Red Hat OpenStack Platform 9

ベアメタルプロビジョニング

Bare Metal Provisioning (Ironic) のインストール、設定、使用方法

Last Updated: 2017-09-18

Red Hat OpenStack Platform 9 ベアメタルプロビジョニング

Bare Metal Provisioning (Ironic) のインストール、設定、使用方法

OpenStack Team
rhos-docs@redhat.com

法律上の通知

概要

本ガイドには、Red Hat OpenStack Platform 環境のオーバークラウドにおける Bare Metal Provisioning をインストール、設定、使用するための手順を記載しています。

目次

前書き

本ガイドは、OpenStack サービスとしてオーバークラウドに Bare Metal Provisioning (ironic) をインストールして設定し、そのサービスを使用してエンドユーザー向けの物理マシンのプロビジョニングと管理を行う手順を説明します。このサービスを設定することにより、ユーザーは仮想マシンインスタンスを起動するのと同じ方法で物理マシン上のインスタンスを起動することができるようになります。

Bare Metal Provisioning のコンポーネントは、Red Hat OpenStack Platform director で OpenStack 環境 (オーバークラウド) を構成するベアメタルノードのプロビジョニングと管理を行うためにアンダークラウドの一部としても使用されます。director での Bare Metal Provisioning の使用については、『director のインストールと使用方法』を参照してください。

注記

本ガイドでは、すでにデプロイ済みのオーバークラウド上に Bare Metal Provisioning を手動でインストールする方法について説明します。この手動による手順は、Red Hat OpenStack Platform director の 10 以降のバージョンに統合された Bare Metal Provisioning の方法とは異なります。このプロセスを実行したオーバークラウドのアップグレードはサポートされていないため、アップグレードを試みることによってどのような結果をもたらすかは明らかではありません。本書は、Red Hat OpenStack Platform director の 10 以降のバージョンに統合された機能をプレビューするためのガイドとして利用してください。

第1章 OPENSTACK BARE METAL PROVISIONING (IRONIC) のインストールと設定

OpenStack Bare Metal Provisioning (ironic) は、エンドユーザー向けの物理 (ベアメタル) マシンのプロビジョニングと管理に必要なコンポーネントを提供します。オーバークラウド内の Bare Metal Provisioning は、以下の OpenStack サービスと対話します。

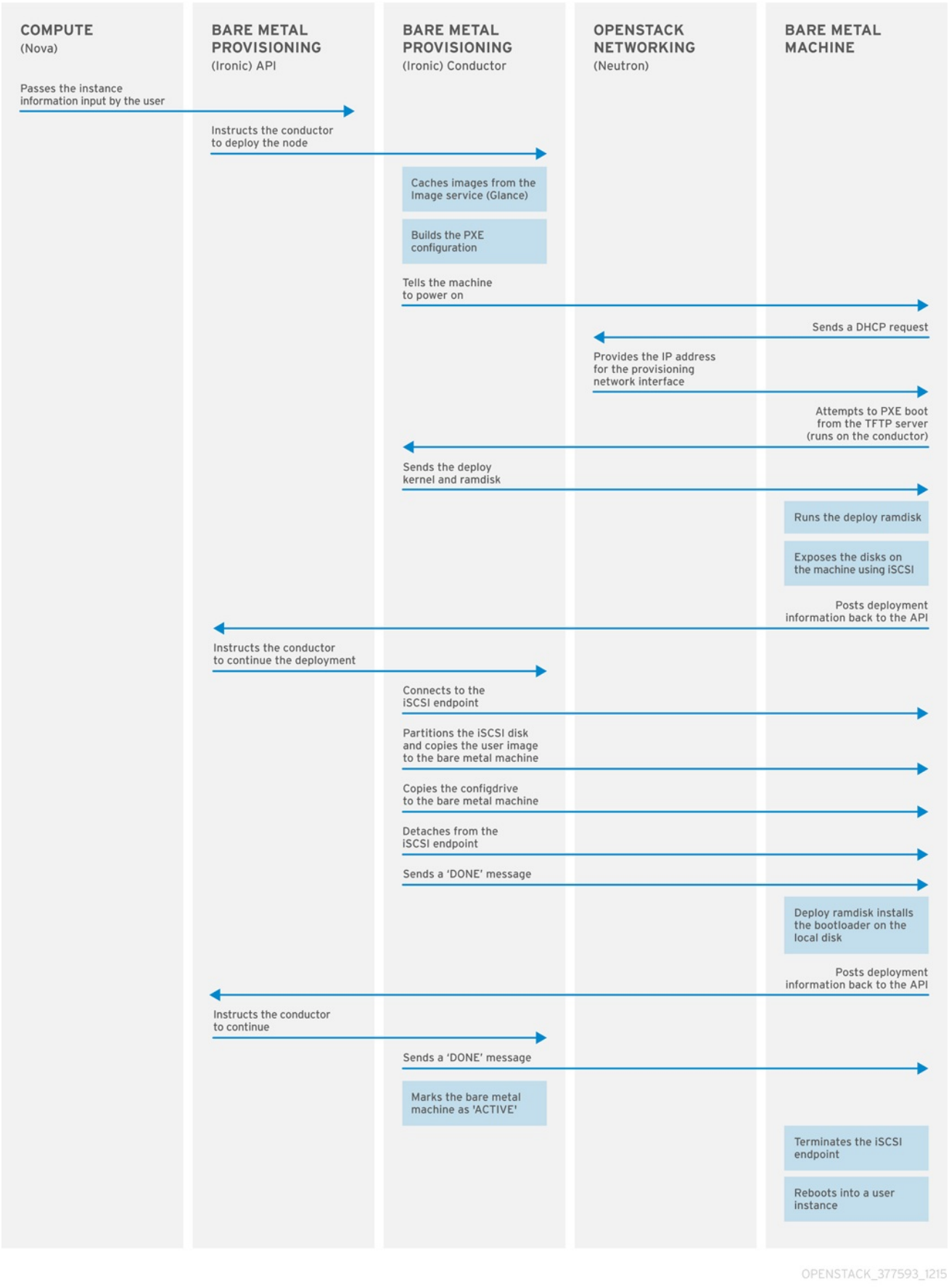
- OpenStack Compute (nova) は、スケジューリング、テナントレベルのクォータ設定、IP の割り当ての機能と、仮想マシンインスタンスを管理するためのユーザー向けの API を提供します。一方、Bare Metal Provisioning は、ハードウェア管理のための管理 API を提供します。Bare Metal Provisioning ドライバーを使用して Bare Metal Provisioning の要求を処理するには、専用の **openstack-nova-compute** ホストを 1 台選択してください。

- OpenStack Identity (keystone) は、要求の認証機能を提供し、Bare Metal Provisioning が他の OpenStack サービスを特定するのを補助します。

- OpenStack Image サービス (glance) は、ベアメタルマシンの起動に使用するイメージとイメージのメタデータを管理します。

- OpenStack Networking (neutron) は、必須のベアメタルプロビジョニングネットワークの DHCP とネットワーク設定を提供します。

Bare Metal Provisioning は、PXE を使用して物理マシンをプロビジョニングします。以下の図は、ユーザーが新規ベアメタルマシンを起動した場合に、プロビジョニングプロセス中に OpenStack サービスがどのように対話するかを示しています。



1.1. 要件

本章では、インストールの前提条件、ハードウェア要件、ネットワーク要件など、Bare Metal Provisioning を設定するための要件について説明します。

1.1.1. Bare Metal Provisioning のインストールの前提条件

Bare Metal Provisioning は、同じノードまたは別のノードで実行するように設定することが可能なコンポーネントのコレクションです。本ガイドの構成例では、全 Bare Metal Provisioning コンポーネントを単一のノード上にインストールします。本ガイドでは、OpenStack Identity、OpenStack Image、OpenStack Compute、OpenStack Networking のサービスがインストール/設定済みであることを前提としています。Bare Metal Provisioning には、以下の外部サービスも必須となるため、事前にインストール/設定しておく必要があります。

- ハードウェア情報と状態を保管するデータベースサーバー。本ガイドでは、MariaDB データベースサービスが Red Hat OpenStack Platform 環境向けに設定されていることを前提とします。
- メッセージングサービス。本ガイドでは、RabbitMQ が Red Hat OpenStack Platform 環境向けに設定されていることを前提とします。

OpenStack 環境のデプロイに director を使用した場合には、データベースとメッセージングサービスは、オーバークラウドのコントローラーノードにインストールされています。

Red Hat OpenStack Platform には、Red Hat Enterprise Linux 7 を実行するコンピュートノードと OpenStack Networking ノード上に **firewalld** ではなく **iptables** が必要です。本ガイドでは、ファイアウォールルールは、**iptables** を使用して設定します。

注記

ハードウェア検査 (**ironic-inspector**) は **iptables** を使用して、ironic ノードの MAC アドレスをブラックリストに追加します。**ironic-inspector** が変更を加えようと試みている間に、別のプロセスが **iptables** をロックした場合に、**iptables -w** フラグがサポートされていれば (バージョン 1.4.21 以降)、**ironic-inspector** はこのフラグを使用します。

1.1.2. Bare Metal Provisioning のハードウェア要件

Bare Metal Provisioning の全コンポーネントを実行するノードには、以下の条件を満たすハードウェアが必要です。

- 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
- 最小 6 GB の RAM
- 最小 40 GB の空きディスク領域
- 最小で 1 Gbps ネットワークインターフェースカードを 2 枚。 ただし、 特に、多数のベアメタルマシンをプロビジョニングする場合には、ベアメタルプロビジョニングネットワークのトラフィックには 10 Gbps のインターフェースを 1 枚搭載することを推奨します。
- ホストのオペレーティングシステムに Red Hat Enterprise Linux 7.2 (以降) がインストール済みであること

もしくは、Bare Metal Provisioning コンポーネントを専用の **openstack-nova-compute** ノードにインストールします。ハードウェア要件については、**『director のインストールと使用方法』**のガイドの「コンピュートノードの要件」 の項を参照してください。

1.1.3. ベアメタルプロビジョニングネットワークの要件

Bare Metal Provisioning には少なくとも 2 つのネットワークが必要です。

- プロビジョニングネットワーク: これは、Bare Metal Provisioning がベアメタルマシンのプロビジョニング/管理に使用するプライベートネットワークです。ベアメタルプロビジョニングネットワークは、ベアメタルシステムの検出がしやすくなるように、DHCP および PXE ブート機能を提供します。このネットワークは、Bare Metal Provisioning が PXE ブートおよび DHCP の要求に対応できるように、トランク化されたインターフェースでネイティブ VLAN を使用するのが理想的です。また、このネットワークは、プロビジョニングするベアメタルマシン上の帯域外 (OOB) ドライバーで電源管理を制御するのに使用するネットワークでもあります。

- 外部ネットワーク: リモート接続に使用する別個のネットワーク。このネットワークに接続するインターフェースには、静的または外部の DHCP サービス経由で動的に定義された、ルーティング可能な IP アドレスが必要です。

1.1.4. ベアメタルマシンの要件

プロビジョニングするベアメタルマシンには、以下が必要となります。

- 2 x NIC: ベアメタルプロビジョニングネットワーク用に 1 つと、外部接続用に 1 つ。

- ベアメタルプロビジョニングネットワークに接続した電源管理インターフェース (例: IPMI)。テスト目的で SSH を使用する場合には、これは必要ありません。

- システムのブート順序で、ベアメタルプロビジョニングネットワーク上での PXE ブートを 1 番上に設定し、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します。システム上の他のすべての NIC で PXE ブートを無効にします。

プロビジョニングするベアメタルマシンのその他の要件は、インストールするオペレーティングシステムによって異なります。Red Hat Enterprise Linux 7 の場合は、『Red Hat Enterprise Linux 7 インストールガイド』を参照してください。Red Hat Enterprise Linux 6 の場合は、『Red Hat Enterprise Linux 6 インストールガイド』を参照してください。

1.2. BARE METAL PROVISIONING サービスに向けた OPENSTACK の設定

すべての OpenStack サービスには、Identity サービスとの認証を行うためのユーザー名とパスワードがあります。また、各サービスを OpenStack Identity サービスで定義して、それらのサービスの内部、管理、パブリックの接続向けに関連付けられたエンドポイントURL も必要です。

Bare Metal Provisioning サービスを director ノードから設定するには、以下のステップを実行します。

1. **overcloudrc** ファイルを読み込みます。

```
# source ~stack/overcloudrc
```

2. OpenStack Bare Metal Provisioning ユーザーを作成します。

```
# openstack user create --password IRONIC_PASSWORD --enable
IRONIC_USER
# openstack role add --project service --user IRONIC_USER admin
```

ここで、**IRONIC_USER** は Bare Metal Provisioning サービスのユーザー名に、**IRONIC_PASSWORD** はパスワードに置き換えます。

3. OpenStack Bare Metal Provisioning サービスを作成します。

```
# openstack service create --name ironic --description "Ironic bare
```

```
metal provisioning service" baremetal
```

4. 他の OpenStack が使用している仮想 IP (VIP) アドレスを確認します。

```
# openstack endpoint list -c "Service Name" -c "PublicURL" --long
```

このコマンドの出力には、指定したサービスとその **Public URL** が一覧表示されます。これらは通常すべて同じサーバー上にあり、同じ IP アドレスを使用します。

5. Bare Metal Provisioning サービスのインストール先となるコンピュートノードの内部 API ネットワークのアドレスを取得します。

```
# route -n
```

このコマンドの出力には、IP ルーティングテーブルが IP アドレスと各 IP アドレス用の **インターフェース** とともに一覧表示されます。

内部 API ネットワークのアドレスは、次にサービスエンドポイントの作成に使用されます。

6. 以下のコマンドを実行すると、内部および管理用の URL に使用する NIC に関連付けられた IP アドレスをチェックすることができます。

```
# ifconfig INTERFACE
```

7. サービスエンドポイントを作成します。

```
# openstack endpoint create --publicurl http://VIP:6385 --
internalurl http://COMPUTE_INTERNAL_API_IP:6385 --adminurl
http://COMPUTE_INTERNAL_API_IP:6385 --region regionOne SERVICE_ID
```

ここで、**VIP** は HAProxy で設定されている仮想 IP アドレス
に、**COMPUTE_INTERNAL_API_IP** は内部 API ネットワークに接続されている Bare Metal Provisioning サービスを実行するコンピュートノードの IP アドレスに、**SERVICE_ID** は **service create** コマンドを使用して作成した Bare Metal Provisioning サービスの ID に置き換えます。

次に、HAProxy を設定して、以前の手順で作成したエンドポイントのパブリック URL に対する要求を確実に受信するようにします。HAProxy の値を設定するには、コントローラーノードに **root** ユーザーとしてログインしてください。

1. **/etc/haproxy/haproxy.cfg** ファイルを編集して、そのファイルの末尾に以下の行を追記します。

```
listen ironic

  bind VIP:6385 transparent
  server SERVER_NAME COMPUTE_INTERNAL_API_IP:6385 check fall 5 inter
2000 rise 2
```

この例では、

- **VIP** は仮想 IP アドレスです。

- **SERVER_NAME** は、Bare Metal Provisioning サービスのインストール/実行先となる Compute サーバーの HAProxy 識別名です。
- **COMPUTE_INTERNAL_API_IP** は、Bare Metal Provisioning のインストール/実行先となる Compute サーバーの内部 API の IP アドレスです。
- **transparent** により、HAProxy は、コントローラーノード上に存在しない IP アドレスでもバインディングできるようになったので、クラスター環境内において仮想 IP アドレスをコントローラー間で移動できます。
- **check fall 5 inter 2000 rise 2** はバックエンドサーバーに対する以下のヘルスチェックを指します。
  - **fall 5**: ヘルスチェックが連続 5 回失敗すると、そのサーバーは利用不可と見なされます。
  - **inter 2000**: ヘルスチェックの間隔は 2000 ミリ秒または 2 秒です。
  - **rise 2**: ヘルスチェックが 2 回連続で成功すると、そのサーバーは利用可能と見なされます。

2. HAProxy を再起動して、変更が有効になったことを確認します。

```
# systemctl restart haproxy.service
```

バックエンドサーバーが利用できないことを示す「**haproxy[4249]: proxy ironic has no server available!**」というメッセージが表示される場合があります。このサービスのインストール/設定はまだ行っていないので、このメッセージは、現時点では無視してかまいません。

1.3. BARE METAL PROVISIONING サービスに向けたコントローラーノードの設定

以下のステップは、Red Hat OpenStack Platform デプロイメント内の全コントローラーノードで **root** ユーザーとして実行する必要があります。ただし、**Bare Metal Provisioning データベースの作成** セクションは例外です。Bare Metal Provisioning データベースは全コントローラーノードで共有されるので、このセクションの手順は 1 つのコントローラー上で実行する必要があります。

コントローラーノード上で、ベアメタルプロビジョニングネットワークが Open vSwitch に接続されていて、OpenStack デプロイメントが到達可能であることを確認する必要があります。

1. Open vSwitch にブリッジを追加します。

```
# ovs-vsctl add-br br-ironic
# ovs-vsctl add-port br-ironic IRONIC_PROVISIONING_NIC
# ovs-vsctl show
```

ここで、**br-ironic** はブリッジ名に、**IRONIC_PROVISIONING_NIC** はベアメタルプロビジョニングネットワークに接続された NIC に置き換えます。

ovs-vsctl show コマンドを実行すると、関連付けられたポートを使用して新規ブリッジが作成されたことを確認できますが、**br-int** 統合ブリッジには新規ブリッジへのパッチがないことが分かるはずです。

2. 統合ブリッジに新規ブリッジを追加するには、以下のプラグインファイルを更新する必要があります。

a. ML2 設定ファイル (**/etc/neutron/plugins/ml2/ml2_conf.ini**) を以下のように更新します。

- **type_drivers** パラメーターで指定するドライバーに **flat** をリストします (例: **type_drivers = vxlan,vlan,flat,gre**)。これはコンマ区切りリストです。
- **mechanism_drivers** パラメーターで指定するドライバーに **openvswitch** オプションをリストします (例: **mechanism_drivers =openvswitch**)。これはコンマ区切りリストです。
- **flat_networks** パラメーターには、ベアメタルプロビジョニングネットワークを参照する名前 (例: **ironicnet**) を作成して **flat_networks** にリストします (例: **flat_networks =datacentre,ironicnet**)。これはコンマ区切りリストです。
- ベアメタルプロビジョニングネットワークに VLAN を使用する場合には、**network_vlan_ranges** パラメーターを「**ironicnet:VLAN_START:VLAN_END**」の形式で追加します (例: **network_vlan_ranges =datacentre:1:1000**)。これはコンマ区切りリストです。
- **enable_security_group** パラメーターはすでに有効化されているはずですが、設定されていない場合にはその値を **True** に変更します (例: **enable_security_group = True**)。

b. **/etc/neutron/plugins/ml2/openvswitch_agent.ini** ファイルで **bridge_mappings** パラメーターを見つけて以下のように更新します。

```
bridge_mappings =datacentre:br-ex,ironicnet:br-ironic
```

このコンマ区切りのキー/値のペアは、ベアメタルプロビジョニングネットワークの名前を、そのネットワークに接続された物理デバイスにマッピングします。

3. **neutron-openvswitch-agent.service** を再起動して **br-ironic** ブリッジが統合ブリッジの一部になっていることを確認します。

```
# systemctl restart neutron-openvswitch-agent.service
```

4. **neutron-server.service** を再起動して、新規接続を検出します。

```
# systemctl restart neutron-server.service
```

注記

このステップを実行しなかった場合には、OpenStack Networking サービス内でベアメタルプロビジョニングネットワークの作成を試みても、flat ネットワークが存在しないというメッセージが表示されて操作は失敗します。

1.3.1. Bare Metal Provisioning データベースの作成

Bare Metal Provisioning が使用するデータベースとデータベースユーザーを作成します。以下の手順のステップはすべて、データベースサーバーに **root** ユーザーとしてログインして実行する必要があります。

Bare Metal Provisioning データベースの作成

1. データベースサービスに接続します。

```
# mysql -u root
```

2. **ironic** データベースを作成します。

```
mysql> CREATE DATABASE ironic CHARACTER SET utf8;
```

3. **ironic** のデータベースユーザーを作成し、そのユーザーが **ironic** データベースにアクセスするのを許可します。

```
mysql> GRANT ALL PRIVILEGES ON ironic.* TO 'ironic'@'%' IDENTIFIED
BY 'PASSWORD';
mysql> GRANT ALL PRIVILEGES ON ironic.* TO 'ironic'@'localhost'
IDENTIFIED BY 'PASSWORD';
```

PASSWORD は、このユーザーとしてデータベースサーバーとの認証を行う際に使用するセキュアなパスワードに置き換えます。

4. データベースの特権をフラッシュして、設定が即時に反映されるようにします。

```
mysql> FLUSH PRIVILEGES;
```

5. mysql クライアントを終了します。

```
mysql> quit
```

1.3.2. Bare Metal Provisioning に向けた OpenStack Compute サービスの設定

ベアメタルプロビジョニングドライバー向けに Compute サービスを設定します。このドライバーを使用すると、Compute は仮想マシンのプロビジョニングに使用するのと同じ API を使用して物理マシンをプロビジョニングできるようになります。1 つの **openstack-nova-compute** ノードに対して指定できるドライバーは 1 つのみなので、ベアメタルプロビジョニングドライバーを指定したノードでは、物理マシンしかプロビジョニングできません。ベアメタルプロビジョニングドライバーを使用して全ベアメタルノードのプロビジョニングを行う **openstack-nova-compute** ノードを 1 台割り当てることを推奨します。以下の手順の全ステップは、選択したコンピュートノードに **root** ユーザーとしてログインした状態で実行する必要があります。

Bare Metal Provisioning に向けた OpenStack Compute の設定

1. OpenStack Compute が Bare Metal Provisioning スケジューラーホストマネージャーを使用するように設定します。

```
# openstack-config --set /etc/nova/nova.conf \
   DEFAULT scheduler_host_manager
nova.scheduler.ironic_host_manager.IronicHostManager
```

2. コンピュートのスケジューラーがインスタンスの変更をトラッキングする機能を無効にします。

```
# openstack-config --set /etc/nova/nova.conf DEFAULT
scheduler_tracks_instance_changes false
```

3. デフォルトのフィルターを以下のように設定します。

```
# openstack-config --set /etc/nova/nova.conf DEFAULT
baremetal_scheduler_default_filters
AvailabilityZoneFilter,ComputeFilter,ComputeCapabilitiesFilter,Image
PropertiesFilter
```

4. Compute がデフォルトの Bare Metal Provisioning スケジューリングフィルターを使用するように設定します。

```
# openstack-config --set /etc/nova/nova.conf \
   DEFAULT scheduler_use_baremetal_filters True
```

5. Compute が Bare Metal Provisioning の正しい認証情報を使用するように設定します。

```
# openstack-config --set /etc/nova/nova.conf \
   ironic admin_username ironic
# openstack-config --set /etc/nova/nova.conf \
   ironic admin_password PASSWORD
# openstack-config --set /etc/nova/nova.conf \
   ironic admin_url http://IDENTITY_IP:35357/v2.0
# openstack-config --set /etc/nova/nova.conf \
   ironic admin_tenant_name service
# openstack-config --set /etc/nova/nova.conf \
   ironic api_endpoint http://IRONIC_API_IP:6385/v1
```

以下の値を置き換えてください。

- **PASSWORD** は、Bare Metal Provisioning が Identity との認証で使用するパスワードに置き換えます。
- **IDENTITY_IP** は、Identity をホストするサーバーの IP アドレスまたはホスト名に置き換えます。
- **IRONIC_API_IP** は、Bare Metal Provisioning API サービスをホストするサーバーの IP アドレスまたはホスト名に置き換えます。

6. **nova** データベースの認証情報を **ironic** コンピュートノードで設定します。

```
# openstack-config --set /etc/nova/nova.conf database connection
"mysql+pymysql://nova:NOVA_DB_PASSWORD@DB_IP/nova"
```

7. コンピュートコントローラーノードで、コンピュートスケジューラーサービスを再起動します。

```
# systemctl restart openstack-nova-scheduler.service
```

8. コンピュートノードで、Compute サービスを再起動します。

```
# systemctl restart openstack-nova-compute.service
```

1.3.3. OpenStack Networking DHCP エージェントが iPXE 要求をタグ付けするための設定

iPXE からの OpenStack Networking DHCP 要求には、**ipxe** という DHCP タグを付けて、クライアントが HTTP 操作を実行して **boot.ipxe** スクリプトを取得する必要があることを DHCP サーバーに知らせる必要があります。このタグ付けは、OpenStack Networking DHCP エージェントサービスが使用する **dnsmasq** 設定ファイルに **dhcp-userclass** エントリーを追加することによって行うことができます。

1. オーバークラウドコントローラーで DHCP エージェントが使用している **dnsmasq** ファイルを確認します。

```
# grep ^dnsmasq_config_file /etc/neutron/dhcp_agent.ini

dnsmasq_config_file =/etc/neutron/dnsmasq-neutron.conf
```

2. このファイルを編集して、ファイルの末尾に以下の行を追記します。

```
# Create the "ipxe" tag if request comes from iPXE user class

dhcp-userclass=set:ipxe,iPXE
```

3. ファイルを保存してから OpenStack Networking DHCP エージェントサービスを再起動します。

```
# systemctl restart neutron-dhcp-agent.service
```

1.4. BARE METAL PROVISIONING に向けたコンピュートノードの設定

以下の手順は、Bare Metal Provisioning サービスを実行するコンピュートノード**のみに**適用します。これらのステップはコンピュートノード上で **root** ユーザーとして実行する必要があります。

コンピュートノードには、Bare Metal Provisioning 用の NIC (例: **eth6**) が設定されています。この手順は、以下を目的とします。

1. Bare Metal Provisioning 用の NIC (この例では **eth6**) を Open vSwitch に接続します。
2. ベアメタルサーバーは、iPXE プロセスの一環としてベアメタルノードからブートイメージをダウンロードする必要があるため、この接続に IP アドレスを割り当てます。

Open vSwitch への eth6 の接続

1. 「Bare Metal Provisioning サービスに向けたコントローラーノードの設定」に記載のコントローラーノードの設定と同様に、Bare Metal Provisioning サービスを実行するコンピュートノード上の Open vSwitch 内にブリッジを作成します。

```
# ovs-vsctl add-br br-ironic
# ovs-vsctl add-port br-ironic IRONIC_PROVISIONING_NIC
```

ここで、**br-ironic** はブリッジ名に、**IRONIC_PROVISIONING_NIC** はベアメタルプロビジョニングネットワークに接続された NIC (例: **eth6**) に置き換えます。

注記

この手順が「Bare Metal Provisioning サービスに向けたコントローラーノードの設定」と唯一異なる点は、コンピュートノードでは OpenStack Networking サービスを再起動しないことです。

これでブリッジとポートが Open vSwitch に追加され、**ovs-vsctl show** コマンドで確認することができます。ただし、OpenStack が使用するための統合ブリッジ (**br-int**) には接続しません。

2. 接続を作成するには、OpenStack Networking プラグインのファイルを以下のように更新する必要があります。

   a. ML2 設定ファイル (**/etc/neutron/plugins/ml2/ml2_conf.ini**) を以下のように更新します。

      - **type_drivers** パラメーターで指定するドライバーに **flat** をリストします (例: **type_drivers = vxlan,vlan,flat,gre**)。これはコンマ区切りリストです。
      - **mechanism_drivers** パラメーターで指定するドライバーに **openvswitch** オプションをリストします (例: **mechanism_drivers =openvswitch**)。これはコンマ区切りリストです。
      - **flat_networks** パラメーターには、ベアメタルプロビジョニングネットワークを参照する名前 (例: **ironicnet**) を作成して **flat_networks** にリストします (例: **flat_networks =datacentre,ironicnet**)。これはコンマ区切りリストです。
      - ベアメタルプロビジョニングネットワークに VLAN を使用する場合には、**network_vlan_ranges** パラメーターを「**ironicnet:VLAN_START:VLAN_END**」の形式で追加します (例: **network_vlan_ranges =datacentre:1:1000**)。これはコンマ区切りリストです。
      - **enable_security_group** パラメーターはすでに有効化されているはずですが、設定されていない場合にはその値を **True** に変更します (例: **enable_security_group = True**)。

   b. **/etc/neutron/plugins/ml2/openvswitch_agent.ini** ファイルで **bridge_mappings** パラメーターを見つけて以下のように更新します。

      ```
      bridge_mappings =datacentre:br-ex,ironicnet:br-ironic
      ```

      このコンマ区切りのキー/値のペアは、ベアメタルプロビジョニングネットワークの名前を、そのネットワークに接続された物理デバイスにマッピングします。

3. OpenStack Networking Open vSwitch エージェントサービスを再起動します。

   ```
   # systemctl restart neutron-openvswitch-agent.service
   ```

これで、本手順の最初の目的が達成されました。次は **br-ironic** ブリッジに IP アドレスを割り当てて、再起動後も永続するように設定します。

Bare Metal サーバーへの IP アドレスの割り当て

1. **/etc/sysconfig/network-scripts** の場所に標準の設定ファイルを作成します。テナントネットワークですでに利用可能な **ifcfg*** ファイルをコピーして、**device**、**ipaddr**、**ovs_bridge**、**bridge name** と、**br-ironic** および **eth6** の **MAC** アドレスの値を編集します。新規ファイルの更新完了後には、以下のような値が設定されるはずです。
   ifcfg-br-ironic

```
DEVICE=br-ironic
ONBOOT=yes
HOTPLUG=no
NM_CONTROLLED=no
PEERDNS=no
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=static
IPADDR=BARE_METAL_PROVISIONING_IP
NETMASK=255.255.255.0
OVS_EXTRA="set bridge br-ironic other-config:hwaddr=MAC_ADDRESS"
```

   ifcfg-eth6

```
DEVICE=eth6
ONBOOT=yes
HOTPLUG=no
NM_CONTROLLED=no
PEERDNS=no
DEVICETYPE=ovs
TYPE=OVSPort
OVS_BRIDGE=br-ironic
BOOTPROTO=none
```

2. ネットワークブリッジを再起動して、IP アドレスを ping できる状態にします。

```
# ifup br-ironic
```

注記

ネットワークサービスの再起動時にノードからの接続が切断された場合には、サーバーを再起動してください。

1.4.1. 必要なチャンネルのサブスクライブ

Bare Metal Provisioning のパッケージをインストールするには、サーバーを Red Hat サブスクリプションマネージャーに登録して、必要なチャンネルをサブスクライブする必要があります。コンピュートノードに Bare Metal Provisioning をインストールする場合は、サーバーはすでに適切にサブスクライブされている可能性があります。**yum repolist** コマンドを実行して、以下の手順に記載しているチャンネルが有効化されているかどうかを確認してください。

必要なチャンネルのサブスクライブ

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

```
# subscription-manager register
```

2. Bare Metal Provisioning のインストールに必要なチャンネルが含まれたエンタイトルメントプールを特定します。

```
# subscription-manager list --available | grep -A13 "Red Hat
Enterprise Linux Server"
# subscription-manager list --available | grep -A13 "Red Hat
OpenStack Platform"
```

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

```
# subscription-manager attach --pool=POOL_ID
```

4. 必要なチャンネルを有効にします。

```
# subscription-manager repos --enable=rhel-7-server-rpms \
--enable=rhel-7-server-openstack-9-rpms \
--enable=rhel-7-server-rh-common-rpms --enable=rhel-7-server-
optional-rpms \
--enable=rhel-7-server-openstack-9-optools-rpms
```

1.4.2. Bare Metal Provisioning パッケージのインストール

Bare Metal Provisioning には、以下のパッケージが必要です。

openstack-ironic-api

Bare Metal Provisioning API サービスを提供します。

openstack-ironic-conductor

Bare Metal Provisioning コンダクターサービスを提供します。コンダクターにより、ノードの追加/編集/削除、IPMI または SSH を使用した電源オン/オフ、ベアメタルのプロビジョニング/デプロイ/デコミッションが可能となります。

python-ironicclient

Bare Metal Provisioning サービスと対話するためのコマンドラインインターフェースを提供します。

パッケージをインストールします。

```
# yum install openstack-ironic-api openstack-ironic-conductor python-
ironicclient ipxe-bootimgs
```

1.4.3. iPXE の設定

1. iPXE 、map-file に必要なディレクトリーを作成して、**undionly.kpxe** ブートイメージ、iPXE、**map-file** を適切な場所にコピーします。

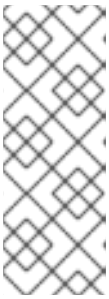
```
# mkdir /httpboot
# mkdir /tftpboot
```

```
# echo 'r ([/]) /tftpboot/\1' > /tftpboot/map-file
# echo 'r ^(/tftpboot/) /tftpboot/\2' >> /tftpboot/map-file
# cp /usr/share/ipxe/undionly.kpxe /tftpboot/
# chown -R ironic:ironic /httpboot
# chown -R ironic:ironic /tftpboot
```

2. デフォルトでは、director でデプロイされたコンピュートノードは、SELinux を **Enforcing** モードで実行します。iPXE ブートでパーミッションエラーが発生するのを避けるには、それらのディレクトリーに適切なラベルを設定するようにしてください。このラベルを適用するには、**root** ユーザーとして以下のコマンドを実行します。

```
# semanage fcontext -a -t httpd_sys_content_t "/httpboot(/.*)?"
# restorecon -Rv /httpboot
# semanage fcontext -a -t tftpdir_t "/tftpboot(/.*)?"
# restorecon -Rv /tftpboot
```

3. HTTP がイメージの要求に対応できるように設定します。**httpd** パッケージはすでにインストール済みなので、適切な仮想ホストのエントリーを作成してサービスを起動するだけです。



注記

/etc/httpd/conf.d には、多数のファイルが含まれています。Red Hat は、単一のオーバークラウドの完全なイメージを全ノードに使用するので、コントローラーノードでのみ使用されている場合でも、全ノードで /etc/httpd/conf.d にファイルが含まれます。これらのファイルは使用されていないので、**/etc/httpd/conf.d** の内容を削除するか、別の場所にコピーすることができます。

iPXE の設定用に新しいファイルを作成します。このファイルには任意の名前を付けることができます。**.conf** の形式で、以下のような内容を記述してください。

```
# cat 10-ipxe_vhost.conf
Listen 8088
<VirtualHost *:8088>
    DocumentRoot "/httpboot"
    <Directory "/httpboot">
        Options Indexes FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
        Require all granted
    </Directory>

    ## Logging
    ErrorLog "/var/log/httpd/ironic_error.log"
    ServerSignature Off
    CustomLog "/var/log/httpd/ironic_access.log" combined
</VirtualHost>
```

上記の仮想ホストの設定は、HTTPD がポート 8088 上の全アドレスにリッスンするように設定します。また、そのポートに対する全要求が **/httpboot** に送られるようにドキュメントルートを設定します。

4. このファイルを保存してから、コンピュートノードで HTTPD サービスを再起動します。

```
# systemctl enable httpd.service
# systemctl start httpd.service
```

1.4.4. Bare Metal Provisioning サービスの設定

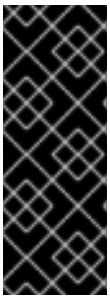
本項では、**/etc/ironic/ironic.conf** ファイルに必要な変更を加えます。

1.4.4.1. Bare Metal Provisioning がデータベースサーバーと通信するための設定

接続の設定キーの値を以下のように設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
database connection mysql+pymysql://ironic:PASSWORD@IP/ironic
```

ここで、**PASSWORD** はデータベースサーバーのパスワードに、**IP** はデータベースサーバーの IP アドレスまたはホスト名に変更します。



重要

この接続設定キーに指定する IP アドレスまたはホスト名は、「Bare Metal Provisioning データベースの作成」の項で OpenStack Networking データベースを作成した際に Bare Metal Provisioning データベースユーザーがアクセスを許可された IP アドレスまたはホスト名と一致する必要があります。また、データベースがローカルでホストされ、Bare Metal Provisioning データベースの作成時に **localhost** へのアクセス権を付与した場合には、**localhost** と入力する必要があります。

1.4.4.2. Bare Metal Provisioning の認証の設定

Bare Metal Provisioning が認証に Identity を使用するように設定します。以下の手順に記載するステップはすべて、Bare Metal Provisioning をホストするサーバー (単一または複数) に **root** ユーザーとしてログインして実行する必要があります。

Bare Metal Provisioning が Identity で認証を行うための設定

1. Bare Metal Provisioning が使用する必要のある Identity のパブリックおよび admin エンドポイントを設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
   keystone_authtoken auth_uri http://IP:5000/v2.0
# openstack-config --set /etc/ironic/ironic.conf \
   keystone_authtoken identity_uri http://IP:35357/
```

   IP は、Identity サーバーの IP アドレスまたはホスト名に置き換えます。

2. Bare Metal Provisioning が **service** テナントとして認証を行うように設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
   keystone_authtoken admin_tenant_name service
```

3. Bare Metal Provisioning が **ironic** の管理ユーザーアカウントを使用して認証を行うように設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
```

```
   keystone_authtoken admin_user ironic
```

4. Bare Metal Provisioning が正しい **ironic** 管理ユーザーアカウントのパスワードを使用するように設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
   keystone_authtoken admin_password PASSWORD
```

PASSWORD は **ironic** ユーザーを作成した際に設定したパスワードに置き換えます。

1.4.4.3. Bare Metal Provisioning のための RabbitMQ Message Broker の設定

RabbitMQ はデフォルト (かつ推奨の) メッセージブローカーです。RabbitMQ メッセージングサービスは、**rabbitmq-server** パッケージにより提供されます。以下の手順で記載するステップはすべて、Bare Metal Provisioning をホストするコントローラーノードまたはコンピュートノードで **root** ユーザーとしてログインして実行する必要があります。

この手順は、RabbitMQ メッセージングサービスのインストールおよび設定が完了済みで、メッセージングサービスをホストするサーバー上で **ironic** ユーザーとそのパスワードが作成済みであることを前提とします。

RabbitMQ Message Broker を使用するための Bare Metal Provisioning の設定

1. RabbitMQ を RPC バックエンドとして設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
   DEFAULT rpc_backend ironic.openstack.common.rpc.impl_kombu
```

2. Bare Metal Provisioning が RabbitMQ ホストに接続するように設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
   oslo_messaging_rabbit rabbit_host RABBITMQ_HOST
```

RABBITMQ_HOST は、メッセージブローカーをホストするサーバーの IP アドレスまたはホスト名に置き換えます。

3. メッセージブローカーのポートを 5672 に設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
   oslo_messaging_rabbit rabbit_port 5672
```

4. RabbitMQ の設定時に Bare Metal Provisioning 用に作成した RabbitMQ ユーザー名とパスワードを設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
   oslo_messaging_rabbit rabbit_userid guest
# openstack-config --set /etc/ironic/ironic.conf \
   oslo_messaging_rabbit rabbit_password RABBIT_GUEST_PASSWORD
```

RABBIT_GUEST_PASSWORD はゲストユーザー用の RabbitMQ パスワードに置き換えます。

5. RabbitMQ の起動時に、全リソースに対するパーミッションが **guest** ユーザーに付与されています。リソースへのアクセスは、特別に仮想ホストを介して行われます。Bare Metal Provisioning がこの仮想ホストにアクセスするように設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
   oslo_messaging_rabbit rabbit_virtual_host /
```

1.4.4.4. Bare Metal Provisioning ドライバーの設定

Bare Metal Provisioning は、ベアメタルサーバーのデプロイと管理用の複数のドライバーをサポートします。一部のドライバーには、ハードウェア要件があり、追加の設定またはパッケージインストールが必要です。ドライバーについての詳しい情報は、「付録A *Bare Metal Provisioning ドライバー*」を参照してください。ドライバー名の前半はデプロイメントの方法 (例: PXE)、後半は電源管理の方法 (例: IPMI) を示しています。

Bare Metal Provisioning ドライバーの設定

1. ベアメタルサーバーのプロビジョニングに使用するドライバー (1 つまたは複数) を指定します。複数のドライバーを指定するには、コンマ区切りリストを使用します。

   ```
   # openstack-config --set /etc/ironic/ironic.conf \
      DEFAULT enabled_drivers DRIVER1,DRIVER2
   ```

   次のドライバーがサポートされています。

   - PXE デプロイを使用する IPMI
     - **pxe_ipmitool**
   - PXE デプロイを使用する DRAC
     - **pxe_drac**
   - PXE デプロイを使用する iLO
     - **pxe_ilo**
   - PXE デプロイを使用する iBoot
     - **pxe_iboot**
   - PXE デプロイを使用する SSH
     - **pxe_ssh**
   - PXE を使用する iRMC
     - **pxe_irmc**
   - PXE を使用する AMT
     - **pxe_amt**
   - HTTP を使用する AMT
     - **agent_amt**

2. Bare Metal コンダクターサービスを再起動します。

   ```
   # systemctl restart openstack-ironic-conductor.service
   ```

1.4.4.5. Bare Metal Provisioning サービスが PXE を使用するための設定

1. Bare Metal Provisioning サービスが PXE テンプレートを使用するように設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
pxe pxe_config_template
\$pybasedir/drivers/modules/ipxe_config.template
```

2. Bare Metal Provisioning サービスが **tftp_server** を使用するように設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
pxe tftp_server BARE_METAL_PROVISIONING_NETWORK_IP
```

3. PXE **tftp_root** を設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
pxe tftp_root /tftpboot
```

4. PXE ブート用のファイルの名前を設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
pxe pxe_bootfile_name undionly.kpxe
```

5. Bare Metal Provisioning サービスが iPXE を使用するように有効化します。

```
# openstack-config --set /etc/ironic/ironic.conf \
pxe ipxe_enabled true
```

6. **http** サーバーの URL を設定します。

```
# openstack-config --set /etc/ironic/ironic.conf deploy http_url
http://BARE_METAL_PROVISIONING_IP:8088
```

7. Bare Metal コンダクターサービスを再起動します。

```
# systemctl restart openstack-ironic-conductor.service
```

1.4.4.6. Bare Metal Provisioning が OpenStack Networking および OpenStack Image と通信するための設定

Bare Metal Provisioning は、DHCP とネットワークの設定には OpenStack Networking を使用し、物理マシンのブートに使用するイメージの管理には Image サービスを使用します。Bare Metal Provisioning が OpenStack Networking と Image サービスに接続して通信するように設定します。以下の手順に記載するステップはすべて、Bare Metal Provisioning をホストするサーバーに **root** ユーザーとしてログインして実行する必要があります。

Bare Metal Provisioning が OpenStack Networking および OpenStack Image と通信するための設定

1. Bare Metal Provisioning が OpenStack Networking エンドポイントを使用するように設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
   neutron url http://NEUTRON_IP:9696
```

NEUTRON_IP は OpenStack Networking をホストするサーバーの IP アドレスまたはホスト名に置き換えます。

2. Bare Metal Provisioning が Image サービスと通信するように設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
   glance glance_host GLANCE_IP
```

GLANCE_IP は、Image サービスをホストするサーバーの IP アドレスまたはホスト名に置き換えます。

3. Bare Metal Provisioning API サービスを起動して、ブート時に開始するように設定します。

```
# systemctl start openstack-ironic-api.service
# systemctl enable openstack-ironic-api.service
```

4. Bare Metal Provisioning データベーステーブルを作成します。

```
# ironic-dbsync --config-file /etc/ironic/ironic.conf create_schema
```

5. Bare Metal Provisioning コンダクターサービスを起動して、ブート時に開始するように設定します。

```
# systemctl restart openstack-ironic-conductor.service
# systemctl enable openstack-ironic-conductor.service
```

1.4.4.7. Image サービスおよび Bare Metal Provisioning サービス用の Ceph Object Gateway の設定

Red Hat Ceph Storage は、Swift 対応の API を実装する Ceph オブジェクト (RADOS) ゲートウェイが含まれた分散型ストレージシステムです。RADOS ゲートウェを使用するには、以下をステップを実行する必要があります。

- Image サービスを設定して、RADOS ゲートウェイへのアクセスを許可します。
- Bare Metal Provisioning サービスが RADOS ゲートウェイの Swift API を使用してベアメタルイメージをプロビジョニングするように設定します。

操作を実行する前の準備

Red Hat Ceph Storage で Ceph Object Gateway の設定を必ず済ませておいてください。詳しくは、「Ceph Object Gateway Installation」を参照してください。

Ceph Object Gateway の Image サービスへのアクセスの設定

1. Ceph Object Gateway の **admin** ホストの Image サービス用にアクセス認証情報を作成します。

```
# radosgw-admin user create --uid=GLANCE_USERNAME --display-
name="User for Glance"
```

```
# radosgw-admin subuser create --uid=GLANCE_USERNAME --
subuser=GLANCE_USERNAME:swift --access
=full

# radosgw-admin key create --subuser=GLANCE_USERNAME:swift --key-
type=swift --secret=STORE_KEY

# radosgw-admin user modify --uid=GLANCE_USERNAME --temp-url-
key=TEMP_URL_KEY
```

GLANCE_USERNAME は Image サービスにアクセスするユーザー名に、**STORE_KEY** と **TEMP_URL_KEY** は適切なキーに置き換えます。

注記

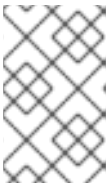
--gen-secret CLI パラメーターは使用しないでください。このパラメーターを使用すると、**radosgw-admin** ユーティリティーは OpenStack Image サービスでは機能しないスラッシュ記号付きのキーを生成します。

2. **/etc/glance/glance-api.conf** ファイルを編集して、Image API サービスが Ceph Object Gateway の Swift API をバックエンドとして使用するように設定します。

```
[glance_store]

stores = file, http, swift
default_store = swift
swift_store_auth_version = 1
swift_store_auth_address = http://RADOS_IP:PORT/auth/1.0
swift_store_user = GLANCE_USERNAME:swift
swift_store_key = STORE_KEY
swift_store_container = glance
swift_store_create_container_on_put = True
```

RADOS_IP と **PORT** は、Ceph Object Gateway の API サービスの IP とポートに置き換えてください。



注記

Ceph Object Gateway は、HTTP サーバーとの対話に FastCGI プロトコルを使用します。HTTPS サポートを有効化する場合には、お使いの HTTP サーバーのドキュメントを参照してください。

3. Image API サービスの再起動

```
# systemctl restart openstack-glance-api.service
```

Bare Metal Provisioning が Ceph Object Gateway を使用するための設定

1. Edit the **/etc/ironic/ironic.conf** ファイルを編集して、Bare Metal Provisioning コンダクターサービスが Ceph Object Gateway を使用するように設定します。

```
[glance]
```

```
swift_container = glance
swift_api_version = v1
swift_endpoint_url = http://RADOS_IP:PORT
swift_temp_url_key = TEMP_URL_KEY
temp_url_endpoint_type=radosgw
```

TEMP_URL_KEY と **_RADOS_IP:PORT** は、前の手順で使用した値に置き換えます。

2. Bare Metal Provisioning コンダクターサービスを再起動します。

```
# systemctl restart openstack-ironic-conductor.service
```

1.4.5. OpenStack Compute が Bare Metal Provisioning サービスを使用するための設定

本セクションでは、**/etc/nova/nova.conf** ファイルを更新して、Compute サービスが Bare Metal Provisioning サービスを使用するように設定します。

OpenStack Compute が Bare Metal Provisioning を使用するための設定

1. Compute が Clustered Compute Manager を使用するように設定します。

```
# openstack-config --set /etc/nova/nova.conf \
   DEFAULT compute_manager
ironic.nova.compute.manager.ClusteredComputeManager
```

2. 物理メモリーに対する仮想メモリーの割り当ての比率を設定します。

```
# openstack-config --set /etc/nova/nova.conf \
   DEFAULT ram_allocation_ratio 1.0
```

3. ホスト用に確保するディスク容量を MB 単位で設定します。

```
# openstack-config --set /etc/nova/nova.conf \
   DEFAULT reserved_host_memory_mb 0
```

4. OpenStack Compute が Bare Metal Provisioning ドライバーを使用するように設定します。

```
# openstack-config --set /etc/nova/nova.conf \
   DEFAULT compute_driver nova.virt.ironic.IronicDriver
```

5. Compute が Bare Metal Provisioning の正しい認証情報を使用するように設定します。

```
# openstack-config --set /etc/nova/nova.conf \
   ironic admin_username ironic
# openstack-config --set /etc/nova/nova.conf \
   ironic admin_password PASSWORD
# openstack-config --set /etc/nova/nova.conf \
   ironic admin_url http://IDENTITY_IP:35357/v2.0
# openstack-config --set /etc/nova/nova.conf \
   ironic admin_tenant_name service
# openstack-config --set /etc/nova/nova.conf \
   ironic api_endpoint http://IRONIC_API_IP:6385/v1
```

以下の値を置き換えてください。

- **PASSWORD** は、Bare Metal Provisioning が Identity との認証で使用するパスワードに置き換えます。
- **IDENTITY_IP** は、Identity をホストするサーバーの IP アドレスまたはホスト名に置き換えます。
- **IRONIC_API_IP** は、Bare Metal Provisioning API サービスをホストするサーバーの IP アドレスまたはホスト名に置き換えます。

6. コンピュートコントローラーノードで、コンピュートスケジューラーサービスを再起動します。

```
# systemctl restart openstack-nova-scheduler.service
```

7. コンピュートノードで、Compute サービスを再起動します。

```
# systemctl restart openstack-nova-compute.service
```

第2章 BARE METAL デプロイメントの設定

OpenStack 環境でのベアメタルデプロイメントが可能となるように Bare Metal Provisioning、Image service、および Compute を設定します。以下の項では、ベアメタルノードのデプロイを正常に実行するのに必要な追加の設定ステップについて説明します。

2.1. BARE METAL PROVISIONING サービス用の OPENSTACK 設定の作成

2.1.1. OpenStack Networking 構成の設定

OpenStack Networking が DHCP、PXE ブートおよびその他の必要な場合に Bare Metal Provisioning と通信するように設定します。以下の手順では、単一のフラットなネットワークをベアメタルにプロビジョニングするユースケースで OpenStack Networking を設定します。この設定には ML2 プラグインと Open vSwitch エージェントを使用します。

プロビジョニングに使用するネットワークインターフェースは、OpenStack Networking ノード上でリモート接続に使用しているネットワークインターフェースとは別であることを確認してください。この手順では、ベアメタルプロビジョニングネットワーク用のインターフェースを使用するブリッジを作成し、リモート接続を解除します。

以下の手順に記載するステップはすべて、OpenStack Networking をホストするサーバーに **root** ユーザーとしてログインして実行する必要があります。

OpenStack Networking が Bare Metal Provisioning と通信するための設定

1. Identity に管理ユーザーとしてアクセスするためのシェルを設定します。

```
# source ~stack/overcloudrc
```

2. ベアメタルインスタンスをプロビジョニングするためのフラットなネットワークを作成します。

```
# neutron net-create --tenant-id TENANT_ID sharednet1 --shared \
--provider:network_type flat --provider:physical_network PHYSNET
```

TENANT_ID はネットワークを作成するテナントの一意識別子に、**PHYSNET** は物理ネットワークの名前に置き換えます。

3. フラットネットワーク上にサブネットを作成します。

```
# neutron subnet-create sharednet1 NETWORK_CIDR --name SUBNET_NAME \
--ip-version 4 --gateway GATEWAY_IP --allocation-pool \
start=START_IP,end=END_IP --enable-dhcp
```

以下の値を置き換えてください。

- **NETWORK_CIDR** は、サブネットが示す IP アドレスブロックの Classless Inter-Domain Routing (CIDR) 表記に置き換えます。 **START_IP** で始まり **END_IP** で終る範囲で指定する IP アドレスブロックは、**NETWORK_CIDR** で指定されている IP アドレスブロックの範囲内に入る必要があります。
- **SUBNET_NAME** はサブネットの名前に置き換えます。

- **GATEWAY_IP** は新しいサブネットのゲートウェイとして機能するシステムの IP アドレスまたはホスト名に置き換えます。このアドレスは、 **NETWORK_CIDR** で指定されている IP アドレスブロック内で、かつ **START_IP** で始まり **END_IP** で終わる範囲で指定されている IP アドレスブロック外である必要があります。

- **START_IP** は、Floating IP アドレスの割り当て元となる新規サブネット内の IP アドレス範囲の開始アドレスを示す IP アドレスに置き換えます。

- **END_IP** は、Floating IP アドレスの割り当て元となる新規サブネット内の IP アドレス範囲の終了アドレスを示す IP アドレスに置き換えます。

4. ネットワークとサブネットをルーターに接続して、メタデータ要求が OpenStack Networking サービスによって応答されるようにします。

```
# neutron router-create ROUTER_NAME
```

ROUTER_NAME はルーターの名前に置き換えます。

5. ベアメタルのサブネットにこのルーターをインターフェースとして追加します。

```
# neutron router-interface-add ROUTER_NAME BAREMETAL_SUBNET
```

ROUTER_NAME は、ルーターの名前に、**BAREMETAL_SUBNET** は以前に作成した ID またはサブネット名に置き換えます。これにより、**cloud-init** からのメタデータ要求に応答することができます。

6. Bare Metal Provisioning サービスを実行するコンピュートノード上の **/etc/ironic/ironic.conf** ファイルを更新して、クリーニングサービスに同じネットワークを使用するようにします。Bare Metal Provisioning を実行しているコンピュートノードにログインして、以下のコマンドを **root** ユーザーとして実行します。

```
# openstack-config --set /etc/ironic/ironic.conf neutron
cleaning_network_uuid NETWORK_UUID
```

NETWORK_UUID は、前のステップで作成したベアメタルプロビジョニングネットワークの ID に置き換えます。

7. Bare Metal Provisioning サービスを再起動します。

```
# systemctl restart openstack-ironic-conductor.service
```

2.1.2. Bare Metal Provisioning フレーバーの作成

デプロイメントの一部として使用するフレーバーを作成する必要があります。このフレーバーの仕様 (メモリー、CPU、ディスク) はベアメタルノードが提供する仕様以下である必要があります。

1. Identity に管理ユーザーとしてアクセスするためのシェルを設定します。

```
# source ~stack/overcloudrc
```

2. 既存のフレーバーを一覧表示します。

```
# openstack flavor list
```

3. Bare Metal Provisioning サービス向けに新規フレーバーを作成します。

```
# openstack flavor create --id auto --ram RAM --vcpus VCPU --disk
DISK --public baremetal
```

RAM は RAM のメモリーに、**VCPU** は仮想 CPU 数に、**DISK** はディスクストレージの値に置き換えます。

4. 指定したそれぞれの値を使用して新規フレーバーが作成されたことを確認します。

```
# openstack flavor list
```

2.1.3. ベアメタルイメージの作成

Bare Metal Provisioning は、ディスク全体のイメージまたはルートパーティションのイメージのデプロイをサポートしています。ディスク全体のイメージにはパーティションテーブル、カーネルイメージ、最終のユーザーイメージが含まれます。ルートパーティションイメージには OS のルートパーティションが含まれ、ベアメタルノードが最終のユーザーイメージのブートに使用する kernel および ramdisk イメージが必要です。サポートされているベアメタルエージェントドライバーはすべて、ディスク全体のイメージまたはルートパーティションのイメージをデプロイすることができます。

ディスク全体のイメージには、パーティションテーブル、ブートローダー、ユーザーイメージが含まれたイメージが 1 つ必要です。ディスク全体のイメージを使用してデプロイしたノードはローカルブートをサポートするので、Bare Metal Provisioning はデプロイ後の再起動は制御しません。

ルートパーティションのデプロイメントには、**deploy** イメージと **user** イメージの 2 セットのイメージが必要です。Bare Metal Provisioning は **deploy** イメージを使用してノードを起動し、**user** イメージをベアメタルノードにコピーします。**deploy** イメージが Image サービスに読み込まれた後には、ベアメタルノードのプロパティーを使用して **deploy** イメージをベアメタルノードに関連付けて、そのノードが **deploy** イメージを使用するように設定することができます。その後のノードの再起動にはネットブートが使用されて、user イメージがダウンロードされます。

本項には、ルートパーティションイメージを使用して、ベアメタルノードをプロビジョニングする例を記載しています。ディスク全体のイメージのデプロイメントの場合は、「Windows 全体のイメージの作成」を参照してください。パーティションベースのデプロイメントの場合は、アンダークラウドがオーバークラウドをデプロイした際に **deploy** イメージはすでに使用されているので、**作成する必要はありません。deploy** イメージは、以下に示すように、**kernel** イメージと **ramdisk** イメージの 2 つで構成されます。

```
ironic-python-agent.kernel
ironic-python-agent.initramfs
```

rhosp-director-images-ipa パッケージをインストール済みの場合には、これらのイメージは、**/usr/share/rhosp-director-images/ironic-python-agent*.el7ost.tar** ファイル内にあります。

イメージを抽出して Image サービスにアップロードします。

```
# openstack image create --container-format aki --disk-format aki --public
--file ./ironic-python-agent.kernel bm-deploy-kernel
# openstack image create --container-format ari --disk-format ari --public
--file ./ironic-python-agent.initramfs bm-deploy-ramdisk
```

最終的に必要なイメージは、Bare Metal Provisioning ノードに実際にデプロイされるイメージです。たとえば、**cloud-init** がすでにあるので、**Red Hat Enterprise Linux KVM** イメージをダウンロードすることができます。

Image サービスへのイメージのアップロード

```
# openstack image create --container-format bare --disk-format qcow2  --
property kernel_id=DEPLOY_KERNEL_ID \
--property ramdisk_id=DEPLOY_RAMDISK_ID --public --file ./IMAGE_FILE rhel
```

DEPLOY_KERNEL_ID が Image サービスにアップロードされた deploy-kernel イメージに関連付けられている UUID に置き換えます。**DEPLOY_RAMDISK_ID** は、Image サービスにアップロードされた deploy-ramdisk に関連付けられたUUID に置き換えます。これらの UUID を確認するには、**openstack image list** を使用してください。

2.1.4. Bare Metal Provisioning サービスへの Bare Metal Provisioning ノードの追加

Bare Metal Provisioning ノードを Bare Metal Provisioning サービスに追加するには、クラウドをインスタンス化するのに使用された **instackenv.json** ファイルのセクションをコピーして、必要に応じて変更します。

1. **overcloudrc** ファイルを読み込んで、**.json** ファイルをインポートします。

```
# source ~stack/overcloudrc
# openstack baremetal import --json ./baremetal.json
```

2. Bare Metal Provisioning サービス内のベアメタルノードの **driver_info** セクションの **deploy_kernel** と **deploy_ramdisk** を指定して、そのノードが初期ブートイメージとしてデプロイ済みのイメージを使用するように設定します。

```
# ironic node-update NODE_UUID add
driver_info/deploy_kernel=DEPLOY_KERNEL_ID
driver_info/deploy_ramdisk=DEPLOY_RAMDISK_ID
```

NODE_UUID は、ベアメタルノードの UUID に置き換えます。この値は、director ノードで **ironic node-list** コマンドを実行すると取得することができます。**DEPLOY_KERNEL_ID** は deploy kernel イメージの ID に置き換えます。この値は、director ノード上で **glance image-list** コマンドを実行すると取得することができます。**DEPLOY_RAMDISK_ID** は deploy ramdisk イメージの ID に置き換えます。この値は、director ノード上で **glance image-list** コマンドを実行すると取得することができます。

2.1.5. Image デプロイメント向けのプロキシーサービス

オプションで、ベアメタルノードが Object Storage で HTTP または HTTPS プロキシーサービスを使用してベアメタルノードにイメージをダウンロードするように設定することが可能です。これにより、ベアメタルノードと同じ物理ネットワークセグメント内にイメージをキャッシュして、全体的なネットワークトラフィックを削減し、デプロイメント時間を短縮することができます。

作業を開始する前に

以下の点を追加で考慮した上で、プロキシーサーバーを設定します。

- 要求された URL に含まれているクエリーの場合でも、コンテンツのキャッシュを使用します。

- キャッシュの最大サイズを大きくして、イメージのサイズが収まるようにします。

注記

イメージに機密情報が含まれていない場合にのみ、プロキシーサーバーが暗号化なしでイメージを保管するようにします。

Image プロキシーの設定

1. Identity に管理ユーザーとしてアクセスするためのシェルを設定します。

```
# source ~stack/overcloudrc
```

2. ベアメタルノードドライバーが HTTP または HTTPS を使用するための設定

```
# openstack baremetal node set NODE_UUID \
   --driver_info image_https_proxy=HTTPS://PROXYIP:PORT
```

この例では、HTTPS プロトコルを使用します。HTTPS の代わりに HTTP を使用する場合には、**driverinfo/image_http_proxy** パラメーターを指定してください。

3. イメージが要求された時に、Bare Metal Provisioning が Object Storage のキャッシュ済み一時 URL を再使用するように設定します。

```
# openstack-config --set /etc/ironic/ironic.conf glance
swift_temp_url_cache_enabled=true
```

プロキシーサーバーは、URL のクエリーの部分に基づいて、要求が生成される度に毎回変更されるクエリーパラメーターが含まれている場合には、同じイメージには新規キャッシュエントリーを作成しません。

4. 生成された URL の有効期間を (秒単位で) 設定します。

```
# openstack-config --set /etc/ironic/ironic.conf glance
swift_temp_url_duration=DURATION
```

Object Storage サービスの一時 URL のキャッシュからは、有効期限が切れていないイメージリンクのみが返されます。swift_temp_url_duration=1200 と設定した場合には、新しいイメージはプロキシーサーバーによって 20分後にキャッシュされます。このオプションの値は、swift_temp_url_expected_download_start_delay の値以上である必要があります。

5. ハードウェアに対して、ダウンロードを開始するまでの遅延時間 (秒単位) を設定します。

```
# openstack-config --set /etc/ironic/ironic.conf glance
swift_temp_url_expected_download_start_delay=DELAY
```

DELAY には、デプロイが要求 (一時 URL が生成) されてから、その URL を使用してイメージがベアメタルにダウンロードされるまでの遅延時間を設定します。この遅延時間により、ramdisk を起動してからイメージのダウンロードを開始することができます。この値により、イメージのダウンロード開始時にキャッシュされたエントリーが引き続き有効な状態となるかどうかが決定します。

2.1.6. Bare Metal Provisioning ノードのデプロイ

nova boot コマンドを使用して Bare Metal Provisioning ノードをデプロイします。

```
# nova  boot --image BAREMETAL_USER_IMAGE --flavor BAREMETAL_FLAVOR --nic
net-id=IRONIC_NETWORK_ID --key default MACHINE_HOSTNAME
```

BAREMETAL_USER_IMAGE は Image サービスにアップロードされたイメージに、**BAREMETAL_FLAVOR** はそのベアメタルデプロイメント用のフレーバーに、**IRONIC_NETWORK_ID** は OpenStack Networking 内のベアメタルプロビジョニングネットワークの ID に、**MACHINE_HOSTNAME** はデプロイ後のマシンのホスト名に置き換えます。

2.2. ハードウェア検査の設定

ハードウェア検査により、Bare Metal Provisioning はノード上のハードウェア情報を検出することができます。検査は、検出したイーサネットの MAC アドレスも作成します。または、各ノードにハードウェア情報を手動で追加することが可能です。詳しくは、「手動によるノードの追加」を参照してください。以下の手順に記載するステップはすべて、Bare Metal Provisioning コンダクターサービスをホストするサーバーで **root** ユーザーとしてログインして実行する必要があります。

ハードウェア検査は、以下のドライバーを使用するインバンドでサポートされます。

- **pxe_drac**
- **pxe_ipmitool**
- **pxe_ssh**
- **pxe_amt**

ハードウェア検査の設定

1. PXE ブートを使用したベアメタルシステムの検出に使用する **Ironic Python Agent** の kernel と ramdisk のイメージを取得します。これらのイメージは、https://access.redhat.com/downloads/content/191/ver=9/rhel---7/9/x86_64/product-software で **Ironic Python Agent Image for RHOSP director 9.0** というラベルの付いた TAR アーカイブで提供されています。この TAR アーカイブをダウンロードして、そこからイメージファイル (**ironic-python-agent.kernel** および **ironic-python-agent.initramfs**) を抽出してから、それらを TFTP サーバーの **/tftpboot** ディレクトリーにコピーしてください。

2. ハードウェア検査をホストするサーバーで **Red Hat OpenStack Platform 9 director for RHEL 7 (RPMs)** チャンネルを有効にします。

   ```
   # subscription-manager repos --enable=rhel-7-server-openstack-9-
   director-rpms
   ```

3. **openstack-ironic-inspector** パッケージをインストールします。

   ```
   # yum install openstack-ironic-inspector
   ```

4. **ironic.conf** ファイルで検査を有効にします。

   ```
   # openstack-config --set /etc/ironic/ironic.conf \
      inspector enabled True
   ```

5. ハードウェア検査サービスを別のサーバーでホストする場合には、コンダクターサービスをホストするサーバーでその URL を設定します。

```
# openstack-config --set /etc/ironic/ironic.conf \
   inspector service_url http://INSPECTOR_IP:5050
```

INSPECTOR_IP はハードウェア検査をホストするサーバーのホスト名または IP アドレスに置き換えます。

6. ハードウェア検査サービスに認証情報を提供します。

```
# openstack-config --set /etc/ironic-inspector/inspector.conf \
   keystone_authtoken identity_uri http://IDENTITY_IP:35357
# openstack-config --set /etc/ironic-inspector/inspector.conf \
   keystone_authtoken auth_uri http://IDENTITY_IP:5000/v2.0
# openstack-config --set /etc/ironic-inspector/inspector.conf \
   keystone_authtoken admin_user ironic
# openstack-config --set /etc/ironic-inspector/inspector.conf \
   keystone_authtoken admin_password PASSWORD
# openstack-config --set /etc/ironic-inspector/inspector.conf \
   keystone_authtoken admin_tenant_name services
# openstack-config --set /etc/ironic-inspector/inspector.conf \
   ironic os_auth_url http://IDENTITY_IP:5000/v2.0
# openstack-config --set /etc/ironic-inspector/inspector.conf \
   ironic os_username ironic
# openstack-config --set /etc/ironic-inspector/inspector.conf \
   ironic os_password PASSWORD
# openstack-config --set /etc/ironic-inspector/inspector.conf \
   ironic os_tenant_name service
# openstack-config --set /etc/ironic-inspector/inspector.conf \
   firewall dnsmasq_interface br-ironic
# openstack-config --set /etc/ironic-inspector/inspector.conf \
   database connection sqlite:////var/lib/ironic-
inspector/inspector.sqlite
```

以下の値を置き換えてください。

- **IDENTITY_IP** は Identity サーバーの IP アドレスまたはホスト名に置き換えます。
- **PASSWORD** は、Bare Metal Provisioning が Identity との認証で使用するパスワードに置き換えます。

7. オプションで、ハードウェア検査が RAM ディスクのログを保管するように設定します。

```
# openstack-config --set /etc/ironic-inspector/inspector.conf \
processing ramdisk_logs_dir /var/log/ironic-inspector/ramdisk
```

8. オプションで、複数のローカルディスクを使用するベアメタルマシンでブロックデバイスを収集してルートデバイスを公開する追加のデータ処理プラグインを有効にします。**ramdisk_error**、**root_disk_selection**、**scheduler**、**validate_interfaces** はデフォルトで有効化され、無効にすべきではありません。以下のコマンドを実行して、**root_device_hint** を一覧に追加します。

```
# openstack-config --set /etc/ironic-inspector/inspector.conf \
processing processing_hooks
'$default_processing_hooks,root_device_hint'
```

9. 初期 **ironic inspector** データベースを生成します。

```
# ironic-inspector-dbsync --config-file /etc/ironic-
inspector/inspector.conf upgrade
```

10. inspector データベースファイルを更新して、ironic-inspector が所有するようにします。

```
# chown ironic-inspector /var/lib/ironic-inspector/inspector.sqlite
```

11. テキストエディターで **/etc/ironic-inspector/dnsmasq.conf** ファイルを開き、以下に示す PXE ブート設定を **openstack-ironic-inspector-dnsmasq** サービスに設定します。

```
port=0
interface=br-ironic
bind-interfaces
dhcp-range=START_IP,END_IP
enable-tftp
tftp-root=/tftpboot
dhcp-boot=pxelinux.0
```

以下の値を置き換えてください。

- **INTERFACE** はプロビジョニングネットワークインターフェースの名前に置き換えます。
- **START_IP** は、Floating IP アドレスを割り当てる元となる IP アドレス範囲の開始アドレスを示す IP アドレスに置き換えます。
- **END_IP** は、Floating IP アドレスを割り当てる元となる IP アドレス範囲の終了アドレスを示す IP アドレスに置き換えます。

12. **syslinux bootloader** を **tftp** ディレクトリーにコピーします。

```
# cp /usr/share/syslinux/pxelinux.0 /tftpboot/pxelinux.0
```

13. オプションで、**swift** section of the **/etc/ironic-inspector/inspector.conf** ファイルの **swift** セクションに、メタデータを保管するためのハードウェア検査サービスを設定することができます。

```
[swift]
username = ironic
password = PASSWORD
tenant_name = service
os_auth_url = http://IDENTITY_IP:5000/v2.0
```

以下の値を置き換えてください。

- **IDENTITY_IP** は Identity サーバーの IP アドレスまたはホスト名に置き換えます。
- **PASSWORD** は、Bare Metal Provisioning が Identity との認証で使用するパスワードに置き換えます。

14. テキストエディターで **/tftpboot/pxelinux.cfg/default** ファイルを開き、以下のオプションを設定します。

```
default discover

label discover
kernel ironic-python-agent.kernel
append initrd=ironic-python-agent.initramfs \
ipa-inspection-callback-url=http://INSPECTOR_IP:5050/v1/continue
ipa-api-url=http://IRONIC_API_IP:6385

ipappend 3
```

INSPECTOR_IP はハードウェア検査をホストするサーバーのホスト名または IP アドレスに置き換えます。上記のブロックに \ で示したように、**append** から **/continue** までのテキストは、一行に記載する必要がある点に注意してください。

15. **/tftpboot/** ディレクトリーおよびそのファイルのセキュリティーコンテキストをリセットします。

```
# restorecon -R /tftpboot/
```

このステップでは、ディレクトリーに正しい SELinux セキュリティーラベルが設定され、dnsmasq サービスがディレクトリーにアクセスできることを確認します。

16. ハードウェア検査サービスと dnsmasq サービスを起動して、ブート時に開始するように設定します。

```
# systemctl start openstack-ironic-inspector.service
# systemctl enable openstack-ironic-inspector.service
# systemctl start openstack-ironic-inspector-dnsmasq.service
# systemctl enable openstack-ironic-inspector-dnsmasq.service
```

ハードウェア検査は、Bare Metal Provisioning に登録された後に、ノードで使用することができます。

2.3. ベアメタルノードとしての物理マシンの追加

インスタンスをプロビジョニングする物理マシンをノードとして追加し、Compute に対して利用可能なハードウェアが表示されることを確認します。Compute のリソーストラッカーは一定の時間間隔で同期するので、新規リソースは Compute に即時には通知されません。変更は次の定期タスクが実行された後に表示されるようになります。この値 (**scheduler_driver_task_period**) は、**/etc/nova/nova.conf** で更新することができます。デフォルトの間隔は 60 秒です。

システムがベアメタルノードとして登録されると、ハードウェアの詳細は、ハードウェア検査を使用して検出するか、手動で追加することができます。

2.3.1. ハードウェア検査を使用したノードの追加

物理マシンをベアメタルノードとして登録してから、**openstack-ironic-inspector** を使用してノードのハードウェア情報を検出し、各イーサネット MAC アドレス用にポートを作成します。以下の手順に記載するステップはすべて、Bare Metal Provisioning コンダクターサービスをホストするサーバーに **root** ユーザーとしてログインして実行する必要があります。

ハードウェア検査を使用してノードを追加する方法

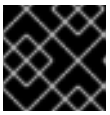
1. Identity を管理ユーザーとして使用するためのシェルを設定します。

```
# source ~/keystonerc_admin
```

2. 新規ノードを追加します。

```
# ironic node-create -d DRIVER_NAME
```

DRIVER_NAME は、Bare Metal Provisioning がこのノードのプロビジョニングに使用するドライバーの名前に置き換えます。このドライバーは、**/etc/ironic/ironic.conf** ファイルで有効にしておく必要があります。ノードを作成するには、少なくともドライバーの名前を指定する必要があります。



重要

ノードの一意識別子を書き留めておきます。

3. ノードは論理名または UUID で参照することができます。オプションで、ノードに論理名を割り当てます。

```
# ironic node-update NODE_UUID add name=NAME
```

NODE_UUID は、ノードの一意識別子に置き換えます。**NAME** は、ノードの論理名に置き換えます。

4. ドライバーが必要とするノードの情報を確認してから、Bare Metal Provisioning がノードを管理するようにノードドライバーの情報を更新します。

```
# ironic driver-properties DRIVER_NAME
# ironic node-update NODE_UUID add \
   driver_info/PROPERTY=VALUE \
   driver_info/PROPERTY=VALUE
```

以下の値を置き換えてください。

- **DRIVER_NAME** は、プロパティーを表示するドライバーの名前に置き換えます。この情報は、**/etc/ironic/ironic.conf** ファイルでドライバーを有効にしていなければ返されません。
- **NODE_UUID** は、ノードの一意識別子に置き換えます。もしくは、ノードの論理名を使用します。
- **PROPERTY** は、**ironic driver-properties** コマンドで返された必要なプロパティーに置き換えます。
- **VALUE** は、プロパティーの有効な値に置き換えます。

5. ノードドライバーのデプロイカーネルとデプロイ RAM ディスクを指定します。

```
# ironic node-update NODE_UUID add \
  driver_info/deploy_kernel=KERNEL_UUID \
  driver_info/deploy_ramdisk=INITRAMFS_UUID
```

以下の値を置き換えてください。

- **NODE_UUID** は、ノードの一意識別子に置き換えます。もしくは、ノードの論理名を使用します。
- **KERNEL_UUID** は、Image サービスにアップロードされた **.kernel** イメージの一意識別子に置き換えます。
- **INITRAMFS_UUID** は、Image サービスにアップロードされた **.initramfs** イメージの一意識別子に置き換えます。

6. ノードを設定して、初回のデプロイの後には、PXE や仮想メディアの代わりに、そのノードのディスクにインストールされたローカルのブートローダーからリブートするようにします。ノードのプロビジョニングに使用するフレーバーでも、ローカルブートの機能を設定する必要があります。ローカルブートを有効にするには、ノードのデプロイに使用するイメージに **grub2** が含まれる必要があります。ローカルブートを以下のように設定します。

```
# ironic node-update NODE_UUID add \
   properties/capabilities="boot_option:local"
```

NODE_UUID は、ノードの一意識別子に置き換えます。もしくは、ノードの論理名を使用します。

7. ベアメタルノードを **manageable** 状態に切り替えます。

```
# ironic node-set-provision-state NODE_UUID manage
```

NODE_UUID は、ノードの一意識別子に置き換えます。もしくは、ノードの論理名を使用します。

8. 検査を開始します。

```
# openstack baremetal introspection start NODE_UUID --discoverd-url
http://overcloud IP:5050
```

- **NODE_UUID** はノードの一意識別子に置き換えます。もしくは、ノードの論理名を使用します。ノードをプロビジョニングする前に、ノードの検出および検査プロセスは完了するまで実行する必要があります。ノードの検査のステータスを確認するには、**ironic node-list** を実行して **Provision State** の箇所を探します。検査が正常に完了すると、ノードは **available** の状態になります。
- **overcloud IP** は **ironic.conf** で以前設定した **service_url** の値に置き換えます。

9. ノードの設定を検証します。

```
# ironic node-validate NODE_UUID
+------------+--------+----------------------------+
| Interface  | Result | Reason                     |
+------------+--------+----------------------------+
| console    | None   | not supported              |
| deploy     | True   |                            |
| inspect    | True   |                            |
| management | True   |                            |
| power      | True   |                            |
+------------+--------+----------------------------+
```

NODE_UUID は、ノードの一意識別子に置き換えます。もしくは、ノードの論理名を使用します。上記のコマンドの出力には、各インターフェースが **True** または **None** のいずれかと報告されるはずです。**None** とマークされたインターフェースは、設定していないか、ドライバーがサポートしていないインターフェースです。

2.3.2. 手動によるノードの追加

物理マシンをベアメタルノードとして登録してから、ノードのハードウェア情報を追加し、各イーサネット MAC アドレス用にポートを作成します。以下の手順に記載するステップはすべて、Bare Metal Provisioning コンダクターサービスをホストするサーバーに **root** ユーザーとしてログインして実行する必要があります。

ハードウェア検査を使用せずにノードを追加する方法

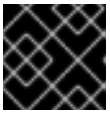
1. Identity を管理ユーザーとして使用するためのシェルを設定します。

```
# source ~/keystonerc_admin
```

2. 新規ノードを追加します。

```
# ironic node-create -d DRIVER_NAME
```

DRIVER_NAME は、Bare Metal Provisioning がこのノードのプロビジョニングに使用するドライバーの名前に置き換えます。このドライバーは、**/etc/ironic/ironic.conf** ファイルで有効にしておく必要があります。ノードを作成するには、少なくともドライバーの名前を指定する必要があります。



重要

ノードの一意識別子を書き留めておきます。

3. ノードは論理名または UUID で参照することができます。オプションで、ノードに論理名を割り当てます。

```
# ironic node-update NODE_UUID add name=NAME
```

NODE_UUID は、ノードの一意識別子に置き換えます。**NAME** は、ノードの論理名に置き換えます。

4. ドライバーが必要とするノードの情報を確認してから、Bare Metal Provisioning がノードを管理するようにノードドライバーの情報を更新します。

```
# ironic driver-properties DRIVER_NAME
# ironic node-update NODE_UUID add \
   driver_info/PROPERTY=VALUE \
   driver_info/PROPERTY=VALUE
```

以下の値を置き換えてください。

- **DRIVER_NAME** は、プロパティーを表示するドライバーの名前に置き換えます。この情報は、**/etc/ironic/ironic.conf** ファイルでドライバーを有効にしていなければ返されません。
- **NODE_UUID** は、ノードの一意識別子に置き換えます。もしくは、ノードの論理名を使用します。

- **PROPERTY** は、**ironic driver-properties** コマンドで返された必要なプロパティーに置き換えます。
- **VALUE** は、プロパティーの有効な値に置き換えます。

5. ノードドライバーのデプロイカーネルとデプロイ RAM ディスクを指定します。

```
# ironic node-update NODE_UUID add \
  driver_info/deploy_kernel=KERNEL_UUID \
  driver_info/deploy_ramdisk=INITRAMFS_UUID
```

以下の値を置き換えてください。

- **NODE_UUID** は、ノードの一意識別子に置き換えます。もしくは、ノードの論理名を使用します。
- **KERNEL_UUID** は、Image サービスにアップロードされた **.kernel** イメージの一意識別子に置き換えます。
- **INITRAMFS_UUID** は、Image サービスにアップロードされた **.initramfs** イメージの一意識別子に置き換えます。

6. ノードのプロパティーを更新して、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。

```
# ironic node-update NODE_UUID add \
   properties/cpus=CPU \
   properties/memory_mb=RAM_MB \
   properties/local_gb=DISK_GB \
   properties/cpu_arch=ARCH
```

以下の値を置き換えてください。

- **NODE_UUID** は、ノードの一意識別子に置き換えます。もしくは、ノードの論理名を使用します。
- **CPU** は、使用する CPU 数に置き換えます。
- **RAM_MB** は、使用するメモリー (MB 単位) に置き換えます。
- **DISK_GB** は、使用するディスクサイズ (GB 単位) に置き換えます。
- **ARCH** は使用するアーキテクチャーのタイプに置き換えます。

7. ノードを設定して、初回のデプロイの後には、PXE や仮想メディアの代わりに、そのノードのディスクにインストールされたローカルのブートローダーからリブートするようにします。ノードのプロビジョニングに使用するフレーバーでも、ローカルブートの機能を設定する必要があります。ローカルブートを有効にするには、ノードのデプロイに使用するイメージに **grub2** が含まれる必要があります。ローカルブートを以下のように設定します。

```
# ironic node-update NODE_UUID add \
   properties/capabilities="boot_option:local"
```

NODE_UUID は、ノードの一意識別子に置き換えます。もしくは、ノードの論理名を使用します。

8. Bare Metal Provisioning にノードのネットワークインターフェースカードの情報を提供します。各 NIC の MAC アドレスでポートを作成します。

```
# ironic port-create -n NODE_UUID -a MAC_ADDRESS
```

NODE_UUID は、ノードの一意識別子に置き換えます。**MAC_ADDRESS** は、ノード上の 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。

```
# ironic node-validate NODE_UUID
+------------+--------+----------------------------+
| Interface  | Result | Reason                     |
+------------+--------+----------------------------+
| console    | None   | not supported              |
| deploy     | True   |                            |
| inspect    | None   | not supported              |
| management | True   |                            |
| power      | True   |                            |
+------------+--------+----------------------------+
```

NODE_UUID は、ノードの一意識別子に置き換えます。もしくは、ノードの論理名を使用します。上記のコマンドの出力には、各インターフェースが **True** または **None** のいずれかと報告されるはずです。**None** とマークされたインターフェースは、設定していないか、ドライバーがサポートしていないインターフェースです。

2.3.3. 手動によるノードのクリーニングの設定

ベアメタルサーバーの初めてプロビジョニングされる時や、そのサーバーがワークロードから解放された後に再プロビジョニングされる時には、Bare Metal Provisioning がそのサーバーを自動的にクリーニングして、そのサーバーが別のワークロードに対応する準備を整えることができます。また、サーバーが管理可能な状態の時には、手動のクリーニングサイクルを開始することも可能です。手動のクリーニングサイクルは、長時間実行されるタスクや破壊的なタスクに役立ちます。ベアメタルサーバー特定のクリーニングステップを設定することができます。

クリーニングネットワークの設定

Bare Metal Provisioning はクリーニングネットワークを使用して、ベアメタルサーバーのインバンドのクリーニングステップを提供します。このクリーニングネットワークには別のネットワークを作成するか、プロビジョニングネットワークを使用することができます。

Bare Metal Provisioning サービスのクリーニングネットワークを設定するには、以下の手順に従ってください。

1. Identity に管理ユーザーとしてアクセスするためのシェルを設定します。

```
# source ~stack/overcloudrc
```

2. Bare Metal Provisioning がベアメタルサーバーのクリーニングに使用するネットワークの UUID を特定します。

```
# openstack network list
```

neutron net-list の出力の **id** フィールドから UUID を選択します。

3. **/etc/ironic/ironic.conf** ファイルの **cleaning_network_uuid** をクリーニングネットワークの UUID に設定します。

```
# openstack-config --set /etc/ironic/ironic.conf neutron
cleaning_network_uuid CLEANING_NETWORK_UUID
```

CLEANING_NETWORK_UUID は、以前のステップで取得したネットワークの **id** に置き換えます。

4. Bare Metal Provisioning サービスを再起動します。

```
# systemctl restart openstack-ironic-conductor
```

手動クリーニングの設定

1. ベアメタルサーバーが管理可能な状態であることを確認します。

```
# ironic node-set-provision-state NODE_ID manage
```

NODE_ID はベアメタルサーバーの UUID またはノード名に置き換えます。

2. ベアメタルサーバーをクリーニングの状態に切り替えてクリーニングのステップを実行します。

```
# ironic node-set-provision-state NODE_ID clean --clean-steps
CLEAN_STEPS
```

NODE_ID は、ベアメタルサーバーの UUID またはノード名に置き換えます。**CLEAN_STEPS** は JSON 形式のクリーニングステップまたはクリーニングステップが記載されたファイルへのパスに置き換えるか、標準入力から直接置き換えます。

```
 '[{"interface": "deploy", "step": "erase_devices"}]'
```

詳しくは、「OpenStack - Node Cleaning」を参照してください。

2.3.4. ベアメタルノード上で優先するルートディスクの指定

deploy **ramdisk** がベアメタル上でブートする際には、Bare Metal Provisioning が最初に検出するディスクが **root** デバイス (イメージが保存されるデバイス) となります。ベアメタルノードに SATA、SCSI、または IDE ディスクコントローラーが複数ある場合には、対応するディスクデバイスが追加される順序は任意で、再起動するたびに変更される場合があります。たとえば、**/dev/sda** および **/dev/sdb** は起動するたびに切り替わるので、ベアメタルノードがデプロイされる度に Bare Metal Provisioning が異なるディスクを選択することになります。

ディスクのヒントを使用すると、deploy **ramdisk** にヒントを渡して、Bare Metal Provisioning がイメージをデプロイする先のディスクを指定することができます。以下の表には、ベアメタル上で優先する **root** ディスクを選択するのに使用するヒントをまとめています。

表2.1 ディスクのヒント

ヒント	型	説明
model	(STRING):	ディスクデバイスの識別子
vendor	(STRING):	ディスクデバイスのベンダー
serial	(STRING):	ディスクデバイスのシリアル番号
size	(INT):	ディスクデバイスのサイズ (GB 単位) **注記**: ノードの **`local_gb`** プロパティーは、パーティショニングを考慮に入れて、実際のディスクサイズよりも 1 GB 少ない値に設定されることが多くあります。ただし、この場合には **`size`** は実際のサイズにすべきです。たとえば、128 GB のディスク **`local_gb`** は 127 ですが、 **`size`** ヒントは 128 です。
wwn	(STRING):	ストレージー意識別子
wwn_with_extension	(STRING):	ベンダー拡張が末尾に付いたストレージー意識別子
wwn_vendor_extension	(STRING):	ベンダーのストレージー意識別子
name	(STRING):	デバイス名 (例: /dev/md0) **警告**: ルートデバイスヒント名は一定の名前が付いたデバイス (例: RAID ボリューム) のみを使用すべきです。SATA、SCSI、IDE ディスクコントローラーを使用すると、Linux 内でデバイスノードが任意の順序で追加されて、/dev/sda や /dev/sdb などのデバイスがブート時に切り替わってしまうので、これらのディスクコントローラーは使用しないでください。詳しくは、「永続的な命名」を参照してください。

1 つのベアメタルノードに 1 つまたは複数のディスクデバイスヒントがある場合は、ノードのプロパティーを **`root_device`** キーで更新します。以下に例を示します。

```
# ironic node-update <node-uuid> add properties/root_device='{"wwn":
"0x4000cca77fc4dba1"}'
```

この例では、指定した WWN 値と等しい **`wwn`** が設定されたディスクデバイスを Bare Metal Provisioning が選択することを保証します。

ヒントには、値の文字列の最初に演算子を付けることができます。演算子が指定されていない場合には、デフォルトは **==** (数値) と **s==** (文字列の値) です。

表2.2 サポートされている演算子

型	演算子	説明
数値	**=**	等しいかより大きい (**>=** と同等)

型	演算子	説明
	==	等しい
	!=	等しくない
	>=	より大きいか等しい
	>	より大きい
	<=	より小さいか等しい
	<	より小さい
文字列 (python の比較)	**s==**	等しい
	s!=	等しくない
	s>=	より大きいか等しい
	s>	より大きい
	s<=	より小さいか等しい
	s<	より小さい
	<in>	サブ文字列
コレクション	**<all-in>**	コレクションに含まれる全要素
	<or>	それらのいずれか 1 つを検索

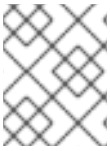
以下の例は、ベアメタルノードのプロパティーを更新して特定のディスクを選択する方法を示しています。

- 60 GB より大きいか等しい非回転 (SSD)ディスクを検索します。

```
# ironic node-update <node-uuid> add properties/root_device='{"size": ">=
60", "rotational": false}'
```

- Samsung または Winsys ディスクを検索します。

```
# ironic node-update <node-uuid> add properties/root_device='{"vendor": "
<or> samsung <or> winsys"}'
```



注記

複数のヒントが指定されている場合には、ディスクデバイスはそれらのヒントをすべて満たす必要があります。

2.4. ホストアグリゲートを使用した物理/仮想マシンのプロビジョニングの分離

ホストアグリゲートは、OpenStack Compute がアベイラビリティーゾーンを分割して、共通する特定のプロパティーでノードをグループ化するのに使用します。キーと値のペアは、ホストアグリゲートとインスタンスのフレーバーの両方で設定され、これらのプロパティーを定義します。インスタンスのプロビジョニング時には、Compute のスケジューラーがフレーバー上のキーと値のペアをホストアグリゲートに割り当てられたキーと値のペアと比較し、物理マシン上または **openstack-nova-compute** ノード上の仮想マシンとして、インスタンスが正しいアグリゲート内かつ正しいホスト上にプロビジョニングされるようにします。

Red Hat OpenStack Platform 環境がベアメタルマシンと仮想マシンの両方をプロビジョニングするように設定されている場合には、ホストアグリゲートを使用してインスタンスが物理マシンまたは仮想マシンとして起動するように指示します。 以下の手順では、ベアメタルホスト用のホストアグリゲートを作成し、ホストの種別を **baremetal** に指定してキーと値のペアを追加します。このアグリゲートに分類されたベアメタルノードはいずれも、このキーと値のペアを継承します。同じキーと値のペアは、その後にインスタンスのプロビジョニングに使用されるフレーバーに追加されます。

ベアメタルのプロビジョニングに使用するイメージ (1 つまたは複数) が **hypervisor_type=ironic** プロパティーを設定して Image サービスにアップロードされた場合には、スケジューラーもスケジューリングの決定でそのキーと値のペアを使用します。イメージのプロパティーが適用されなかった場合に有効なスケジューリングを行うようにするには、イメージプロパティーの設定に加えて、ホストアグリゲートを追加で設定します。イメージのビルド/アップロードについての詳しい情報は、「ベアメタルイメージの作成」を参照してください。

Bare Metal Provisioning 向けのホストアグリゲートの作成

1. デフォルトの **nova** アベイラビリティーゾーンで **baremetal** 用のホストアグリゲートを作成します。

```
# nova aggregate-create baremetal nova
```

2. アグリゲートに追加されたホストに **hypervisor_type=ironic** プロパティーを割り当てる **baremetal** アグリゲート上のメタデータを設定します。

```
# nova aggregate-set-metadata baremetal hypervisor_type=ironic
```

3. **openstack-nova-compute** ノードを Bare Metal Provisioning ドライバーとともに **baremetal** アグリゲートに追加します。

```
# nova aggregate-add-host baremetal COMPUTE_HOSTNAME
```

COMPUTE_HOSTNAME は、**openstack-nova-compute** サービスをホストするシステムのホスト名に置き換えます。Bare Metal Provisioning の全要求を処理するには、専用のコンピュートホストを 1 台使用するべきです。

4. プロビジョニング用に作成したフレーバー (1 つまたは複数) に **ironic** のハイパーバイザーのプロパティーを追加します。

```
# nova flavor-key FLAVOR_NAME set hypervisor_type="ironic"
```

FLAVOR_NAME はフレーバー名に置き換えます。

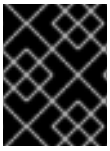
5. **/etc/nova/nova.conf** の **scheduler_default_filters** の下にある既存の一覧に以下の Compute フィルタースケジューラーを追加します。

```
AggregateInstanceExtraSpecsFilter
```

このフィルターは、ホストアグリゲートに割り当てられたキーと値のペアを Compute のスケジューラーが確実に処理するようにします。

2.5. SSH と VIRSH を使用した BARE METAL PROVISIONING のテスト例

単一の物理ホストで、ベアメタルノードとして機能する 2 つの仮想マシン上にインスタンスをデプロイして、Bare Metal Provisioning の設定をテストします。両仮想マシンは **libvirt** と **virsh** を使用して仮想化されます。



重要

この SSH ドライバーは、検証および評価のみを目的としており、Red Hat OpenStack Platform のエンタープライズ環境には推奨されません。

このシナリオには以下のリソースが必要です。

- Bare Metal Provisioning をオーバークラウドノード上に設定した Red Hat OpenStack Platform 環境を 1 つ。本ガイドの全ステップを完了しておく必要があります。
- Red Hat Enterprise Linux 7.2 と **libvirt** 仮想化ツールをインストール済みのベアメタルマシンを 1 つ。このシステムは、仮想ベアメタルノードが属するホストとして機能します。
- Bare Metal Provisioning ノードと、仮想ベアメタルノードが属するホストとの間のネットワーク接続を 1 つ。このネットワークはベアメタルプロビジョニングネットワークとして機能します。

2.5.1. 仮想ベアメタルノードの作成

テストシナリオでベアメタルノードとして機能する仮想マシンを 2 つ作成します。 ノードは **Node1** および **Node2** と呼びます。

仮想ベアメタルノードの作成

1. **libvirt** ホストから仮想マシンマネージャーにアクセスします。
2. 以下の設定で仮想マシンを 2 つ作成します。
   - 1 x 仮想 CPU
   - 2048 MB のメモリー
   - ネットワークブート (PXE)
   - 20 GB のストレージ
   - **ネットワークソース**: **ホストデバイス eth0: macvtap**、**ソースモード**: **Bridge**。

macvtap を選択すると、仮想マシンはホストのイーサネットネットワークインターフェースを共有します。このように設定すると、Bare Metal Provisioning ノードは仮想ノードに直接アクセスすることが可能となります。

3. 両仮想マシンをシャットダウンします。

2.5.2. SSH キーペアの作成

Bare Metal Provisioning ノードが **libvirt** ホストに接続できるようにする SSH キーペアを作成します。

SSH キーペアの作成

1. Bare Metal Provisioning ノードで、SSH キーを作成します。

```
# ssh-keygen -t rsa -b 2048 -C "user@domain.com" -f ./virtkey
```

user@domain.com は、このキーを特定するメールアドレスまたはその他のコメントに置き換えます。コマンドがパスフレーズの入力を要求したら、パスフレーズは入力せずに **Enter** を押して次に進みます。このコマンドにより、秘密鍵 (**virtkey**) と公開鍵 (**virtkey.pub**) の 2 つのファイルが作成されます。

2. 公開鍵の内容を **libvirt** ホストの **root** ユーザーの **/root/.ssh/authorized_keys** ファイルにコピーします。

```
# ssh-copy-id -i virtkey root@LIBVIRT_HOST
```

LIBVIRT_HOST は **libvirt** ホストの IP アドレスまたはホスト名に置き換えます。

秘密鍵 (**virtkey**) は、ノードの登録時に使用されます。

2.5.3. ベアメタルノードとしての仮想ノードの追加

インスタンスをプロビジョニングする仮想マシンをノードとして追加します。以下の例では、ドライバーの詳細は手動で指定し、ノードの詳細はハードウェア検査を使用して検出します。ノードの詳細は、ノード別に手動で追加することも可能です。詳しくは、「手動によるノードの追加」を参照してください。

仮想ノードをベアメタルノードとして追加する手順

1. Bare Metal Provisioning コンダクターサービスノードで、**pxe_ssh** ドライバーを有効にします。

```
# openstack-config --set /etc/ironic/ironic.conf \
   DEFAULT enabled_drivers pxe_ssh
```

pxe_ssh を既存のドライバーの一覧に追加する場合には、ファイルを開いて、**enabled_drivers** に記載されているリストにドライバーをコンマ区切りで追加します。

2. Identity を管理ユーザーとして使用するためのシェルを設定します。

```
# source ~/keystonerc_admin
```

3. 最初のノードを追加して、**libvirt** ホストの SSH 情報を登録します。

```
# ironic node-create -d pxe_ssh -n Node1 \
   -i ssh_virt_type=virsh \
   -i ssh_username=root \
   -i ssh_key_filename=VIRTKEY_FILE_PATH \
   -i ssh_address=LIBVIRT_HOST_IP \
   -i deploy_kernel=KERNEL_UUID \
   -i deploy_ramdisk=INITRAMFS_UUID
```

以下の値を置き換えてください。

- **VIRTKEY_FILE_PATH** は **virtkey** SSH 秘密鍵ファイルの絶対パスに置き換えます。
- **LIBVIRT_HOST_IP** は、**libvirt** ホストの IP アドレスまたはホスト名に置き換えます。
- **KERNEL_UUID** は、Image サービスにアップロードされた **.kernel** イメージの一意識別子に置き換えます。
- **INITRAMFS_UUID** は、Image サービスにアップロードされた **.initramfs** イメージの一意識別子に置き換えます。

4. 上記と同じコマンドで、 **Node1** を **Node2** に置き換えて実行し、2 番目のノードを追加します。

5. ノードを設定して、初回のデプロイの後には、PXE や仮想メディアの代わりに、そのノードのディスクにインストールされたローカルのブートローダーからリブートするようにします。ノードのプロビジョニングに使用するフレーバーでも、ローカルブートの機能を設定しておく必要があります。ローカルブートを有効にするには、ノードのデプロイに使用するイメージに **grub2** が含まれる必要があります。ローカルブートを以下のように設定します。

```
# ironic node-update Node1 add \
   properties/capabilities="boot_option:local"
# ironic node-update Node2 add \
   properties/capabilities="boot_option:local"
```

6. ノードを管理可能な状態に切り替えます。

```
# ironic node-set-provision-state Node1 manage
# ironic node-set-provision-state Node2 manage
```

7. ノードに対する検査を開始します。

```
# ironic node-set-provision-state Node1 inspect
# ironic node-set-provision-state Node2 inspect
```

ノードをプロビジョニングする前に、ノードの検出および検査プロセスは完了するまで実行する必要があります。ノードの検査のステータスを確認するには、**ironic node-list** を実行して **Provision State** の箇所を探します。検査が正常に完了すると、ノードは **available** の状態になります。

8. ノードの設定を検証します。

```
# ironic node-validate Node1
# ironic node-validate Node2
+------------+--------+----------------------------+
| Interface  | Result | Reason                     |
```

```
+------------+--------+----------------------------+
| console    | None   | not supported              |
| deploy     | True   |                            |
| inspect    | True   |                            |
| management | True   |                            |
| power      | True   |                            |
+------------+--------+----------------------------+
```

上記のコマンドの出力には、各インターフェースが **True** または **None** と報告されるはずです。**None** とマークされたインターフェースは、設定していないか、ドライバーがサポートしていないインターフェースです。

9. ノードの追加が正常に完了したら、「3章 *ベアメタルインスタンスの起動*」の手順に従って 2 つのノードを起動します。

第3章 ベアメタルインスタンスの起動

登録済みのベアメタルノードで、物理マシンをプロビジョニングします。インスタンスは、コマンドラインまたは OpenStack Dashboard で起動することができます。

3.1. コマンドラインインターフェースを使用したインスタンスのデプロイ

nova コマンドラインインターフェースを使用してベアメタルインスタンスをデプロイします。

コマンドライン上でのインスタンスのデプロイ

1. Identity に管理ユーザーとしてアクセスするためのシェルを設定します。

```
# source ~/keystonerc_admin
```

2. インスタンスをデプロイします。

```
# nova boot --nic net-id=NETWORK_UUID --flavor FLAVOR_NAME --image
IMAGE_UUID INSTANCE_NAME
```

以下の値を置き換えてください。

- **NETWORK_UUID** は、Bare Metal Provisioning で使用するために作成したネットワークの一意識別子に置き換えます。
- **FLAVOR_NAME** は、そのノード用に作成されたフレーバーの名前に置き換えます。
- **IMAGE_UUID** は、Image サービスにアップロードされているディスクイメージの一意識別子に置き換えます。
- **INSTANCE_NAME** は、ベアメタルインスタンスの名前に置き換えます。

3. インスタンスのステータスを確認します。

```
# nova list
```

3.2. DASHBOARD を使用したインスタンスのデプロイ

Dashboard のグラフィカルユーザーインターフェースを使用してベアメタルインスタンスをデプロイします。

Dashboard でのインスタンスのデプロイ

1. **https://DASHBOARD_IP/dashboard** で Dashboard にログインします。
2. **プロジェクト > コンピュート > インスタンス** をクリックします。
3. **インスタンスの起動** をクリックします。
4. **詳細** タブで、以下のフィールドに必要事項を入力します。
   - **インスタンス名** を指定します。
   - ベアメタルノード用に作成した **フレーバー** を選択します。

- **インスタンス数** のリストから **1** を選択します。
- **インスタンスのブートソース** のリストから **イメージから起動** を選択します。
- **イメージ名** のリストからオペレーティングシステムのディスクイメージを選択します。

5. **ネットワーク** タブで、**利用可能なネットワーク** から **選択済みネットワーク** に必要なネットワークをドラッグアンドドロップします。Bare Metal Provisioning 用に作成した共有ネットワークを選択するようにしてください。

6. **起動** をクリックします。

3.3. WINDOWS 全体のイメージの作成

以下の手順では、Windows Server 2012 のデプロイメントイメージを作成します。Red Hat Enterprise Linux システムで以下のステップを実行します。

1. virtio-win ドライバーをダウンロードします。必要な手順 については、Red Hat カスタマーポータルを参照してください。
この手順を実行すると、**virtio-win** .iso ファイルが /**usr**/**share**/**virtio-win**/ にダウンロードされます (例: /**usr**/**share**/**virtio-win**/**virtio-win-1.8.0.iso**)。

2. **virt-manager** を使用して Windows Server 2012 のテンプレートを作成します。
   - Windows テンプレートの仮想ハードウェア要件を決定します。この例では、1 x CPU、4 GB メモリー、10 GB HDD、2 x NIC、2 x 仮想 CD-ROM ドライブを使用します。
   - **qcow2** ファイルとしてディスクを作成します。
   - 仮想 HDD と NIC ドライバーの両方を **`virtio`** に設定します。
   - 仮想 CD-ROM ドライブ 2 台をアタッチして、Windows Server 2012 R2 **.iso** ファイルと、**virtio-win-1.8.0.iso** ファイルをマウントします。Windows のインストールプロセスで **virtio-win** ドライブをインストールできるようにするために、仮想 CD-ROM が 2 台必要です。

3. ISO イメージから Windows を手動でインストールします。
   - Windows Server 2012 R2 評価版の .iso イメージから Windows のインストールを起動します。
   - HDD ドライバーを選択するように促された場合は、 virtio-win-1.8.0.iso ファイルを含む 2 番目の仮想 CD-ROM からドライバーを選択してください。

4. Windows のインストール後のチェックを実行します。
デバイスマネージャー を開き、すべてのデバイスが適切に認識されていることと、**感嘆符** の警告が表示されていないことを確認します。特に、VirtIO を使用する NIC、シリアル、バルーンドライバーを確認してください。正しく認識されていないデバイスがある場合は、virtio-win ドライバーディスクからドライバーを適用します。

5. **sysprep** を実行します。
Sysprep は、Windows のインストールを一般化することで、以前実行した単一のインストールに固有のインストール情報を削除します。これにより、仮想マシンの仮想ハードディスクをテンプレートとして使用して、他のシステムに対して複数回インストールすることができます。

   a. **C:\Windows\System32\sysprep\sysprep.exe** から **Sysprep** を起動します。

b. 以下の情報を Sysprep ツールに入力します。

- **システムクリーンアップアクション** 下で、**システムの OOBE (Out-of-Box-Experience) に入る** を選択します。
- **一般化** のチェックボックスを選択します。
- **シャットダウン** のオプションで **シャットダウン** を選択します。

c. **OK** をクリックして sysprep プロセスを完了します。プロセスが完了すると、仮想マシンは自動的にシャットダウンします。

6. Image サービス (glance) に Windows のイメージを登録します。このステップで、glance に Windows インストールの qcow2 HDD を登録します。この例で は、**/root/win2012r2.qcow2** というディスクファイルを使用します。

```
$ glance image-create --file /root/win2012r2.qcow2 --disk-format
qcow2 --container-format bare --name win2012r2 --is-public True --
progress
[==============================>] 100%
```

3.3.1. Windows の物理サーバーへのデプロイ

1. ハードウェアコンポーネントを指定して ironic に物理ノードを登録します。以下に例を示します。

```
# ironic node-create -d fake_pxe -p cpus=1 -p memory_mb=1024 -p
local_gb=15 -p cpu_arch=amd64
```

2. 新規ノードの UUID を取得します。

```
# ironic node-list
```

3. Windows の物理ノードの MAC アドレスに一致するノードのポートを作成します。

```
# ironic port-create -n [NODE_UUID] -a [NIC_MAC]
```

例:

```
# ironic port-create -n 3b3d3fd4-4621-427f-a78d-054a1ef17a0a -a
52:54:00:85:76:53
```

4. ノードをメンテナンスモードに切り替えます。

```
# ironic node-set-maintenance [NODE_UUID] true
# ironic node-set-provision-state [NODE_UUID] manage
```

5. 新規ノードの検査を行います。

```
# openstack baremetal introspection start [NODE_UUID] --discoverd-
url http://[KEYSTONE_URL]:5050
```

- **[KEYSTONE_URL]** は、keystone サービスの IP アドレスに置き換えます。

注記: FAKE_PXE ドライバーを使用する場合は、**ironic node-list** の状態が **deploy wait-callback** に変化した後に、手動でノードの電源を入れてください。

6. 検査が完了したら、メンテナンスモードをオフにしてください。

```
# ironic node-set-maintenance [NODE UUID] false
# ironic node-set-provision-state [NODE UUID] provide
```

7. インスタンスへの認証用キーペアを作成します。

```
# nova keypair-add --pub_key ~/.ssh/id_rsa.pub [KEY_NAME]
```

8. ノードのローカルブートを有効化します。

```
# ironic node-update [NODE_UUID] add
properties/capabilities="boot_option:local"
```

9. **nova** を使用してノードを起動します。

```
# nova boot --nic net-id=[NETWORK_UUID] --flavor [FLAVOR_NAME] --
image [IMAGE_UUID/IMAGE_NAME] --keyname [INSTANCE_NAME]
```

以下の値を置き換えてください。

- **[NETWORK_UUID]** は、Bare Metal Provisioning で使用するために作成したネットワークの一意識別子に置き換えます。
- **[FLAVOR_NAME]** は、そのノード用に作成されたフレーバーの名前に置き換えます。
- **[IMAGE_UUID]** は、Image サービスにアップロードされているディスクイメージの一意識別子に置き換えます。
- **[INSTANCE_NAME]** は、ベアメタルインスタンスの名前に置き換えます。

10. インスタンスのパスワードを取得します。

```
# nova get-password [INSTANCE_NAME]
/path/to/your/keypairs/private/key
```

11. インスタンスのステータスを確認します。

```
# nova list
```

Dashboard のコンソールを使用してインスタンスにアクセスすることで、さらにテストすることができます。

3.3.2. リモートデスクトップアクセスの有効化

1. **TCP/UDP 3389** で **リモートデスクトップ** のトラフィックを 許可するセキュリティーグループルールを追加します。
2. **nova** コマンドを使用して、セキュリティーキーを取得します。

```
# nova get-password [INSTANCE_NAME]
/path/to/your/keypairs/private/key
```

- **[INSTANCE_NAME]** は、ベアメタルインスタンスの名前に置き換えます。

第4章 BARE METAL PROVISIONING のトラブルシューティング

以下の項には、Bare Metal Provisioning の設定における問題を診断するのに役立つ可能性のある情報と手順を記載します。

Bare Metal Provisioning は検査で **openstack-ironic-api**、**openstack-ironic-conductor**、**openstack-ironic-inspector**、**openstack-ironic-inspector-dnsmasq** の 4 つのサービスを使用します。大半の OpenStack コンポーネントのログは、**/var/log** ディレクトリーで確認することができます。

4.1. PXE ブートのエラー

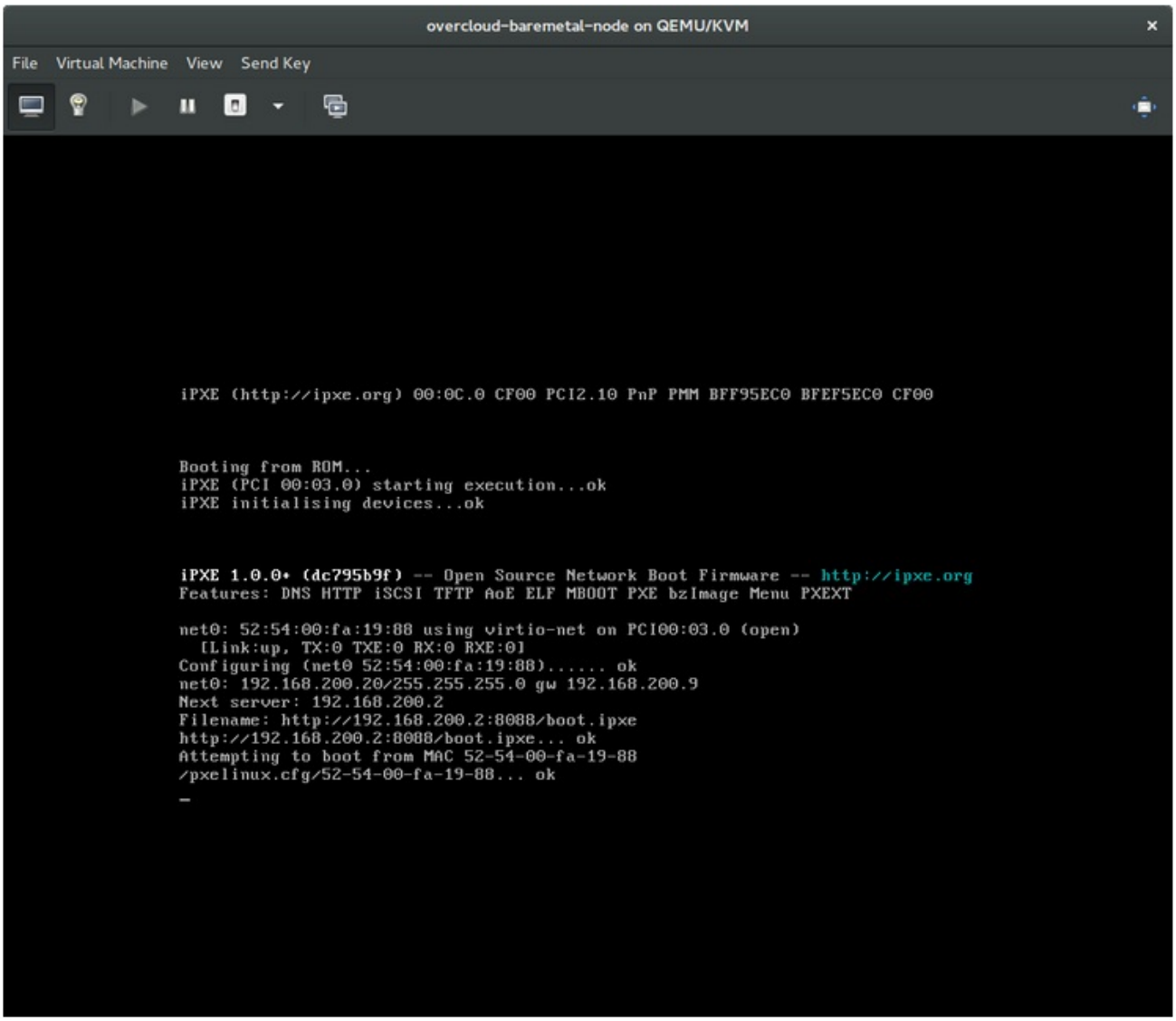
Permission Denied エラー

Bare Metal Provisioning ノードのコンソールで「Permission Denied」エラーが表示された場合には、以下に示したように、適切な SELinux の内容を **/httpboot** と **/tftpboot** のディレクトリーに必ず適用してください。

```
# semanage fcontext -a -t httpd_sys_content_t "/httpboot(/.*)?"
# semanage fcontext -a -t tftpdir_t "/tftpboot(/.*)?"
```

/pxelinux.cfg/XX-XX-XX-XX-XX-XX でのブートプロセスのフリーズ

以下の図に示したように、ノードのコンソールで、IP アドレスは取得されているように表示されており、プロセスが停止している場合:



これは、**ironic.conf** ファイルで誤った PXE ブートテンプレートを使用している可能性があることを示しています。

```
# grep ^pxe_config_template ironic.conf
pxe_config_template=$pybasedir/drivers/modules/ipxe_config.template
```

デフォルトのテンプレートは **pxe_config.template** なので、これを **ipxe_config.template** に変更するための **i** を忘れがちです。

4.2. ベアメタルノードの起動後のログインエラーのトラブルシュート

設定ステップで設定した **root** パスワードを使用して、ノードのコンソールのログインプロンプトでログインを試みてもログインできない場合には、デプロイしたイメージでブートしていないことを意味します。**deploy-kernel/deploy-ramdisk** イメージにスタックしてしまって、システムが正しいイメージを取得していない可能性があります。

この問題を修正するには、Compute または Bare Metal Provisioning ノードの **/httpboot/pxelinux.cfg/MAC_ADDRESS** にある PXE ブートの設定ファイルをチェックして、このファイルにリストされている全 IP アドレスがベアメタルプロビジョニングネットワークの IP アドレスに対応していることを確認してください。

注記

Bare Metal Provisinoning ノードが認識している唯一のネットワークはベアメタルプロビジョニングネットワークです。エンドポイントの 1 つがこのネットワーク上にない場合には、そのエンドポイントはブートプロセスの一環として Bare Metal Provisioning ノードに到達することはできません。

たとえば、ファイルのカーネルの行は以下のようになります。

```
kernel http://192.168.200.2:8088/5a6cdbe3-2c90-4a90-b3c6-
85b449b30512/deploy_kernel selinux=0 disk=cciss/c0d0,sda,hda,vda
iscsi_target_iqn=iqn.2008-10.org.openstack:5a6cdbe3-2c90-4a90-b3c6-
85b449b30512 deployment_id=5a6cdbe3-2c90-4a90-b3c6-85b449b30512
deployment_key=VWDYDVVEFCQJNOSTO9R67HKUXUGP77CK
ironic_api_url=http://192.168.200.2:6385 troubleshoot=0 text nofb
nomodeset vga=normal boot_option=netboot ip=${ip}:${next-
server}:${gateway}:${netmask} BOOTIF=${mac}  ipa-api-
url=http://192.168.200.2:6385 ipa-driver-name=pxe_ssh boot_mode=bios
initrd=deploy_ramdisk coreos.configdrive=0 || goto deploy
```

上記の例の **kernel** 行の値	対応する情報
http://192.168.200.2:8088	**/etc/ironic/ironic.conf** ファイルのパラメーター **http_url**。この IP アドレスはベアメタル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。
5a6cdbe3-2c90-4a90-b3c6-85b449b30512	**ironic node-list** 内のベアメタルノードの UUID
deploy_kernel	これは、**/httpboot/<NODE_UUID>/deploy_kernel** としてコピーされた Image サービス内の deply kernel イメージです。
http://192.168.200.2:6385	**/etc/ironic/ironic.conf** ファイル内のパラメーター **api_url**。この IP アドレスはベアメタル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。
pxe_ssh	このノードの Bare Metal Provisioning サービスが使用している IPMI ドライバー
deploy_ramdisk	これは、**/httpboot/<NODE_UUID>/deploy_ramdisk** としてコピーされた Image サービス内の deploy ramdisk イメージです。

これらの値のいずれかが **/httpboot/pxelinux.cfg/MAC_ADDRESS** と **ironic.conf** ファイル間で一致していない場合には、**ironic.conf** ファイルを更新して Bare Metal Provisioning サービスを再起動してから Bare Metal Provisioning ノードを再デプロイする必要があります。

4.3. BARE METAL PROVISIONING サービスが正しいホスト名を取得しない場合のトラブルシューティング

Bare Metal Provisioning システムが正しいホスト名を取得しない場合は、**cloud-init** でエラーが発生

していることを意味します。この問題を修正するには、Bare Metal Provisioning のサブネットを OpenStack Networking サービス内のルーターに接続します。meta-data エージェントへの要求はこれで正しくルーティングされるようになるはずです。

4.4. BARE METAL PROVISIONING コマンドの実行時に OPENSTACK IDENTITY サービスの認証情報が無効な場合のトラブルシューティング

Identity サービスへの認証で問題がある場合には、**ironic.conf** ファイルの **identity_uri** パラメーターをチェックして、 **keystone** AdminURL から **/v2.0** が削除されていることを確認してください。たとえば、**identity_uri** は **http://IP:PORT** に設定する必要があります。

4.5. ハードウェア登録のトラブルシューティング

ハードウェア登録での問題は、ノードの登録情報が誤っていることが原因となっている場合があります。プロパティー名と値が正しく入力されていることを確認してください。プロパティー名が誤っていたり、タイプエラーがある場合でも、ノードの情報には正常に追加されますが、無視されます。

ノードの情報を更新します。以下の例では、登録するノードのメモリー使用量を 2 GB に更新します。

```
# ironic node-update NODE_UUID replace properties/memory_mb=2048
```

4.6. NO VALID HOST エラーのトラブルシューティング

Compute のスケジューラーがインスタンスを起動するのに適切な Bare Metal Provisioning ノードを見つけられない場合には、**NoValidHost** エラーが **/var/log/nova/nova-conductor.log** に表示されるか、起動に失敗した直後に Dashboard に表示されます。これは、通常 Compute が想定するリソースと、Bare Metal Provisioning ノードが提供するリソースが一致しないことが原因です。

1. 利用可能なハイパーバイザーのリソースを確認します。

   ```
   # nova hypervisor-stats
   ```

   このコマンドで返されるリソースは、Bare Metal Provisioning が提供するリソースと一致する必要があります。

2. Compute がBare Metal Provisioning ノードをハイパーバイザーとして認識していることを確認します。

   ```
   # nova hypervisor-list
   ```

   ノードは UUID で識別され、一覧に表示されるはずです。

3. Bare Metal Provisioning ノードの詳細を確認します。

   ```
   # ironic node-list
   # ironic node-show NODE_UUID
   ```

   ノードの詳細が、Compute によって返された情報と一致することを確認します。

4. 選択したフレーバーが Bare Metal Provisioning ノードで利用可能なリソースを超えていないことを確認します。

   ```
   nova flavor-show FLAVOR_NAME
   ```

5. **ironic node-list** の出力をチェックして、Bare Metal Provisioning ノードがメンテナンスモードに入っていないことを確認します。必要な場合はメンテナンスモードを解除します。

```
# ironic node-set-maintenance NODE_UUID off
```

6. **ironic node-list** の出力をチェックして、Bare Metal Provisioning ノードが **available** の状態であることを確認します。必要な場合には **available** に切り替えます。

```
# ironic node-set-provision-state NODE_UUID provide
```

4.7. ハードウェア検査のトラブルシューティング

ハードウェア検査は、**available** のプロビジョニング状態の Bare Metal Provisioning ノードで失敗する場合があります。

1. 全ノードのプロビジョニング状態を確認します。

```
# ironic node-list
```

2. 検査を開始する前に、ノードを **available** から **manageable** に切り替えます。

```
# ironic node-set-provision-state NODE_UUID manage
```

付録A BARE METAL PROVISIONING ドライバー

Bare Metal Provisioning は、多数のドライバーを使用するように設定することができます。各ドライバーは、プロビジョニングメソッドと電源管理タイプで構成されます。一部のドライバーには追加の設定が必要です。本項で説明する各ドライバーは、プロビジョニングに PXE を使用します。ドライバーは電源管理タイプ別にリストされます。エージェントドライバーは、ディスク全体のイメージとパーティションイメージのデプロイメントをサポートしています。ドライバー (1 つまたは複数) を Bare Metal Provisioning 用に有効化する方法については、「Bare Metal Provisioning ドライバーの設定」を参照してください。

A.1. INTELLIGENT PLATFORM MANAGEMENT INTERFACE (IPMI)

IPMI は、電源管理やサーバーのモニタリングを含む帯域外 (OOB) リモート管理機能を提供するインターフェースです。この電源管理タイプを使用するには、全 Bare Metal Provisioning ノードで IPMI が共有ベアメタルプロビジョニングネットワークに接続されている必要があります。**`pxe_ipmitool`** ドライバーを有効化し、ノードの **`driver_info`** に以下の情報を設定します。

- **`ipmi_address`**: IPMI NIC の IP アドレス
- **`ipmi_username`**: IPMI のユーザー名
- **`ipmi_password`**: IPMI のパスワード

A.2. DELL REMOTE ACCESS CONTROLLER (DRAC)

DRAC は、電源管理やサーバーのモニタリングを含む帯域外 (OOB) リモート管理機能を提供するインターフェースです。この電源管理タイプを使用するには、全 Bare Metal Provisioning ノードで DRAC が共有ベアメタルプロビジョニングネットワークに接続されている必要があります。**`pxe_drac`** ドライバーを有効化し、ノードの **`driver_info`** に以下の情報を設定します。

- **`drac_address`**: DRAC NIC の IP アドレス
- **`drac_username`**: DRAC のユーザー名
- **`drac_password`**: DRAC のパスワード

A.3. INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC)

富士通の iRMC は、電源管理やサーバーのモニタリングを含む帯域外 (OOB) リモート管理機能を提供するインターフェースです。Bare Metal Provisioning ノードでこの電源管理タイプを使用するには、このノードに、共有ベアメタルプロビジョニングネットワークに接続された iRMC インターフェースが 1 つ必要です。**`pxe_irmc`** ドライバーを有効化し、ノードの **`driver_info`** に以下の情報を設定します。

- **`irmc_address`**: iRMC インターフェースの NIC の IP アドレス
- **`irmc_username`**: iRMC のユーザー名
- **`irmc_password`**: iRMC のパスワード

IPMI を使用してブートモードを設定するか、SCCI を使用してセンサーデータを取得するには、追加で以下のステップを完了する必要があります。

1. **ironic.conf** でセンサーメソッドを有効にします。

```
# openstack-config --set /etc/ironic/ironic.conf \
   irmc sensor_method METHOD
```

METHOD は **scci** または **ipmitool** に置き換えます。

2. SCCI を有効にした場合は、**python-scciclient** パッケージをインストールします。

```
# yum install python-scciclient
```

3. Bare Metal Provisioning コンダクターサービスを再起動します。

```
# systemctl restart openstack-ironic-conductor.service
```

注記

iRMC ドライバーを使用するには、 iRMC S4 以降が必要です。

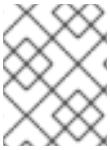
A.4. INTEGRATED LIGHTS-OUT (ILO)

iLO は、電源管理やサーバーのモニタリングを含む帯域外 (OOB) リモート管理機能を提供するインターフェースです。この電源管理タイプを使用するには、全 Bare Metal Provisioning ノードで、インターフェースが共有ベアメタルプロビジョニングネットワークに接続された iLO インターフェースが 1 つ必要です。**pxe_ilo** ドライバーを有効化し、ノードの **driver_info** に以下の情報を設定します。

- **ilo_address**: iLO インターフェースの NIC の IP アドレス
- **ilo_username**: iLO のユーザー名
- **ilo_password**: iLO のパスワード

python-proliantutils パッケージもインストールして、Bare Metal Provisioning コンダクターサービスを再起動する必要があります。

```
# yum install python-proliantutils
# systemctl restart openstack-ironic-conductor.service
```



注記

検査を正常に実行するには、HP ノードが 2015 のファームウェアバージョンである必要があります。

A.5. ACTIVE MANAGEMENT TECHNOLOGY (AMT)

Intel の AMT は、サーバーにおける IPMI の使用方法と同様に、デスクトップの電源制御を含むデスクトップの監視および管理に使用される帯域外リモート管理テクノロジーです

AMT ドライバーは WS-MAN プロトコルを使用して AMT クライアントと対話します。

AMT は 2 つのドライバーで構成されます。

- **pxe_amt** は AMT を使用して電源を管理し、コンダクターから iSCSI を介して user イメージをデプロイします。

- **agent_amt** は AMT を使用して電源を管理し、HTTP を介してユーザーイメージをノードにデプロイします。

警告

非推奨に関する通知: Red Hat OpenStack Platform 11 より、AMT ドライバーは非推奨になり、今後のリリースではサポートされなくなります。

作業を開始する前に

デスクトップ環境と AMT クライアントを設定します。詳しくは、「Manually configuring the AMT Client」を参照してください

環境の設定

1. Bare Metal Provisioning サービスのノードに **openwsman-python** をインストールします。

```
# yum install openwsman-python
```

2. **ironic.conf** で AMT ドライバーを有効化します。

```
# openstack-config --set /etc/ironic/ironic.conf \
  enabled_drivers DRIVER
```

DRIVER は **pxe_amt** または **agent_amt** です。

3. Bare Metal Provisioning コンダクターサービスを再起動します。

```
# systemctl restart openstack-ironic-conductor.service
```

AMT ドライバーを有効化した後には、ノードの **driver_info** で以下の情報を設定する必要があります。

- **amt_address**: AMT NIC の IP アドレス
- **amt_username**: AMT のユーザー名
- **amt_password**: AMT のパスワード

注記

AMT を使用する Bare metal ノードは、ローカルブートオプションを有効にしてデプロイする必要があります。AMT は現在、永続的な起動デバイスはサポートしていません。ローカルブートオプションを有効にせずにデプロイされたノードは、Bare Metal Provisioning サービスの制御外で再起動された場合に起動に失敗する可能性があります。たとえば、ローカルユーザーによって再起動されたノードは、カーネルの PXE またはネットワークブートは試みません。ローカルブートオプションを有効にしてデプロイされた AMT ノードは、この問題を解決します。

A.6. SSH と VIRSH

Bare Metal Provisioning は、libvirt を実行するホストにアクセスして、仮想マシンをノードとして使用することができます。virsh はノードの電源管理機能を制御します。

重要

この SSH ドライバーは、検証および評価のみを目的としており、Red Hat OpenStack Platform のエンタープライズ環境には推奨されません。

この電源管理タイプを使用するには、Bare Metal Provisioning は仮想ノードを設定するホスト上の libvirt 環境に完全にアクセス可能なアカウントに SSH でアクセスできる必要があります。**pxe_ssh** ドライバーを有効にして、ノードの **driver_info** で以下の情報を設定します。

- **ssh_virt_type**: このオプションは **virsh** に設定します。
- **ssh_address**: virsh ホストの IP アドレス
- **ssh_username**: SSH ユーザー名
- **ssh_key_contents**: Bare Metal Provisioning コンダクターノード上の SSH 秘密鍵の内容。対応する公開鍵が virsh ホストにコピーされている必要があります。