルは、次に署名する証明書に使用する次のシリアル番号を特定します。このファイルが存在しているかどうかを確認してください。ファイルが存在しない場合には、新規ファイルを作成して新しい開始値を指定します。

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

20.2. 認証局の作成

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

手順

1. 鍵と証明書のペアを生成して、認証局として機能するようにします。

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

2. **openssl req** コマンドは、認証局に関する特定の情報を要求します。要求されたら、それらの情報を入力してください。これらのコマンドにより、**ca.crt.pem** という名前の認証局ファイルが作成されます。

3. 証明書の場所を **enable-tls.yaml** ファイルの **PublicTLSCAFile** パラメーターの値として設定します。証明書の場所を **PublicTLSCAFile** パラメーターの値として設定する場合、CA 証明書パスが **clouds.yaml** 認証ファイルに追加されていることを確認してください。

```
parameter_defaults:
  PublicTLSCAFile: /etc/pki/ca-trust/source/anchors/cacert.pem
```

20.3. クライアントへの認証局の追加

SSL/TLS を使用して通信する外部クライアントについては、Red Hat OpenStack Platform 環境にアクセスする必要のある各クライアントに認証局ファイルをコピーします。

手順

1. 認証局をクライアントシステムにコピーします。

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

2. 各クライアントに認証局ファイルをコピーしたら、それぞれのクライアントで以下のコマンドを実行し、証明書を認証局のトラストバンドルに追加します。

```
$ sudo update-ca-trust extract
```

20.4. SSL/TLS 鍵の作成

OpenStack 環境で SSL/TLS を有効にするには、証明書を生成するための SSL/TLS 鍵が必要です。

手順

1. 以下のコマンドを実行し、SSL/TLS 鍵 (**server.key.pem**) を生成します。

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

20.5. SSL/TLS 証明書署名要求の作成

証明書署名要求を作成するには、以下の手順を実施します。

手順

1. 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 .
```

2. 新しい **openssl.cnf** ファイルを編集して、director に使用する SSL パラメーターを設定します。変更するパラメーターの種別には以下のような例が含まれます。

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]
countryName = Country Name (2 letter code)
countryName_default = AU
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Queensland
```

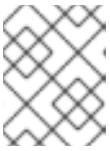
```
localityName = Locality Name (eg, city)
localityName_default = Brisbane
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = Red Hat
commonName = Common Name
commonName_default = 192.168.0.1
commonName_max = 64

[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
IP.1 = 192.168.0.1
DNS.1 = instack.localdomain
DNS.2 = vip.localdomain
DNS.3 = 192.168.0.1
```

commonName_default を、以下のエントリーのいずれかに設定します。

- IP アドレスを使用して SSL/TLS 経由で director にアクセスする場合には、**undercloud.conf** ファイルの **undercloud_public_host** パラメーターを使用します。
- 完全修飾ドメイン名を使用して SSL/TLS 経由で director にアクセスする場合には、ドメイン名を使用します。

alt_names セクションを編集して、以下のエントリーを追加します。

- **IP**: SSL 経由で director にアクセスするためにクライアントが使用する IP アドレスリスト
- **DNS**: SSL 経由で director にアクセスするためにクライアントが使用するドメイン名リスト。**alt_names** セクションの最後に DNS エントリーとしてパブリック API の IP アドレスも追加します。



注記

openssl.cnf に関する詳しい情報については、**man openssl.cnf** コマンドを実行してください。

3. 以下のコマンドを実行し、証明書署名要求 (**server.csr.pem**) を生成します。

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out server.csr.pem
```

-key オプションを使用して、OpenStack SSL/TLS 鍵を指定するようにしてください。

このコマンドにより、証明書署名要求として **server.csr.pem** ファイルが生成されます。このファイルを使用して OpenStack SSL/TLS 証明書を作成します。

20.6. SSL/TLS 証明書の作成

OpenStack 環境の SSL/TLS 証明書を生成するには、以下のファイルが必要です。

openssl.cnf

v3 拡張機能を指定するカスタム設定ファイル

server.csr.pem

証明書を生成して認証局を使用して署名するための証明書署名要求

ca.crt.pem

証明書への署名を行う認証局

ca.key.pem

認証局の秘密鍵

手順

1. 以下のコマンドを実行し、アンダークラウドまたはオーバークラウドの証明書を作成します。

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

コマンドは、以下のオプションを使用します。

-config

カスタム設定ファイルを使用します (ここでは、v3 拡張機能を指定した **openssl.cnf** ファイル)。

-extensions v3_req

v3 拡張機能を有効にします。

-days

証明書の有効期限が切れるまでの日数を定義します。

-in

証明書署名要求

-out

作成される署名済み証明書

-cert

認証局ファイル

-keyfile

認証局の秘密鍵

上記のコマンドにより、**server.crt.pem** という名前の新規証明書が作成されます。OpenStack SSL/TLS 鍵と共にこの証明書を使用します。

20.7. アンダークラウドへの証明書の追加

OpenStack SSL/TLS 証明書をアンダークラウドのトラストバンドルに追加するには、以下の手順を実施します。

手順

1. 以下のコマンドを実行して、証明書と鍵を統合します。

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

このコマンドにより、**undercloud.pem** ファイルが作成されます。

2. **undercloud.pem** ファイルを **/etc/pki** ディレクトリー内の場所にコピーし、HAProxy が読み取ることができるように必要な SELinux コンテキストを設定します。

```
$ sudo mkdir /etc/pki/undercloud-certs
$ sudo cp ~/undercloud.pem /etc/pki/undercloud-certs/.
$ sudo semanage fcontext -a -t etc_t "/etc/pki/undercloud-certs(/.*)?"
$ sudo restorecon -R /etc/pki/undercloud-certs
```

3. **undercloud.conf** ファイルの **undercloud_service_certificate** オプションに **undercloud.pem** ファイルの場所を追加します。

```
undercloud_service_certificate = /etc/pki/undercloud-certs/undercloud.pem
```

generate_service_certificate および **certificate_generation_ca** パラメーターを設定または有効にしないでください。director は、手動で作成した **undercloud.pem** 証明書を使用する代わりに、これらのパラメーターを使用して証明書を自動的に生成します。

4. アンダークラウド内の別のサービスが認証局にアクセスできるように、証明書に署名した認証局をアンダークラウドの信頼済み認証局のリストに追加します。

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

認証局がアンダークラウドに追加されたことを確認するには、**openssl** を使用してトラストバンドルを確認します。

```
$ openssl crl2pkcs7 -nocrl -certfile /etc/pki/tls/certs/ca-bundle.crt | openssl pkcs7 -print_certs -text | grep <CN of the CA issuer> -A 10 -B 10
```

- **<CN of the CA issuer>** を CA の発行者の一般名に置き換えます。このコマンドにより、有効期間を含むメインの証明書の詳細が出力されます。

第21章 その他のイントロスペクション操作

状況によっては、標準のオーバークラウドデプロイメントワークフローの外部でイントロスペクションを実行したい場合があります。たとえば、既存の未使用ノードのハードウェアを交換した後、新しいノードをイントロスペクトしたり、イントロスペクションデータを更新したりすることができます。

21.1. ノードイントロスペクションの個別実行

available の状態のノードで個別にイントロスペクションを実行するには、ノードを管理モードに設定して、イントロスペクションを実行します。

手順

1. すべてのノードを **manageable** 状態に設定します。

```
(undercloud) $ openstack baremetal node manage [NODE UUID]
```

2. イントロスペクションを実行します。

```
(undercloud) $ openstack overcloud node introspect [NODE UUID] --provide
```

イントロスペクションが完了すると、ノードの状態が **available** に変わります。

21.2. 初回のイントロスペクション後のノードイントロスペクションの実行

--provide オプションを指定したので、初回のイントロスペクションの後には、全ノードが **available** の状態になります。最初のイントロスペクションの後にすべてのノードでイントロスペクションを実行するには、ノードを管理モードに設定してイントロスペクションを実行します。

手順

1. すべてのノードを **manageable** 状態に設定します

```
(undercloud) $ for node in $(openstack baremetal node list --fields uuid -f value) ; do
openstack baremetal node manage $node ; done
```

2. bulk introspection コマンドを実行します。

```
(undercloud) $ openstack overcloud node introspect --all-manageable --provide
```

イントロスペクション完了後には、すべてのノードが **available** の状態に変わります。

21.3. ネットワークイントロスペクションの実行によるインターフェイス情報の取得

ネットワークイントロスペクションにより、Link Layer Discovery Protocol (LLDP) データがネットワークスイッチから取得されます。以下のコマンドにより、ノード上の全インターフェイスに関する LLDP 情報のサブセット、または特定のノードおよびインターフェイスに関するすべての情報が表示されます。この情報は、トラブルシューティングに役立ちます。director では、デフォルトで LLDP データ収集が有効になっています。

手順

1. ノード上のインターフェイスをリスト表示するには、以下のコマンドを実行します。

```
(undercloud) $ openstack baremetal introspection interface list [NODE UUID]
```

以下に例を示します。

```
(undercloud) $ openstack baremetal introspection interface list c89397b7-a326-41a0-907d-79f8b86c7cd9
+-----------+-------------------+-----------------------+-------------------+----------------+
| Interface | MAC Address       | Switch Port VLAN IDs  | Switch Chassis ID | Switch Port ID |
+-----------+-------------------+-----------------------+-------------------+----------------+
| p2p2      | 00:0a:f7:79:93:19 | [103, 102, 18, 20, 42] | 64:64:9b:31:12:00 | 510            |
| p2p1      | 00:0a:f7:79:93:18 | [101]                  | 64:64:9b:31:12:00 | 507            |
| em1       | c8:1f:66:c7:e8:2f | [162]                  | 08:81:f4:a6:b3:80 | 515            |
| em2       | c8:1f:66:c7:e8:30 | [182, 183]             | 08:81:f4:a6:b3:80 | 559            |
+-----------+-------------------+-----------------------+-------------------+----------------+
```

2. インターフェイスのデータおよびスイッチポートの情報を表示するには、以下のコマンドを実行します。

```
(undercloud) $ openstack baremetal introspection interface show [NODE UUID] [INTERFACE]
```

以下に例を示します。

```
(undercloud) $ openstack baremetal introspection interface show c89397b7-a326-41a0-907d-79f8b86c7cd9 p2p1
+--------------------------------------+------------------------------------------------------------------------------------------------+
| Field                                | Value                                                                                          |
+--------------------------------------+------------------------------------------------------------------------------------------------+
| interface                            | p2p1                                                                                           |
| mac                                  | 00:0a:f7:79:93:18                                                                              |
| node_ident                           | c89397b7-a326-41a0-907d-79f8b86c7cd9                                                           |
| switch_capabilities_enabled          | [u'Bridge', u'Router']                                                                         |
| switch_capabilities_support          | [u'Bridge', u'Router']                                                                         |
| switch_chassis_id                    | 64:64:9b:31:12:00                                                                              |
| switch_port_autonegotiation_enabled  | True                                                                                           |
| switch_port_autonegotiation_support  | True                                                                                           |
| switch_port_description              | ge-0/0/2.0                                                                                     |
| switch_port_id                       | 507                                                                                            |
```

```
| switch_port_link_aggregation_enabled | False
|
| switch_port_link_aggregation_id      | 0
|
| switch_port_link_aggregation_support | True
|
| switch_port_management_vlan_id       | None
|
| switch_port_mau_type                 | Unknown
|
| switch_port_mtu                      | 1514
|
| switch_port_physical_capabilities    | [u'1000BASE-T fdx', u'100BASE-TX fdx', u'100BASE-
TX hdx', u'10BASE-T fdx', u'10BASE-T hdx', u'Asym and Sym PAUSE fdx'] |
| switch_port_protocol_vlan_enabled    | None
|
| switch_port_protocol_vlan_ids        | None
|
| switch_port_protocol_vlan_support    | None
|
| switch_port_untagged_vlan_id         | 101
|
| switch_port_vlan_ids                 | [101]
|
| switch_port_vlans                    | [{u'name': u'RHOS13-PXE', u'id': 101}]
|
| switch_protocol_identities           | None
|
| switch_system_name                   | rhos-compute-node-sw1
|
+--------------------------------------+-------------------------------------------------------------------
-------------------------------------------------+
```

21.4. ハードウェアイントロスペクション情報の取得

Bare Metal サービスでは、オーバークラウド設定の追加ハードウェア情報を取得する機能がデフォルトで有効です。**undercloud.conf** ファイルの **inspection_extras** パラメーターについての詳しい情報は、Director Installation and Usage の Configuring director を参照してください。

たとえば、**numa_topology** コレクターは、追加ハードウェアイントロスペクションの一部で、各 NUMA ノードに関する以下の情報が含まれます。

- RAM (キロバイト単位)
- 物理 CPU コアおよびそのシブリングスレッド
- NUMA ノードに関連付けられた NIC

手順

- 上記の情報を取得するには、<UUID> をベアメタルノードの UUID に置き換えて、以下のコマンドを実行します。

```
# openstack baremetal introspection data save <UUID> | jq .numa_topology
```

取得されるベアメタルノードの NUMA 情報の例を以下に示します。


```
{
  "cpus": [
    {
      "cpu": 1,
      "thread_siblings": [
        1,
        17
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        10,
        26
      ],
      "numa_node": 1
    },
    {
      "cpu": 0,
      "thread_siblings": [
        0,
        16
      ],
      "numa_node": 0
    },
    {
      "cpu": 5,
      "thread_siblings": [
        13,
        29
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        15,
        31
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        7,
        23
      ],
      "numa_node": 0
    },
    {
      "cpu": 1,
      "thread_siblings": [
        9,
```

```
      25
    ],
    "numa_node": 1
  },
  {
    "cpu": 6,
    "thread_siblings": [
      6,
      22
    ],
    "numa_node": 0
  },
  {
    "cpu": 3,
    "thread_siblings": [
      11,
      27
    ],
    "numa_node": 1
  },
  {
    "cpu": 5,
    "thread_siblings": [
      5,
      21
    ],
    "numa_node": 0
  },
  {
    "cpu": 4,
    "thread_siblings": [
      12,
      28
    ],
    "numa_node": 1
  },
  {
    "cpu": 4,
    "thread_siblings": [
      4,
      20
    ],
    "numa_node": 0
  },
  {
    "cpu": 0,
    "thread_siblings": [
      8,
      24
    ],
    "numa_node": 1
  },
  {
    "cpu": 6,
    "thread_siblings": [
      14,
```

```
      30
     ],
     "numa_node": 1
    },
    {
     "cpu": 3,
     "thread_siblings": [
      3,
      19
     ],
     "numa_node": 0
    },
    {
     "cpu": 2,
     "thread_siblings": [
      2,
      18
     ],
     "numa_node": 0
    }
  ],
  "ram": [
    {
     "size_kb": 66980172,
     "numa_node": 0
    },
    {
     "size_kb": 67108864,
     "numa_node": 1
    }
  ],
  "nics": [
    {
     "name": "ens3f1",
     "numa_node": 1
    },
    {
     "name": "ens3f0",
     "numa_node": 1
    },
    {
     "name": "ens2f0",
     "numa_node": 0
    },
    {
     "name": "ens2f1",
     "numa_node": 0
    },
    {
     "name": "ens1f1",
     "numa_node": 0
    },
    {
     "name": "ens1f0",
     "numa_node": 0
    },
```

```
    {
      "name": "eno4",
      "numa_node": 0
    },
    {
      "name": "eno1",
      "numa_node": 0
    },
    {
      "name": "eno3",
      "numa_node": 0
    },
    {
      "name": "eno2",
      "numa_node": 0
    }
  ]
}
```

第22章 ベアメタルノードの自動検出

自動検出を使用すると、オーバークラウドノードを登録してそのメタデータを生成するのに、**instackenv.json** ファイルを作成する必要がありません。この改善は、ノードに関する情報を取得するのに費す時間を短縮するのに役立ちます。たとえば、自動検出を使用する場合、IPMI IP アドレスを照合し、その後に **instackenv.json** を作成する必要がありません。

22.1. 自動検出の有効化

Bare Metal 自動検出を有効にして設定し、PXE でブートするときにプロビジョニングネットワークに参加するノードを自動的に検出してインポートします。

手順

1. **undercloud.conf** ファイルで、ベアメタルの自動検出を有効にします。

```
enable_node_discovery = True
discovery_default_driver = ipmi
```

   - **enable_node_discovery**: 有効にすると、PXE を使用して introspection ramdisk をブートするすべてのノードが、自動的に Bare Metal サービス (ironic) に登録されます。
   - **discovery_default_driver**: 検出されたノードに使用するドライバーを設定します。例: **ipmi**

2. IPMI の認証情報を ironic に追加します。

   a. IPMI の認証情報を **ipmi-credentials.json** という名前のファイルに追加します。この例の **SampleUsername**、**RedactedSecurePassword**、および **bmc_address** の値を、実際の環境に応じて置き換えてください。

```
[
  {
    "description": "Set default IPMI credentials",
    "conditions": [
      {"op": "eq", "field": "data://auto_discovered", "value": true}
    ],
    "actions": [
      {"action": "set-attribute", "path": "driver_info/ipmi_username",
       "value": "SampleUsername"},
      {"action": "set-attribute", "path": "driver_info/ipmi_password",
       "value": "RedactedSecurePassword"},
      {"action": "set-attribute", "path": "driver_info/ipmi_address",
       "value": "{data[inventory][bmc_address]}"}
    ]
  }
]
```

3. IPMI の認証情報ファイルを ironic にインポートします。

```
$ openstack baremetal introspection rule import ipmi-credentials.json
```

22.2. 自動検出のテスト

PXE は、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を起動して、Bare Metal 自動検出機能をテストします。

手順

1. 必要なノードの電源をオンにします。
2. **openstack baremetal node list** コマンドを実行します。新しいノードが **enroll** の状態でリストに表示されるはずです。

```
$ openstack baremetal node list
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| UUID                                 | Name | Instance UUID | Power State | Provisioning State |
Maintenance |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| c6e63aec-e5ba-4d63-8d37-bd57628258e8 | None | None          | power off   | enroll             |
False       |
| 0362b7b2-5b9c-4113-92e1-0b34a2535d9b | None | None          | power off   | enroll             |
False       |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
```

3. 各ノードにリソースクラスを設定します。

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node set $NODE --resource-class baremetal ; done
```

4. 各ノードにカーネルと ramdisk を設定します。

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node manage $NODE ; done
$ openstack overcloud node configure --all-manageable
```

5. 全ノードを利用可能な状態に設定します。

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node provide $NODE ; done
```

22.3. ルールを使用した異なるベンダーハードウェアの検出

異種のハードウェアが混在する環境では、イントロスペクションルールを使用して、認証情報の割り当てやリモート管理を行うことができます。たとえば、DRAC を使用する Dell ノードを処理するには、別の検出ルールが必要になる場合があります。

手順

1. 以下の内容で、**dell-drac-rules.json** という名前のファイルを作成します。

```
[
  {
    "description": "Set default IPMI credentials",
```

```
        "conditions": [
            {"op": "eq", "field": "data://auto_discovered", "value": true},
            {"op": "ne", "field": "data://inventory.system_vendor.manufacturer",
             "value": "Dell Inc."}
        ],
        "actions": [
            {"action": "set-attribute", "path": "driver_info/ipmi_username",
             "value": "SampleUsername"},
            {"action": "set-attribute", "path": "driver_info/ipmi_password",
             "value": "RedactedSecurePassword"},
            {"action": "set-attribute", "path": "driver_info/ipmi_address",
             "value": "{data[inventory][bmc_address]}"}
        ]
    },
    {
        "description": "Set the vendor driver for Dell hardware",
        "conditions": [
            {"op": "eq", "field": "data://auto_discovered", "value": true},
            {"op": "eq", "field": "data://inventory.system_vendor.manufacturer",
             "value": "Dell Inc."}
        ],
        "actions": [
            {"action": "set-attribute", "path": "driver", "value": "idrac"},
            {"action": "set-attribute", "path": "driver_info/drac_username",
             "value": "SampleUsername"},
            {"action": "set-attribute", "path": "driver_info/drac_password",
             "value": "RedactedSecurePassword"},
            {"action": "set-attribute", "path": "driver_info/drac_address",
             "value": "{data[inventory][bmc_address]}"}
        ]
    }
]
```

- この例のユーザー名およびパスワードの値を、実際の環境に応じて置き換えてください。

2. ルールを ironic にインポートします。

```
$ openstack baremetal introspection rule import dell-drac-rules.json
```

第23章 プロファイルの自動タグ付けの設定

イントロスペクションプロセスでは、一連のベンチマークテストを実行します。director は、これらのテストからデータを保存します。このデータをさまざまな方法で使用するポリシーセットを作成することができます。

- ポリシーにより、パフォーマンスの低いノードまたは不安定なノードを特定して、これらのノードがオーバークラウドで使用されないように隔離することができます。
- ポリシーにより、ノードを自動的に特定のプロファイルにタグ付けするかどうかを定義することができます。

23.1. ポリシーファイルの構文

ポリシーファイルは JSON 形式で、ルールセットが記載されます。各ルールでは、説明、条件、およびアクションが定義されます。**説明** はプレーンテキストで記述されたルールの説明で、**条件** はキー/値のパターンを使用して評価を定義し、**アクション** は条件のパフォーマンスを表します。

説明

これは、プレーンテキストで記述されたルールの説明です。

例:

```
"description": "A new rule for my node tagging policy"
```

conditions

ここでは、以下のキー/値のパターンを使用して評価を定義します。

field

評価するフィールドを定義します。

- **memory_mb**: ノードのメモリーサイズ (MB 単位)
- **cpus**: ノードの CPU の合計スレッド数
- **cpu_arch**: ノードの CPU のアーキテクチャー
- **local_gb**: ノードのルートディスクの合計ストレージ容量

op

評価に使用する演算を定義します。これには、以下の属性が含まれます。

- **eq**: 等しい
- **ne**: 等しくない
- **lt**: より小さい
- **gt**: より大きい
- **le**: より小さいか等しい
- **ge**: より大きいか等しい

- **in-net**: IP アドレスが指定のネットワーク内にあることを確認します。
- **matches**: 指定の正規表現と完全に一致する必要があります。
- **contains**: 値には、指定の正規表現が含まれる必要があります。
- **is-empty**: **field** が空欄であることを確認します。

invert
: 評価の結果をインバージョン (反転) するかどうかを定義するブール値

multiple
: 複数の結果が存在する場合に、使用する評価を定義します。このパラメーターには以下の属性が含まれます。

  - **any**: いずれかの結果が一致する必要があります。
  - **all**: すべての結果が一致する必要があります。
  - **first**: 最初の結果が一致する必要があります。

value
: 評価する値を定義します。フィールド、演算および値の条件が満たされる場合には、true の結果を返します。そうでない場合には、条件は false の結果を返します。

例:

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

アクション

条件が **true** の場合には、ポリシーはアクションを実行します。アクションでは、**action** キーおよび **action** の値に応じて追加のキーが使用されます。

- **fail**: イントロスペクションが失敗します。失敗のメッセージには、**message** パラメーターが必要です。
- **set-attribute**: ironic ノードの属性を設定します。ironic の属性へのパス (例: **/driver_info/ipmi_address**) を指定する **path** フィールドおよび設定する **value** が必要です。
- **set-capability**: ironic ノードのケイパビリティーを設定します。新しいケイパビリティーの名前と値を指定する **name** および **value** フィールドが必要です。これにより、このケイパビリティーの既存の値が置き換えられます。たとえば、これを使用してノードのプロファイルを定義します。
- **extend-attribute**: **set-attribute** と同じですが、既存の値をリストとして扱い、そのリストに値を追記します。オプションの **unique** パラメーターを True に設定すると、対象の値がすでにリストに含まれている場合には何も追加しません。

例:

```
"actions": [
  {
    "action": "set-capability",
    "name": "profile",
    "value": "swift-storage"
  }
]
```

23.2. ポリシーファイルの例

イントロスペクションルールを記載した JSON ファイル (**rules.json**) の例を以下に示します。

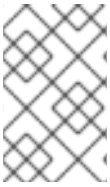
```
[
  {
    "description": "Fail introspection for unexpected nodes",
    "conditions": [
      {
        "op": "lt",
        "field": "memory_mb",
        "value": 4096
      }
    ],
    "actions": [
      {
        "action": "fail",
        "message": "Memory too low, expected at least 4 GiB"
      }
    ]
  },
  {
    "description": "Assign profile for object storage",
    "conditions": [
      {
        "op": "ge",
        "field": "local_gb",
        "value": 1024
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "profile",
        "value": "swift-storage"
      }
    ]
  },
  {
    "description": "Assign possible profiles for compute and controller",
    "conditions": [
      {
        "op": "lt",
        "field": "local_gb",
        "value": 1024
      },
      {
```

```
        "op": "ge",
        "field": "local_gb",
        "value": 40
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "compute_profile",
        "value": "1"
      },
      {
        "action": "set-capability",
        "name": "control_profile",
        "value": "1"
      },
      {
        "action": "set-capability",
        "name": "profile",
        "value": null
      }
    ]
  }
]
```

上記の例は、3 つのルールで設定されています。

- メモリーが 4096 MiB 未満の場合には、イントロスペクションが失敗します。クラウドから特定のノードを除外する場合は、このルール種別を適用することができます。
- ハードドライブのサイズが 1 TiB 以上のノードの場合は swift-storage プロファイルが無条件で割り当てられます。
- ハードドライブが 1 TiB 未満だが 40 GiB を超えているノードは、コンピュートノードまたはコントローラーノードのいずれかに割り当てることができます。**openstack overcloud profiles match** コマンドを使用して後で最終選択できるように、2 つのケイパビリティー (**compute_profile** および **control_profile**) を割り当てています。このプロセスが機能するためには、既存のプロファイルケイパビリティーを削除する必要があります。削除しないと、既存のプロファイルケイパビリティーが優先されます。

プロファイルマッチングルールは、他のノードを変更しません。



注記

イントロスペクションルールを使用して **profile** 機能を割り当てる場合は常に、既存の値よりこの割り当てた値が優先されます。ただし、すでにプロファイルケイパビリティーを持つノードについては、**[PROFILE]_profile** ケイパビリティーは無視されます。

23.3. ポリシーファイルのインポート

ポリシーファイルを director にインポートするには、以下の手順を実施します。

手順

1. ポリシーファイルを director にインポートします。

```
$ openstack baremetal introspection rule import rules.json
```

2. イントロスペクションのプロセスを実行します。

```
$ openstack overcloud node introspect --all-manageable
```

3. イントロスペクションが完了したら、ノードとノードに割り当てられたプロファイルを確認します。

```
$ openstack overcloud profiles list
```

4. イントロスペクションルールに間違いがあった場合には、以下のコマンドを実行してすべてのルールを削除します。

```
$ openstack baremetal introspection rule purge
```

第24章 完全なディスクイメージの作成

メインのオーバークラウドイメージは、パーティション情報またはブートローダーが含まれないフラットパーティションイメージです。director は、ノードをブートする時には別のカーネルおよび ramdisk を使用し、オーバークラウドイメージをディスクに書き込む時に基本的なパーティションレイアウトを作成します。ただし、パーティションレイアウト、ブートローダー、および強化されたセキュリティー機能が含まれる完全なディスクイメージを作成することができます。

重要

以下のプロセスでは、director のイメージビルド機能を使用します。Red Hat では、本項に記載の指針に従うイメージのみをサポートしています。これらとは異なる仕様でビルドされたカスタムイメージはサポートされていません。

24.1. セキュリティー強化手段

完全なディスクイメージには、セキュリティーが重要な機能となる Red Hat OpenStack Platform のデプロイメントに必要な、追加のセキュリティー強化手段が含まれます。

イメージを作成する際のセキュリティーに関する推奨事項

- **/tmp** ディレクトリーを別のボリュームまたはパーティションにマウントし、**rw**、**nosuid**、**nodev**、**noexec**、および **relatime** のフラグを付ける。
- **/var**、**/var/log**、および **/var/log/audit** ディレクトリーを別のボリュームまたはパーティションにマウントし、**rw** および **relatime** のフラグを付ける。
- **/home** ディレクトリーを別のパーティションまたはボリュームにマウントし、**rw**、**nodev**、および **relatime** のフラグを付ける。
- **GRUB_CMDLINE_LINUX** の設定に以下の変更を加える。
  - 監査を有効にするには、**audit=1** カーネルブートフラグを追加します。
  - ブートローダー設定を使用した USB のカーネルサポートを無効にするには、**nousb** を追加します
  - セキュアでないブートフラグを削除するには、**crashkernel=auto** を削除します。
- セキュアでないモジュール (**usb-storage**、**cramfs**、**freevxfs**、**jffs2**、**hfs**、**hfsplus**、**squashfs**、**udf**、**vfat**) をブラックリストに登録して、読み込まれないようにする。
- **telnet** などの安全でないパッケージは、デフォルトでインストールされているため、イメージから削除します。

24.2. 完全なディスクイメージに関するワークフロー

完全なディスクイメージをビルドするには、以下のワークフローに従います。

1. ベースの Red Hat Enterprise Linux 8 2 イメージをダウンロードする。
2. 登録固有の環境変数を設定する。
3. パーティションスキーマとサイズを変更してイメージをカスタマイズする。

4. イメージを作成する。

5. イメージを director にアップロードする。

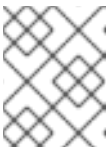
24.3. ベースのクラウドイメージのダウンロード

完全なディスクイメージをビルドする前に、ベースとして使用する Red Hat Enterprise Linux の既存のクラウドイメージをダウンロードする必要があります。

手順

1. Red Hat Enterprise Linux 8.2 ダウンロードページに移動します。Red Hat OpenStack Platform 16.2 は Red Hat Enterprise Linux 8.2 でサポートされています。

   - https://access.redhat.com/downloads/content/479/ver=/rhel---8/8.4/x86_64/product-software

   注記

   プロンプトが表示されたら、カスタマーポータルのログイン情報を入力します。

2. **Red Hat Enterprise Linux 8.2 KVM Guest Image**の横にある **Download Now** をクリックします。

24.4. 一貫したインターフェイスの命名を有効にする

一貫性のあるネットワークインターフェイスデバイスの命名は、デフォルトで KVM ゲストイメージで無効になっています。**virt-customize** を使用して、一貫した命名を有効にします。

手順

1. KVM ゲストイメージを **/var/lib/libvirt/images** に移動します。

```
$ sudo mv <kvm_guest_image> /var/lib/libvirt/images/
```

2. **libvirtd** を開始します。

```
$ sudo systemctl start libvirtd
```

3. KVM ゲストイメージで一貫したインターフェイスの命名を有効にします。

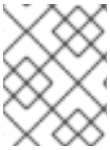
```
$ sudo virt-customize -a /var/lib/libvirt/images/<kvm guest image> --edit
/etc/default/grub:s/net.ifnames=0/net.ifnames=1/
```

4. **libvirtd** を停止します。

```
$ sudo systemctl stop libvirtd
```

24.5. ディスクイメージの環境変数

ディスクイメージのビルドプロセスとして、director にはベースイメージと、新規オーバークラウドイメージのパッケージを取得するための登録情報が必要です。これらの属性は、以下に示す Linux の環境変数を使用して定義します。



注記

イメージのビルドプロセスにより、イメージは一時的に Red Hat サブスクリプションに登録され、イメージのビルドプロセスが完了するとシステムの登録が解除されます。

ディスクイメージをビルドするには、Linux の環境変数をお使いの環境と要件に応じて設定します。

DIB_LOCAL_IMAGE
: 完全なディスクイメージのベースに使用するローカルイメージを設定します。

REG_ACTIVATION_KEY
: 登録プロセスにおいて、ログイン情報の代わりにアクティベーションキーを使用します。

REG_AUTO_ATTACH
: 最も互換性のあるサブスクリプションを自動的にアタッチするかどうかを定義します。

REG_BASE_URL
: イメージのパッケージが含まれるコンテンツ配信サーバーのベース URL。カスタマーポータル Subscription Management のデフォルトプロセスでは **https://cdn.redhat.com** を使用します。Red Hat Satellite 6 サーバーを使用している場合は、このパラメーターをお使いの Satellite サーバーのベース URL に設定します。

REG_ENVIRONMENT
: 組織内の環境に登録します。

REG_METHOD
: 登録の方法を設定します。Red Hat カスタマーポータルに登録するには **portal** を使用します。Red Hat Satellite 6 で登録するには、**satellite** を使用します。

REG_ORG
: イメージを登録する組織

REG_POOL_ID
: 製品のサブスクリプション情報のプール ID

REG_PASSWORD
: イメージを登録するユーザーアカウントのパスワードを設定します。

REG_RELEASE
: Red Hat Enterprise Linux のマイナーリリースバージョンを設定します。**REG_AUTO_ATTACH** または **REG_POOL_ID** 環境変数でこれを使用する必要があります。

REG_REPOS
: リポジトリー名のコンマ区切り文字列。この文字列の各リポジトリーは **subscription-manager** で有効化されます。
以下に示すセキュリティーが強化された完全なディスクイメージのリポジトリーを使用します。

- **rhel-8-for-x86_64-baseos-eus-rpms**
- **rhel-8-for-x86_64-appstream-eus-rpms**
- **rhel-8-for-x86_64-highavailability-eus-rpms**
- **ansible-2.9-for-rhel-8-x86_64-rpms**

- **fast-datapath-for-rhel-8-x86_64-rpms**
- **openstack-16.1-for-rhel-8-x86_64-rpms**

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。

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設定します。Red Hat カスタマーポータルの場合、デフォルトホスト名は **subscription.rhn.redhat.com** です。Red Hat Satellite 6 サーバーを使用している場合は、このパラメーターをお使いの Satellite サーバーのホスト名に設定します。

REG_USER
: イメージを登録するアカウントのユーザー名を設定します。

環境変数のセットをエクスポートし、ローカルの QCOW2 イメージを一時的に Red Hat カスタマーポータルに登録するには、以下の例に示すコマンドのセットを使用します。

```
$ export DIB_LOCAL_IMAGE=./rhel-8.2-x86_64-kvm.qcow2
$ export REG_METHOD=portal
$ export REG_USER=<your_name>
$ export REG_PASSWORD=<your_password>
$ export REG_RELEASE="8.2"
$ export REG_POOL_ID=<pool_id>
$ export REG_REPOS="rhel-8-for-x86_64-baseos-eus-rpms \
    rhel-8-for-x86_64-appstream-eus-rpms \
    rhel-8-for-x86_64-highavailability-eus-rpms \
    ansible-2.9-for-rhel-8-x86_64-rpms \
    fast-datapath-for-rhel-8-x86_64-rpms \
    openstack-16.1-for-rhel-8-x86_64-rpms"
```

24.6. ディスクレイアウトのカスタマイズ

デフォルトでは、セキュリティーが強化されたイメージのサイズは 20 GB で、事前定義されたパーティショニングサイズを使用します。ただし、オーバークラウドのコンテナーイメージを収容するには、パーティションレイアウトを変更する必要があります。以降のセクションで説明する手順を実施して、イメージのサイズを 40 GB に増やします。より厳密にご自分のニーズに合わせるために、パーティションレイアウトやディスクのサイズを変更することができます。

パーティションレイアウトとディスクサイズを変更するには、以下の手順に従ってください。

- **DIB_BLOCK_DEVICE_CONFIG** 環境変数を使用してパーティショニングスキーマを変更する。
- **DIB_IMAGE_SIZE** 環境変数を更新して、イメージのグローバルサイズを変更する。

24.7. パーティショニングスキーマの変更

パーティショニングスキーマを編集して、パーティショニングサイズを変更したり、新規パーティションの作成や既存パーティションの削除を行うことができます。新規パーティショニングスキーマを定義するには、以下の環境変数を使用します。

```
$ export DIB_BLOCK_DEVICE_CONFIG='<yaml_schema_with_partitions>'
```

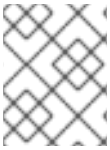
BIOS の例

以下の YAML 設定は、オーバークラウドのコンテナーイメージをプルするのに十分なスペースを提供する、論理ボリュームの変更後のパーティションレイアウトを示しています。

```
export DIB_BLOCK_DEVICE_CONFIG='"
- local_loop:
    name: image0
- partitioning:
    base: image0
    label: mbr
    partitions:
      - name: root
        flags: [ boot,primary ]
        size: 40G
- lvm:
    name: lvm
    base: [ root ]
    pvs:
        - name: pv
          base: root
          options: [ "--force" ]
    vgs:
        - name: vg
          base: [ "pv" ]
          options: [ "--force" ]
    lvs:
        - name: lv_root
          base: vg
          extents: 23%VG
        - name: lv_tmp
          base: vg
          extents: 4%VG
        - name: lv_var
          base: vg
          extents: 45%VG
        - name: lv_log
          base: vg
          extents: 23%VG
        - name: lv_audit
          base: vg
          extents: 4%VG
        - name: lv_home
          base: vg
          extents: 1%VG
- mkfs:
    name: fs_root
    base: lv_root
    type: xfs
    label: "img-rootfs"
    mount:
        mount_point: /
        fstab:
            options: "rw,relatime"
```

```
        fsck-passno: 1
- mkfs:
    name: fs_tmp
    base: lv_tmp
    type: xfs
    mount:
        mount_point: /tmp
        fstab:
            options: "rw,nosuid,nodev,noexec,relatime"
            fsck-passno: 2
- mkfs:
    name: fs_var
    base: lv_var
    type: xfs
    mount:
        mount_point: /var
        fstab:
            options: "rw,relatime"
            fsck-passno: 2
- mkfs:
    name: fs_log
    base: lv_log
    type: xfs
    mount:
        mount_point: /var/log
        fstab:
            options: "rw,relatime"
            fsck-passno: 3
- mkfs:
    name: fs_audit
    base: lv_audit
    type: xfs
    mount:
        mount_point: /var/log/audit
        fstab:
            options: "rw,relatime"
            fsck-passno: 4
- mkfs:
    name: fs_home
    base: lv_home
    type: xfs
    mount:
        mount_point: /home
        fstab:
            options: "rw,nodev,relatime"
            fsck-passno: 2
'"
```

このサンプル YAML コンテンツをイメージのパーティションスキーマのベースとして使用します。パーティションサイズとレイアウトを必要に応じて変更します。



注記

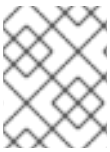
デプロイメント後にパーティションサイズを変更することはできないので、イメージ用に正しいパーティションサイズを定義する必要があります。

UEFI の例

以下の YAML 設定は、オーバークラウドのコンテナーイメージをプルするのに十分なスペースを提供する、論理ボリュームの変更後のパーティションレイアウトを示しています。

```
export DIB_BLOCK_DEVICE_CONFIG='''
- local_loop:
    name: image0
- partitioning:
    base: image0
    label: gpt
    partitions:
      - name: ESP
        type: 'EF00'
        size: 200MiB
      - name: BSP
        type: 'EF02'
        size: 1MiB
      - name: ROOT
        type: '8300'
        size: 100%
- mkfs:
    name: fs_esp
    base: ESP
    type: vfat
    mount:
      mount_point: /boot/efi
      fstab:
        options: "defaults"
        fsck-passno: 1
- mkfs:
    name: fs_root
    label: "img-rootfs"
    base: ROOT
    type: xfs
    mount:
      mount_point: /
      fstab:
        options: "defaults"
        fsck-passno: 1
'''
```

このサンプル YAML コンテンツをイメージのパーティションスキーマのベースとして使用します。ご使用の環境に合わせて、パーティションのサイズとレイアウトを変更してください。



注記

デプロイメント後にイメージのサイズを変更することはできないため、デプロイメント前にイメージの正しいパーティションサイズを定義する必要があります。

24.8. イメージサイズの変更

変更後のパーティショニングスキーマの合計は、デフォルトのディスクサイズ (20 GB) を超える可能性があります。そのような場合には、イメージサイズを変更しなければならない場合があります。イメージサイズを変更するには、イメージを作成する設定ファイルを編集します。

手順

1. **/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-python3.yaml** のコピーを作成します。

```
# cp /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-python3.yaml \
/home/stack/overcloud-hardened-images-python3-custom.yaml
```

注記

UEFI の完全なディスクイメージの場合は、**/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-uefi-python3.yaml** を使用します。

2. 設定ファイルで **DIB_IMAGE_SIZE** を編集して、必要な値に調整します。

```
...

environment:
  DIB_PYTHON_VERSION: '3'
  DIB_MODPROBE_BLACKLIST: 'usb-storage cramfs freevxfs jffs2 hfs hfsplus squashfs udf vfat bluetooth'
  DIB_BOOTLOADER_DEFAULT_CMDLINE: 'nofb nomodeset vga=normal console=tty0 console=ttyS0,115200 audit=1 nousb'
  DIB_IMAGE_SIZE: '40' (1)
  COMPRESS_IMAGE: '1'
```

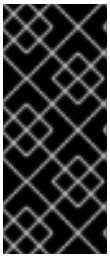
(1) この値は、新しいディスクサイズの合計に応じて調整してください。

3. オプション:プロキシーを設定するには、**http_proxy** と **https:_proxy** 環境変数も含める必要があります。

```
environment:
  http_proxy: <proxy_server>
  https_proxy: <proxy_server>
```

- **<proxy_server>** をプロキシーのアドレスに置き換えます。

4. ファイルを保存します。



重要

オーバークラウドをデプロイする際に、director はオーバークラウドイメージの RAW バージョンを作成します。これは、アンダークラウドに、その RAW イメージを収容するのに十分な空き容量がなければならないことを意味します。たとえば、セキュリティーが強化されたイメージのサイズを 40 GB に設定した場合には、アンダークラウドのハードディスクに 40 GB の空き容量が必要となります。

重要

director が物理ディスクにイメージを書き込む際に、ディスクの最後に 64 MB のコンフィグドライブ一次パーティションが作成されます。完全なディスクイメージを作成する場合には、物理ディスクをこの追加パーティションを収容することのできるサイズにしてください。

24.9. 完全なディスクイメージのビルド

環境変数を設定してイメージをカスタマイズしたら、**openstack overcloud image build** コマンドを使用してイメージを作成します。

手順

1. 必要なすべての設定ファイルを指定して、**openstack overcloud image build** コマンドを実行します。

```
# openstack overcloud image build \
--image-name overcloud-hardened-full \ 1
--config-file /home/stack/overcloud-hardened-images-python3-custom.yaml \ 2
--config-file /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-rhel8.yaml 3
```

1. UEFI ディスク全体のイメージの場合は、**overcloud-hardened-uefi-full** を使用します。
2. **overcloud-hardened-images-python3-custom.yaml** ファイルは、新しいディスクサイズを含むカスタム設定ファイルです。異なるカスタムディスクサイズを使用していない場合は、代わりに元の **/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-python3.yaml** ファイルを使用してください。標準の UEFI の完全なディスクイメージの場合は、**overcloud-hardened-images-uefi-python3.yaml** を使用します。
3. UEFI の完全なディスクイメージの場合は、**overcloud-hardened-images-uefi-rhel8.yaml** を使用します。

このコマンドにより、必要なセキュリティー機能がすべて含まれた、**overcloud-hardened-full.qcow2** という名前のイメージが作成されます。

24.10. 完全なディスクイメージのアップロード

OpenStack Image (glance) サービスにイメージをアップロードして、Red Hat OpenStack Platform director から使用を開始します。セキュリティーが強化されたイメージをアップロードするには、以下の手順を実施してください。

1. 新たに生成したイメージの名前を変更し、**images** ディレクトリーに移動します。

```
# mv overcloud-hardened-full.qcow2 ~/images/overcloud-full.qcow2
```

2. オーバークラウドの古いイメージをすべて削除します。

```
# openstack image delete overcloud-full
# openstack image delete overcloud-full-initrd
# openstack image delete overcloud-full-vmlinuz
```

3. 新規オーバークラウドイメージをアップロードします。

```
# openstack overcloud image upload --image-path /home/stack/images --whole-disk
```

既存のイメージをセキュリティーが強化されたイメージに置き換える場合は、**--update-existing** フラグを使用します。このフラグを使用することで、元の **overcloud-full** イメージがセキュリティーの強化された新しいイメージに書き換えられます。

第25章 直接デプロイの設定

ノードをプロビジョニングする場合、director は iSCSI マウントにオーバークラウドのベースオペレーティングシステムのイメージをマウントし、そのイメージを各ノードのディスクにコピーします。直接デプロイとは、ディスクイメージを HTTP の場所から直接ベアメタルノード上のディスクに書き込む代替方法です。

25.1. アンダークラウドへの直接デプロイインターフェイスの設定

iSCSI デプロイインターフェイスがデフォルトのデプロイインターフェイスです。ただし、直接デプロイインターフェイスを有効にして、イメージを HTTP の保管場所からターゲットディスクにダウンロードすることができます。

注記

オーバークラウドノードのメモリー **tmpfs** には、少なくとも 8 GB の RAM が必要です。

手順

1. カスタム環境ファイル **/home/stack/undercloud_custom_env.yaml** を作成または変更して、**IronicDefaultDeployInterface** を指定します。

```
parameter_defaults:
  IronicDefaultDeployInterface: direct
```

2. デフォルトでは、各ノードの Bare Metal サービス (ironic) エージェントは、HTTP リンクを通じて Object Storage サービス (swift) に保管されているイメージを取得します。あるいは、ironic は、**ironic-conductor** HTTP サーバーを通じて、このイメージを直接ノードにストリーミングすることもできます。イメージを提供するサービスを変更するには、**/home/stack/undercloud_custom_env.yaml** ファイルの **IronicImageDownloadSource** を **http** に設定します。

```
parameter_defaults:
  IronicDefaultDeployInterface: direct
  IronicImageDownloadSource: http
```

3. カスタム環境ファイルを **undercloud.conf** ファイルの **DEFAULT** セクションに追加します。

```
custom_env_files = /home/stack/undercloud_custom_env.yaml
```

4. アンダークラウドのインストールを実施します。

```
$ openstack undercloud install
```

第26章 仮想コントロールプレーンの作成

仮想コントロールプレーンは、ベアメタルではなく仮想マシン (VM) 上にあるコントロールプレーンです。仮想コントロールプレーンを使用することで、コントロールプレーンに必要なベアメタルマシンの数を減らすことができます。

本章では、Red Hat OpenStack Platform (RHOSP) および Red Hat Virtualization を使用して、オーバークラウドの RHOSP コントロールプレーンを仮想化する方法について説明します。

26.1. 仮想コントロールプレーンのアーキテクチャー

director を使用して、Red Hat Virtualization クラスターにデプロイされたコントローラーノードを使用するオーバークラウドをプロビジョニングします。その後、これらの仮想コントローラーを仮想コントロールプレーンノードとしてデプロイできます。

注記

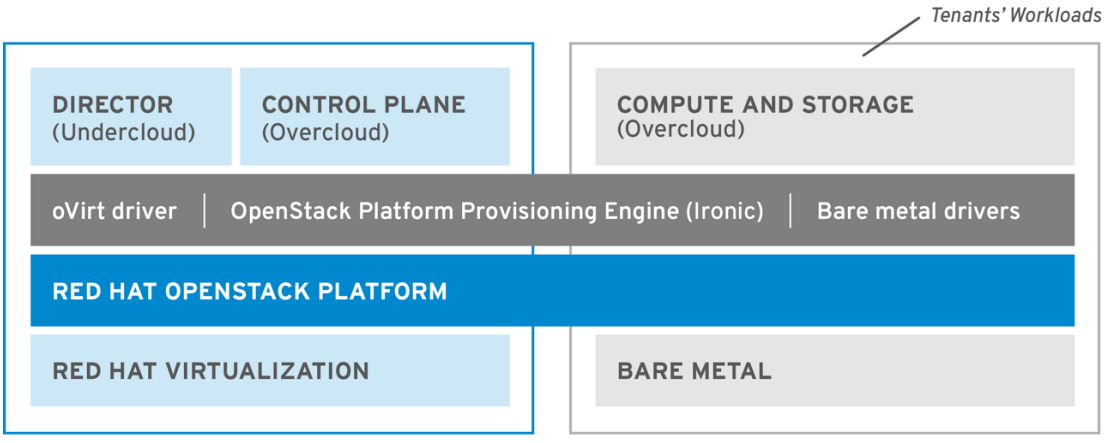
仮想コントローラーノードは、Red Hat Virtualization 上でのみサポートされます。

以下のアーキテクチャー図は、仮想コントロールプレーンのデプロイ方法を示しています。オーバークラウドを Red Hat Virtualization の仮想マシン上で実行中のコントローラーノードに分散し、コンピュートノードおよびストレージノードをベアメタル上で実行します。

注記

OpenStack 仮想アンダークラウドは、Red Hat Virtualization 上で実行されます。

仮想コントロールプレーンのアーキテクチャー



OPENSTACK_477985_1018

OpenStack Bare Metal Provisioning サービス (ironic) には、Red Hat Virtualization の仮想マシン用ドライバー **staging-ovirt** が含まれています。このドライバーを使用して、Red Hat Virtualization 環境内の仮想ノードを管理できます。このドライバーを使用して、Red Hat Virtualization 環境内の仮想マシンとしてオーバークラウドのコントローラーをデプロイすることもできます。

RHOSP オーバークラウドのコントロールプレーンを仮想化する際の利点と制限

RHOSP オーバークラウドのコントロールプレーンを仮想化する利点は数多くありますが、すべての設定に適用できる訳ではありません。

利点

オーバークラウドのコントロールプレーンを仮想化することには、ダウンタイムを回避し、パフォーマンスを向上させる数多くの利点があります。

- ホットプラグおよびホットアンプラグを使用して CPU およびメモリーを必要に応じてスケーリングし、リソースを仮想コントローラーに動的に割り当てることができます。これにより、ダウンタイムを防ぎ、プラットフォームの拡張に合わせて増大した能力を活用できます。
- 同じ Red Hat Virtualization クラスターに追加のインフラストラクチャー仮想マシンをデプロイすることができます。これにより、データセンターのサーバーフットプリントが最小限に抑えられ、物理ノードの効率が最大化されます。
- コンポーザブルロールを使用すると、より複雑な RHOSP コントロールプレーンを定義することができ、リソースをコントロールプレーンの特定のコンポーネントに割り当てることができます。
- 仮想マシンのライブマイグレーション機能により、サービスを中断せずにシステムをメンテナンスすることができます。
- Red Hat Virtualization がサポートするサードパーティーまたはカスタムツールを統合することができます。

制限

仮想コントロールプレーンには、使用できる設定の種類に制限があります。

- 仮想 Ceph Storage ノードおよびコンピュートノードはサポートされません。
- ファイバーチャネルを使用するバックエンドについては、Block Storage (cinder) のイメージからボリュームへの転送はサポートされません。Red Hat Virtualization は N_Port ID Virtualization (NPIV) をサポートしていません。したがって、ストレージのバックエンドからコントローラー (デフォルトで **cinder-volume** を実行) に LUN をマッピングする必要のある Block Storage (cinder) ドライバーは機能しません。仮想化されたコントローラーに含めるのではなく、**cinder-volume** 専用のロールを作成し、そのロールを使用して物理ノードを作成する必要があります。詳しい情報は、オーバークラウドの高度なカスタマイズの コンポーザブルサービスとカスタムロール を参照してください。

26.2. RED HAT VIRTUALIZATION ドライバーを使用した仮想コントローラーのプロビジョニング

RHOSP および Red Hat Virtualization を使用してオーバークラウドの仮想 RHOSP コントロールプレーンをプロビジョニングするには、以下の手順を実施します。

前提条件

- Intel 64 または AMD64 CPU 拡張機能をサポートする、64 ビット x86 プロセッサーが必要です。
- 以下のソフトウェアがすでにインストールされ、設定されている必要があります。
  - Red Hat Virtualization。詳しい情報は、Red Hat Virtualization ドキュメントスイート を参照してください。
  - Red Hat OpenStack Platform (RHOSP)。詳しい情報は、director のインストールと使用方法 を参照してください。

- 事前に仮想コントローラーノードを準備しておく必要があります。これらの要件は、ベアメタルのコントローラーノードの要件と同じです。詳しい情報は、コントローラーノードの要件 を参照してください。
- オーバークラウドのコンピュートノードおよびストレージノードとして使用するベアメタルノードを、事前に準備しておく必要があります。ハードウェアの仕様については、コンピュートノードの要件 および Ceph Storage ノードの要件 を参照してください。POWER (ppc64le) ハードウェアにオーバークラウドのコンピュートノードをデプロイするには、Red Hat OpenStack Platform for POWER を参照してください。
- 論理ネットワークが作成され、ホストネットワークのクラスターで複数ネットワークによるネットワーク分離を使用する用意ができている必要があります。詳しい情報は、Red Hat Virtualization Administration Guide の Logical Networks を参照してください。
- 各ノードの内部 BIOS クロックを UTC に設定する必要があります。これにより、タイムゾーンオフセットを適用する前に hwclock が BIOS クロックを同期するとファイルのタイムスタンプに未来の日時が設定される問題を防ぐことができます。

ヒント

パフォーマンスのボトルネックを防ぐために、コンポーザブルロールを使用しデータプレーンサービスをベアメタルのコントローラーノード上に維持します。

手順

1. director で **staging-ovirt** ドライバーを有効にするには、**undercloud.conf** 設定ファイルの **enabled_hardware_types** パラメーターにこのドライバーを追加します。

```
enabled_hardware_types = ipmi,redfish,ilo,idrac,staging-ovirt
```

2. アンダークラウドに **staging-ovirt** ドライバーが含まれることを確認します。

```
(undercloud) [stack@undercloud ~]$ openstack baremetal driver list
```

アンダークラウドを正しく設定していれば、コマンドにより以下の結果が返されます。

```
+---------------------+-----------------------+
| Supported driver(s) | Active host(s)        |
+---------------------+-----------------------+
| idrac               | localhost.localdomain |
| ilo                 | localhost.localdomain |
| ipmi                | localhost.localdomain |
| pxe_drac            | localhost.localdomain |
| pxe_ilo             | localhost.localdomain |
| pxe_ipmitool        | localhost.localdomain |
| redfish             | localhost.localdomain |
| staging-ovirt       | localhost.localdomain |
```

3. オーバークラウドノードの定義テンプレート (例: **nodes.json**) を更新し、Red Hat Virtualization がホストする仮想マシンを director に登録します。詳しい情報は、Registering Nodes for the Overcloud を参照してください。以下のキー/値のペアを使用して、オーバークラウドでデプロイする仮想マシンの特性を定義します。

表26.1 オーバークラウド用仮想マシンの設定

キー	この値に設定します
pm_type	oVirt/RHV 仮想マシン用の OpenStack Bare Metal Provisioning (ironic) サービスドライバー **staging-ovirt**
pm_user	Red Hat Virtualization Manager のユーザー名
pm_password	Red Hat Virtualization Manager のパスワード
pm_addr	Red Hat Virtualization Manager サーバーのホスト名または IP
pm_vm_name	コントローラーが作成される Red Hat Virtualization Manager の仮想マシンの名前

以下に例を示します。

```
{
    "nodes": [
        {
            "name":"osp13-controller-0",
            "pm_type":"staging-ovirt",
            "mac":[
                "00:1a:4a:16:01:56"
            ],
            "cpu":"2",
            "memory":"4096",
            "disk":"40",
            "arch":"x86_64",
            "pm_user":"admin@internal",
            "pm_password":"password",
            "pm_addr":"rhvm.example.com",
            "pm_vm_name":"{osp_curr_ver}-controller-0",
            "capabilities": "profile:control,boot_option:local"
        },
        ...
}
```

Red Hat Virtualization Host ごとに 1 つのコントローラーを設定します。

4. Red Hat Virtualization でアフィニティーグループをソフトネガティブアフィニティーに設定し、コントローラー用仮想マシンの高可用性を確保します。詳しい情報は、Red Hat Virtualization Virtual Machine Management Guide の Affinity Groups を参照してください。

5. Red Hat Virtualization Manager のインターフェイスにアクセスし、これを使用してそれぞれの VLAN をコントローラー用仮想マシンの個別の論理仮想 NIC にマッピングします。詳しい情報は、Red Hat Virtualization Administration Guide の Logical Networks を参照してください。

6. director とコントローラー用仮想マシンの仮想 NIC で **no_filter** を設定し、仮想マシンを再起動します。これにより、コントローラー用仮想マシンにアタッチされたネットワークで MAC スプーフィングフィルターが無効化されます。詳しい情報は、Red Hat Virtualization Administration Guide の Virtual Network Interface Cards を参照してください。

7. オーバークラウドをデプロイして、新しい仮想コントローラーノードを環境に追加します。

```
(undercloud) [stack@undercloud ~]$ openstack overcloud deploy --templates
```

第27章 高度なコンテナーイメージ管理の実施

デフォルトのコンテナーイメージ設定は、ほとんどの環境に対応します。状況によっては、コンテナーイメージ設定にバージョンの固定などのカスタマイズが必要になる場合があります。

27.1. アンダークラウド用コンテナーイメージの固定

特定の状況では、アンダークラウド用に特定のコンテナーイメージバージョンのセットが必要な場合があります。そのような場合には、イメージを特定のバージョンに固定する必要があります。イメージに固定するには、コンテナー設定ファイルを生成および変更し、続いてアンダークラウドのロールデータをコンテナー設定ファイルと組み合わせ、サービスとコンテナーイメージのマッピングが含まれる環境ファイルを生成する必要があります。次に、この環境ファイルを **undercloud.conf** ファイルの **custom_env_files** パラメーターに追加します。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **--output-env-file** オプションを指定して **openstack tripleo container image prepare default** コマンドを実行し、デフォルトのイメージ設定が含まれるファイルを生成します。

```
$ sudo openstack tripleo container image prepare default \
--output-env-file undercloud-container-image-prepare.yaml
```

3. 環境の要件に応じて、**undercloud-container-image-prepare.yaml** ファイルを変更します。
   a. **tag:** パラメーターを削除して、director が **tag_from_label:** パラメーターを使用できるようにします。director はこのパラメーターを使用して各コンテナーイメージの最新バージョンを特定し、それぞれのイメージをプルし、director のコンテナーレジストリーの各イメージをタグ付けします。
   b. アンダークラウドの Ceph ラベルを削除します。
   c. **neutron_driver:** パラメーターが空であることを確認します。OVN はアンダークラウドでサポートされないため、このパラメーターを **OVN** に設定しないでください。
   d. コンテナーイメージレジストリーの認証情報を追加します。

```
ContainerImageRegistryCredentials:
  registry.redhat.io
    myser: 'p@55w0rd!'
```

注記

image-serve レジストリーがまだインストールされていないので、新しいアンダークラウドのアンダークラウドレジストリーにコンテナーイメージをプッシュすることはできません。**push_destination** の値を **false** に設定するか、カスタム値を使用して、イメージを直接ソースからプルする必要があります。詳細は、コンテナーイメージ準備のパラメーター を参照してください。

4. カスタムの **undercloud-container-image-prepare.yaml** ファイルと組み合わせたアンダークラウドのロールファイルを使用する、新たなコンテナーイメージ設定ファイルを作成します。

```
$ sudo openstack tripleo container image prepare \
-r /usr/share/openstack-tripleo-heat-templates/roles_data_undercloud.yaml \
-e undercloud-container-image-prepare.yaml \
--output-env-file undercloud-container-images.yaml
```

undercloud-container-images.yaml ファイルは、サービスパラメーターのコンテナーイメージへのマッピングが含まれる環境ファイルです。たとえば、OpenStack Identity (keystone) は、**ContainerKeystoneImage** パラメーターを使用してそのコンテナーイメージを定義します。

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel8/openstack-keystone:16.1.4-5
```

コンテナーイメージタグは **{version}-{release}** 形式に一致することに注意してください。

5. **undercloud.conf** ファイルの **custom_env_files** パラメーターに **undercloud-container-images.yaml** ファイルを追加します。アンダークラウドのインストールを実施する際に、アンダークラウドサービスはこのファイルから固定されたコンテナーイメージのマッピングを使用します。

27.2. オーバークラウド用コンテナーイメージのピニング

特定の状況では、オーバークラウド用に特定のコンテナーイメージバージョンのセットが必要な場合があります。そのような場合には、イメージを特定のバージョンに固定する必要があります。イメージに固定するには、**containers-prepare-parameter.yaml** ファイルを作成して、このファイルを使用してアンダークラウドレジストリーにコンテナーイメージをプルし、固定されたイメージのリストが含まれる環境ファイルを生成する必要があります。

たとえば、**containers-prepare-parameter.yaml** ファイルに以下の内容が含まれる場合があります。

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
      set:
        name_prefix: openstack-
        name_suffix: ''
        namespace: registry.redhat.io/rhosp-rhel8
        neutron_driver: ovn
      tag_from_label: '{version}-{release}'

  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
```

ContainerImagePrepare パラメーターには、単一のルール **set** が含まれます。このルール **set** には **tag** パラメーターを含めないでください。各コンテナーイメージの最新バージョンとリリースを特定するのに、**tag_from_label** パラメーターを使用する必要があります。director はこのルール **set** を使用して各コンテナーイメージの最新バージョンを特定し、それぞれのイメージをプルし、director のコンテナーレジストリーの各イメージをタグ付けします。

手順

1. **openstack tripleo container image prepare** コマンドを実行します。このコマンドは、**containers-prepare-parameter.yaml** ファイルで定義されたソースからすべてのイメージ

をプルします。固定されたコンテナーイメージのリストが含まれる出力ファイルを指定するには、**--output-env-file** を追加します。

```
$ sudo openstack tripleo container image prepare -e /home/stack/templates/containers-prepare-parameter.yaml --output-env-file overcloud-images.yaml
```

overcloud-images.yaml ファイルは、サービスパラメーターのコンテナーイメージへのマッピングが含まれる環境ファイルです。たとえば、OpenStack Identity (keystone) は、**ContainerKeystoneImage** パラメーターを使用してそのコンテナーイメージを定義します。

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel8/openstack-keystone:16.2.4-5
```

コンテナーイメージタグは **{version}-{release}** 形式に一致することに注意してください。

2. **openstack overcloud deploy** コマンドを実行する際に、**containers-prepare-parameter.yaml** および **overcloud-images.yaml** ファイルを特定の順序で環境ファイルコレクションに含めます。

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/containers-prepare-parameter.yaml \
    -e /home/stack/overcloud-images.yaml \
    ...
```

オーバークラウドサービスは、**overcloud-images.yaml** ファイルにリスト表示されている固定されたイメージを使用します。

第28章 DIRECTOR のエラーに関する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、典型的な問題の診断について説明します。

28.1. ノードの登録に関する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、ノードの情報が含まれるテンプレートファイルを検証して、インポートされたノードの情報を修正します。

手順

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. **--validate-only** オプションを指定して、ノードのインポートコマンドを実行します。このオプションを指定した場合には、インポートを実施せずにノードのテンプレートを検証します。

```
(undercloud) $ openstack overcloud node import --validate-only ~/nodes.json
Waiting for messages on queue 'tripleo' with no timeout.

Successfully validated environment file
```

3. インポートしたノードの誤った情報を修正するには、**openstack baremetal** コマンドを実行してノードの情報を更新します。ネットワーク設定の詳細を変更する方法を、以下の例に示します。

   a. インポートしたノードに割り当てられたポートの UUID を特定します。

```
$ source ~/stackrc
(undercloud) $ openstack baremetal port list --node [NODE UUID]
```

   b. MAC アドレスを更新します。

```
(undercloud) $ openstack baremetal port set --address=[NEW MAC] [PORT UUID]
```

   c. ノードの新しい IPMI アドレスを設定します。

```
(undercloud) $ openstack baremetal node set --driver-info ipmi_address=[NEW IPMI ADDRESS] [NODE UUID]
```

28.2. ハードウェアのイントロスペクションに関するトラブルシューティング

イントロスペクションのプロセスは最後まで実行する必要があります。ただし、イントロスペクション ramdisk が応答しない場合には、**ironic-inspector** がデフォルトの 1 時間が経過した後にタイムアウトします。イントロスペクション ramdisk にバグがあることを示している可能性もありますが、通常は環境の設定ミス (特に、BIOS のブート設定) によりこのタイムアウトが発生します。

典型的な環境の誤設定の問題を診断して解決するには、以下の手順を実施します。

手順

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. director は OpenStack Object Storage (swift) を使用して、イントロスペクションプロセス中に取得するハードウェアデータを保存します。このサービスが稼働していない場合には、イントロスペクションは失敗する場合があります。以下のコマンドを実行して、OpenStack Object Storage に関連したサービスをすべてチェックし、このサービスが稼働中であることを確認します。

```
(undercloud) $ sudo systemctl list-units tripleo_swift*
```

3. ノードを **manageable** の状態にします。イントロスペクションは、デプロイメント用を意味する **available** の状態にあるノードを検査しません。**available** の状態にあるノードを検査するには、イントロスペクションの前にノードのステータスを **manageable** の状態に変更します。

```
(undercloud) $ openstack baremetal node manage [NODE UUID]
```

4. イントロスペクション ramdisk への一時的なアクセスを設定します。一時パスワードまたは SSH 鍵のいずれかを指定して、イントロスペクションのデバッグ中にノードにアクセスすることができます。ramdisk へのアクセスを設定するには、以下の手順を実施します。

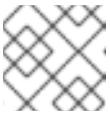
   a. 一時パスワードを指定して **openssl passwd -1** コマンドを実行し、MD5 ハッシュを生成します。

```
(undercloud) $ openssl passwd -1 mytestpassword
$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/
```

   b. **/var/lib/ironic/httpboot/inspector.ipxe** ファイルを編集し、**kernel** で始まる行を特定し、**rootpwd** パラメーターおよび MD5 ハッシュを追記します。

```
kernel http://192.2.0.1:8088/agent.kernel ipa-inspection-callback-
url=http://192.168.0.1:5050/v1/continue ipa-inspection-collectors=default,extra-
hardware,logs systemd.journald.forward_to_console=yes BOOTIF=${mac} ipa-debug=1
ipa-inspection-benchmarks=cpu,mem,disk
rootpwd="$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/" selinux=0
```

   あるいは、**sshkey** パラメーターに公開 SSH 鍵 を追加します。



注記

rootpwd および **sshkey** パラメーターには、いずれも引用符を含めます。

5. ノード上でイントロスペクションを実行します。

```
(undercloud) $ openstack overcloud node introspect [NODE UUID] --provide
```

   イントロスペクションの完了後にノードの状態を **available** に変更するには、**--provide** オプションを使用します。

6. **dnsmasq** ログでノードの IP アドレスを特定します。

```
(undercloud) $ sudo tail -f /var/log/containers/ironic-inspector/dnsmasq.log
```

7. エラーが発生する場合には、root ユーザーおよび一時アクセス用の詳細を使用してノードにアクセスします。

```
$ ssh root@192.168.24.105
```

イントロスペクション中にノードにアクセスして、診断コマンドを実行してイントロスペクションの失敗に関するトラブルシューティングを行います。

8. イントロスペクションのプロセスを停止するには、以下のコマンドを実行します。

```
(undercloud) $ openstack baremetal introspection abort [NODE UUID]
```

プロセスがタイムアウトするまで待つことも可能です。

注記

イントロスペクションの最初の中止後、Red Hat OpenStack Platform director は実行を 3 回試みます。イントロスペクションを完全に中止するには、それぞれの試行時に **openstack baremetal introspection abort** コマンドを実行します。

28.3. ワークフローおよび実行に関するトラブルシューティング

OpenStack Workflow (mistral) サービスは、複数の OpenStack タスクをワークフローにグループ化します。Red Hat OpenStack Platform は、これらのワークフローのセットを使用して、director 全体を通じて共通の機能を実行します。これには、ベアメタルノードの制御、検証、プラン管理、およびオーバークラウドのデプロイメントが含まれます。

たとえば **openstack overcloud deploy** コマンドを実行すると、OpenStack Workflow サービスは 2 つのワークフローを実行します。最初のワークフローは、デプロイメントプランをアップロードします。

```
Removing the current plan files
Uploading new plan files
Started Mistral Workflow. Execution ID: aef1e8c6-a862-42de-8bce-073744ed5e6b
Plan updated
```

2 つ目のワークフローは、オーバークラウドのデプロイメントを開始します。

```
Deploying templates in the directory /tmp/tripleoclient-LhRlHX/tripleo-heat-templates
Started Mistral Workflow. Execution ID: 97b64abe-d8fc-414a-837a-1380631c764d
2016-11-28 06:29:26Z [overcloud]: CREATE_IN_PROGRESS  Stack CREATE started
2016-11-28 06:29:26Z [overcloud.Networks]: CREATE_IN_PROGRESS  state changed
2016-11-28 06:29:26Z [overcloud.HeatAuthEncryptionKey]: CREATE_IN_PROGRESS  state
changed
2016-11-28 06:29:26Z [overcloud.ServiceNetMap]: CREATE_IN_PROGRESS  state changed
...
```

OpenStack Workflow サービスは、以下のオブジェクトを使用してワークフローを追跡します。

アクション

関連タスクが実行される際に OpenStack が実施する特定の指示。これには、シェルスクリプトの実行や HTTP リクエストの実行などが含まれます。OpenStack の一部のコンポーネントには、OpenStack Workflow が使用するアクションが組み込まれています。

タスク

実行するアクションと、アクションの実行後の結果を定義します。これらのタスクには通常、アクションまたはアクションに関連付けられたワークフローが含まれます。タスクが完了すると、ワークフローは、タスクが成功したか否かによって、別のタスクに指示を出します。

ワークフロー

グループ化されて特定の順番で実行されるタスクのセット

実行

実行する特定のアクション、タスク、またはワークフローを定義します。

また、OpenStack Workflow では、実行が確実にログとして記録されるので、特定のコマンドが失敗した場合に問題を特定しやすくなります。たとえば、ワークフローの実行に失敗した場合には、どの部分で失敗したかを特定することができます。

手順

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. 失敗した状態 (**ERROR**) のワークフローの実行をリスト表示します。

```
(undercloud) $ openstack workflow execution list | grep "ERROR"
```

3. 失敗したワークフロー実行の UUID を取得して (例: **dffa96b0-f679-4cd2-a490-4769a3825262**)、実行と出力を表示します。

```
(undercloud) $ openstack workflow execution show dffa96b0-f679-4cd2-a490-4769a3825262
(undercloud) $ openstack workflow execution output show dffa96b0-f679-4cd2-a490-4769a3825262
```

4. これらのコマンドにより、実行に失敗したタスクに関する情報が返されます。**openstack workflow execution show** コマンドは、実行に使用したワークフローも表示します (例: **tripleo.plan_management.v1.publish_ui_logs_to_swift**)。以下のコマンドを使用して完全なワークフロー定義を表示することができます。

```
(undercloud) $ openstack workflow definition show
tripleo.plan_management.v1.publish_ui_logs_to_swift
```

これは、特定のタスクがワークフローのどの部分で発生するかを特定する際に便利です。

5. 同様のコマンド構文を使用して、アクションの実行とその結果を表示します。

```
(undercloud) $ openstack action execution list
(undercloud) $ openstack action execution show 8a68eba3-0fec-4b2a-adc9-5561b007e886
(undercloud) $ openstack action execution output show 8a68eba3-0fec-4b2a-adc9-5561b007e886
```

これは、問題を引き起こす具体的なアクションを特定する際に便利です。

28.4. オーバークラウドの作成およびデプロイメントに関するトラブルシューティング

オーバークラウドの初回作成は、OpenStack Orchestration (heat) サービスにより実行されます。オーバークラウドのデプロイメントに失敗した場合には、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。

手順

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. デプロイメントのエラー発生ステップ確認コマンドを実行します。

```
$ openstack overcloud failures
```

3. 以下のコマンドを実行して、エラーの詳細を表示します。

```
(undercloud) $ openstack stack failures list <OVERCLOUD_NAME> --long
```

- **<OVERCLOUD_NAME>** を実際のオーバークラウドの名前に置き換えてください。

4. 以下のコマンドを実行し、エラーが発生したスタックを特定します。

```
(undercloud) $ openstack stack list --nested --property status=FAILED
```

28.5. ノードのプロビジョニングに関するトラブルシューティング

OpenStack Orchestration (heat) サービスは、プロビジョニングプロセスを制御します。ノードのプロビジョニングに失敗した場合には、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、問題の診断を行います。

手順

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. Bare Metal サービスをチェックして、全登録ノードおよびそれらの現在の状態を表示します。

```
(undercloud) $ openstack baremetal 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       |
+----------+------+---------------+-------------+-----------------+-------------+
```

プロビジョニングに使用できるすべてのノードは、以下の状態でなければなりません。

- Maintenance: **False** に設定。
- Provision State: プロビジョニングの前に **available** に設定。

3. ノードの **Maintenance** が **False** に設定されていない場合、または **Provision State** が **available** に設定されていない場合は、次の表を使用して問題と解決策を特定します。

問題	原因	ソリューション
Maintenance が自動的に **True** に設定される。	director がノードの電源管理にアクセスすることができない。	ノードの電源管理の認証情報を確認します。
Provision State は **available** に設定されているが、ノードがプロビジョニングされない。	ベアメタルのデプロイメントが開始される前に問題が生じた。	プロファイルおよびフレーバーのマッピングを含め、ノードの詳細を確認します。ノードのハードウェア詳細がフレーバーの要件を満たしていることを確認します。
Provision State がノードの **wait call-back** に設定される。	このノードのプロビジョニングプロセスがまだ終了していない。	このステータスが変わるまで待ちます。あるいは、ノードの仮想コンソールに接続し、出力を確認します。
Provision State および Power State はそれぞれ **active** および **power on** だが、ノードが応答しない。	ノードのプロビジョニングは正常に終了したが、デプロイメント後の設定ステップで問題が生じている。	ノード設定のプロセスを診断します。ノードの仮想コンソールに接続し、出力を確認します。
Provision State が **error** または **deploy failed** である。	ノードのプロビジョニングに失敗している。	**openstack baremetal node show** コマンドを使用してベアメタルノードの詳細を表示し、エラーに関する説明が含まれる **last_error** フィールドを確認します。

関連情報

- ベアメタルノードのプロビジョニング状態

28.6. プロビジョニング時の IP アドレス競合に関するトラブルシューティング

対象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、イントロスペクションおよびデプロイメントのタスクは失敗します。このタスク失敗を防ぐには、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します。

手順

1. **nmap** をインストールします。

```
$ sudo dnf 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、192.168.24.0/24 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)。

```
$ sudo nmap -sn 192.168.24.0/24
```

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。

```
$ sudo nmap -sn 192.168.24.0/24

Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-02 15:14 EDT
Nmap scan report for 192.168.24.1
Host is up (0.00057s latency).
Nmap scan report for 192.168.24.2
Host is up (0.00048s latency).
Nmap scan report for 192.168.24.3
Host is up (0.00045s latency).
Nmap scan report for 192.168.24.5
Host is up (0.00040s latency).
Nmap scan report for 192.168.24.9
Host is up (0.00019s latency).
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.45 seconds
```

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。

28.7. NO VALID HOST FOUND エラーに関する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.
```

このエラーは、Compute Scheduler が新規インスタンスをブートするのに適したベアメタルノードを検出できない場合に発生します。通常これは、Compute サービスが検出を想定しているリソースと、Bare Metal サービスが Compute に通知するリソースが、一致していないことを意味します。不一致によるエラーがあることを確認するには、以下の手順を実施します。

手順

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. ノードのイントロスペクションが成功したことを確認します。イントロスペクションが失敗する場合には、各ノードに必要な ironic ノードの属性が含まれていることを確認してください。

```
(undercloud) $ openstack baremetal node show [NODE UUID]
```

properties JSON フィールドの **cpus**、**cpu_arch**、**memory_mb**、および **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。

3. ノードにマッピングされた Compute フレーバーが、必要なノード数のノード属性を超えないようにしてください。

```
(undercloud) $ openstack flavor show [FLAVOR NAME]
```

4. **openstack baremetal node list** コマンドを実行して、available の状態にあるノードが十分にあることを確認します。ノードの状態が **manageable** の場合には、通常イントロスペクションに失敗します。

5. **openstack baremetal node list** コマンドを実行し、ノードがメンテナンスモードに設定されていないことを確認します。ノードが自動的にメンテナンスモードに切り替わる場合には、電源管理の認証情報が間違っていることが一般的な原因として考えられます。電源管理の認証情報を確認し、メンテナンスモードを解除します。

```
(undercloud) $ openstack baremetal node maintenance unset [NODE UUID]
```

6. プロファイルの自動タグ付けを使用している場合には、各フレーバー/プロファイルに対応するノードが十分に存在することを確認します。ノードで **openstack baremetal node show** コマンドを実行し、**properties** フィールドの **capabilities** キーを確認します。たとえば、Compute ロールにタグ付けされたノードには、**profile:compute** の値が含まれます。

7. イントロスペクションの後にノードの情報が Bare Metal から Compute に反映されるのを待つ必要があります。ただし、一部のステップを手動で実行した場合、短時間 Compute サービス (nova) がノードを利用できない状態になる可能性があります。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。

```
(undercloud) $ openstack hypervisor stats show
```

28.8. オーバークラウドの設定に関するトラブルシューティング

Red Hat OpenStack Platform director は、Ansible を使用してオーバークラウドを設定します。オーバークラウドでの Ansible Playbook のエラー (**config-download**) を診断するには、以下の手順を実施します。

手順

1. **stack** ユーザーが **アンダークラウド** 上の **/var/lib/mistral** ディレクトリー内のファイルにアクセスできるようにします。

```
$ sudo setfacl -R -m u:stack:rwx /var/lib/mistral
```

このコマンドを実行しても、**mistral** ユーザーのディレクトリーへのアクセス権限は維持されます。

2. **config-download** ファイルの作業ディレクトリーに移動します。通常これは **/var/lib/mistral/overcloud/** です。

```
$ cd /var/lib/mistral/overcloud/
```

3. **ansible.log** ファイルを確認し、異常のあった場所を探します。

```
$ less ansible.log
```

異常のあったステップを書き留めます。

4. 作業ディレクトリー内の **config-download** Playbook で異常のあったステップを探し、実行していたアクションを特定します。

28.9. コンテナーの設定に関するトラブルシューティング

Red Hat OpenStack Platform director は、**paunch** を使用してコンテナーを起動し、**podman** を使用してコンテナーを管理し、また **puppet** を使用してコンテナー設定を作成します。以下の手順で、エラーが発生した場合にコンテナーを診断する方法について説明します。

ホストへのアクセス

1. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

2. 障害の発生したコンテナーがあるノードの IP アドレスを取得します。

```
(undercloud) $ openstack server list
```

3. ノードにログインします。

```
(undercloud) $ ssh heat-admin@192.168.24.60
```

4. root ユーザーに変更します。

```
$ sudo -i
```

障害が発生したコンテナーの識別

1. すべてのコンテナーを表示します。

```
$ podman ps --all
```

障害の発生したコンテナーを特定します。障害の発生したコンテナーは、通常ゼロ以外のステータスで終了します。

コンテナーログの確認

1. 各コンテナーは、主要プロセスからの標準出力を保持します。この出力をログとして使用し、コンテナー実行時に実際に何が発生したのかを特定するのに役立てます。たとえば、**keystone** コンテナーのログを確認するには、以下のコマンドを実行します。

```
$ sudo podman logs keystone
```

多くの場合、このログにコンテナー障害の原因に関する情報が含まれます。

2. ホストには、失敗したサービスの **stdout** ログも保持されます。**stdout** ログは、**/var/log/containers/stdouts/** に保存されます。たとえば、障害の発生した **keystone** コンテナーのログを確認するには、以下のコマンドを使用します。

```
$ cat /var/log/containers/stdouts/keystone.log
```

コンテナーの検査

状況によっては、コンテナーに関する情報を検証する必要がある場合があります。たとえば、以下のコマンドを使用して **keystone** コンテナーのデータを確認します。

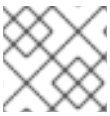
```
$ sudo podman inspect keystone
```

このコマンドにより、ローレベルの設定データが含まれた JSON オブジェクトが返されます。その出力を **jq** コマンドにパイプして、特定のデータを解析することが可能です。たとえば、**keystone** コンテナーのマウントを確認するには、以下のコマンドを実行します。

```
$ sudo podman inspect keystone | jq .[0].Mounts
```

--format オプションを使用して、データを単一行に解析することもできます。これは、コンテナーデータのセットに対してコマンドを実行する場合に役立ちます。たとえば、**keystone** コンテナーを実行するのに使用するオプションを再生成するには、以下のように **inspect** コマンドに **--format** オプションを指定して実行します。

```
$ sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range .Mounts}} -v
{{.Source}}:{{.Destination}}:{{ join .Options "," }}{{end}} -ti {{.Config.Image}}' keystone
```



注記

--format オプションは、Go 構文を使用してクエリーを作成します。

これらのオプションを **podman run** コマンドと共に使用して、トラブルシューティング目的のコンテナーを再度作成します。

```
$ OPTIONS=$( sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range
.Mounts}} -v {{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}'
keystone )
$ sudo podman run --rm $OPTIONS /bin/bash
```

コンテナー内でのコマンドの実行

状況によっては、特定の Bash コマンドでコンテナー内の情報を取得する必要がある場合があります。このような場合には、以下の **podman** コマンドを使用して、稼働中のコンテナー内でコマンドを実行します。たとえば、**podman exec** コマンドを実行して、**keystone** コンテナー内でコマンドを実行します。

```
$ sudo podman exec -ti keystone <COMMAND>
```

注記

-ti オプションを指定すると、コマンドは対話式の擬似ターミナルで実行されます。

- **<COMMAND>** を実行するコマンドに置き換えてください。たとえば、各コンテナーには、サービスの接続を確認するためのヘルスチェックスクリプトがあります。**keystone** にヘルスチェックスクリプトを実行するには、以下のコマンドを実行します。

```
$ sudo podman exec -ti keystone /openstack/healthcheck
```

コンテナーのシェルにアクセスするには、コンテナー内で実行するコマンドとして **/bin/bash** を使用して **podman exec** を実行します。

```
$ sudo podman exec -ti keystone /bin/bash
```

コンテナーファイルシステムの確認

1. 障害の発生したコンテナーのファイルシステムを確認するには、**podman mount** コマンドを実行します。たとえば、障害の発生した **keystone** コンテナーのファイルシステムを確認するには、以下のコマンドを実行します。

   ```
   $ podman mount keystone
   ```

   これによりマウント位置が表示され、ファイルシステムの内容を確認することができます。

   ```
   /var/lib/containers/storage/overlay/78946a109085aeb8b3a350fc20bd8049a08918d74f573396d7358270e711c610/merged
   ```

   これは、コンテナー内の Puppet レポートを確認する際に役立ちます。これらのレポートは、コンテナーのマウント内の **var/lib/puppet/** ディレクトリーにあります。

コンテナーのエクスポート

コンテナーに障害が発生した場合には、ファイルの内容を詳細に調べる必要があります。この場合は、コンテナーの全ファイルシステムを **tar** アーカイブとしてエクスポートすることができます。たとえば、**keystone** コンテナーのファイルシステムをエクスポートするには、以下のコマンドを実行します。

```
$ sudo podman export keystone -o keystone.tar
```

このコマンドにより **keystone.tar** アーカイブが作成されます。これを抽出して、調べることができます。

28.10. コンピュートノードの異常に関するトラブルシューティング

コンピュートノードは、Compute サービスを使用して、ハイパーバイザーベースの操作を実行します。これは、このサービスを中心にコンピュートノードのメインの診断が行われていることを意味します。

手順

1. **stackrc** ファイルを取得します。

   ```
   $ source ~/stackrc
   ```

2. 障害が発生したコンピュートノードの IP アドレスを取得します。

   ```
   (undercloud) $ openstack server list
   ```

3. ノードにログインします。

```
(undercloud) $ ssh heat-admin@192.168.24.60
```

4. root ユーザーに変更します。

```
$ sudo -i
```

5. コンテナーのステータスを表示します。

```
$ sudo podman ps -f name=nova_compute
```

6. コンピュートノードの主なログファイルは **/var/log/containers/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、このファイルを使用して診断を始めます。

7. コンピュートノードでメンテナンスを実行する場合には、既存のインスタンスをホストから稼働中のコンピュートノードに移行し、ノードを無効にします。

28.11. SOS レポートの作成

Red Hat に連絡して Red Hat OpenStack Platform に関するサポートを受ける必要がある場合は、**SOS レポート** を生成しなければならない場合があります。**SOS レポート** 作成についての詳しい情報は、以下のドキュメントを参照してください。

- How to collect all required logs for Red Hat Support to investigate an OpenStack issue

28.12. ログの場所

問題のトラブルシューティングを行う場合、以下のログを使用してアンダークラウドおよびオーバークラウドについての情報を収集します。

表28.1 アンダークラウドおよびオーバークラウドノード両方に関するログ

情報	ログの場所
コンテナー化されたサービスに関するログ	**/var/log/containers/**
コンテナー化されたサービスからの標準出力	**/var/log/containers/stdouts**
Ansible の設定に関するログ	**/var/lib/mistral/overcloud/ansible.log**

表28.2 アンダークラウドノードに関する追加のログ

情報	ログの場所
openstack overcloud deploy のコマンド履歴	**/home/stack/.tripleo/history**
アンダークラウドのインストールに関するログ	**/home/stack/install-undercloud.log**

表28.3 オーバークラウドノードに関する追加のログ

情報	ログの場所
cloud-init に関するログ	**/var/log/cloud-init.log**
高可用性に関するログ	**/var/log/pacemaker.log**

第29章 アンダークラウドおよびオーバークラウドサービスについてのヒント

本項では、アンダークラウド上の特定の OpenStack サービスのチューニングと管理についてアドバイスを行います。

29.1. デプロイメントパフォーマンスのチューニング

Red Hat OpenStack Platform director は、OpenStack Orchestration (heat) を使用してメインのデプロイメントおよびプロビジョニング機能を実施します。heat は一連のワーカーを使用してデプロイメントタスクを実行します。デフォルトのワーカー数を計算するには、director の heat 設定ではアンダークラウドの合計 CPU スレッド数を 2 で割ります。ここでは、スレッド数とは CPU コア数にハイパースレッディングの値を掛けたものを指します。たとえば、アンダークラウドの CPU スレッド数が 16 の場合には、デフォルトでは heat により 8 つのワーカーが提供されます。デフォルトでは、director の設定に最小および最大のワーカー数も適用されます。

サービス	最小値	最大値
OpenStack Orchestration (heat)	4	24

ただし、環境ファイルの **HeatWorkers** パラメーターを使用して、手動でワーカー数を設定することができます。

heat-workers.yaml

```
parameter_defaults:
  HeatWorkers: 16
```

undercloud.conf

```
custom_env_files: heat-workers.yaml
```

29.2. コンテナーでの SWIFT-RING-BUILDER の実行

Object Storage (swift) リングを管理するには、サーバーコンテナー内で **swift-ring-builder** コマンドを使用します。

- **swift_object_server**
- **swift_container_server**
- **swift_account_server**

たとえば、swift オブジェクトリングの情報を表示するには、以下のコマンドを実行します。

```
$ sudo podman exec -ti -u swift swift_object_server swift-ring-builder /etc/swift/object.builder
```

アンダークラウドおよびオーバークラウドノードの両方で、このコマンドを実行することができます。

29.3. HAPROXY の SSL/TLS 暗号ルールの変更

アンダークラウドで SSL/TLS を有効にした場合には (「director の設定パラメーター」 を参照)、HAProxy 設定で使用する SSL/TLS の暗号とルールを強化することを推奨します。この強化は、POODLE TLS 脆弱性 などの SSL/TLS の脆弱性を回避するのに役立ちます。

hieradata_override のアンダークラウド設定オプションを使用して、以下の hieradata を設定します。

tripleo::haproxy::ssl_cipher_suite

HAProxy で使用する暗号スイート

tripleo::haproxy::ssl_options

HAProxy で使用する SSL/TLS ルール

たとえば、以下のような暗号およびルールを使用することができます。

- 暗号: **ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS**
- ルール: **no-sslv3 no-tls-tickets**

hieradata オーバーライドファイル (**haproxy-hiera-overrides.yaml**) を作成して、以下の内容を記載します。

```
tripleo::haproxy::ssl_cipher_suite: ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS
tripleo::haproxy::ssl_options: no-sslv3 no-tls-tickets
```

注記

暗号のコレクションは、改行せずに 1 行に記述します。

openstack undercloud install を実行する前に作成した hierradata オーバーライドファイルを使用するように、**undercloud.conf** ファイルの **hieradata_override** パラメーターを設定します。

```
[DEFAULT]
...
hieradata_override = haproxy-hiera-overrides.yaml
...
```

第30章 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。この付録には、director がサポートする電源管理機能のリストが記載されています。オーバークラウドのノードを登録する際に、これらの電源管理設定を使用します。詳しい情報は、Registering nodes for the overcloud を参照してください。

30.1. INTELLIGENT PLATFORM MANAGEMENT INTERFACE (IPMI)

Baseboard Management Controller (BMC) を使用する際の標準的な電源管理手法

pm_type
: このオプションを **ipmi** に設定します。

pm_user、pm_password
: IPMI のユーザー名およびパスワード

pm_addr
: IPMI コントローラーの IP アドレス

pm_port (オプション)
: IPMI コントローラーに接続するためのポート

30.2. REDFISH

Distributed Management Task Force (DMTF) の開発した、IT インフラストラクチャー向け標準 RESTful API

pm_type
: このオプションを **redfish** に設定します。

pm_user、pm_password
: Redfish のユーザー名およびパスワード

pm_addr
: Redfish コントローラーの IP アドレス

pm_system_id
: システムリソースへの正規パス。このパスには、そのシステムの root サービス、バージョン、パス/一意 ID を含める必要があります。たとえば、**/redfish/v1/Systems/CX34R87**。

redfish_verify_ca
: ベースボード管理コントローラー (BMC) の Redfish サービスが、認識済み認証局 (CA) により署名された有効な TLS 証明書を使用するように設定されていない場合、ironic の Redfish クライアントは BMC への接続に失敗します。**redfish_verify_ca** オプションを **false** に設定して、エラーをミュートします。ただし、BMC 認証を無効にすると、BMC のアクセスセキュリティーが低下するので、注意してください。

30.3. 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イスです。

pm_type
: このオプションを **idrac** 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

30.4. 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イスです。

pm_type

このオプションを **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 インターフェイスの IP アドレス

- このドライバーを有効にするには、**undercloud.conf** の **enabled_hardware_types** オプションに **ilo** を追加してから、**openstack undercloud install** を再実行します。
- 正常にイントロスペクションを実施するためには、HP ノードの ILO ファームウェアバージョンは、最低でも 1.85 (2015 年 5 月 13 日版) でなければなりません。この ILO ファームウェアバージョンを使用するノードで、director は正常にテストされています。
- 共有 iLO ポートの使用はサポートされません。

30.5. FUJITSU INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC)

Fujitsu 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 **irmc** に設定します。

pm_user、pm_password

iRMC インターフェイスのユーザー名とパスワード

pm_addr

iRMC インターフェイスの 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 **scci** です。デフォルトは **ipmitool** です。

- このドライバーを有効にするには、**undercloud.conf** の **enabled_hardware_types** オプションに **irmc** を追加してから、**openstack undercloud install** コマンドを再実行します。

30.6. RED HAT VIRTUALIZATION

このドライバーにより、RESTful API を介して、Red Hat Virtualization (RHV) 内の仮想マシンを制御することができるようになります。

pm_type

このオプションを **staging-ovirt** に設定します。

pm_user、pm_password

RHV 環境のユーザー名およびパスワード。ユーザー名には、認証プロバイダーも含めます。たとえば、**admin@internal**。

pm_addr

RHV REST API の IP アドレス

pm_vm_name

制御する仮想マシンの名前

mac

ノード上のネットワークインターフェイスの MAC アドレスリスト。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。

- このドライバーを有効にするには、**undercloud.conf** の **enabled_hardware_types** オプションに **staging-ovirt** を追加してから、**openstack undercloud install** コマンドを再実行します。

30.7. 手動管理ドライバー

電源管理を持たないベアメタルデバイスを制御するには、**manual-management** ドライバーを使用します。director は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およびデプロイメントの特定の時点で、手動で電源管理操作を実施する必要があります。

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することができます。Red Hat OpenStack Platform のエンタープライズ環境には推奨していません。

pm_type

このオプションを **manual-management** に設定します。

- このドライバーは、電源管理を制御しないので、認証情報は使用しません。
- このドライバーを有効にするには、**undercloud.conf** の **enabled_hardware_types** オプションに **manual-management** を追加してから、**openstack undercloud install** コマンドを再実行します。

- **instackenv.json** ノードインベントリーファイルで、手動で管理するノードの **pm_type** を **manual-management** に設定します。

イントロスペクション

- ノードのイントロスペクションを実行する際には、**openstack overcloud node introspect** コマンドを実行した後に手動でノードを起動します。ノードが PXE を介して起動することを確認します。
- ノードクリーニングを有効にしている場合は、**openstack baremetal node list** コマンドを実行した際に **Introspection completed** メッセージが表示され、各ノードの状態が **clean wait** になった後、手動でノードを再起動するようにしてください。ノードが PXE を介して起動することを確認します。
- イントロスペクションとクリーニングのプロセスが完了したら、ノードをシャットダウンします。

Deployment

- オーバークラウドデプロイメントを実行する場合は、**openstack baremetal node list** コマンドを使用してノードのステータスを確認してください。ノードのステータスが **deploying** から **wait call-back** に変わるまで待ってから、ノードを手動で開始します。ノードが PXE を介して起動することを確認します。
- オーバークラウドプロビジョニングプロセスが完了すると、ノードはシャットダウンします。設定プロセスを開始するには、ディスクからノードを起動する必要があります。プロビジョニングの完了を確認するには、**openstack baremetal node list** コマンドを使用してノードのステータスを確認し、ノードのステータスが各ノードで **active** に変わるまで待ちます。ノードステータスが **active** な場合、プロビジョニングされたオーバークラウドノードを手動で起動します。