Red Hat OpenStack Platform 17.0

Bare Metal Provisioning

Bare Metal Provisioning サービス (ironic) のインストールと設定

Last Updated: 2023-10-12

Red Hat OpenStack Platform 17.0 Bare Metal Provisioning

Bare Metal Provisioning サービス (ironic) のインストールと設定

OpenStack Team
rhos-docs@redhat.com

法律上の通知

概要

Red Hat OpenStack Platform 環境のオーバークラウドに Bare Metal Provisioning サービスをインストールして設定し、クラウドユーザーの物理マシンをプロビジョニングおよび管理します。

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。

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに対するご意見をお聞かせください。ドキュメントの改善点があればお知らせください。

ドキュメントへのダイレクトフィードバック (DDF) 機能の使用 (英語版のみ)

特定の文章、段落、またはコードブロックに対して直接コメントを送付するには、DDF の **Add Feedback** 機能を使用してください。なお、この機能は英語版のドキュメントでのみご利用いただけます。

1. **Multi-page HTML** 形式でドキュメントを表示します。
2. ドキュメントの右上隅に **Feedback** ボタンが表示されていることを確認してください。
3. コメントするテキスト部分をハイライト表示します。
4. **Add Feedback** をクリックします。
5. **Add Feedback** フィールドにコメントを入力します。
6. オプション:ドキュメントチームが問題の詳細を確認する際に使用できるメールアドレスを記入してください。
7. **Submit** をクリックします。

第1章 BARE METAL PROVISIONING サービス (IRONIC) の機能

Bare Metal Provisioning サービス (ironic) コンポーネントを使用して、クラウドユーザーのベアメタルインスタンスとして物理マシンをプロビジョニングおよび管理します。ベアメタルインスタンスをプロビジョニングおよび管理するために、Bare Metal Provisioning サービスは、オーバークラウド内の次の Red Hat OpenStack Platform (RHOSP) サービスと対話します。

- Compute サービス (nova) は、仮想マシンインスタンス管理用のスケジューリング、テナントクォータ、およびユーザー向け API を提供します。Bare Metal Provisioning サービスは、ハードウェア管理用の管理 API を提供します。
- Identity サービス (keystone) は、要求認証を提供し、Bare Metal Provisioning サービスが他の RHOSP サービスを見つけることを支援します。
- Image サービス (glance) は、ディスクとインスタンスのイメージおよびイメージメタデータを管理します。
- Networking サービス (neutron) は、DHCP とネットワーク設定を提供し、インスタンスが起動時に接続する仮想ネットワークまたは物理ネットワークをプロビジョニングします。
- Object Storage サービス (swift) は、一部のドライバーの一時的なイメージ URL を公開します。

Bare Metal Provisioning サービスコンポーネント

Bare Metal Provisioning サービスは、**ironic-*** という名前のサービスで設定されています。以下のサービスは、コア Bare Metal Provisioning サービスになります。

Bare Metal Provisioning API (ironic-api)

このサービスは、ユーザーに外部 REST API を提供します。API は、リモートプロシージャコール (RPC) を介して Bare Metal Provisioning コンダクターにアプリケーション要求を送信します。

Bare Metal Provisioning コンダクター (ironic-conductor)

このサービスは、ドライバーを使用して以下のベアメタルノード管理タスクを実行します。

- ベアメタルノードを追加、編集、および削除します。
- IPMI、Redfish、またはその他のベンダー固有のプロトコルを使用して、ベアメタルノードの電源をオン/オフにします。
- ベアメタルノードをプロビジョニング、デプロイ、およびクリーニングします。

Bare Metal Provisioning インスペクター (ironic-inspector)

このサービスは、ベアメタルインスタンスのスケジューリングに必要なベアメタルノードのハードウェアプロパティーを検出し、検出されたイーサネット MAC 用の Bare Metal Provisioning サービスポートを作成します。

Bare Metal Provisioning データベース

このデータベースは、ハードウェア情報と状態を追跡します。

メッセージキュー

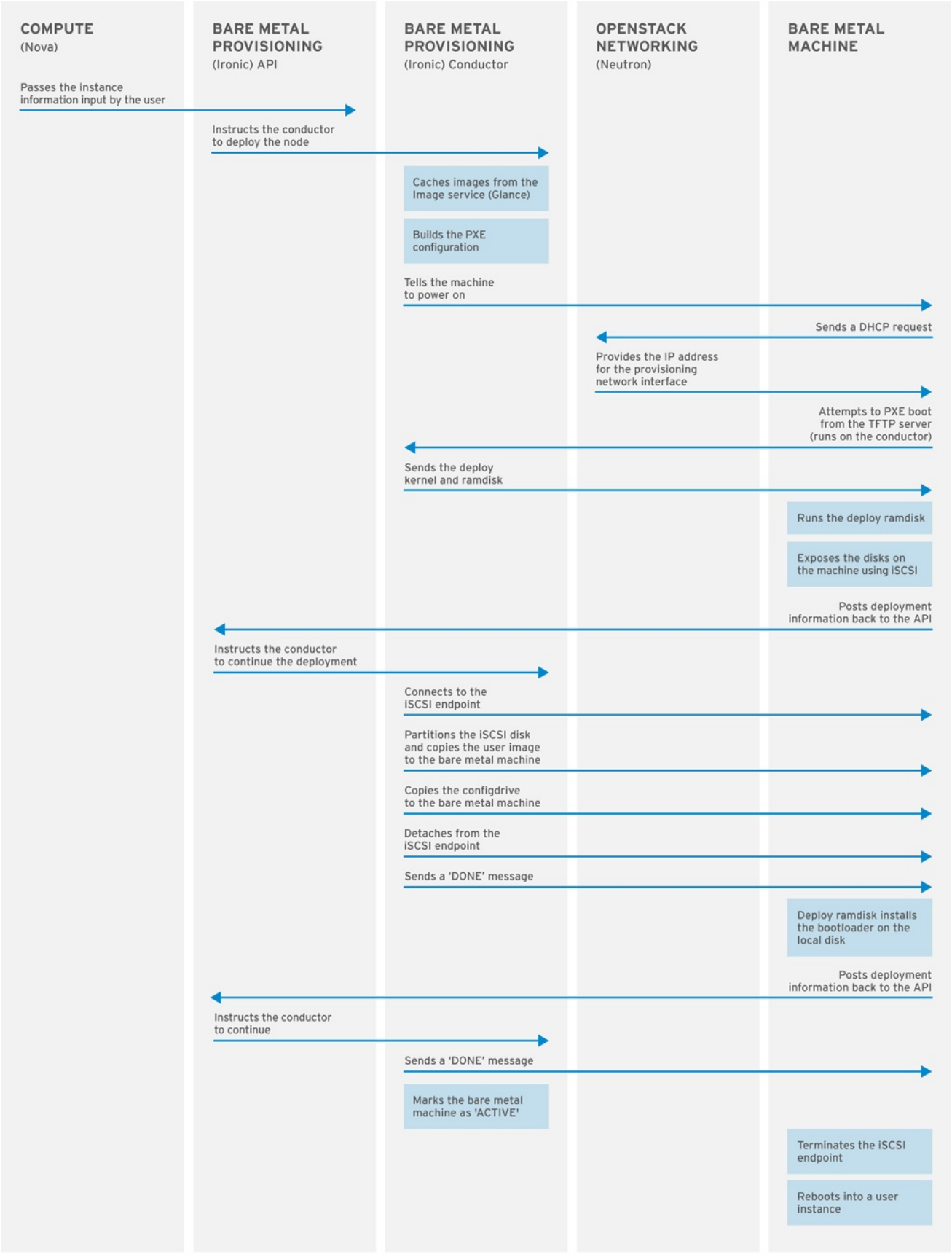
すべてのサービスは、このメッセージングサービスを使用して、**ironic-api** と **ironic-conductor** 間の RPC の実装を含め、相互に通信します。

Bare Metal Provisioning エージェント (ironic-python-agent)

このサービスは一時的な RAM ディスクで実行され、**ironic-conductor** サービスおよび **ironic-inspector** サービスにリモートアクセス、帯域内ハードウェア制御、およびハードウェアイントロスペクションを提供します。

ベアメタルインスタンスのプロビジョニング

Bare Metal Provisioning サービスは iPXE を使用して、物理マシンをベアメタルインスタンスとしてプロビジョニングします。以下の図は、クラウドユーザーがデフォルトのドライバーを使用して新しいベアメタルインスタンスを起動したときに、プロビジョニングプロセス中に RHOSP サービスがどのように相互作用するかを示しています。



OPENSTACK_377593_1215

第2章 BARE METAL PROVISIONING の要件

クラウドユーザーがベアメタルインスタンスを起動できるオーバークラウドを提供するには、Red Hat OpenStack Platform (RHOSP) 環境に必要なハードウェアとネットワーク設定が必要です。

2.1. ハードウェア要件

プロビジョニングのためにクラウドユーザーが利用できるベアメタルマシンのハードウェア要件は、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 インストールのハードウェア要件の詳細は、Product Documentation for Red Hat Enterprise Linux を参照してください。

プロビジョニングのためにクラウドユーザーが利用できるベアメタルマシンには、以下の機能が必要です。

- ベアメタルネットワークに接続するための NIC 1 つ。
- **ironic-conductor** サービスから到達可能なネットワークに接続された電源管理インターフェイス (例: Redfish または IPMI)。コンポーザブルロールを使用して **ironic-conductor** を別の場所で実行する場合以外は、デフォルトでは **ironic-conductor** は全コントローラーノード上で実行されます。
- ベアメタルネットワーク上での PXE ブート。デプロイメント内のその他すべての NIC については PXE ブートを無効にしてください。

2.2. ネットワーク要件

ベアメタルネットワークは、Bare Metal Provisioning サービスが以下の操作に使用するためにプライベートネットワークである必要があります。

- オーバークラウド上のベアメタルマシンのプロビジョニングと管理。
- ノードがプロビジョニングされていない場合のベアメタルノードのクリーニング。
- ベアメタルマシンへのテナントアクセス。

ベアメタルネットワークは、ベアメタルシステムを検出するための DHCP および PXE ブートの機能を提供します。このネットワークは、Bare Metal Provisioning サービスが PXE ブートと DHCP 要求に対応できるように、トランキングされたインターフェイスでネイティブの VLAN を使用する必要があります。

ベアメタルマシンは Red Hat OpenStack Platform (RHOSP) 環境のコントロールプレーンネットワークに直接アクセスできるため、オーバークラウドの Bare Metal Provisioning サービスは、信頼できるテナント環境向けに設計されています。したがって、デフォルトのベアメタルネットワークは、**ironic-conductor** サービスにフラットネットワークを使用します。

デフォルトのフラットプロビジョニングネットワークは、テナントがコントロールプレーンネットワークに干渉する可能性があるため、お客様の環境にセキュリティー上の懸念をもたらす可能性があります。このリスクを避けるために、コントロールプレーンにアクセスできない Bare Metal Provisioning サービス用に、カスタムの設定可能なベアメタルプロビジョニングネットワークを設定できます。

ベアメタルネットワークは、プロビジョニングのためにタグ付けされていない必要があり、Bare Metal Provisioning API にもアクセスできる必要があります。コントロールプレーンネットワーク (別名: director のプロビジョニングネットワーク) は常にタグなしです。その他のネットワークはタグ付けすることができます。

Bare Metal Provisioning サービスをホストするコントローラーノードは、ベアメタルネットワークにアクセス可能である必要があります。

ベアメタルマシンが PXE ブートするように設定されている NIC は、ベアメタルネットワークにアクセスできる必要があります。

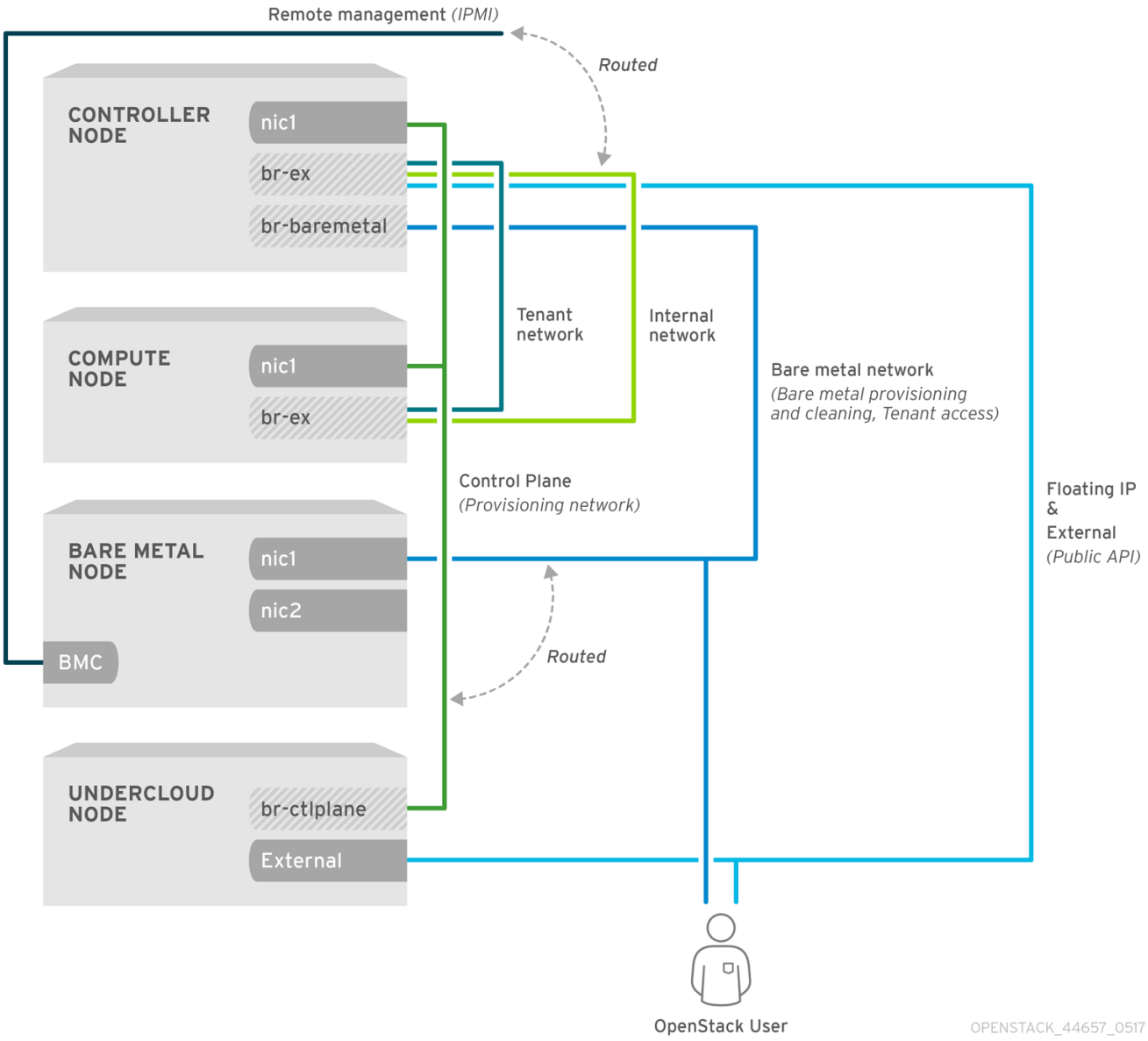
ベアメタルネットワークは、OpenStack のオペレーターが作成します。クラウドユーザーは、パブリック OpenStackAPI とベアメタルネットワークに直接アクセスできます。デフォルトのフラットベアメタルネットワークでは、クラウドユーザーはコントロールプレーンに間接的にアクセスすることもできます。

Bare Metal Provisioning サービスは、ノードのクリーニングにベアメタルネットワークを使用します。

2.2.1. デフォルトのベアメタルネットワーク

デフォルトの Bare Metal Provisioning サービスのデプロイメントアーキテクチャーでは、ベアメタルネットワークはコントロールプレーンネットワークから分離されています。ベアメタルネットワークは、テナントネットワークとしても機能するフラットネットワークです。このネットワークは、director プロビジョニングネットワークと呼ばれる、コントロールプレーン上の Bare Metal Provisioning サービスにルーティングする必要があります。分離したベアメタルネットワークを定義すると、ベアメタルノードは PXE ブートすることができません。

デフォルトのベアメタルネットワークアーキテクチャー図



2.2.2. カスタムコンポーザブルベアメタルネットワーク

Bare Metal Provisioning サービスのデプロイメントアーキテクチャーで、カスタムコンポーザブルベアメタルネットワークを使用する場合、ベアメタルネットワークは、コントロールプレーンにアクセスできないカスタムコンポーザブルネットワークです。コントロールプレーンへのアクセスを制限する場合は、カスタムの設定可能なベアメタルネットワークを使用します。

第3章 BARE METAL PROVISIONING サービスを有効にしたオーバークラウドのデプロイ

Bare Metal Provisioning サービス (ironic) を使用してオーバークラウドをデプロイするには、ベアメタルネットワークを作成して設定し、ベアメタルプロビジョニングを有効に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。

1. ベアメタルネットワークを作成します。コントローラーノードでプロビジョニングネットワークインターフェイスを再利用してフラットネットワークを作成するか、カスタムネットワークを作成できます。
   - デフォルトのフラットネットワークの設定
   - カスタムの IPv4 プロビジョニングネットワークの設定
   - カスタムの IPv6 プロビジョニングネットワークの設定
2. Bare Metal Provisioning を有効化するようにオーバークラウドを設定します。
   - Bare Metal Provisioning を有効化するようにオーバークラウドを設定する

注記

Open Virtual Network (OVN) を使用する場合、Bare Metal Provisioning サービスは、**ironic-overcloud.yaml** ファイルで定義されている DHCP エージェント **neutron-dhcp-agent** でのみサポートされます。OVN の組み込まれている DHCP サーバーは、ベアメタルノードをプロビジョニングしたり、プロビジョニングネットワークに DHCP を提供したりすることはできません。iPXE チェーンロードを有効にするには、dnsmasq で **--dhcp-match** タグを設定する必要があります。これは、OVN DHCP サーバーではサポートされていません。

前提条件

- お使いの環境が最小要件を満たしていること。詳細は、ベアメタルプロビジョニングの要件 を参照してください。

3.1. デフォルトのフラットネットワークの設定

デフォルトのフラットベアメタルネットワークを使用するには、コントローラーノードのプロビジョニングネットワークインターフェイスを再利用して、Bare Metal Provisioning サービス (ironic) のブリッジを作成します。

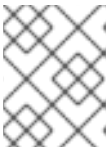
手順

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. **/home/stack/templates/nic-configs/controller.yaml** ファイルを変更して、コントローラーノード **eth1** のプロビジョニングネットワークインターフェイスを再利用し、ベアメタルネットワークのブリッジを作成します。

```
network_config:
- type: ovs_bridge
  name: br-baremetal
  use_dhcp: false
  members:
    - type: interface
      name: eth1
  addresses:
    - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
```



注記

プロビジョニングネットワークを再利用してベアメタルネットワークを作成するときに、VLAN タグを付けることはできません。

4. **network-environment.yaml** ファイルの **NeutronBridgeMappings** パラメーターに **br-baremetal** を追加します。

```
parameter_defaults:
  NeutronBridgeMappings: datacentre:br-ex,baremetal:br-baremetal
```

5. **network-environment.yaml** ファイルの **NeutronFlatNetworks** パラメーターで指定されたネットワークのリストに **baremetal** を追加します。

```
parameter_defaults:
  NeutronBridgeMappings: datacentre:br-ex,baremetal:br-baremetal
  NeutronFlatNetworks: datacentre,baremetal
```

次のステップ

- Bare Metal Provisioning を有効化するようにオーバークラウドを設定する

3.2. カスタムの IPV4 プロビジョニングネットワークの設定

カスタムの IPv4 プロビジョニングネットワークを作成し、IPv4 を介してオーバークラウドをプロビジョニングおよびデプロイします。

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. **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. **network_data.yaml** ファイルを環境ファイルディレクトリーにコピーします。

```
(undercloud) [stack@host01 ~]$ cp /usr/share/openstack-tripleo-heat-templates/network_data.yaml /home/stack/templates/network_data.yaml
```

4. オーバークラウドプロビジョニング用の新しいネットワークを **network_data.yaml** ファイルに追加します。

```
# custom network for overcloud provisioning
- name: OcProvisioning
  name_lower: oc_provisioning
  vip: true
  vlan: 205
  ip_subnet: '<ipv4_subnet_address>/<ipv4_mask>'
  allocation_pools: [{'start': '<ipv4_start_address>', 'end': '<ipv4_end_address>'}]
```

- **<ipv4_subnet_address>** を IPv4 サブネットの IPv4 アドレスに置き換えます。
- **<ipv4_mask>** を IPv4 サブネットの IPv4 ネットワークマスクに置き換えます。
- **<ipv4_start_address>** と **<ipv4_end_address>** は、アドレス割り当てに使用する IPv4 範囲に置き換えます。

5. 新しい IPv4 プロビジョニングネットワークを使用するには、**ServiceNetMap** 設定で **IronicApiNetwork** と **IronicNetwork** を設定します。

```
ServiceNetMap:
  IronicApiNetwork: oc_provisioning
  IronicNetwork: oc_provisioning
```

6. 新しいネットワークをインターフェイスとして、ローカルコントローラー NIC 設定ファイルに追加します。

```
network_config:
- type: vlan
  vlan_id:
    get_param: OcProvisioningNetworkVlanID
  addresses:
  - ip_netmask:
      get_param: OcProvisioningIpSubnet
```

7. **roles_data.yaml** ファイルを環境ファイルディレクトリーにコピーします。

```
(undercloud) [stack@host01 ~]$ cp /usr/share/openstack-tripleo-heat-
templates/roles_data.yaml /home/stack/templates/roles_data.yaml
```

8. コントローラーの新しいネットワークを **roles_data.yaml** ファイルに追加します。

```
networks:
  ...
  OcProvisioning:
    subnet: oc_provisioning_subnet
```

9. **IronicInspector** サービスがまだ存在しない場合は、**roles_data.yaml** ファイル内の **Ironic** ロールに含めます。

```
ServicesDefault:
  OS::TripleO::Services::IronicInspector
```

次のステップ

- Bare Metal Provisioning を有効化するようにオーバークラウドを設定する

3.3. カスタムの IPV6 プロビジョニングネットワークの設定

カスタムの IPv6 プロビジョニングネットワークを作成し、IPv6 を使用してオーバークラウドのプロビジョニングとデプロイを行います。

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. **network_data.yaml** ファイルを環境ファイルディレクトリーにコピーします。

```
(undercloud) [stack@host01 ~]$ cp /usr/share/openstack-tripleo-heat-
templates/network_data.yaml /home/stack/templates/network_data.yaml
```

4. オーバークラウドプロビジョニング用の新しい IPv6 ネットワークを **network_data.yaml** ファイルに追加します。

```
# custom network for IPv6 overcloud provisioning
- name: OcProvisioningIPv6
  vip: true
  name_lower: oc_provisioning_ipv6
  vlan: 10
  ipv6: true
  ipv6_subnet: '<ipv6_subnet_address>/<ipv6_prefix>'
  ipv6_allocation_pools: [{'start': '<ipv6_start_address>', 'end': '<ipv6_end_address>'}]
  gateway_ipv6: '<ipv6_gw_address>'
```

   - **<ipv6_subnet_address>** は、IPv6 サブネットの IPv6 アドレスに置き換えます。
   - **<ipv6_prefix>** は、IPv6 サブネットの IPv6 ネットワーク接頭辞に置き換えます。
   - **<ipv6_start_address>** と **<ipv6_end_address>** は、アドレス割り当てに使用する IPv6 範囲に置き換えます。
   - **<ipv6_gw_address>** は、ゲートウェイの IPv6 アドレスに置き換えます。

5. 環境ファイルディレクトリーに新しいファイル **network_environment_overrides.yaml** を作成します。

```
$ touch /home/stack/templates/network_environment_overrides.yaml
```

6. 新しい IPv6 プロビジョニングネットワークを使用するには、**network_environment_overrides.yaml** ファイルで **IronicApiNetwork** と **IronicNetwork** を設定します。

```
ServiceNetMap:
  IronicApiNetwork: oc_provisioning_ipv6
  IronicNetwork: oc_provisioning_ipv6
```

7. **IronicIpVersion** パラメーターを **6** に設定します。

```
parameter_defaults:
  IronicIpVersion: 6
```

8. **RabbitIPv6**、**MysqlIPv6**、および **RedisIPv6** の各パラメーターを有効化します。

```
parameter_defaults:
  RabbitIPv6: True
  MysqlIPv6: True
  RedisIPv6: True
```

9. 新しいネットワークをインターフェイスとして、ローカルコントローラー NIC 設定ファイルに追加します。

```
network_config:
- type: vlan
  vlan_id:
    get_param: OcProvisioningIPv6NetworkVlanID
  addresses:
  - ip_netmask:
      get_param: OcProvisioningIPv6IpSubnet
```

10. **roles_data.yaml** ファイルを環境ファイルディレクトリーにコピーします。

```
(undercloud) [stack@host01 ~]$ cp /usr/share/openstack-tripleo-heat-
templates/roles_data.yaml /home/stack/templates/roles_data.yaml
```

11. コントローラーロールの新しいネットワークを **roles_data.yaml** ファイルに追加します。

```
networks:
  ...
  - OcProvisioningIPv6
```

12. **IronicInspector** サービスがまだ存在しない場合は、**roles_data.yaml** ファイル内の **Ironic** ロールに含めます。

```
ServicesDefault:
  OS::TripleO::Services::IronicInspector
```

次のステップ

- Bare Metal Provisioning を有効化するようにオーバークラウドを設定する

3.4. BARE METAL PROVISIONING を有効化するようにオーバークラウドを設定する

/usr/share/openstack-tripleo-heat-templates/environments/services ディレクトリーにあるデフォルトのテンプレートの 1 つを使用して、Bare Metal Provisioning サービス (ironic) を有効にして、オーバークラウドをデプロイします。

- OVS を使用するデプロイメントの場合: **ironic.yaml**

- OVN を使用するデプロイの場合: **ironic-overcloud.yaml**

デプロイメントの必要に応じて、ローカル環境ファイルを作成して、デフォルトの設定をオーバーライドできます。

手順

1. デプロイメント用に Bare Metal Provisioning サービスを設定する環境ファイル (例: **ironic-overrides.yaml**) をローカルディレクトリーに作成します。
2. オプション:プロビジョニング前およびプロビジョニング中にベアメタルマシンで実行されるクリーニングのタイプを設定します。

```
parameter_defaults:
  IronicCleaningDiskErase: <cleaning_type>
```

<cleaning_type> を次のいずれかの値に置き換えます。

- **full**:(デフォルト) 完全なクリーニングを実行します。
- **metadata**:パーティションテーブルのみをクリーニングします。このタイプの洗浄は、クリーニングプロセスを大幅にスピードアップします。ただし、マルチテナント環境ではデプロイメントのセキュリティーレベルが低くなるため、信頼済みのテナント環境でのみこのオプションを使用します。

3. オプション:デフォルトのドライバーにドライバーを追加します。

```
parameter_defaults:
  IronicEnabledHardwareTypes: ipmi,idrac,ilo,[additional_driver_1],...,[additional_driver_n]
```

[additional_driver_1] およびオプションで **[additional_driver_n]** までのすべてのドライバーは、有効にする追加のドライバーに置き換えます。

4. ベアメタルイントロスペクションを有効にするには、次の設定をローカルの Bare Metal Provisioning サービス環境ファイル **ironic-overrides.yaml** に追加します。

```
parameter_defaults:
  IronicInspectorSubnets:
    - ip_range: <ip_range>
  IPAImageURLs: '["http://<ip_address>:<port>/agent.kernel", "http://<ip_address>:
<port>/agent.ramdisk"]'
  IronicInspectorInterface: '<baremetal_interface>'
```

- **<ip_range>** を環境の IP 範囲に置き換えます (例: **192.168.0.100,192.168.0.120**)。
- **<ip_address>:<port>** は、IPA カーネルと RAM ディスクをホストする Web サーバーの IP アドレスとポートに置き換えます。アンダークラウドで使用するのと同じイメージを使用するには、IP アドレスをアンダークラウドの IP アドレスに設定し、ポートを **8088** に設定します。このパラメーターを省略する場合は、各コントローラーノードに代替のパラメーターを含める必要があります。
- **<baremetal_interface>** をベアメタルネットワークインターフェイス (**br-baremetal** など) に置き換えます。

5. 新しいロールとカスタム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、オーバークラウドをデプロイします。

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/node-info.yaml \
  -r /home/stack/templates/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/services/<default_ironic_template> \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/ironic-inspector.yaml
\
  -e /home/stack/templates/network_environment_overrides.yaml \
  -n /home/stack/templates/network_data.yaml \
  -e /home/stack/templates/ironic-overrides.yaml
```

- デプロイメントのネットワークサービスメカニズムドライバーに応じて、**<default_ironic_template>** を **ironic.yaml** または **ironic-overcloud.yaml** のいずれかに置き換えます。

注記

環境ファイルを **openstack overcloud deploy** コマンドに渡す順序は重要です。これは、後のファイルの設定が優先されるためです。したがって、オーバークラウドで Bare Metal Provisioning を有効にして設定する環境ファイルは、ネットワーク設定ファイルの後にコマンドに渡す必要があります。

3.5. BARE METAL PROVISIONING サービスのテスト

OpenStack Integration Test Suite を使用して、Red Hat OpenStack デプロイメントを検証することができます。詳細は、OpenStack Integration Test Suite Guide を参照してください。

Bare Metal Provisioning サービスの追加検証方法:

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 ~/overcloudrc
```

2. **nova-compute** サービスがコントローラーノードで実行中であることを確認します。

```
$ openstack compute service list -c Binary -c Host -c Status
```

3. デフォルトの ironic ドライバーを変更した場合には、必要なドライバーを必ず有効にしてください。

```
$ openstack baremetal driver list
```

4. ironic のエンドポイントがリストされていることを確認します。

```
$ openstack catalog list
```

3.6. 関連情報

- **director のインストールと使用** ガイドの デプロイコマンドオプション

- オーバークラウドの IPv6 ネットワーク
- **Overcloud Parameters** ガイドの Bare Metal (ironic) Parameters

第4章 デプロイ後の BARE METAL PROVISIONING サービスの設定

Bare Metal Provisioning サービス (ironic) を使用して、オーバークラウドをデプロイしたら、ベアメタルワークロード用にオーバークラウドを準備する必要があります。ベアメタルワークロード用にオーバークラウドを準備し、クラウドユーザーがベアメタルインスタンスを作成できるようにするには、以下のタスクを完了します。

- Bare Metal Provisioning サービスと統合するように、Networking サービス (neutron) を設定します。
- ノードのクリーニングの設定
- ベアメタルフレーバーとリソースクラスの作成
- オプション:ベアメタルイメージの作成
- 物理マシンをベアメタルノードとして追加
- オプション:Redfish 仮想メディアブートの設定
- オプション:ホストアグリゲートを作成して、物理マシンと仮想マシンのプロビジョニングを分離

4.1. ベアメタルプロビジョニング用の NETWORKING サービスの設定

Bare Metal Provisioning サービス (ironic) と統合するように、Networking サービス (neutron) を設定できます。次のいずれかの方法を使用して、ベアメタルネットワークを設定できます。

- Bare Metal Provisioning のコンダクターサービスである **ironic-conductor** 用に 1 つのフラットベアメタルネットワークを作成します。このネットワークは、コントロールプレーンネットワーク上の Bare Metal Provisioning サービスにルーティングする必要があります。
- カスタムのコンポーザブルネットワークを作成して、オーバークラウドに Bare Metal Provisioning サービスを実装します。

4.1.1. フラットなネットワークで Bare Metal Provisioning サービスと統合するための Networking サービスの設定

Bare Metal Provisioning コンダクターサービスの 1 つのフラットベアメタルネットワークである **ironic-conductor** を作成して、Bare Metal Provisioning サービス (ironic) と統合するように、Networking サービス (neutron) を設定できます。このネットワークは、コントロールプレーンネットワーク上の Bare Metal Provisioning サービスにルーティングする必要があります。

手順

1. Networking サービス (neutron) をホスティングするノードに **root** ユーザーとしてログインします。
2. オーバークラウド認証情報ファイルを入手します。

```
# source ~/<credentials_file>
```

   - **<credentials_file>** を認証情報ファイルの名前 (**overcloudrc** など) に置き換えます。

3. ベアメタルインスタンスをプロビジョニングするフラットネットワークを作成します。

```
# openstack network create \
  --provider-network-type flat \
  --provider-physical-network <provider_physical_network> \
  --share <network_name>
```

- **<provider_physical_network>** を仮想ネットワークを実装する物理ネットワークの名前に置き換えます。これは、**network-environment.yaml** ファイルのパラメーター **NeutronBridgeMappings** で設定されます。
- **<network_name>** をこのネットワークの名前に置き換えます。

4. フラットネットワーク上にサブネットを作成します。

```
# openstack subnet create \
  --network <network_name> \
  --subnet-range <network_cidr> \
  --ip-version 4 \
  --gateway <gateway_ip> \
  --allocation-pool start=<start_ip>,end=<end_ip> \
  --dhcp <subnet_name>
```

- **<network_name>** を前の手順で作成したプロビジョニングネットワークの名前に置き換えます。
- **<network_cidr>** をサブネットが表す IP アドレスのブロックの Classless Inter-Domain Routing (CIDR) 表現に置き換えます。**<start_ip>** で始まり **<end_ip>** で終わる範囲で指定する IP アドレスのブロックは、**<network_cidr>** で指定された IP アドレスのブロック内にある必要があります。
- **<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ルーターインターフェイスの 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>** で指定された IP アドレスのブロック内にある必要がありますが、**<start_ip>** で始まり **<end_ip>** で終わる範囲で指定された IP アドレスのブロック外にある必要があります。
- **<start_ip>** をフローティング IP アドレスが割り当てられる新しいサブネット内の IP アドレス範囲の開始を示す IP アドレスに置き換えます。
- **<end_ip>** をフローティング IP アドレスが割り当てられる新しいサブネット内の IP アドレスの範囲の終了を示す IP アドレスに置き換えます。
- **<subnet_name>** をサブネットの名前に置き換えます。

5. ネットワーキングサービスがメタデータリクエストを確実に処理するように、ネットワークとサブネット用のルーターを作成します。

```
# openstack router create <router_name>
```

- **<router_name>** をルーターの名前に置き換えます。

6. **cloud-init** からのメタデータリクエストを処理し、ノードを設定できるように、サブネットを新しいルーターに接続します。

```
# openstack router add subnet <router_name> <subnet>
```

- **<router_name>** をルーターの名前に置き換えます。
- **<subnet>** を手順 4 で作成したベアメタルサブネットの ID または名前に置き換えます。

4.1.2. カスタムコンポーザブルネットワーク上の Bare Metal Provisioning サービスと統合するための Networking サービスの設定

オーバークラウドに Bare Metal Provisioning サービスを実装するためのカスタムコンポーザブルネットワークを作成して、Bare Metal Provisioning サービス (ironic) と統合するように、Networking サービス (neutron) を設定できます。

手順

1. アンダークラウドのホストにログインします。
2. オーバークラウド認証情報ファイルを入手します。

```
$ source ~/<credentials_file>
```

   - **<credentials_file>** を認証情報ファイルの名前 (**overcloudrc** など) に置き換えます。

3. Bare Metal Provisioning サービスをホスティングするプロバイダーネットワークの UUID を取得します。

```
(overcloud)$ openstack network show <network_name> -f value -c id
```

   - **<network_name>** をベアメタルインスタンスプロビジョニングネットワークに使用するプロバイダーネットワークの名前に置き換えます。

4. デプロイメント用の Bare Metal Provisioning サービスを設定するローカル環境ファイル (例: **ironic-overrides.yaml**) を開きます。
5. ベアメタルインスタンスのプロビジョニングネットワークとして使用するネットワークを設定します。

```
parameter_defaults:
  IronicProvisioningNetwork: <network_uuid>
```

   - **<network_uuid>** を手順 3 で取得したプロバイダーネットワークの UUID に置き換えます。

6. **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
```

7. ベアメタルインスタンスのプロビジョニングネットワーク設定を適用するには、ベアメタルプロビジョニング環境ファイルを他の環境ファイルとともにスタックに追加し、オーバークラウドをデプロイします。

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/node-info.yaml \
  -r /home/stack/templates/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/network-environment.yaml \
```

```
  -e /usr/share/openstack-tripleo-heat-
templates/environments/services/<default_ironic_template> \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/ironic-inspector.yaml
\
  -e /home/stack/templates/network_environment_overrides.yaml \
  -n /home/stack/templates/network_data.yaml \
  -e /home/stack/templates/ironic-overrides.yaml
```

- デプロイメントのネットワークサービスメカニズムドライバーに応じて、**<default_ironic_template>** を **ironic.yaml** または **ironic-overcloud.yaml** のいずれかに置き換えます。

4.2. ベアメタルノードのクリーニング

Bare Metal Provisioning サービスは、ノードをクリーニングして、プロビジョニングの準備をします。次のいずれかの方法を使用して、ベアメタルノードをクリーニングできます。

- 自動:ノードをプロビジョニング解除する際、ノードクリーニングを自動的に実行するように、オーバークラウドを設定できます。
- 手動:必要に応じて、個別のノードを手動でクリーニングできます。

4.2.1. 自動ノードクリーニングの設定

自動ベアメタルノードクリーニングは、ノードを登録した後、ノードがプロビジョニング状態 **available** に達する前に、実行されます。ノードがプロビジョニング解除されるたびに、自動クリーニングが実行されます。

デフォルトでは、Bare Metal Provisioning サービスは、ノードのクリーニングに **provisioning** という名前のネットワークを使用します。ただし、Networking サービス (neutron) では、ネットワーク名は一意ではないため、プロジェクトが同じ名前のネットワークを作成する可能性があり、Bare Metal Provisioning サービスとの競合が発生します。競合を回避するには、ネットワーク UUID を使用して、ノードクリーニングネットワークを設定します。

手順

1. アンダークラウドのホストにログインします。
2. オーバークラウド認証情報ファイルを入手します。

```
$ source ~/<credentials_file>
```

   - **<credentials_file>** を認証情報ファイルの名前 (**overcloudrc** など) に置き換えます。

3. Bare Metal Provisioning サービスをホスティングするプロバイダーネットワークの UUID を取得します。

```
(overcloud)$ openstack network show <network_name> -f value -c id
```

   - **<network_name>** をベアメタルノードのクリーニングネットワークに使用するネットワークの名前に置き換えます。

4. デプロイメント用の Bare Metal Provisioning サービスを設定するローカル環境ファイル (例: **ironic-overrides.yaml**) を開きます。

5. ノードクリーニングネットワークとして使用するネットワークを設定します。

```
parameter_defaults:
  IronicCleaningNetwork: <network_uuid>
```

   - **<network_uuid>** を手順 3 で取得したプロバイダーネットワークの UUID に置き換えます。

6. **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
```

7. ノードクリーニングネットワーク設定を適用するには、Bare Metal Provisioning 環境ファイルを他の環境ファイルとともにスタックに追加し、オーバークラウドをデプロイします。

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/node-info.yaml \
  -r /home/stack/templates/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/services/<default_ironic_template> \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/ironic-inspector.yaml
\
  -e /home/stack/templates/network_environment_overrides.yaml \
  -n /home/stack/templates/network_data.yaml \
  -e /home/stack/templates/ironic-overrides.yaml
```

   - デプロイメントのネットワークサービスメカニズムドライバーに応じて、**<default_ironic_template>** を **ironic.yaml** または **ironic-overcloud.yaml** のいずれかに置き換えます。

4.2.2. ノードの手動によるクリーニング

必要に応じて、特定のノードを手動でクリーニングできます。ノードのクリーニングには 2 つのモードがあります。

- メタデータのみのクリーニング:ノード上のすべてのディスクからパーティションを削除します。メタデータのみのクリーニングモードは、完全なクリーニングより高速ですが、パーティションテーブルのみを消去するため、セキュリティは低くなります。このモードは、信頼済みのテナント環境でのみ使用してください。

- 完全なクリーニング:ATA セキュア消去またはシュレッドを使用して、すべてのディスクからすべてのデータを削除します。完全なクリーニングが完了するまでは、数時間かかる場合があります。

手順

1. オーバークラウド認証情報ファイルを入手します。

```
$ source ~/<credentials_file>
```

   - **<credentials_file>** を認証情報ファイルの名前 (**overcloudrc** など) に置き換えます。

2. ノードの現在の状態を確認します。

```
$ openstack baremetal node show \
 -f value -c provision_state <node>
```

- **<node>** をクリーニングするノードの名前または UUID に置き換えます。

3. ノードが **manageable** 状態ではない場合は、**manageable** に設定します。

```
$ openstack baremetal node manage <node>
```

4. ノードをクリーニングします。

```
$ openstack baremetal node clean <node> \
  --clean-steps '[{"interface": "deploy", "step": "<clean_mode>"}]'
```

- **<node>** をクリーニングするノードの名前または UUID に置き換えます。
- **<clean_mode>** をノードで実行するクリーニングのタイプに置き換えます。
  - **erase_devices**:完全なクリーニングを実行します。
  - **erase_devices_metadata**:メタデータのみのクリーニングを実行します。

5. クリーニングが完了するまで待ってから、ノードのステータスを確認します。
   - **manageable**:クリーニングが成功し、ノードをプロビジョニングする準備ができました。
   - **clean failed**:クリーニングは失敗しました。失敗の原因については、**last_error** フィールドを調べてください。

4.3. ベアメタルインスタンスを起動するためのフレーバーの作成

クラウドユーザーがベアメタルインスタンスをリクエストするために使用できるフレーバーを作成する必要があります。リソースクラスを使用して、特定のフレーバーで起動されるベアメタルインスタンスにどのベアメタルノードを使用するかを指定できます。ベアメタルノードには、ノード上のハードウェアリソース (GPU など) を識別するリソースクラスをタグ付けできます。クラウドユーザーは、GPU リソースクラスのフレーバーを選択して、vGPU ワークロードのインスタンスを作成できます。コンピューティングスケジューラーは、リソースクラスを使用して、インスタンスに適したホストベアメタルノードを識別します。

手順

1. 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。

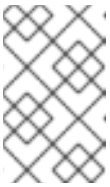
```
$ source ~/overcloudrc
```

2. ベアメタルインスタンスのフレーバーを作成します。

```
(overcloud)$ openstack flavor create --id auto \
 --ram <ram_size_mb> --disk <disk_size_gb> \
 --vcpus <no_vcpus> baremetal
```

- **<ram_size_mb>** をベアメタルノードの RAM (MB 単位) に置き換えます。
- **<disk_size_gb>** をベアメタルノード上のディスク容量 (GB 単位) に置き換えます。

- **<no_vcpus>** をベアメタルノードの CPU 数に置き換えます。



注記

これらの属性は、インスタンスのスケジューリングには使用されません。ただし Compute スケジューラーは、ディスク容量を使用してルートパーティションのサイズを決定します。

3. ノードリストを取得して UUID を把握します。

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

4. 各ベアメタルノードにカスタムベアメタルリソースクラスのタグを付けます。

```
(overcloud)$ openstack baremetal node set \
 --resource-class baremetal.<CUSTOM> <node>
```

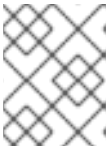
- **<CUSTOM>** は、リソースクラスの目的を識別する文字列に置き換えます。たとえば、**GPU** に設定して、GPU ワークロード用に指定するベアメタルノードにタグを付けるために使用できるカスタム GPU リソースクラスを作成します。

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

5. ベアメタルインスタンスのフレーバーをカスタムリソースクラスに関連付けます。

```
(overcloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_<CUSTOM>=1 \
 baremetal
```

ベアメタルノードのリソースクラスに対応するカスタムリソースクラスの名前を指定するには、リソースクラスを大文字に変換し、それぞれの句読点をアンダースコアに置き換え、**CUSTOM_** の接頭辞を追加します。



注記

フレーバーが要求できるのは、ベアメタルリソースクラスの 1 つのインスタンスだけです。

6. 以下のフレーバー属性を設定して、Compute スケジューラーがインスタンスのスケジューリングにベアメタルフレーバー属性を使用するのを防ぎます。

```
(overcloud)$ openstack flavor set \
 --property resources:VCPU=0 \
 --property resources:MEMORY_MB=0 \
 --property resources:DISK_GB=0 baremetal
```

7. 新規フレーバーの値が正しいことを確認します。

```
(overcloud)$ openstack flavor list
```

4.4. ベアメタルインスタンスを起動するためのイメージの作成

Bare Metal Provisioning サービス (ironic) が含まれるオーバークラウドには、2 つのイメージセットが必要です。

- イメージをデプロイメントします。デプロイイメージは、Bare Metal Provisioning エージェント (**Ironic-python-agent**) がネットワーク経由で RAM ディスクを起動し、オーバークラウドノードのユーザーイメージをディスクにコピーするために必要な、**agent.ramdisk** イメージと **Agent.kernel** イメージです。デプロイイメージはアンダークラウドインストールの一部としてインストールします。詳細は、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。

- ユーザーイメージ:クラウドユーザーがベアメタルインスタンスをプロビジョニングするために使用するイメージ。ユーザーイメージは、**kernel** イメージ、**ramdisk** イメージ、**main** イメージで設定されます。メインイメージは、ルートパーティションイメージまたは完全なディスクイメージのいずれかです。

  - ディスク全体のイメージ:パーティションテーブルとブートローダーを含むイメージ。

  - ルートパーティションイメージ:オペレーティングシステムのルートパーティションのみが含まれます。

互換性のあるディスク全体の RHEL ゲストイメージは、変更せずに動作するはずです。独自のカスタムディスクイメージを作成するには、**イメージの作成と管理ガイドのイメージ**の作成 を参照してください。

4.4.1. デプロイイメージをイメージサービスにアップロードする

Director によってインストールされたデプロイメントイメージをイメージサービスにアップロードする必要があります。デプロイイメージは次の 2 つのイメージで設定されます。

- カーネルイメージ: **/tftpboot/agent.kernel**

- ramdisk イメージ: **/tftpboot/agent.ramdisk**

これらのイメージはホームディレクトリーにインストールされます。デプロイイメージのインストール方法の詳細は、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。

手順

- 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 ./tftpboot/agent.kernel bm-deploy-kernel
$ openstack image create \
  --container-format ari \
  --disk-format ari \
  --public \
  --file ./tftpboot/agent.ramdisk bm-deploy-ramdisk
```

4.5. ベアメタルノードとしての物理マシンの追加

次のいずれかの方法を使用して、ベアメタルノードを登録します。

- ノードの詳細情報を記載したインベントリーファイルを作成し、そのファイルを Bare Metal Provisioning サービスにインポートしてノードを利用できるようにします。
- 物理マシンをベアメタルノードとして登録してから、手動でハードウェア情報を追加し、各イーサネットの MAC アドレス用にポートを作成します。これらの手順は、**overcloudrc** ファイルがある任意のノードで実行できます。

4.5.1. インベントリーファイルを使用したベアメタルノードの登録

ノードの詳細情報を記載したインベントリーファイルを作成し、そのファイルを Bare Metal Provisioning サービス (ironic) にインポートしてノードを利用できるようにします。

前提条件

- Bare Metal Provisioning サービスを含むオーバークラウドのデプロイメント。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

手順

1. ノードの詳細を含むインベントリーファイル **overcloud-nodes.yaml** を作成します。1 つのファイルで複数のノードを登録することが可能です。

```
nodes:
    - name: node0
      driver: ipmi
      driver_info:
        ipmi_address: <ipmi_ip>
        ipmi_username: <user>
        ipmi_password: <password>
        [<property>: <value>]
      properties:
        cpus: <cpu_count>
        cpu_arch: <cpu_arch>
        memory_mb: <memory>
        local_gb: <root_disk>
        root_device:
            serial: <serial>
      ports:
        - address: <mac_address>
```

   - **<ipmi_ip>** は、Bare Metal コントローラーのアドレスに置き換えます。
   - **<user>** は、自分のユーザー名に置き換えます。
   - **<password>** は、自分のパスワードに置き換えます。
   - オプション:**<property>: <value>** を設定する IPMI プロパティーとプロパティー値に置き換えます。使用可能なプロパティーについては、Intelligent Platform Management Interface (IPMI) 電源管理ドライバー を参照してください。
   - **<cpu_count>** は、CPU の数に置き換えます。
   - **<cpu_arch>** は、CPU のアーキテクチャーのタイプに置き換えます。
   - **<memory>** は、メモリー容量 (MiB 単位) に置き換えます。

- **<root_disk>** は、root ディスクの容量 (GiB 単位) に置き換えます。マシンに複数のディスクがある場合にのみ必要です。
- **<serial>** は、デプロイメントに使用するディスクのシリアル番号に置き換えます。
- **<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。
- --driver-info <property>=<value>

2. source コマンドで **overcloudrc** ファイルを読み込みます。

```
$ source ~/overcloudrc
```

3. インベントリーファイルを BareMetalProvisioning サービスにインポートします。

```
$ openstack baremetal create overcloud-nodes.yaml
```

これで、ノードは **enroll** の状態となります。

4. 各ノードでデプロイカーネルとデプロイ ramdisk を指定します。

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

- **<node>** は、ノードの名前または ID に置き換えます。
- **<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. オプション:各ノードの IPMI 暗号スイートを指定します。

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

- **<node>** は、ノードの名前または ID に置き換えます。
- **<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。

6. ノードのプロビジョニング状態を **available** に設定します。

```
$ openstack baremetal node manage <node>
$ openstack baremetal node provide <node>
```

ノードのクリーニングを有効にしている場合には、Bare Metal Provisioning サービスがノードをクリーニングします。

7. ノードにローカルブートオプションを設定します。

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

8. ノードが登録されていることを確認します。

```
$ openstack baremetal node list
```

ノードを登録した後にその状態が表示されるまで時間がかかる場合があります。

4.5.2. ベアメタルノードの手動登録

物理マシンをベアメタルノードとして登録してから、手動でハードウェア情報を追加し、各イーサネットの MAC アドレス用にポートを作成します。これらの手順は、**overcloudrc** ファイルがある任意のノードで実行できます。

前提条件

- Bare Metal Provisioning サービスを含むオーバークラウドのデプロイメント。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。
- **IronicEnabledHardwareTypes** パラメーターを使用して、新しいノードのドライバーを有効にする必要があります。サポートされているドライバーの詳細については、ベアメタルドライバー 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 コマンドでオーバークラウドの認証情報ファイルを読み込みます。

```
(undercloud)$ source ~/overcloudrc
```

3. 新しいノードを追加します。

```
$ openstack baremetal node create --driver <driver_name> --name <node_name>
```

- **<driver_name>** をドライバーの名前 (たとえば、**ipmi**) に置き換えます。
- **<node_name>** を新しいベアメタルノードの名前に置き換えます。

4. ノードの作成時にノードに割り当てられた UUID を書き留めます。
5. 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。

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

<node> をベアメタルノードの UUID に置き換えてください。

6. 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。

```
$ 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。

7. 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。

```
$ 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。

8. オプション:各ノードの IPMI 暗号スイートを指定します。

```
$ 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。

9. オプション:ノードごとの IPMI の詳細を指定します。

```
$ openstack baremetal node set <node> \
 --driver-info <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>** を設定する IPMI プロパティーに置き換えます。使用可能なプロパティーについては、Intelligent Platform Management Interface (IPMI) 電源管理ドライバー を参照してください。
- **<value>** をプロパティー値に置き換えます。

10. オプション:複数のディスクがある場合は、root 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。

```
$ 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>** および **<value>** は、デプロイメントに使用するディスクの情報に置き換えます (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):デバイス識別子。
  - **vendor** (文字列):デバイスベンダー。
  - **serial** (文字列):ディスクのシリアル番号。
  - **hctl** (文字列):SCSI の Host:Channel:Target:Lun
  - **size** (整数):デバイスのサイズ (GB)。
  - **wwn** (文字列):一意のストレージ ID。
  - **wwn_with_extension** (文字列):ベンダーエクステンション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):ローテーションデバイス (HDD) の場合は true、それ以外の場合は false (SSD)。
  - **name** (文字列)デバイス名 (例: /dev/sdb1)。このプロパティーは、永続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。

11. 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。

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

- **<node>** 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。

12. ノードの設定を検証します。

```
$ 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**:インターフェイスの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。

4.5.3. ベアメタルノードのプロビジョニング状態

ベアメタルノードは、そのライフタイム中に複数のプロビジョニング状態を移行します。ノードで実行される API 要求およびコンダクターイベントが移行を開始します。プロビジョニング状態には、"stable" と "in transition" の 2 つのカテゴリーがあります。

以下の表を使用して、ノードが配置されている可能性のあるプロビジョニングの状態と、ノードをあるプロビジョニング状態から別の状態に移行するために使用できるアクションを説明します。

表4.1 プロビジョニングの状態

State	カテゴリー	説明
enroll	安定	各ノードの初期状態。ノードの登録に関する情報は、ベアメタルノードとしての物理マシンの追加 を参照してください。
verifying	移行中	Bare Metal Provisioning サービスは、ノードの登録時に提供された **driver_info** 設定を使用してノードを管理できることを検証します。

State	カテゴリー	説明
manageable	安定	ノードは、Bare Metal Provisioning サービスがノードを管理できることを確認したら、manageable の状態に移行します。以下のコマンドを使用して、ノードを **manageable** 状態から以下のいずれかの状態に移行できます。 ● **openstack baremetal node adopt →adopting → active** ● **openstack baremetal node provide →cleaning → available** ● **openstack baremetal node clean →cleaning → available** ● **openstack baremetal node inspect → inspecting →manageable** ノードを以下の failed 状態のいずれかに移行した後、そのノードを **manageable** の状態に移行する必要があります。 ● **adopt failed** ● **clean failed** ● **inspect failed** ノードを更新する必要がある場合は、ノードを **manageable** 状態にします。
inspecting	移行中	Bare Metal Provisioning サービスは、ノードのイントロスペクションを使用して、ハードウェアから派生したノードプロパティーを更新し、ハードウェアの現在の状態を反映します。ノードは同期検査の場合は **manageable** に、非同期検査の場合は 非同期検査の **inspect wait** に移行します。エラーが発生すると、ノードは**inspect failed** に移行します。
inspect wait	移行中	非同期検査が進行中であることを示すプロビジョニング状態。ノードの検査に成功すると、ノードは **manageable** の状態に移行します。
inspect failed	安定	ノードの検査が失敗したことを示すプロビジョニング状態。以下のコマンドを使用して、ノードを **inspect failed** 状態から以下のいずれかの状態に移行することができます。 ● **openstack baremetal node inspect → inspecting →manageable** ● **openstack baremetal node manage → manageable**

State	カテゴリー	説明
cleaning	移行中	**cleaning** 状態のノードはスクラブされ、既知の設定に再プログラムされます。ノードが **cleaning** 状態にあると、ネットワーク管理に応じて、コンダクターは以下のタスクを実行します。 ● 帯域外:コンダクターは **clean** ステップを実行します。 ● 帯域内:コンダクターは、帯域内クリーンステップを実行するために RAM ディスクをブートする環境を準備します。準備タスクには、PXE 設定ファイルの構築と DHCP の設定が含まれます。
clean wait	移行中	**clean wait** 状態のノードはスクラブされ、既知の設定に再プログラムされます。この状態は、**clean wait** 状態では、コンダクターが RAM ディスクの起動または **clean** ステップの終了を待機している点を除けば、**cleaning** 状態と似ています。 **openstack baremetal node abort** を実行して、**clean wait** 状態のノードのクリーニングプロセスを中断できます。
available	安定	ノードが正常に事前設定され、クリーンアップされると、それらは **available** 状態に移行し、プロビジョニングの準備が整います。以下のコマンドを使用して、ノードを **available** 状態から以下のいずれかの状態に移行することができます。 ● **openstack baremetal node deploy → deploying → active** ● **openstack baremetal node manage → manageable**
deploying	移行中	**deploying** 状態のノードは、ワークロード用に準備されているため、以下のタスクを実行します。 ● ノードのデプロイメントに適切な BIOS オプションを設定する。 ● ドライブにパーティションを設定しファイルシステムを作成する。 ● ノード固有のネットワーク設定や設定ドライブパーティションなど、追加のサブシステムで必要になる可能性のある追加のリソースを作成します。

State	カテゴリー	説明
wait call-back	移行中	**wait call-back** 状態のノードは、ワークロード用に準備されています。この状態は **deploying** 状態と似ていますが、**wait call-back** 状態では、コンダクターはノードの準備前にタスクの完了を待っている点が異なります。たとえば、コンダクターがノードを準備する前に、以下のタスクを完了する必要があります。 ● ramdisk が起動している。 ● ブートローダーがインストールされている。 ● イメージがディスクに書き込まれている。 **openstack baremetal node delete** または **openstack baremetal node undeploy** を実行して、**wait call-back** 状態のノードのデプロイメントを中断できます。
deploy failed	安定	ノードのデプロイメントが失敗したことを示すプロビジョニング状態。以下のコマンドを使用して、ノードを **deploy failed** 状態から以下のいずれかの状態に移行できます。 ● **openstack baremetal node deploy → deploying → active** ● **openstack baremetal node rebuild → deploying → active** ● **openstack baremetal node delete → deleting → cleaning → clean wait → cleaning → available** ● **openstack baremetal node undeploy → deleting → cleaning → clean wait → cleaning → available**
active	安定	**active** 状態のノードで、ワークロードが実行しています。Bare Metal Provisioning サービスは、電源状態を含む帯域外センサー情報を定期的に収集する場合があります。以下のコマンドを使用して、ノードを **active** 状態から以下のいずれかの状態に移行できます。 ● **openstack baremetal node delete → deleting → available** ● **openstack baremetal node undeploy → cleaning → available** ● **openstack baremetal node rebuild → deploying → active** ● **openstack baremetal node rescue → rescuing → rescue**

State	カテゴリー	説明
deleting	移行中	ノードが **deleting** 状態の場合、Bare Metal Provisioning サービスはアクティブなワークロードを逆アセンブルし、ノードのデプロイメントまたはレスキュー時にノードに追加された設定およびリソースを削除します。ノードは **deleting** 状態から **cleaning** 状態へとすばやく移行し、次に **clean wait** 状態に移行します。
error	安定	ノードの削除に失敗した場合は、ノードが **error** 状態に移行します。以下のコマンドを使用して、ノードを **error** 状態から以下のいずれかの状態に移行できます。 ● **openstack baremetal node delete → deleting → available** ● **openstack baremetal node undeploy → cleaning → available**
adopting	移行中	**openstack baremetal node adopt** コマンドを使用すると、最初のクリーニングやデプロイを行わずに、既存のワークロードを持つノードを **manageable** から **active** 状態に直接移行できます。ノードが **adopting** 状態の場合、Bare Metal Provisioning サービスは既存のワークロードでノードの管理を引き継ぎます。
rescuing	移行中	**rescuing** 状態のノードは、以下のレスキュー操作を実行する準備ができています。 ● ノードのデプロイメントに適切な BIOS オプションを設定する。 ● ノード固有のネットワーク設定など、追加のサブシステムで必要になる可能性のある追加のリソースを作成する。
rescue wait	移行中	**rescue wait** 状態のノードはレスキューされます。この状態は、**rescue wait** 状態では、コンダクターが ramdisk のブートを待機するか、rescue という名前のユーザーのパスワードを設定するなど、ノードでインバンドの実行に必要なレスキューの一部を実行することを除いて、**rescuing** の状態と似ています。 **openstack baremetal node abort** を実行して、**rescue wait** 状態でノードのレスキュー操作を中断できます。

State	カテゴリー	説明
rescue failed	安定	ノードのレスキューが失敗したことを示すプロビジョニング状態。以下のコマンドを使用して、**rescue failed** 状態から以下のいずれかの状態にノードを移行できます。 • **openstack baremetal node rescue → rescuing → rescue** • **openstack baremetal node unrescue → unrescuing → active** • **openstack baremetal node delete → deleting → available**
rescue	安定	**rescue** 状態のノードは、レスキュー ramdisk を実行している。Bare Metal Provisioning サービスは、電源状態を含む帯域外センサー情報を定期的に収集する場合があります。以下のコマンドを使用して、ノードを **rescue** 状態から以下のいずれかの状態に移行できます。 • **openstack baremetal node unrescue → unrescuing → active** • **openstack baremetal node delete → deleting → available**
unrescuing	移行中	**unrescuing** 状態のノードは、**rescue** 状態から **active** 状態に移行する準備ができています。
unrescue failed	安定	ノードのアンレスキュー操作が失敗したことを示すプロビジョニング状態。以下のコマンドを使用して、**unrescue failed** 状態から以下のいずれかの状態にノードを移行できます。 • **openstack baremetal node rescue → rescuing → rescue** • **openstack baremetal node unrescue → unrescuing → active** • **openstack baremetal node delete → deleting → available**

4.6. REDFISH 仮想メディアブートの設定

重要

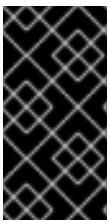
この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト用途にのみご利用いただく機能です。実稼働環境にはデプロイしないでください。テクノロジープレビュー機能についての詳しい情報は、対象範囲の詳細 を参照してください。

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 Provisioning 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。

4.6.1. Redfish 仮想メディアブートを使用するベアメタルサーバーのデプロイ



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。

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
- Sushy ライブラリーがインストールされている。

```
$ sudo yum install sushy
```

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。

```
$ openstack baremetal node set --property capabilities="boot_mode:uefi" $NODE_NAME
```

$NODE_NAME 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。

5. 新しいベアメタルサーバーを作成します。

```
$ openstack server create \
    --flavor baremetal \
    --image $IMAGE \
    --network $NETWORK \
    test_instance
```

$IMAGE および **$NETWORK** は、使用するイメージおよびネットワークの名前に置き換えます。

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

OpenStack Compute は、ホストアグリゲートを使用してアベイラビリティーゾーンをパーティション分割し、特定の共有属性が指定されたノードをグループ化します。インスタンスがプロビジョニングされると、Compute のスケジューラーがフレーバーのプロパティーをホストアグリゲートに割り当てられたプロパティーと比較して、インスタンスが正しいアグリゲート内の正しいホストに (物理マシン上または仮想マシンとして) プロビジョニングされたことを確認します。

本項の手順を実施して、以下の操作を行います。

- **baremetal** プロパティーをフレーバーに追加して、**true** または **false** に設定する。

- 一致する **baremetal** プロパティーを設定して、ベアメタルホスト用とコンピュートノード用のホストアグリゲートを別々に作成する。1 つのアグリゲートでグループ化されたノードは、このプロパティーを継承します。

前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

手順

1. ベアメタル用のフレーバーで **baremetal** プロパティーを **true** に設定します。

```
$ openstack flavor set baremetal --property baremetal=true
```

2. 仮想インスタンスに使用するフレーバーで **baremetal** プロパティーを **false** に設定します。

```
$ openstack flavor set FLAVOR_NAME --property baremetal=false
```

3. **baremetal-hosts** という名前のホストアグリゲートを作成します。

```
$ openstack aggregate create --property baremetal=true baremetal-hosts
```

4. 各コントローラーノードを **baremetal-hosts** アグリゲートに追加します。

```
$ openstack aggregate add host baremetal-hosts HOSTNAME
```

注記

NovaIronic サービスでコンポーザブルロールを作成していた場合には、このサービスがあるノードをすべて **baremetal-hosts** アグリゲートに追加します。デフォルトでは、**NovaIronic** サービスがあるのはコントローラーノードのみです。

5. **virtual-hosts** という名前のホストアグリゲートを作成します。

```
$ openstack aggregate create --property baremetal=false virtual-hosts
```

6. 各コンピュートノードを **virtual-hosts** アグリゲートに追加します。

```
$ openstack aggregate add host virtual-hosts HOSTNAME
```

7. オーバークラウドのデプロイ時に以下の Compute フィルタースケジューラーを追加していなかった場合には、この時点で _/**etc**/**nova**/**nova.conf**_ の **scheduler_default_filters** セクションの既存リストに追加します。

```
AggregateInstanceExtraSpecsFilter
```

第5章 ベアメタルノードの管理

Bare Metal Provisioning サービス (ironic) が含まれるオーバークラウドをデプロイしたら、登録済みのベアメタルノードに物理マシンをプロビジョニングして、オーバークラウドでベアメタルインスタンスを起動することができます。

前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

5.1. ベアメタルインスタンスの起動

コマンドラインまたは OpenStack Dashboard のいずれかで、インスタンスを起動することができます。

前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

5.1.1. コマンドラインインターフェイスを使用したインスタンスの起動

OpenStack クライアント CLI を使用して、ベアメタルインスタンスを作成できます。

前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

手順

1. Identity サービス (keystone) 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 ~/overcloudrc
```

2. ベアメタルインスタンスを作成します。

```
$ openstack server create \
 --nic net-id=<network_uuid> \
 --flavor baremetal \
 --image <image_uuid> \
 myBareMetalInstance
```

   - **<network_uuid>** は、Bare Metal Provisioning サービス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。
   - **<image_uuid>** を、インスタンスが必要とするソフトウェアプロファイルを持つイメージの一意の識別子に置き換えます。

3. インスタンスのステータスを確認します。

```
$ openstack server list --name myBareMetalInstance
```

5.1.2. Dashboard を使用したインスタンスの起動

Dashboard のグラフィカルユーザーインターフェイ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。

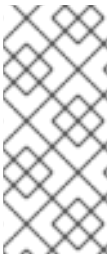
前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

手順

1. **http[s]://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**の順にクリックします。
3. **インスタンスの起動** をクリックします。
   - **詳細** タブで **インスタンス名** を指定して、**インスタンス数** に **1** を選択します。
   - **ソース** タブで **ブートソースを選択してください** のドロップダウンメニューから **イメージ** を選択し、続いて ↑ (上向き矢印) の記号をクリックして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。選択したイメージが **割り当て済み** に移動します。
   - **フレーバー** タブで **baremetal** を選択します。
   - **ネットワーク** タブで、↑ (上向き矢印) および ↓ (下向き矢印) ボタンを使用して必要なネットワークを **利用可能** から **割り当て済み** に移動します。ここでは、必ず Bare Metal Provisioning サービス用に作成した共有ネットワークを選択してください。
   - インスタンスをセキュリティーグループに割り当てるには、**セキュリティーグループ** タブで矢印を使用してそのグループを **割り当て済み** に移動します。
4. **インスタンスの起動** をクリックします。

5.2. BARE METAL PROVISIONING サービスでのポートグループの設定

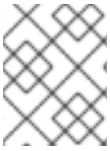


注記

ベアメタルノード向けのポートグループ機能は、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、テスト目的にのみご利用いただく機能で、実稼働環境にデプロイ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。

ポートグループ (ボンディング) の機能により、複数のネットワークインターフェイスを単一のボンディングされたインターフェイスに統合することができます。ポートグループの設定は常に、個別のポート設定に優先します。

ポートグループに物理ネットワークがある場合には、そのポートグループ内の全ポートに同じ物理ネットワークを使用すべきです。Bare Metal Provisioning サービスは、**configdrive** を使用してインスタンスでのポートグループの設定をサポートしています。



注記

Bare Metal Provisioning サービス API バージョン 1.26 は、ポートグループの設定をサポートしています。前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

5.2.1. 手動によるスイッチ上のポートグループの設定

ベアメタルのデプロイメントでポートグループを設定するには、スイッチ上でポートグループを手動設定する必要があります。スイッチによって名前が異なる場合があるため、スイッチ上のモードとプロパティーが、ベアメタル側のモードとプロパティーに対応している状態にする必要があります。

注記

iPXE を使用してデプロイメントを起動する必要がある場合、プロビジョニングとクリーニングにはポートグループを使用できません。

ポートグループのフォールバック機能により、接続でエラーが発生した際に、1 つのポートグループ内の全ポートを個々のスイッチポートにフォールバックさせることができます。スイッチがポートグループのフォールバックをサポートしているかどうかに応じて、**--support-standalone-ports** と **--unsupport-standalone-ports** のオプションを使用することができます。

前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

5.2.2. Bare Metal Provisioning サービスでのポートグループの設定

複数のネットワークインターフェイスを単一の **ボンディングインターフェイス** に統合するポートグループを作成します。

前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

手順

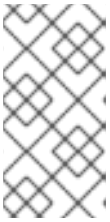
1. ポートグループが属する先のノード、その名前、アドレス、モード、プロパティー、スタンドアロンポートへのフォールバックをサポートするかどうかを指定して、ポートグループを作成します。

```
# openstack baremetal port group create --node NODE_UUID --name NAME --address
MAC_ADDRESS --mode MODE  --property miimon=100 --property
xmit_hash_policy="layer2+3" --support-standalone-ports
```

また、**openstack baremetal port group set** コマンドを使用してポートグループを更新することもできます。

アドレスを指定しない場合には、デプロイされるインスタンスのポートグループアドレスは OpenStack Networking のポートと同じになります。neutron ポートを接続しないと、ポートグループの設定は失敗します。

インターフェイスの接続中には、ポートグループの優先度はポートよりも高くなるので、最初に使用されます。現在、インターフェイスの接続要求で、ポートグループとポートのどちらを優先するかを指定することはできません。ポートのないポートグループは無視されます。



注記

ポートグループは、手動でスタンドアロンモードに設定する必要があります。そのためには、イメージ内で設定するか、**configdrive** を生成してノードの **instance_info** に追加します。ポートグループの設定が機能するには、**cloud-init** バージョン 0.7.7 以降を使用している必要があります。

2. ポートをポートグループに関連付けます。

   - ポートの作成中

     ```
     # openstack baremetal port create --node NODE_UUID --address MAC_ADDRESS --
     port-group test
     ```

   - ポートの更新中

     ```
     # openstack baremetal port set PORT_UUID --port-group PORT_GROUP_UUID
     ```

3. **cloud-init** 対応のイメージまたはボンディングをサポートしているイメージを提供することにより、インスタンスを起動します。
   ポートグループが適切に設定されているかを確認するには、以下のコマンドを実行します。

   ```
   # cat /proc/net/bonding/bondX
   ```

   X は、**cloud-init** が設定済みの各ポートグループに対して自動生成する番号です。**0** で始まり、ポートグループを設定するたびに 1 つずつ増えます。

5.3. ホストから IP アドレスへのマッピングの確認

各 IP アドレスが割り当てられているホストおよびベアメタルノードを確認するには、以下のコマンドを使用します。これらのコマンドにより、ホストに直接アクセスせずに、ホストから IP へのマッピングをアンダークラウドで確認することが可能です。

前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

手順

1. 以下のコマンドを実行して、各ホストの IP アドレスを表示します。

```
(undercloud) [stack@host01 ~]$ openstack stack output show overcloud HostsEntry --max-
width 80

+--------------+------------------------------------------------------------+
| Field        | Value                                                      |
+--------------+------------------------------------------------------------+
| description  | The content that should be appended to your /etc/hosts if you |
|              | want to get                                                |
|              | hostname-based access to the deployed nodes (useful for    |
|              | testing without                                            |
|              | setting up a DNS).                                         |
|              |                                                            |
| output_key   | HostsEntry                                                 |
| output_value | 172.17.0.10 overcloud-controller-0.localdomain overcloud-  |
|              | controller-0                                               |
|              | 10.8.145.18 overcloud-controller-0.external.localdomain    |
|              | overcloud-controller-0.external                            |
|              | 172.17.0.10 overcloud-controller-0.internalapi.localdomain |
|              | overcloud-controller-0.internalapi                         |
|              | 172.18.0.15 overcloud-controller-0.storage.localdomain     |
|              | overcloud-controller-0.storage                             |
|              | 172.21.2.12 overcloud-controller-0.storagemgmt.localdomain |
|              | overcloud-controller-0.storagemgmt                         |
|              | 172.16.0.15 overcloud-controller-0.tenant.localdomain      |
|              | overcloud-controller-0.tenant                              |
|              | 10.8.146.13 overcloud-controller-0.management.localdomain  |
|              | overcloud-controller-0.management                          |
|              | 10.8.146.13 overcloud-controller-0.ctlplane.localdomain    |
|              | overcloud-controller-0.ctlplane                            |
|              |                                                            |
|              | 172.17.0.21 overcloud-compute-0.localdomain overcloud-     |
|              | compute-0                                                  |
|              | 10.8.146.12 overcloud-compute-0.external.localdomain       |
|              | overcloud-compute-0.external                               |
|              | 172.17.0.21 overcloud-compute-0.internalapi.localdomain    |
|              | overcloud-compute-0.internalapi                            |
|              | 172.18.0.20 overcloud-compute-0.storage.localdomain        |
|              | overcloud-compute-0.storage                                |
|              | 10.8.146.12 overcloud-compute-0.storagemgmt.localdomain    |
|              | overcloud-compute-0.storagemgmt                            |
|              | 172.16.0.16 overcloud-compute-0.tenant.localdomain overcloud- |
|              | compute-0.tenant                                           |
|              | 10.8.146.12 overcloud-compute-0.management.localdomain     |
|              | overcloud-compute-0.management                             |
|              | 10.8.146.12 overcloud-compute-0.ctlplane.localdomain       |
|              | overcloud-compute-0.ctlplane                               |
|              |                                                            |
|              |                                                            |
|              |                                                            |
|              |                                                            |
|              | 10.8.145.16  overcloud.localdomain                         |
|              | 10.8.146.7  overcloud.ctlplane.localdomain                 |
```

```
|             | 172.17.0.19  overcloud.internalapi.localdomain             |
|             | 172.18.0.19  overcloud.storage.localdomain                 |
|             | 172.21.2.16  overcloud.storagemgmt.localdomain               |
+-------------+------------------------------------------------------------+
```

2. 特定のホストに絞り込むには、以下のコマンドを実行します。

```
(undercloud) [stack@host01 ~]$ openstack stack output show overcloud HostsEntry -c
output_value -f value | grep overcloud-controller-0

172.17.0.12 overcloud-controller-0.localdomain overcloud-controller-0
10.8.145.18 overcloud-controller-0.external.localdomain overcloud-controller-0.external
172.17.0.12 overcloud-controller-0.internalapi.localdomain overcloud-controller-0.internalapi
172.18.0.12 overcloud-controller-0.storage.localdomain overcloud-controller-0.storage
172.21.2.13 overcloud-controller-0.storagemgmt.localdomain overcloud-controller-
0.storagemgmt
172.16.0.19 overcloud-controller-0.tenant.localdomain overcloud-controller-0.tenant
10.8.146.13 overcloud-controller-0.management.localdomain overcloud-controller-
0.management
10.8.146.13 overcloud-controller-0.ctlplane.localdomain overcloud-controller-0.ctlplane
```

3. ホストをベアメタルノードにマッピングするには、以下のコマンドを実行します。

```
(undercloud) [stack@host01 ~]$ openstack baremetal node list --fields uuid name
instance_info -f json
[
  {
    "UUID": "c0d2568e-1825-4d34-96ec-f08bbf0ba7ae",
    "Instance Info": {
      "root_gb": "40",
      "display_name": "overcloud-compute-0",
      "image_source": "24a33990-e65a-4235-9620-9243bcff67a2",
      "capabilities": "{\"boot_option\": \"local\"}",
      "memory_mb": "4096",
      "vcpus": "1",
      "local_gb": "557",
      "configdrive": "******",
      "swap_mb": "0",
      "nova_host_id": "host01.lab.local"
    },
    "Name": "host2"
  },
  {
    "UUID": "8c3faec8-bc05-401c-8956-99c40cdea97d",
    "Instance Info": {
      "root_gb": "40",
      "display_name": "overcloud-controller-0",
      "image_source": "24a33990-e65a-4235-9620-9243bcff67a2",
      "capabilities": "{\"boot_option\": \"local\"}",
      "memory_mb": "4096",
      "vcpus": "1",
      "local_gb": "557",
      "configdrive": "******",
      "swap_mb": "0",
      "nova_host_id": "host01.lab.local"
```

```
    },
    "Name": "host3"
  }
]
```

5.4. 仮想ネットワークインターフェイスの接続と切断

Bare Metal Provisioning サービスには、仮想ネットワークインターフェイス間のマッピングを管理するための API があります。たとえば、OpenStack Networking サービスのインターフェイスと実際の物理インターフェイス (NIC) などです。これらのインターフェイスを各 Bare Metal Provisioning ノードに設定して、仮想ネットワークインターフェイス (VIF) から物理ネットワークインターフェイス (PIF) へのマッピングロジックを設定することができます。インターフェイスを設定するには、**openstack baremetal node vif*** コマンドを使用します。

前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

手順

1. ベアメタルノードに現在接続されている VIF の ID をリスト表示します。

```
$ openstack baremetal node vif list baremetal-0
+--------------------------------------+
| ID                                   |
+--------------------------------------+
| 4475bc5a-6f6e-466d-bcb6-6c2dce0fba16 |
+--------------------------------------+
```

2. VIF がアタッチされた後に、Bare Metal Provisioning サービスは OpenStack Networking サービス内の仮想ポートを実際の物理ポートの MAC アドレスで更新します。このポートアドレスを確認します。

```
$ openstack port show 4475bc5a-6f6e-466d-bcb6-6c2dce0fba16 -c mac_address -c fixed_ips
+-------------+-------------------------------------------------------------------------------+
| Field       | Value                                                                         |
+-------------+-------------------------------------------------------------------------------+
| fixed_ips   | ip_address='192.168.24.9', subnet_id='1d11c677-5946-4733-87c3-
23a9e06077aa' |
| mac_address | 00:2d:28:2f:8d:95                                                             |
+-------------+-------------------------------------------------------------------------------+
```

3. **baremetal-0** ノードを作成したネットワーク上に新規ポートを作成します。

```
$ openstack port create --network baremetal --fixed-ip ip-address=192.168.24.24 baremetal-
0-extra
```

4. インスタンスからポートを削除します。

```
$ openstack server remove port overcloud-baremetal-0 4475bc5a-6f6e-466d-bcb6-
6c2dce0fba16
```

5. その IP アドレスがリストには存在しなくなったことを確認します。

```
$ openstack server list
```

6. そのノードに接続されている VIF があるかどうかを確認します。

```
$ openstack baremetal node vif list baremetal-0
$ openstack port list
```

7. 新規作成されたポートを追加します。

```
$ openstack server add port overcloud-baremetal-0 baremetal-0-extra
```

8. 新しい IP アドレスに新しいポートが表示されることを確認します。

```
$ openstack server list
+--------------------------------------+------------------------+--------+------------------------+----------
------------------+---------+
| ID                                   | Name                   | Status | Networks               | Image
| Flavor  |
+--------------------------------------+------------------------+--------+------------------------+----------
------------------+---------+
| 53095a64-1646-4dd1-bbf3-b51cbcc38789 | overcloud-controller-2  | ACTIVE |
ctlplane=192.168.24.7  | overcloud-hardened-uefi-full | control |
| 3a1bc89c-5d0d-44c7-a569-f2a3b4c73d65 | overcloud-controller-0  | ACTIVE |
ctlplane=192.168.24.8  | overcloud-hardened-uefi-full | control |
| 6b01531a-f55d-40e9-b3a2-6d02be0b915b | overcloud-controller-1  | ACTIVE |
ctlplane=192.168.24.16 | overcloud-hardened-uefi-full | control |
| c61cc52b-cc48-4903-a971-073c60f53091 | overcloud-novacompute-0overcloud-baremetal-
0 | ACTIVE | ctlplane=192.168.24.24 | overcloud-hardened-uefi-full | compute |
+--------------------------------------+------------------------+--------+------------------------+----------
------------------+---------+
```

9. VIF ID が新規ポートの UUID であるかどうかを確認します。

```
$ openstack baremetal node vif list baremetal-0
+--------------------------------------+
| ID                                   |
+--------------------------------------+
| 6181c089-7e33-4f1c-b8fe-2523ff431ffc |
+--------------------------------------+
```

10. OpenStack Networking ポートの MAC アドレスが更新され、Bare Metal Provisioning サービスポートの中の 1 つと一致しているかどうかを確認します。

```
$ openstack port show 6181c089-7e33-4f1c-b8fe-2523ff431ffc -c mac_address -c fixed_ips
+-------------+-------------------------------------------------------------------------------+
| Field       | Value                                                                         |
+-------------+-------------------------------------------------------------------------------+
| fixed_ips   | ip_address='192.168.24.24', subnet_id='1d11c677-5946-4733-87c3-
23a9e06077aa' |
| mac_address | 00:2d:28:2f:8d:95                                                             |
+-------------+-------------------------------------------------------------------------------+
```

11. 新規 IP アドレスを認識するように、ベアメタルノードを再起動します。

```
$ openstack server reboot overcloud-baremetal-0
```

インターフェイスを接続または切断した後には、ベアメタルの OS は変更されたネットワークインターフェイスを削除/追加/変更します。ポートを置き換える場合、DHCP 要求が新規 IP アドレスを取得しますが、古い DHCP リースがまだ有効なので、多少時間がかかる場合があります。変更を即時に適用するには、ベアメタルホストをリブートします。

5.5. BARE METAL PROVISIONING サービスの通知の設定

Bare Metal Provisioning サービス (ironic) を設定して、サービス内で発生するさまざまなイベントの通知を表示することができます。外部サービスは、請求目的、データストアの監視、およびその他の目的でこれらの通知を使用することができます。Bare Metal Provisioning サービスの通知を有効にするには、**ironic.conf** 設定ファイルで以下のオプションを設定する必要があります。

前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

手順

- **[DEFAULT]** セクションの **notification_level** オプションは、通知送信の最小の優先度を決定します。このオプションの値は、**debug**、**info**、**warning**、**error**、**critical** のいずれかに設定することができます。オプションが **warning** に設定されると、優先度が **warning**、**error**、または **critical** のいずれかである通知はすべて送信されますが、優先度が **debug** または **info** の通知は送信させません。このオプションが設定されていない場合には、通知は一切送信されません。利用可能な各通知の優先度は、以下に記載しています。

- **[oslo_messaging_notifications]** セクションの **transport_url** のオプションは、通知の送信に使用されるメッセージバスを決定します。このオプションが設定されていない場合には、RPC に使われるデフォルトのトランスポートが使用されます。

通知はすべて、メッセージバス内の **ironic_versioned_notifications** トピックで発行されます。通常は、メッセージバスを通過する各種別のメッセージは、メッセージの内容を説明しているトピックに関連付けられます。

5.6. 電源異常からの自動復帰の設定

Bare Metal Provisioning サービス (ironic) には、ノードの電源、クリーニング、およびレスキューアボートの失敗を記録する文字列フィールド **fault** があります。

表5.1 Ironic ノードの異常

異常	説明
power failure	電源の同期に失敗したため (リトライ回数の最大値の超過)、ノードはメンテナンスモードに移行しています。

異常	説明
clean failure	クリーニング操作に失敗したため、ノードはメンテナンスモードに移行しています。
rescue abort failure	レスキューアボート時のクリーニング操作に失敗したため、ノードはメンテナンスモードに移行しています。
none	異常は発生していません。

Conductor は、このフィールドの値を定期的に確認します。Conductor が **power failure** の状態を検出し、ノードの電源の復旧に成功すると、ノードはメンテナンスモードから抜け出し動作状態に戻ります。

注記

オペレーターが手動でノードをメンテナンスモードに切り替えた場合には、Conductor が自動的にノードをメンテナンスモードから移行させることはありません。

デフォルトの間隔は 300 秒ですが、hieradata を使用して director からこの間隔を設定することができます。

前提条件

- Bare Metal Provisioning サービスが含まれるオーバークラウドのデプロイメントが正常に完了していること。詳細は、Bare Metal Provisioning サービスを有効にしたオーバークラウドのデプロイ を参照してください。

手順

- 以下の hieradata を追加して、カスタムの復帰間隔を設定します。

```
ironic::conductor::power_failure_recovery_interval
```

  電源異常からの自動復帰を無効にするには、値を **0** に設定します。

5.7. オーバークラウドノードのイントロスペクション

オーバークラウドノードのイントロスペクションを実行して、ノードの仕様を識別し、director に保存します。

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. **overcloudrc** 認証情報ファイルを入手します。

```
$ source ~/overcloudrc
```

3. イントロスペクションコマンドを実行します。

```
$ openstack baremetal introspection start [--wait] <NODENAME>
```

<NODENAME> は、検査するノードの名前または UUID に置き換えます。

4. イントロスペクションのステータスを確認します。

```
$ openstack baremetal introspection status <NODENAME>
```

<NODENAME> は、ノードの名前または UUID に置き換えます。

次のステップ

- イントロスペクションデータを抽出します。

```
$ openstack baremetal introspection data save <NODE-UUID>
```

<NODENAME> は、ノードの名前または UUID に置き換えます。

第6章 ブート可能ボリュームからベアメタルインスタンスを作成できるようにベアメタルノードを設定する

重要

この機能は Red Hat OpenStack Platform 17.0 では非推奨になりました。RHOSP 17.0 ではバグ修正とサポートが提供されますが、新しい機能の拡張は行われません。

Block Storage サービス (cinder) にボリュームを作成し、これらのボリュームを Bare Metal Provisioning サービス (ironic) で作成するベアメタルインスタンスに接続することができます。

クラウドユーザーがブート可能ボリュームからベアメタルインスタンスを作成できるようにするには、次のタスクを実行します。

1. ブート可能ボリュームからベアメタルインスタンスを起動できるように各ベアメタルノードを設定します。
2. ブートディスク上で iSCSI カーネルパラメーターを設定します。

6.1. 前提条件

- Bare Metal Provisioning サービス (ironic) は、iSCSI インターフェイスを介してベアメタルノードをブロックストレージボリュームに接続します。したがって、オーバークラウドは、Block Storage サービス (cinder) の iSCSI バックエンドを使用してデプロイする必要があります。Block Storage サービスの iSCSI バックエンドを有効にするには、**CinderEnableIscsiBackend** パラメーターを **true** に設定し、オーバークラウドをデプロイします。

注記

Red Hat Ceph Storage バックエンドでは Block Storage ボリュームブート機能を使用できません。

6.2. ブート可能ボリュームからベアメタルインスタンスを作成するためのノードの設定

各ベアメタルノードを設定して、ブート可能ボリュームからベアメタルインスタンスを起動できるようにする必要があります。

手順

1. オーバークラウド認証情報ファイルを入手します。

```
$ source ~/<credentials_file>
```

   - **<credentials_file>** を認証情報ファイルの名前 (**overcloudrc** など) に置き換えます。

2. 各ベアメタルノードの **iscsi_boot** 機能を **true** に設定します。

```
$ openstack baremetal node set --property capabilities=iscsi_boot:true <node_uuid>
```

   - **<node_uuid>** をベアメタルノードの ID に置き換えます。

3. 各ベアメタルノードの **storage-interface** を **cinder** に設定します。

```
$ openstack baremetal node set --storage-interface cinder <node_uuid>
```

4. ノードの iSCSI コネクターを作成します。

```
$ openstack baremetal volume connector create --node <node_uuid> \
 --type iqn --connector-id <connector_id>
```

- **<connector_id>** を各ノードの一意の ID (例: **iqn.2010-10.org.openstack.node<NUM>)** に置き換えます。ここで、**<NUM>** は各ノードの増分番号です。

6.3. ブートディスクでの ISCSI カーネルパラメーターの設定

カーネルで iSCSI ブートを有効にするには、インスタンスイメージを設定する必要があります。

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. **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. Red Hat Enterprise Linux 製品ソフトウェアダウンロードページ から、QCOW2 形式の Red Hat Enterprise Linux KVM イメージをダウンロードします。
4. イメージをアンダークラウドの **/home/stack/** ディレクトリーにコピーします。
5. QCOW2 イメージをマウントし、**root** ユーザーとしてアクセスします。
   a. **nbd** カーネルモジュールを読み込みます。

```
$ sudo modprobe nbd
```

   b. QCOW イメージを **/dev/nbd0** として接続します。

```
$ sudo qemu-nbd --connect=/dev/nbd0 <image>
```

   c. NBD 上のパーティションを確認します。

```
$ sudo fdisk /dev/nbd0 -l
```

   新しい Red Hat Enterprise Linux QCOW2 イメージには、パーティションが 1 つだけ含まれます。通常、そのパーティションは NBD の **/dev/nbd0p1** という名前です。

   d. イメージのマウントポイントを作成します。

```
$ mkdir /tmp/mountpoint
```

   e. イメージをマウントします。

```
$ sudo mount /dev/nbd0p1 /tmp/mountpoint/
```

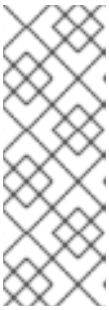
f. イメージがホストのデバイス情報にアクセスできるように、**dev** ディレクトリーをマウントします。

```
$ sudo mount -o bind /dev /tmp/mountpoint/dev
```

g. ルートディレクトリーをマウントポイントに変更します。

```
$ sudo chroot /tmp/mountpoint /bin/bash
```

6. イメージ上で iSCSI を設定します。



注記

このステップの一部のコマンドにより、以下のエラーが返される場合があります。

```
lscpu: cannot open /proc/cpuinfo: No such file or directory
```

このエラーは重要ではないので、エラーを無視して構いません。

a. **resolv.conf** ファイルを一時的な場所に移動します。

```
# mv /etc/resolv.conf /etc/resolv.conf.bak
```

b. Red Hat コンテンツ配信ネットワークの DNS 要求を解決するために、一時的な **resolv.conf** ファイルを作成します。以下の例では、ネームサーバーに **8.8.8.8** を使用しています。

```
# echo "nameserver 8.8.8.8" > /etc/resolv.conf
```

c. マウントしたイメージを Red Hat コンテンツ配信ネットワークに登録します。

```
# subscription-manager register
```

コマンドにより要求されたら、ユーザー名およびパスワードを入力します。

d. Red Hat Enterprise Linux が含まれるサブスクリプションをアタッチします。

```
# subscription-manager list --all --available
# subscription-manager attach --pool <POOLID>
```

<POOLID> をサブスクリプションのプール ID に置き換えます。

e. デフォルトのリポジトリーを無効にします。

```
# subscription-manager repos --disable "*"
```

f. Red Hat Enterprise Linux リポジトリーを有効にします。

- Red Hat Enterprise Linux 7:

```
# subscription-manager repos --enable "rhel-7-server-rpms"
```

- Red Hat Enterprise Linux 8:

```
# subscription-manager repos --enable "rhel-8-for-x86_64-baseos-eus-rpms"
```

g. **iscsi-initiator-utils** パッケージをインストールします。

```
# yum install -y iscsi-initiator-utils
```

h. マウントしたイメージの登録を解除します。

```
# subscription-manager unregister
```

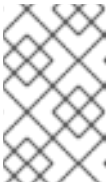
i. 元の **resolv.conf** ファイルを復元します。

```
# mv /etc/resolv.conf.bak /etc/resolv.conf
```

j. マウントされたイメージのカーネルバージョンを確認します。

```
# rpm -qa kernel
```

たとえば、出力が **kernel-3.10.0-1062.el7.x86_64** の場合、カーネルバージョンは **3.10.0-1062.el7.x86_64** になります。次のステップのために、このカーネルバージョンを書き留めておきます。



注記

新しい Red Hat Enterprise Linux QCOW2 イメージには、1 つのカーネルバージョンしかインストールされません。複数のカーネルバージョンがインストールされている場合は、最新のものを使用してください。

k. initramfs イメージに **network** および **iscsi** dracut モジュールを追加します。

```
# dracut --force --add "network iscsi" /boot/initramfs-<KERNELVERSION>.img
<KERNELVERSION>
```

<KERNELVERSION> を **rpm -qa kernel** から取得したバージョン番号に置き換えます。以下の例では、カーネルバージョンに **3.10.0-1062.el7.x86_64** を使用しています。

```
# dracut --force --add "network iscsi" /boot/initramfs-3.10.0-1062.el7.x86_64.img 3.10.0-
1062.el7.x86_64
```

l. マウントされたイメージからホストオペレーティングシステムに戻ります。

```
# exit
```

7. イメージをアンマウントします。

a. 一時的なマウントポイントから **dev** ディレクトリーをアンマウントします。

```
$ sudo umount /tmp/mountpoint/dev
```

b. マウントポイントからイメージをアンマウントします。

```
$ sudo umount /tmp/mountpoint
```

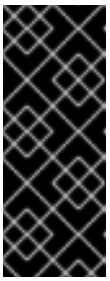
c. QCOW2 イメージを **/dev/nbd0/** から切断します。

```
$ sudo qemu-nbd --disconnect /dev/nbd0
```

8. イメージ上で **grub** メニュー設定を再ビルドします。

a. **libguestfs-tools** パッケージをインストールします。

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



重要

アンダークラウドに **libguestfs-tools** パッケージをインストールする場合は、アンダークラウドの **tripleo_iscsid** サービスとのポートの競合を避けるために **iscsid.socket** を無効にします。

```
$ sudo systemctl disable --now iscsid.socket
```

b. QEMU を直接使用するように **libguestfs** バックエンドを設定します。

```
$ export LIBGUESTFS_BACKEND=direct
```

c. イメージの grub 設定を更新し、ブートディスクに **rd.iscsi.firmware=1** カーネルパラメーターを設定します。

```
$ guestfish -a /tmp/images/{{ dib_image }} -m /dev/sda3 sh "mount /dev/sda2 /boot/efi &&
rm /boot/grub2/grubenv && /sbin/grub2-mkconfig -o /boot/grub2/grub.cfg && cp
/boot/grub2/grub.cfg /boot/efi/EFI/redhat/grub.cfg && grubby --update-kernel=ALL --
args=\"rd.iscsi.firmware=1\" && cp /boot/grub2/grubenv /boot/efi/EFI/redhat/grubenv &&
echo Success"
```

9. iSCSI 対応イメージをイメージサービスにアップロードします (glance):

```
$ openstack image create --disk-format qcow2 --container-format bare \
 --file <image> <image_name>
```

- **<image>** を iSCSI 対応イメージの名前 (例: **rhel-server-7.7-x86_64-kvm.qcow2**) に置き換えます。
- **** を、イメージの参照に使用する名前 (**rhel-server-7.7-iscsi** など) に置き換えます。

6.4. ブート可能ボリュームからのベアメタルインスタンスの作成

ベアメタルノードがブート可能ボリュームから作成されたベアメタルインスタンスをホストできることを確認するには、ブート可能ボリュームを作成し、インスタンスを起動します。

手順

1. オーバークラウド認証情報ファイルを入手します。

```
$ source ~/<credentials_file>
```

- **<credentials_file>** を認証情報ファイルの名前 (**overcloudrc** など) に置き換えます。

2. iSCSI 対応のインスタンスイメージからボリュームを作成します。

```
$ openstack volume create --size 10 --image  --bootable myBootableVolume
```

- **** を、ボリュームに書き込むイメージの名前または ID (**rhel-server-7.7-iscsi** など) に置き換えます。

3. ブートボリュームを使用するベアメタルインスタンスを作成します。

```
$ openstack server create --flavor baremetal --volume myBootableVolume --key default
myBareMetalInstance
```

第7章 BARE METAL PROVISIONING サービスのトラブルシューティング

Bare Metal Provisioning サービス (ironic) が含まれる環境内の問題を診断します。

7.1. PXE ブートエラー

PXE ブートで直面する問題を評価し、修正するには、以下のトラブルシューティング手順を使用します。

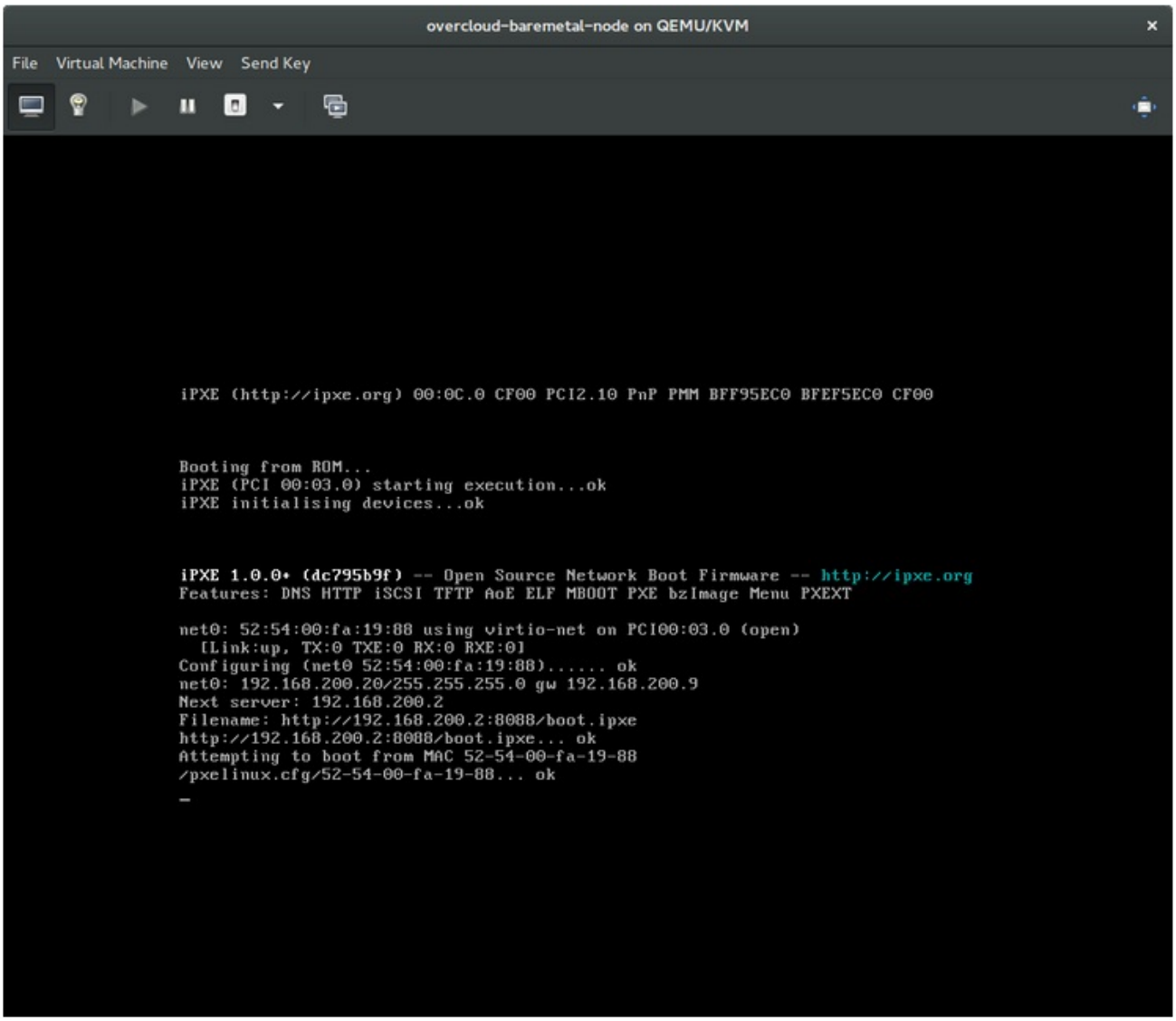
Permission Denied エラー

ベアメタル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、以下に示すように、必ず適切な SELinux コンテキストを **/httpboot** および **/tftpboot** ディレクトリーに適用してください。

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

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

ノードのコンソールで、IP アドレスは取得しているがプロセスが停止しているように表示されている場合は、**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ります。




```
$ 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** であるため、**i** を省略して、誤って **ipxe_config.template** と入力しがちです。

7.2. ベアメタルノードブート後のログインエラー

設定時に指定した root パスワードを使用してノードにログインできない場合、デプロイされたイメージにブートしていないことを示しています。**deploy-kernel/deploy-ramdisk** イメージにログインしており、システムが正しい イメージを読み込んで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 サービスノードの **/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 Provisioning サービスノードが使用する唯一のネットワークは、ベアメタルネットワークです。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 サービス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 kernel 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=ipmi 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 サービス内のデプロイカーネルイメージです。
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ネットワーク上にある必要があります。
ipmi	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 サービス内のデプロイ ramdisk イメージです。

/httpboot/pxelinux.cfg/MAC_ADDRESS と **ironic.conf** ファイルの間で値が一致していない場合:

1. **ironic.conf** ファイル内の値を更新します。
2. Bare Metal Provisioning サービスを再起動します。
3. ベアメタルインスタンスを再デプロイします。

7.3. デプロイされたノードでの BOOT-TO-DISK エラー

特定のハードウェアでは、デプロイされたノードで問題が発生し、デプロイメントの一環としての以降

のブート操作中にノードがディスクからブートできない可能性があります。これは通常、director がノードで要求する永続的なブート設定を BMC が反映しないために発生します。代わりに、ノードは PXE ターゲットからブートします。

この場合、ノードの BIOS でブート順序を更新する必要があります。HDD を最初のブートデバイスに設定し、次に PXE を非デフォルトのオプションに設定します。これにより、デフォルトではノードはディスクからブートし、必要に応じてイントロスペクションまたはデプロイメント時にネットワークからブートすることができます。

注記

このエラーは、LegacyBIOS ファームウェアを使用するほとんどのノードに該当します。

7.4. BARE METAL PROVISIONING サービスが正しいホスト名を受信しない

Bare Metal Provisioning サービスが正しいホスト名を受信しない場合は、**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、ベアメタルのサブネットを OpenStack Networking サービス内のルーターに接続します。この設定により、要求が meta-data エージェントに正しくルーティングされます。

7.5. BARE METAL PROVISIONING サービスのコマンド実行時に OPENSTACK IDENTITY サービスの認証情報が無効

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** を **http://IP:PORT** に設定します。

7.6. ハードウェアの登録

ノードの登録情報が間違っていると、登録したハードウェアで問題が発生する可能性があります。属性名と値を正しく入力してください。属性名を誤って入力すると、システムは属性をノードの詳細に追加しますが、無視されます。

openstack baremetal node set コマンドを使用して、ノードの情報を更新します。以下の例では、ノードの登録されたメモリー使用量を 2 GB に更新します。

```
$ openstack baremetal node set --property memory_mb=2048 NODE_UUID
```

7.7. IDRAC に関する問題のトラブルシューティング

Redfish 管理インターフェイスがブートデバイスの設定に失敗する

特定の iDRAC ファームウェアバージョンの **idrac-redfish** 管理インターフェイスを使用する場合、UEFI ブートのベアメタルサーバーでブートデバイスの設定を試みると、iDRAC は以下のエラーを返します。

```
Unable to Process the request because the value entered for the
parameter Continuous is not supported by the implementation.
```

この問題が発生した場合には、ノードの **driver-info** の **force_persistent_boot_device** パラメーターを **Never** に設定します。

```
openstack baremetal node set --driver-info force_persistent_boot_device=Never ${node_uuid}
```

電源オフ時のタイムアウト

一部のサーバーで、電源のオフに長時間を要し、タイムアウトする場合があります。デフォルトのリトライ回数は **6** で、その結果 30 秒でタイムアウトになります。タイムアウトの時間を 90 秒に増やすには、アンダークラウドの hieradata オーバーライドファイルで **ironic::agent::rpc_response_timeout** の値を **18** に設定して、**openstack undercloud install** コマンドを再実行します。

```
ironic::agent::rpc_response_timeout: 18
```

ベンダーパススルーのタイムアウト

ベンダーパススルーコマンドを実行するのに iDRAC が利用できない場合、これらのコマンドの実行に非常に長い時間がかかり、タイムアウトします。

```
openstack baremetal node passthru call --http-method GET \
aed58dca-1b25-409a-a32f-3a817d59e1e0 list_unfinished_jobs
Timed out waiting for a reply to message ID 547ce7995342418c99ef1ea4a0054572 (HTTP 500)
```

メッセージングのタイムアウト時間を増やすには、アンダークラウドの hieradata オーバーライドファイルで **ironic::default::rpc_response_timeout** パラメーターの値を増やし、**openstack undercloud install** コマンドを再実行します。

```
ironic::default::rpc_response_timeout: 600
```

7.8. サーバーコンソールの設定

オーバークラウドノードからのコンソール出力は、常にサーバーコンソールに送信される訳ではありません。サーバーコンソールでこの出力を表示するには、ハードウェアの正しいコンソールを使用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。この設定を行うには、以下のいずれかの方法を使用します。

- オーバークラウドロールごとに **KernelArgs** heat パラメーターを変更する
- director がオーバークラウドノードのプロビジョニングに使用する **overcloud-hardened-uefi-full.qcow2** イメージをカスタマイズします。

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、director のインストールと使用方法 を参照してください。
- デプロイ可能なオーバークラウドノード

デプロイメント時の heat を使用した KernelArgs の変更

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. 環境ファイル **overcloud-console.yaml** を作成して、以下の内容を記載します。

```
parameter_defaults:
  <role>Parameters:
    KernelArgs: "console=<console-name>"
```

<role> を設定するオーバークラウドロールの名前に置き換え、**<console-name>** を使用するコンソールの ID に置き換えます。たとえば、デフォルトロールのすべてのオーバークラウドノードが **tty0** を使用するように設定するには、以下のスニペットを使用します。

```
parameter_defaults:
  ControllerParameters:
    KernelArgs: "console=tty0"
  ComputeParameters:
    KernelArgs: "console=tty0"
  BlockStorageParameters:
    KernelArgs: "console=tty0"
  ObjectStorageParameters:
    KernelArgs: "console=tty0"
  CephStorageParameters:
    KernelArgs: "console=tty0"
```

4. **-e** オプションを使用して、**overcloud-console-tty0.yaml** ファイルをデプロイメントコマンドに追加します。

overcloud-hardened-uefi-full.qcow2 イメージの変更

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. **overcloud-hardened-uefi-full.qcow2** イメージのカーネル引数を変更して、ハードウェアの正しいコンソールを設定します。たとえば、コンソールを **tty1** に設定します。

```
$ virt-customize --selinux-relabel -a overcloud-hardened-uefi-full.qcow2 --run-command
'grubby --update-kernel=ALL --args="console=tty1"'
```

4. イメージを director にインポートします。

```
$ openstack overcloud image upload --image-path overcloud-hardened-uefi-full.qcow2
```

5. オーバークラウドをデプロイします。

検証

1. アンダークラウドからオーバークラウドノードにログインします。

```
$ ssh tripleo-admin@<IP-address>
```

<IP-address> をオーバークラウドノードの IP アドレスに置き換えます。

2. **/proc/cmdline** ファイルの内容を調べ、**console=** パラメーターが使用するコンソールの値に設定されていることを確認します。

```
[tripleo-admin@controller-0 ~]$ cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos2)/boot/vmlinuz-4.18.0-193.29.1.el8_2.x86_64
root=UUID=0ec3dea5-f293-4729-b676-5d38a611ce81 ro console=tty0
console=ttyS0,115200n81 no_timer_check crashkernel=auto rhgb quiet
```

第8章 BARE METAL のドライバー

ベアメタルノードが Bare Metal Provisioning サービスで有効にしたドライバーの 1 つを使用するように設定することができます。各ドライバーには、プロビジョニング方法と電源管理タイプが含まれます。ドライバーによっては追加の設定が必要な場合があります。このセクション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表示されます。

ironic.yaml ファイルの **IronicEnabledHardwareTypes** パラメーターを設定して、ドライバーを追加することができます。デフォルトでは、**ipmi** と **redfish** が有効になっています。

サポートされているプラグインとドライバーの全リストは、Component, Plug-In, and Driver Support in Red Hat OpenStack Platform のアーティクルを参照してください。

8.1. INTELLIGENT PLATFORM MANAGEMENT INTERFACE (IPMI) 電源管理ドライバー

IPMI 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。この電源管理タイプを使用するには、全 Bare Metal Provisioning サービスノードで IPMI が共有ベアメタルネットワークに接続されている必要があります。IPMI 電源管理ドライバーは、**ipmitool** ユーティリティーを使用して、ハードウェアをリモートから管理します。次の **driver_info** プロパティーを使用して、ノードの IPMI 電源管理ドライバーを設定できます。

表8.1 IPMI driver_info プロパティー

プロパティー	説明	同等の ipmitool オプション
ipmi_address	(必須) ノードの IP アドレスまたはホスト名。	**-H**
ipmi_username	IPMI ユーザー名。	**-U**
ipmi_password	IPMI パスワード。パスワードは一時ファイルに書き込まれます。**-f** オプションを使用して、ファイル名を **ipmitool** に渡します。	**-f**
ipmi_hex_kg_key	IPMIv2 認証用の 16 進数の Kg キー。	**-y**
ipmi_port	リモート IPMI RMCP ポート。	**-p**
ipmi_priv_level	IPMI 権限レベル。以下の有効な値のいずれかに設定します。 • **ADMINISTRATOR** (デフォルト) • **CALLBACK** • **OPERATOR** • **USER**	**-L**

プロパティー	説明	同等の ipmitool オプション
ipmi_protocol_version	IPMI プロトコルのバージョン。以下の有効な値のいずれかに設定します。 ● lan の場合は **1.5** ● lanplus の場合は **2.0** (デフォルト)	**-I**
ipmi_bridging	ブリッジのタイプ。ネストされたシャーシ管理コントローラー (CMC) で使用します。以下の有効な値のいずれかに設定します。 ● **single** ● **dual** ● **no** (デフォルト)	該当なし
ipmi_target_channel	ブリッジされたリクエストの宛先チャネル。**ipmi_bridging** が **single** または **dual** に設定されている場合のみ必要です。	**-b**
ipmi_target_address	ブリッジされたリクエストの宛先アドレス。**ipmi_bridging** が **single** または **dual** に設定されている場合のみ必要です。	**-t**
ipmi_transit_channel	ブリッジされたリクエストの転送チャネル。**ipmi_bridging** が **dual** に設定されている場合のみ必要です。	**-B**
ipmi_transit_address	ブリッジされたリクエストの転送アドレス。**ipmi_bridging** が **dual** に設定されている場合のみ必要です。	**-T**
ipmi_local_address	ブリッジされたリクエストのローカル IPMB アドレス。**ipmi_bridging** が **single** または **dual** に設定されている場合のみ使用します。	**-m**
ipmi_force_boot_device	サーバーの電源を入れるたびに、Bare Metal Provisioning サービスが BMC に起動デバイスを指定する必要があるかどうかを指定するには、**true** に設定します。BMC は、電源の再投入後、選択した起動デバイスを記憶できません。デフォルトでは無効になっています。	該当なし
ipmi_disable_boot_timeout	ノードでの起動の 60 秒のタイムアウトを無効にするために、raw IPMI コマンドを送信しない場合は、**false** に設定します。	該当なし

プロパティー	説明	同等の ipmitool オプション
ipmi_cipher_suite	ノードで使用する IPMI 暗号スイートのバージョン。以下の有効な値のいずれかに設定します。 ● SHA1 を使用する AES-128 の場合は **3** ● SHA256 を使用する AES-128 の場合は **17**	該当なし

8.2. REDFISH

Distributed Management Task Force (DMTF) によって開発された IT インフラストラクチャー用の標準 RESTful API。以下の **driver_info** プロパティーを使用して、Redfish への Bare Metal Provisioning サービス (ironic) 接続を設定できます。

表8.2 Redfish driver_info プロパティー

プロパティー	説明
redfish_address	(必須) Redfish コントローラーの IP アドレス。アドレスには、URL のオーソリティー部分を含める必要があります。スキームを含めない場合は、デフォルトで **https** になります。
redfish_system_id	Redfish ドライバーがやりとりするシステムリソースへの標準パス。パスには、ルートサービス、バージョン、および **redfish_address** プロパティーと同じオーソリティー内のシステムへの一意のパスが含まれている必要があります。たとえば、**/redfish/v1/Systems/CX34R87**。このプロパティーは、ターゲット BMC が複数のリソースを管理する場合のみ必要です。
redfish_username	Redfish ユーザー名。
redfish_password	Redfish パスワード。
redfish_verify_ca	ブール値、CA_BUNDLE ファイルへのパス、または信頼できる CA の証明書を含むディレクトリーのいずれかです。この値を **True** に設定すると、ドライバーはホストの証明書を検証します。この値を **False** に設定すると、ドライバーは SSL 証明書の検証を無視します。この値をパスに設定すると、ドライバーは指定された証明書またはディレクトリー内の証明書の 1 つを使用します。デフォルトは **True** です。
redfish_auth_type	Redfish HTTP クライアント認証方法。以下の有効な値のいずれかに設定します。 ● **basic** ● **session** (推奨) ● **auto** (デフォルト) - 利用可能な場合は **session** 認証方法を使用し、**session** 方法を利用できない場合は **basic** 認証方法を使用します。

8.3. 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。この電源管理タイプを使用するには、すべての Bare Metal Provisioning サービスノードに、共有 Bare Metal Provisioning ネットワークに接続された DRAC が必要です。**idrac** ドライバーを有効にし、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
- オプション: **drac_port** - WS-Management エンドポイントに使用するポート。デフォルトはポート **443** です。
- オプション: **drac_path** - WS-Management エンドポイントに使用するパス。デフォルトのパスは /**wsman** です。
- オプション: **drac_protocol** - WS-Management エンドポイントに使用するプロトコル。有効な値: **http**、**https**。デフォルトのプロトコルは **https** です。

8.4. INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC)

富士通の iRMC 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。Bare Metal Provisioning サービスノードでこの電源管理タイプを使用するには、このノードで iRMC インターフェイスが共有ベアメタルネットワークに接続されている必要があります。**irmc** ドライバーを有効にし、ノードの **driver_info** に以下の情報を設定します。

- **irmc_address**: iRMC インターフェイスの 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場合、または SCCI を使用してセンサーデータを取得する場合には、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。

```
# dnf install python-scciclient
```

3. Bare Metal Conductor 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。

8.5. 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。この電源管理タイプを使用するには、全ベアメタルノードで iLO インターフェイスが共有ベアメタルネットワークに接続されている必要があります。**ilo** ドライバーを有効にし、ノードの **driver_info** に以下の情報を設定します。

- **ilo_address**: iLO インターフェイスの 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 Conductor サービスを再起動する必要があります。

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