Red Hat OpenStack Platform 17.1

インスタンス作成のための Compute サービスの設定

インスタンスを作成するための Red Hat OpenStack Platform Compute サービス (nova) の設定と管理

Last Updated: 2024-04-20

Red Hat OpenStack Platform 17.1 インスタンス作成のための Compute サービスの設定

インスタンスを作成するための Red Hat OpenStack Platform Compute サービス (nova) の設定と管理

OpenStack Team
rhos-docs@redhat.com

法律上の通知

概要

本ガイドでは、クラウド管理者が OpenStack クライアント CLI を使用して Red Hat OpenStack Platform Compute (nova) サービスを設定および管理するための概念および手順について説明します。

目次

多様性を受け入れるオープンソースの強化 4

RED HAT ドキュメントへのフィードバック (英語のみ) 5

第1章 COMPUTE サービス (NOVA) の機能 6

第2章 COMPUTE サービス (NOVA) の設定 8

第3章 インスタンス起動用のフレーバーの作成 10
3.1. フレーバーの作成 10
3.2. フレーバーの引数 12
3.3. フレーバーのメタデータ 13

第4章 コンピュートノードで CPU を設定する 30
4.1. コンピュートノードでの CPU ピニングの設定 30
4.2. エミュレータースレッドの設定 39
4.3. インスタンスの CPU 機能フラグの設定 40

第5章 コンピュートノードでメモリーを設定する 43
5.1. オーバーコミットのためのメモリー設定 43
5.2. コンピュートノード上で確保するホストメモリーの計算 44
5.3. スワップサイズの計算 44
5.4. コンピュートノードでの HUGE PAGE の設定 45
5.5. インスタンスにファイルベースのメモリーを使用するコンピュートノードの設定 50
5.6. インスタンスのメモリーを暗号化するための AMD SEV コンピュートノードの設定 51

第6章 COMPUTE サービスのストレージの設定 60
6.1. イメージキャッシュの設定オプション 60
6.2. インスタンスの一時ストレージ属性の設定オプション 62
6.3. 1 つのインスタンスにアタッチすることのできる最大ストレージデバイス数の設定 65
6.4. 共有インスタンスストレージの設定 67
6.5. RED HAT CEPH RADOS BLOCK DEVICE (RBD) からの直接イメージダウンロードの設定 68
6.6. 関連情報 69

第7章 インスタンスのスケジューリングと配置の設定 70
7.1. PLACEMENT サービスを使用した事前絞り込み 70
7.2. COMPUTE スケジューラーサービス用フィルターおよび重みの設定 76
7.3. COMPUTE スケジューラーのフィルター 78
7.4. COMPUTE スケジューラーの重み 83
7.5. カスタム特性とリソースクラスの宣言 91
7.6. ホストアグリゲートの作成と管理 94

第8章 PCI パススルーの設定 101
8.1. PCI パススルー用コンピュートノードの指定 101
8.2. PCI パススルー用コンピュートノードの設定 104
8.3. PCI パススルーデバイス種別フィールド 107
8.4. NOVAPCIPASSTHROUGH 設定のガイドライン 107

第9章 VDPA ポートを使用するインスタンスを有効にするための VDPA コンピュートノードの設定 110
9.1. VDPA 用コンピュートノードの指定 110
9.2. VDPA コンピュートノードの設定 113

第10章 インスタンス用の仮想 GPU の設定 116
10.1. サポートされる設定および制限 116

10.2. コンピュートノードでの仮想 GPU の設定 117
10.3. カスタム VGPU リソースプロバイダー特性の作成 122
10.4. カスタム GPU インスタンスイメージの作成 123
10.5. インスタンス用の仮想 GPU フレーバーの作成 124
10.6. 仮想 GPU インスタンスの起動 124
10.7. GPU デバイスの PCI パススルーの有効化 125

第11章 インスタンスへのメタデータの追加 130
11.1. インスタンスメタデータの種別 130
11.2. 全インスタンスへのコンフィグドライブの追加 130
11.3. インスタンスへの動的メタデータの追加 132

第12章 KERNELARGS を定義するための手動でのノード再起動の設定 134
12.1. KERNELARGS を定義するための手動でのノード再起動の設定 134

第13章 インスタンスのセキュリティーを設定する 136
13.1. インスタンスの VNC コンソールへの接続のセキュリティー保護 136
13.2. エミュレートされた TRUSTED PLATFORM MODULE (TPM) デバイスをインスタンスに提供するためのコンピュートノードの設定 137

第14章 データベースのクリーニング 141
14.1. データベース管理の設定 141
14.2. COMPUTE サービスのデータベース自動管理用設定オプション 141

第15章 コンピュートノード間の仮想マシンインスタンスの移行 145
15.1. 移行の種別 145
15.2. 移行の制約 147
15.3. 移行の準備 149
15.4. インスタンスのコールドマイグレーション 149
15.5. インスタンスのライブマイグレーション 150
15.6. 移行ステータスの確認 151
15.7. インスタンスの退避 153
15.8. 移行に関するトラブルシューティング 155

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。

Jira でドキュメントのフィードバックを提供する

ドキュメントに関するフィードバックを提供するには、Create Issue フォームを使用します。Red Hat OpenStack Platform Jira プロジェクトで Jira Issue が作成され、フィードバックの進行状況を追跡できます。

1. Jira にログインしていることを確認してください。Jira アカウントをお持ちでない場合は、アカウントを作成してフィードバックを送信してください。
2. Create Issue をクリックして、**Create Issue** ページを開きます。
3. **Summary** フィールドと **Description** フィールドに入力します。**Description** フィールドに、ドキュメントの URL、章またはセクション番号、および問題の詳しい説明を入力します。フォーム内の他のフィールドは変更しないでください。
4. **Create** をクリックします。

第1章 COMPUTE サービス (NOVA) の機能

Compute (nova) サービスを使用して、Red Hat OpenStack Platform (RHOSP) 環境で仮想マシンインスタンスおよびベアメタルサーバーを作成、プロビジョニング、および管理します。Compute サービスは、ベースとなるホストプラットフォームの詳細を公開するのではなく、Compute サービスを実行しているベースとなるハードウェアを抽象化します。たとえば、ホスト上で動作中の CPU の種別およびトポロジーを公開するのではなく、Compute サービスは多数の仮想 CPU (vCPU) を公開し、これらの仮想 CPU のオーバーコミットに対応します。

Compute サービスは、KVM ハイパーバイザーを使用して Compute サービスのワークロードを実行します。libvirt ドライバーは QEMU と協調して KVM との相互のやり取りをすべて処理し、仮想マシンインスタンスの作成を可能にします。インスタンスを作成およびプロビジョニングするために、Compute サービスは以下の RHOSP サービスと協調します。

- 認証のための Identity (keystone) サービス
- リソースインベントリーをトラッキングおよび選択するための Placement サービス
- ディスクおよびインスタンスイメージのための Image サービス (glance)
- インスタンスがブート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するための Networking (neutron) サービス

Compute サービスは、**nova-*** という名前のデーモンプロセスおよびサービスで構成されます。コアの Compute サービスを以下に示します。

Compute サービス (nova-compute)

このサービスは、KVM または QEMU ハイパーバイザー API の libvirt を使用してインスタンスを作成、管理、および削除し、インスタンスの状態でデータベースを更新します。

Compute コンダクター (nova-conductor)

このサービスは、Compute サービスとデータベースの協調を仲介します。これにより、コンピュートノードをデータベースへの直接アクセスから隔離します。このサービスを **nova-compute** サービスが実行されているノードにデプロイしないでください。

Compute スケジューラー (nova-scheduler)

このサービスはキューからインスタンスの要求を受け取り、インスタンスをホストするコンピュートノードを決定します。

Compute API (nova-api)

このサービスは、ユーザーに外部 REST API を提供します。

API データベース

このデータベースはインスタンスの場所の情報をトラッキングします。また、ビルドされているがスケジュールされていないインスタンスの一時的な場所を提供します。マルチセルのデプロイメントでは、このデータベースには各セルのデータベース接続を指定するセルのマッピングも含まれます。

セルデータベース

このデータベースには、インスタンスに関するほとんどの情報が含まれます。API データベース、コンダクター、および Compute サービスによって使用されます。

メッセージキュー

このメッセージングサービスは、セル内の各サービス間の通信およびグローバルなサービスとの通信のために、すべてのサービスによって使用されます。

Compute メタデータ

このサービスは、インスタンス固有のデータを保管します。インスタンス

は、http://169.254.169.254 または IPv6 を通じてリンクローカルアドレス 80::a9fe:a9fe から、メタデータサービスにアクセスします。Networking (neutron) サービスは、要求をメタデータ API サーバーに転送するロールを持ちます。**NeutronMetadataProxySharedSecret** パラメーターを使用して、Networking サービスと Compute サービス両方の設定でシークレットキーワードを設定する必要があります。これにより、これらのサービスが通信を行うことができます。Compute メタデータサービスは、Compute API の一部としてグローバルに実行することや、それぞれのセルで実行することができます。

複数のコンピュートノードをデプロイすることができます。インスタンスを操作するハイパーバイザーは、それぞれのコンピュートノードで実行されます。それぞれのコンピュートノードには、少なくとも 2 つのネットワークインターフェイスが必要です。コンピュートノードでは、インスタンスを仮想ネットワークに接続し、セキュリティーグループを介してインスタンスにファイアウォールサービスを提供する Networking サービスエージェントも実行されます。

デフォルトでは、director はすべてのコンピュートノードを単一のセルとしてオーバークラウドをインストールします。このセルには、仮想マシンインスタンスを制御および管理するすべての Compute サービスおよびデータベース、ならびにすべてのインスタンスおよびインスタンスのメタデータが含まれます。大規模なデプロイメントでは、複数のセルでオーバークラウドをデプロイし、多数のコンピュートノードに対応することができます。新しいオーバークラウドをインストールする際に、またはその後の任意の時に、セルを環境に追加することができます。

第2章 COMPUTE サービス (NOVA) の設定

クラウド管理者は、環境ファイルを使用して Compute (nova) サービスをカスタマイズします。Puppet は、この設定を生成して **/var/lib/config-data/puppet-generated/<nova_container>/etc/nova/nova.conf** ファイルに保存します。Compute サービスの設定をカスタマイズするには、以下の設定方法を順番どおりに使用します。

1. **heat パラメーター** - **オーバークラウドパラメーター** ガイドの Compute (nova) パラメーターセクションに詳細が記載されています。以下の例では、heat パラメーターを使用して、デフォルトのスケジューラーフィルターを設定し、Compute サービスの NFS バックエンドを設定します。

```
parameter_defaults:
  NovaNfsEnabled: true
  NovaNfsOptions: "context=system_u:object_r:nfs_t:s0"
  NovaNfsShare: "192.0.2.254:/export/nova"
  NovaNfsVersion: "4.2"
  NovaSchedulerEnabledFilters:
    - AggregateInstanceExtraSpecsFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
```

2. **Puppet パラメーター**: **/etc/puppet/modules/nova/manifests/*** で定義されます。

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::force_raw_images: True
```

> **注記**
>
> この方法は、等価な heat パラメーターが存在しない場合にのみ使用してください。

3. **手動での hieradata の上書き**: heat パラメーターまたは Puppet パラメーターが存在しない場合の、パラメーターカスタマイズ用。たとえば、以下の設定により Compute ロールの **[DEFAULT]** セクションに **timeout_nbd** が定義されます。

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      DEFAULT/timeout_nbd:
        value: '20'
```

警告

heat パラメーターが存在する場合は、Puppet パラメーターではなく heat パラメーターを使用します。Puppet パラメーターは存在するが heat パラメーターは存在しない場合は、手動で上書きする方法ではなく Puppet パラメーターを使用します。等価な heat パラメーターまたは Puppet パラメーターがない場合に限り、手動で上書きする方法を使用してください。

ヒント

変更するパラメーター の特定のガイダンスに従って、特定の設定をカスタマイズする ために、heat または Puppet パラメーターを使用できるかどうかを判断します。

オーバークラウドサービスの設定方法の詳細は、**Red Hat OpenStack Platform デプロイメントのカスタマイズ** ガイドの Heat パラメーター を参照してください。

第3章 インスタンス起動用のフレーバーの作成

インスタンスのフレーバーは、インスタンス用の仮想ハードウェアプロファイルを指定するリソースのテンプレートです。クラウドユーザーは、インスタンスを起動する際にフレーバーを指定する必要があります。

フレーバーにより、Compute サービスがインスタンスに割り当てる必要のある以下のリソースの量を指定することができます。

- 仮想 CPU の数
- RAM (MB 単位)
- ルートディスク (GB 単位)
- セカンダリー一時ストレージおよびスワップディスクを含む仮想ストレージ

フレーバーを全プロジェクトに公開したり、特定のプロジェクトまたはドメインを対象にプライベートに設定したりすることで、フレーバーを使用できるユーザーを指定することができます。

フレーバーでは、メタデータ (追加スペックとも呼ばれる) を使用して、インスタンス用ハードウェアのサポートおよびクォータを指定することができます。フレーバーのメタデータは、インスタンスの配置、リソースの使用上限、およびパフォーマンスに影響を及ぼします。利用可能なメタデータ属性の完全なリストは、Flavor metadata を参照してください。

フレーバーメタデータキーを使用することで、ホストアグリゲートで設定した **extra_specs** メタデータを照合して、インスタンスをホストするのに適したホストアグリゲートを探すこともできます。ホストアグリゲートでインスタンスをスケジュールするには、**extra_specs** キーに **aggregate_instance_extra_specs:** 名前空間の接頭辞を指定して、フレーバーのメタデータのスコープを定義する必要があります。詳細は、Creating and managing host aggregates を参照してください。

Red Hat OpenStack Platform (RHOSP) のデプロイメントには、クラウドユーザーが使用可能な以下のデフォルトパブリックフレーバーのセットが含まれています。

表3.1 デフォルトのフレーバー

名前	仮想 CPU の数	メモリー	ルートディスクのサイズ
m1.nano	1	128 MB	1 GB
m1.micro	1	192 MB	1 GB

注記

フレーバー属性を使用して設定した動作は、イメージを使用して設定した動作よりも優先されます。クラウドユーザーがインスタンスを起動する際、指定したフレーバーの属性がイメージの属性よりも優先されます。

3.1. フレーバーの作成

特定の機能や動作のために特化したフレーバーを作成および管理することができます。以下に例を示します。

- 基になるハードウェアの要件に応じて、デフォルトのメモリーと容量を変更する

- インスタンスに特定の I/O レートを強制するためのメタデータ、またはホストアグリゲートと一致させるためのメターデータを追加する

手順

1. インスタンスが利用可能な基本的なリソースを指定するフレーバーを作成します。

```
(overcloud)$ openstack flavor create --ram <size_mb> \
 --disk <size_gb> --vcpus <no_vcpus> \
 [--private --project <project_id>] <flavor_name>
```

- **<size_mb>** を、このフレーバーで作成するインスタンスに割り当てる RAM の容量に置き換えます。
- **<size_gb>** を、このフレーバーで作成するインスタンスに割り当てるルートディスクのサイズに置き換えます。
- **<no_vcpus>** を、このフレーバーで作成するインスタンスに確保する仮想 CPU の数に置き換えます。
- (オプション) **--private** および **--project** オプションを指定して、特定のプロジェクトまたはユーザーグループだけがフレーバーにアクセスできるようにします。**<project_id>** を、このフレーバーを使用してインスタンスを作成できるプロジェクトの ID に置き換えます。アクセシビリティーを指定しない場合は、フレーバーはデフォルトで public に設定されます。つまり、すべてのプロジェクトで使用可能になります。

注記

パブリックフレーバーの作成後は、そのフレーバーをプライベートにすることはできません。

- **<flavor_name>** を、一意のフレーバー名に置き換えます。
フレーバーの引数についての詳細は、Flavor arguments を参照してください。

2. (オプション) フレーバーのメタデータを指定するには、キー/値のペアを使用して必要な属性を設定します。

```
(overcloud)$ openstack flavor set \
 --property <key=value> --property <key=value> ... <flavor_name>
```

- **<key>** を、このフレーバーで作成するインスタンスに割り当てる属性のメタデータキーに置き換えます。利用可能なメタデータキーのリストは、Flavor metadata を参照してください。
- **<value>** を、このフレーバーで作成するインスタンスに割り当てるメタデータキーの値に置き換えます。
- **<flavor_name>** を、フレーバーの名前に置き換えます。
たとえば、以下のフレーバーを使用して起動されるインスタンスはに 2 つの CPU ソケットがあり、それぞれに 2 つの CPU が割り当てられます。

```
(overcloud)$ openstack flavor set \
 --property hw:cpu_sockets=2 \
 --property hw:cpu_cores=2 processor_topology_flavor
```

3.2. フレーバーの引数

openstack flavor create コマンドには、新しいフレーバーの名前を指定する 1 つの位置引数 **<flavor_name>** を使用することができます。

以下の表には、新規フレーバーを作成する際に必要に応じて指定することのできる、オプションの引数の詳細をまとめています。

表3.2 オプションのフレーバー引数

オプションの引数	説明
--id	一意のフレーバー ID。デフォルト値は **auto** で、UUID4 値を生成します。この引数を使用して、整数値を手動で指定することや、UUID4 値を生成することができます。
--ram	(必須) インスタンスが利用可能なメモリーの容量 (MB 単位) デフォルト: 256 MB
--disk	(必須) ルート (/) パーティションに使用するディスク容量 (GB 単位)。ルートディスクは、ベースイメージがコピーされる一時ディスクです。インスタンスが永続ボリュームからブートする場合、ルートディスクは使用されません。 **注記** **--disk** が **0** に設定されたフレーバーでインスタンスを作成する場合、インスタンスはボリュームからブートする必要があります。 デフォルト: 0 GB
--ephemeral	一時ディスクに使用するディスク容量 (GB 単位)デフォルト値は 0 GB で、セカンダリー一時ディスクは作成されません。一時ディスクは、インスタンスのライフサイクルにリンクしたマシンのローカルディスクストレージを提供します。一時ディスクは、いずれのスナップショットにも含まれません。インスタンスが削除されると、このディスクは破棄されすべてのデータが失われます。 デフォルト: 0 GB
--swap	スワップディスクのサイズ (MB 単位)Compute サービスのバックエンドストレージがローカルストレージでない場合は、フレーバーに **swap** を指定しないでください。 デフォルト: 0 GB
--vcpus	(必須) インスタンス用の仮想 CPU の数 デフォルト: 1

オプションの引数	説明
--public	フレーバーは、すべてのプロジェクトで利用可能です。デフォルトでは、フレーバーはパブリックですべてのプロジェクトで利用することができます。
--private	フレーバーは、**--project** オプションで指定したプロジェクトでのみ利用可能です。プライベートのフレーバーを作成し、フレーバーにプロジェクトを追加しない場合、クラウド管理者だけがそのフレーバーを利用することができます。
--property	以下の形式のキー/値のペアで指定されるメタデータまたは追加スペック **--property <key=value>** 複数の属性を設定するには、このオプションを繰り返します。
--project	プライベートのフレーバーを使用することができるプロジェクトを指定します。この引数は、**--private** オプションと共に使用する必要があります。プロジェクトを指定しないと、フレーバーは **admin** ユーザーだけに表示されます。 複数のプロジェクトにアクセスを許可するには、このオプションを繰り返します。
--project-domain	プライベートのフレーバーを使用することができるプロジェクトドメインを指定します。この引数は、**--private** オプションと共に使用する必要があります。 複数のプロジェクトドメインにアクセスを許可するには、このオプションを繰り返します。
--description	フレーバーの説明。長さは 65535 文字に制限されます。使用できるのは印刷可能文字だけです。

3.3. フレーバーのメタデータ

フレーバーを作成する際に、**--property** オプションを使用してフレーバーのメタデータを指定します。フレーバーのメタデータは **追加スペック** とも呼ばれます。フレーバーのメタデータで指定するインスタンス用ハードウェアのサポートおよびクォータは、インスタンスの配置、インスタンスの制限、およびパフォーマンスに影響を及ぼします。

インスタンスによるリソースの使用

以下の表に示す属性キーを使用して、インスタンスによる CPU、メモリー、およびディスク I/O の使用に制限を設定します。

注記

インスタンスの CPU リソース使用量を制限するための追加の仕様は、libvirt に直接渡されるホスト固有の調整可能なプロパティーであり、libvirt は制限をホスト OS に渡します。したがって、サポートされるインスタンスの CPU リソース制限の設定は、基盤となるホスト OS によって異なります。

RHOSP デプロイメントでコンピュートノードのインスタンス CPU リソース使用を設定する方法の詳細については、RHEL 9 ドキュメントの cgroup について および Libvirt ドキュメントの CPU チューニング を参照してください。

表3.3 リソースの使用を制限するためのフレーバーメタデータ

キー	説明
quota:cpu_shares	ドメイン内の CPU 時間の配分に使用する重みを指定します。デフォルトは OS が提供するデフォルト値です。Compute スケジューラーは、この属性の設定と同じドメイン内にある他のインスタンスの設定を比較して、相対的な重み付けを行います。たとえば、設定が **quota:cpu_shares=2048** のインスタンスには、設定が **quota:cpu_shares=1024** のインスタンスの 2 倍の CPU 時間が割り当てられます。
quota:cpu_period	**cpu_quota** を適用する期間を指定します (マイクロ秒単位)。この **cpu_period** の期間、各仮想 CPU は **cpu_quota** を超えるランタイムを使用することはできません。1000 - 1000000 の範囲で値を設定します。無効にするには **0** に設定します。

quota:cpu_quota	各 **cpu_period** における仮想 CPU の最大許容帯域幅を指定します (マイクロ秒単位)。
	● 1000 - 18446744073709551 の範囲で値を設定します。
	● 無効にするには **0** に設定します。
	● 帯域幅に制限を設けない場合は、負の値に設定します。
	cpu_quota および **cpu_period** を使用して、全仮想 CPU が同じ速度で実行されるようにすることができます。たとえば、以下のフレーバーを使用して、物理 CPU の処理能力の最大 50% しか消費できないインスタンスを起動することができます。
	`$ openstack flavor set cpu_limits_flavor \`
	`  --property quota:cpu_quota=10000 \`
	`  --property quota:cpu_period=20000`

インスタンスディスクのチューニング

以下の表に示す属性キーを使用して、インスタンスのディスクのパフォーマンスをチューニングします。

注記

Compute サービスは、以下の QoS 設定を、Compute サービスがプロビジョニングしたストレージ (一時ストレージなど) に適用します。Block Storage (cinder) ボリュームのパフォーマンスを調整するには、ボリュームタイプのサービス品質 (QoS) 仕様を設定して関連付ける必要もあります。詳細は、**永続ストレージの設定** ガイドの Block Storage サービス (cinder) のサービス品質仕様 を参照してください。

表3.4 ディスクチューニング用のフレーバーメタデータ

キー	説明
quota:disk_read_bytes_sec	インスタンスが利用可能な最大ディスク読み取りを指定します (バイト毎秒単位)。
quota:disk_read_iops_sec	インスタンスが利用可能な最大ディスク読み取りを指定します (IOPS 単位)。
quota:disk_write_bytes_sec	インスタンスが利用可能な最大ディスク書き込みを指定します (バイト毎秒単位)。
quota:disk_write_iops_sec	インスタンスが利用可能な最大ディスク書き込みを指定します (IOPS 単位)。
quota:disk_total_bytes_sec	インスタンスが利用可能な最大 I/O 操作を指定します (バイト毎秒単位)。
quota:disk_total_iops_sec	インスタンスが利用可能な最大 I/O 操作を指定します (IOPS 単位)。

インスタンスのネットワークトラフィックの帯域幅

以下の表に示す属性キーを使用して、VIF I/O オプションの設定により、インスタンスのネットワークトラフィックの帯域幅上限を設定します。

注記

quota :vif_* 属性は非推奨になりました。この属性の代わりに、Networking (neutron) サービスの Quality of Service (QoS) ポリシーを使用する必要があります。QoS ポリシーの詳細は、**Red Hat OpenStack Platform ネットワークの設定** ガイドの サービス品質 (QoS) ポリシーの設定 を参照してください。**quota:vif_*** プロパティーは、**NeutronOVSFirewallDriver** が **iptables_hybrid** に設定されている ML2/OVS メカニズムドライバーを使用する場合にのみサポートされます。

表3.5 帯域幅を制限するためのフレーバーメタデータ

キー	説明
quota:vif_inbound_average	(非推奨) インスタンスに送付されるトラフィックに要求される平均ビットレートを指定します (kbps 単位)。

キー	説明
quota:vif_inbound_burst	(非推奨) ピークの速度でバースト処理することができる受信トラフィックの最大量を指定します (KB 単位)。
quota:vif_inbound_peak	(非推奨) インスタンスが受信することのできるトラフィックの最大レートを指定します (kbps 単位)。
quota:vif_outbound_average	(非推奨) インスタンスから送信されるトラフィックに要求される平均ビットレートを指定します (kbps 単位)。
quota:vif_outbound_burst	(非推奨) ピークの速度でバースト処理することができる送信トラフィックの最大量を指定します (KB 単位)。
quota:vif_outbound_peak	(非推奨) インスタンスが送信することのできるトラフィックの最大レートを指定します (kbps 単位)。

ハードウェアビデオ RAM

以下の表に示す属性キーを使用して、ビデオデバイスに使用するインスタンス RAM の上限を設定します。

表3.6 ビデオデバイス用のフレーバーメタデータ

キー	説明
hw_video:ram_max_mb	ビデオデバイスに使用する最大 RAM を指定します (MB 単位)。**hw_video_ram** イメージ属性で使用します。**hw_video_ram** は **hw_video:ram_max_mb** 以下でなければなりません。

ウォッチドッグの動作

以下の表に示す属性キーを使用して、インスタンスで仮想ハードウェアのウォッチドッグデバイスを有効にします。

表3.7 ウォッチドッグの動作を設定するためのフレーバーメタデータ

キー	説明

キー	説明
hw:watchdog_action	仮想ハードウェアのウォッチドッグデバイスを有効にするかどうかを指定し、その動作を設定します。インスタンスがハングアップまたはエラーが発生した場合に、ウォッチドッグデバイスは設定されたアクションを実行します。ウォッチドッグは i6300esb デバイスを使用し、PCI Intel 6300ESB をエミュレートします。**hw:watchdog_action** が指定されていない場合には、ウォッチドッグは無効になります。 以下の有効な値のいずれかに設定します。 ● **disabled**: (デフォルト) デバイスは接続されません。 ● **reset**: インスタンスを強制的にリセットします。 ● **poweroff**: インスタンスを強制的にシャットダウンします。 ● **pause**: インスタンスを一時停止します。 ● **none**: ウォッチドッグを有効にしますが、インスタンスがハングアップまたはエラーが発生しても何も実行しません。 **注記** 特定のイメージの属性を使用して設定するウォッチドッグの動作は、フレーバーを使用して設定する動作よりも優先されます。

乱数ジェネレーター (RNG)

以下の表に示す属性キーを使用して、インスタンスで RNG デバイスを有効にします。

表3.8 RNG 用のフレーバーメタデータ

キー	説明
hw_rng:allowed	イメージ属性によりインスタンスに追加された RNG デバイスを無効にするには、**False** に設定します。 デフォルト: **True**
hw_rng:rate_bytes	期間中、インスタンスがホストのエントロピーから読み取ることのできる最大バイト数を指定します。
hw_rng:rate_period	読み取り期間を指定します (ミリ秒単位)。

仮想パフォーマンス監視ユニット (vPMU)

以下の表に示す属性キーを使用して、インスタンスの仮想 PMU を有効にします。

表3.9 仮想 PMU 用のフレーバーメタデータ

キー	説明
hw:pmu	インスタンスの仮想 PMU を有効にするには、**True** に設定します。 **perf** 等のツールは、インスタンスの仮想 PMU を使用して、インスタンスのパフォーマンスをプロファイリングおよびモニタリングするためのより正確な情報を提供します。リアルタイム負荷のケースでは、仮想 PMU のエミュレーションにより、望ましくないレイテンシーが付加される場合があります。テレメトリーの提供が不要な場合は、**hw:pmu=False** に設定します。

Virtual Trusted Platform Module (vTPM) デバイス

次の表のプロパティーキーを使用して、インスタンスの vTPM デバイスを有効にします。

表3.10 vTPM 用のフレーバーメタデータ

キー	説明
hw:tpm_version	使用する TPM のバージョンを設定します。TPM バージョン **2.0** が唯一サポートされているバージョンです。
hw:tpm_model	使用する TPM デバイスのモデルに設定します。**hw:tpm_version** が設定されていない場合は無視されます。以下の有効な値のいずれかに設定します。 ● **tpm-tis**: (デフォルト) TPM インターフェイス仕様。 ● **tpm-crb**: コマンド応答バッファー。TPM バージョン 2.0 とのみ互換性があります。

インスタンスの CPU トポロジー

以下の表に示す属性キーを使用して、インスタンス内のプロセッサーのトポロジーを定義します。

表3.11 CPU トポロジー用のフレーバーメタデータ

キー	説明
hw:cpu_sockets	インスタンスの推奨ソケット数を指定します。 デフォルト: 要求される仮想 CPU の数
hw:cpu_cores	インスタンスの 1 ソケットあたりの推奨コア数を指定します。 デフォルト: **1**

キー	説明
hw:cpu_threads	インスタンスの1コアあたりの推奨スレッド数を指定します。 デフォルト: **1**
hw:cpu_max_sockets	イメージ属性を使用してユーザーがインスタンスに選択できる最大ソケット数を指定します。 例: **hw:cpu_max_sockets=2**
hw:cpu_max_cores	イメージ属性を使用してユーザーがインスタンスに選択できる、1ソケットあたりの最大コア数を指定します。
hw:cpu_max_threads	イメージ属性を使用してユーザーがインスタンスに選択できる、1コアあたりの最大スレッド数を指定します。

シリアルポート

以下の表に示す属性キーを使用して、1インスタンスあたりのシリアルポートの数を設定します。

表3.12 シリアルポート用のフレーバーメタデータ

キー	説明
hw:serial_port_count	1インスタンスあたりの最大シリアルポート数

CPU ピニングポリシー

デフォルトでは、インスタンスの仮想 CPU (vCPU) は1コア1スレッドのソケットです。属性を使用して、インスタンスの仮想 CPU をホストの物理 CPU コア (pCPU) に固定するフレーバーを作成することができます。1つまたは複数のコアがスレッドシブリングを持つ同時マルチスレッド (SMT) アーキテクチャーで、ハードウェア CPU スレッドの動作を設定することもできます。

以下の表に示す属性キーを使用して、インスタンスの CPU ピニングポリシーを定義します。

表3.13 CPU ピニング用のフレーバーメタデータ

キー	説明

キー	説明
hw:cpu_policy	使用する CPU ポリシーを指定します。以下の有効な値のいずれかに設定します。 ● **shared**: (デフォルト) インスタンスの仮想 CPU は、ホストの物理 CPU 全体で共有されます。 ● **dedicated**: インスタンスの仮想 CPU をホストの物理 CPU のセットに固定します。これにより、インスタンスが固定される CPU のトポロジーに一致するインスタンス CPU のトポロジーが作成されます。このオプションの場合、必然的にオーバーコミット比は 1.0 になります。 ● **mixed**: インスタンス vCPU は、専用 (ピニングされた) ホスト pCPU と共有 (ピニングされていない) ホスト pCPU を組み合わせて使用します。

キー	説明

hw:cpu_thread_policy	設定が **hw:cpu_policy=dedicated** の場合に使用する CPU スレッドポリシーを指定します。以下の有効な値のいずれかに設定します。
	
	● **prefer**: (デフォルト) ホストは SMT アーキテクチャーを持つ場合と持たない場合があります。SMT アーキテクチャーが存在する場合、Compute スケジューラーはスレッドシブリングを優先します。
	
	● **isolate**: ホストは SMT アーキテクチャーを持たないか、あるいは SMT 以外のアーキテクチャーをエミュレートする必要があります。このポリシーにより、Compute スケジューラーは **HW_CPU_HYPERTHREADING** 特性が設定されていないホストを要求して、SMT を持たないホストにインスタンスを配置するようになります。以下の属性を使用して、この特性を明示的に要求することもできます。
	
	`--property trait:HW_CPU_HYPERTHREADING=forbidden`
	
	ホストが SMT アーキテクチャーを持たない場合、Compute サービスはそれぞれの仮想 CPU を想定どおりに異なるコアに配置します。ホストが SMT アーキテクチャーを持つ場合は、動作は **[workarounds]/disable_fallback_pcpu_query** パラメーターの設定により決定されます。
	
	○ **True**: SMT アーキテクチャーを持つホストは使用されず、スケジューリングに失敗します。
	
	○ **False**: Compute サービスはそれぞれの仮想 CPU を異なる物理コアに配置します。Compute サービスは、別のインスタンスからの仮想 CPU を同じコア上に配置しません。したがって、使用される各コアのスレッドシブリングは、1 つを除きすべて使用できなくなります。
	
	● **require**: ホストは SMT アーキテクチャーを持つ必要があります。このポリシーにより、Compute スケジューラーは **HW_CPU_HYPERTHREADING** 特性が設定されたホストを要求して、SMT を持つホストにインスタンスを配置するようになります。以下の属性を使用して、この特性を明示的に要求することもできます。
	
	`--property trait:HW_CPU_HYPERTHREADING=required`
	
	Compute サービスは、それぞれの仮想 CPU をスレッドシブリングに割り当てます。ホストが SMT アーキテクチャーを持たない場合、そのホストは使用されません。ホストは SMT アーキテクチャーを持つが、スレッドシブリングが使用されていないコアが十分にない場合、スケジューリングに失敗します。

キー	説明
hw:cpu_dedicated_mask	どの CPU が専用 (ピニングされた) CPU か共有 (ピニングされていない/フローティング) CPU かを指定します。 ● 専用 CPU を指定するには、CPU 番号または CPU 範囲を指定します。たとえば、CPU 2 と 3 を専用に指定し、残りのすべての CPU を共有に指定するには、プロパティーを **2-3** に設定します。 ● 共有 CPU を指定するには、CPU 番号または CPU 範囲の前にキャレット (^) を付けます。たとえば、CPU 0 と 1 を共有に指定し、残りのすべての CPU を専用に指定するには、プロパティーを **^0-1** に設定します。

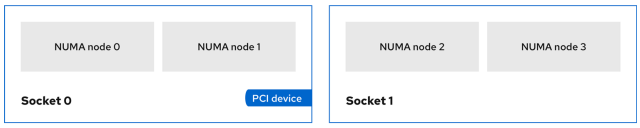
インスタンス PCI NUMA アフィニティーポリシー

以下の表に示す属性キーを使用して、PCI パススルーデバイスおよび SR-IOV インターフェイスの NUMA アフィニティーポリシーを指定するフレーバーを作成します。

表3.14 PCI NUMA アフィニティーポリシー用のフレーバーメタデータ

キー	説明

hw:pci_numa_affinity_policy	PCI パススルーデバイスおよび SR-IOV インターフェイスの NUMA アフィニティーポリシーを指定します。以下の有効な値のいずれかに設定します。
	● **required**: インスタンスの NUMA ノードの少なくとも 1 つが PCI デバイスとのアフィニティーを持つ場合に限り、Compute サービスは PCI デバイスを要求するインスタンスを作成します。このオプションは、最高のパフォーマンスを提供します。
	● **preferred**: Compute サービスは、NUMA アフィニティーに基づきベストエフォートで PCI デバイスの選択を試みます。これができない場合には、Compute サービスは PCI デバイスとのアフィニティーを持たない NUMA ノード上でインスタンスをスケジュールします。
	● **legacy**: (デフォルト) 以下のどちらかのケースで、Compute サービスは PCI デバイスを要求するインスタンスを作成します。
	  ○ PCI デバイスが少なくとも 1 つの NUMA ノードとのアフィニティーを持つ。
	  ○ PCI デバイスが NUMA アフィニティーに関する情報を提供しない。
	● **socket**: インスタンス NUMA ノード少なくとも 1 つが PCI デバイスと同じホストソケット内の NUMA ノードとのアフィニティーを持つ場合に限り、Compute サービスは PCI デバイスを要求するインスタンスを作成します。たとえば、次のホストアーキテクチャーには 2

キー	説明
	つのソケットがあります。各ソケットには 2 つの NUMA ノードがあり、PCI デバイスはソケットの 1 つのノードの 1 つに接続されています。  Compute サービスは、2 つの NUMA ノードと **socket** PCI NUMA アフィニティーポリシーを持つインスタンスを次のホストノードの組み合わせにのみピニングできます。これは、すべてのホストノードに、PCI デバイスのソケットにピニングされているインスタンス NUMA ノードが少なくとも 1 つあるためです。 ○ ノード 0 とノード 1 ○ ノード 0 とノード 2 ○ ノード 0 とノード 3 ○ ノード 1 とノード 2 ○ ノード 1 とノード 3 インスタンスをピニングできないホストノードの唯一の組み合わせはノード 2 とノード 3 です。これらのノードはいずれも PCI デバイスと同じソケット上にないためです。他のノードが他のインスタンスによって使用されており、ノード 2 と 3 のみが使用可能な場合、インスタンスは起動しません。

インスタンスの NUMA トポロジー

属性を使用して、インスタンスの仮想 CPU スレッドのホスト NUMA 配置、ならびにホスト NUMA ノードからのインスタンスの仮想 CPU およびメモリーの割り当てを定義するフレーバーを作成することができます。

メモリーおよび仮想 CPU の割り当てがコンピュートホスト内の NUMA ノードのサイズよりも大きいフレーバーの場合、インスタンスの NUMA トポロジーを定義するとインスタンス OS のパフォーマンスが向上します。

Compute スケジューラーは、これらの属性を使用してインスタンスに適したホストを決定します。たとえば、クラウドユーザーは以下のフレーバーを使用してインスタンスを起動します。

```
$ openstack flavor set numa_top_flavor \
  --property hw:numa_nodes=2 \
  --property hw:numa_cpus.0=0,1,2,3,4,5 \
  --property hw:numa_cpus.1=6,7 \
  --property hw:numa_mem.0=3072 \
  --property hw:numa_mem.1=1024
```

Compute スケジューラーは、2 つの NUMA ノード (1 つは 3 GB の RAM を持ち 6 つの CPU を実行できるノード、もう 1 つは 1GB の RAM を持ち 2 つの CPU を実行できるノード) を持つホストを探します。4 GB の RAM を持ち 8 つの CPU を実行できる 1 つの NUMA ノードを持つホストの場合、Compute スケジューラーは有効な一致とは見なしません。

注記

フレーバーで定義された NUMA トポロジーは、イメージで定義された NUMA トポロジーでオーバーライドされることはありません。イメージの NUMA トポロジーがフレーバーの NUMA トポロジーと競合する場合、Compute サービスは **ImageNUMATopologyForbidden** エラーを報告します。

注意

この機能を使用して、インスタンスを特定のホスト CPU または NUMA ノードに制限することはできません。包括的なテストおよびパフォーマンス計測が完了した後にのみ、この機能を使用してください。代わりに **hw:pci_numa_affinity_policy** プロパティーを使用することができます。

以下の表に示す属性キーを使用して、インスタンスの NUMA トポロジーを定義します。

表3.15 NUMA トポロジー用のフレーバーメタデータ

キー	説明
hw:numa_nodes	インスタンスの仮想 CPU スレッドの実行先として制限するホスト NUMA ノードの数を指定します。指定しない場合は、利用可能な任意の数のホスト NUMA ノード上で仮想 CPU スレッドを実行することができます。
hw:numa_cpus.N	インスタンスの NUMA ノード N にマッピングするインスタンスの仮想 CPU のコンマ区切りリスト。このキーを指定しない場合、仮想 CPU は利用可能な NUMA ノード間で均等に配分されます。 N は 0 から始まります。*.N 値を使用する場合には注意が必要です。NUMA ノードが少なくとも 2 つある場合に限り使用してください。 この属性は **hw:numa_nodes** を設定している場合にのみ有効で、インスタンスの NUMA ノードに CPU および RAM が対称的に割り当てられていない場合 (一部の NFV 負荷で重要) にのみ必要です。
hw:numa_mem.N	インスタンスの NUMA ノード N にマッピングするインスタンスのメモリー容量 (MB 単位)。このキーを指定しない場合、メモリーは利用可能な NUMA ノード間で均等に配分されます。 N は 0 から始まります。*.N 値を使用する場合には注意が必要です。NUMA ノードが少なくとも 2 つある場合に限り使用してください。 この属性は **hw:numa_nodes** を設定している場合にのみ有効で、インスタンスの NUMA ノードに CPU および RAM が対称的に割り当てられていない場合 (一部の NFV 負荷で重要) にのみ必要です。

警告

hw:numa_cpus.N で指定する仮想 CPU の総数または **hw:numa_mem.N** で指定するメモリー容量が、それぞれ利用可能な CPU の数またはメモリー容量よりも大きい場合、Compute サービスは例外を発生させます。

CPU リアルタイムポリシー

以下の表に示す属性キーを使用して、インスタンス内のプロセッサーのリアルタイムポリシーを定義します。

注記

- インスタンスのほとんどの仮想 CPU は、リアルタイムポリシーを設定して実行することができますが、非リアルタイムのゲストプロセスとエミュレーターのオーバーヘッドプロセスの両方に使用するために、少なくとも 1 つの仮想 CPU を非リアルタイムと識別する必要があります。
- この追加スペックを使用するには、ピニングされた CPU を有効にする必要があります。

表3.16 CPU リアルタイムポリシー用のフレーバーメタデータ

キー	説明
hw:cpu_realtime	リアルタイムポリシーをインスタンスの仮想 CPU に割り当てるフレーバーを作成するには、**yes** に設定します。 デフォルト: **no**

hw:cpu_realtime_mask	リアルタイムポリシーを割り当てない仮想 CPU を指定します。マスクする値の前にキャレット記号 (^) を追加する必要があります。仮想 CPU 0 および 1 を除くすべての仮想 CPU にリアルタイムポリシーを設定する場合の例を以下に示します。
	
	`$ openstack flavor set <flavor> \`
	`  --property hw:cpu_realtime="yes" \`
	`  --property hw:cpu_realtime_mask=^0-1`
	
	注記
	
	hw_cpu_realtime_mask 属性がイメージで設定されている場合、フレーバーで設定した **hw:cpu_realtime_mask** 属性よりも優先されます。

エミュレータースレッドポリシー

物理 CPU をインスタンスに割り当てて、エミュレータースレッドに使用することができます。エミュレータースレッドとは、インスタンスと直接関係しないエミュレータープロセスを指します。リアルタイム負荷には、専用のエミュレータースレッド用物理 CPU が必要です。エミュレータースレッドポリシーを使用するには、以下の属性を設定してピニングされた CPU を有効にする必要があります。

```
--property hw:cpu_policy=dedicated
```

以下の表に示す属性キーを使用して、インスタンスのエミュレータースレッドポリシーを定義します。

表3.17 エミュレータースレッドポリシー用のフレーバーメタデータ

キー	説明
hw:emulator_threads_policy	インスタンスに使用するエミュレータースレッドポリシーを指定します。以下の有効な値のいずれかに設定します。 ● **share**: エミュレータースレッドは、**NovaComputeCpuSharedSet** heat パラメーターで定義される物理 CPU 全体で共有されます。**NovaComputeCpuSharedSet** が設定されていない場合、エミュレータースレッドはインスタンスに関連付けられたピニングされた CPU 全体で共有されます。 ● **isolate**: エミュレータースレッド用に、インスタンスごとに専用の物理 CPU をさらに確保します。このポリシーは過度にリソースを消費するので、使用には注意が必要です。 ● unset: (デフォルト) エミュレータースレッドポリシーは有効ではなく、エミュレータースレッドはインスタンスに関連付けられたピニングされた CPU 全体で共有されます。

インスタンスのメモリーページサイズ

以下の表に示す属性キーを使用して、明示的なメモリーページサイズでインスタンスを作成します。

表3.18 メモリーページサイズ用のフレーバーメタデータ

キー	説明

キー	説明
hw:mem_page_size	インスタンスをサポートするのに使用するラージページのサイズを指定します。このオプションを使用すると、**hw:numa_nodes** で特に指定しない限り、1 NUMA ノードの暗黙的な NUMA トポロジーが作成されます。以下の有効な値のいずれかに設定します。 ● **large**: ホストでサポートされる最小のページサイズより大きいページサイズを選択します。x86_64 システムでは 2 MB または 1 GB です。 ● **small**: ホストでサポートされる最小のページサイズを選択します。X86_64 システムでは、4 kB (通常のページ) です。 ● **any**: libvirt ドライバーで決定される、利用可能な最大のヒュージページサイズを選択します。 ● **<pagesize>**: (文字列) ワークロードに具体的な要件がある場合、ページサイズを明示的に設定します。ページサイズには整数値を使用し、kB またはその他の標準単位で指定します。(例: **4KB**、**2MB**、**2048**、**1GB**)。 ● unset: (デフォルト) インスタンスをサポートするのにラージページは使用されず、暗黙的な NUMA トポロジーは生成されません。

PCI パススルー

以下の表に示す属性キーを使用して、グラフィックカードやネットワークデバイス等の物理 PCI デバイスをインスタンスにアタッチします。PCI パススルーの使用に関する詳細は、Configuring PCI passthrough を参照してください。

表3.19 PCI パススルー用のフレーバーメタデータ

キー	説明
pci_passthrough:alias	以下の形式を使用して、インスタンスに割り当てる PCI デバイスを指定します。 `<alias>:<count>` ● **<alias>** を、特定の PCI デバイスクラスに対応するエイリアスに置き換えます。 ● **<count>** を、インスタンスに割り当てる種別 **<alias>** の PCI デバイスの数に置き換えます。

ハイパーバイザーの署名

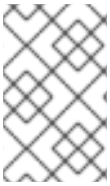
以下の表に示す属性キーを使用して、ハイパーバイザーの署名をインスタンスからは非表示にします。

表3.20 ハイパーバイザーの署名を非表示にするためのフレーバーメタデータ

キー	説明
hide_hypervisor_id	ハイパーバイザーの署名をインスタンスからは非表示にするには、**True** に設定します。これにより、すべてのドライバーがインスタンスで読み込みおよび操作を行うことができます。

UEFI セキュアブート

次の表のプロパティーキーを使用して、UEFI セキュアブートで保護されたインスタンスを作成します。



注記

UEFI セキュアブートを使用するインスタンスは、UEFI および GUID パーティションテーブル (GPT) 標準をサポートし、EFI システムパーティションを含める必要があります。

表3.21 UEFI セキュアブートのフレーバーメタデータ

キー	説明
os:secure_boot	このフレーバーで起動されたインスタンスのセキュアブートを有効にするには、**required** に設定します。デフォルトでは無効になっています。

インスタンスのリソース特性

各リソースプロバイダーには特性のセットがあります。特性は、ストレージディスクの種別や Intel CPU 拡張命令セットなど、リソースプロバイダーの機能的な要素です。インスタンスは、これらの中から要求する特性を指定することができます。

指定することのできる特性は **os-traits** ライブラリーで定義されます。特性の例を以下に示します。

- **COMPUTE_TRUSTED_CERTS**
- **COMPUTE_NET_ATTACH_INTERFACE_WITH_TAG**
- **COMPUTE_IMAGE_TYPE_RAW**
- **HW_CPU_X86_AVX**
- **HW_CPU_X86_AVX512VL**
- **HW_CPU_X86_AVX512CD**

os-traits ライブラリーの使用方法の詳細は、Usage を参照してください。

以下の表に示す属性キーを使用して、インスタンスのリソース特性を定義します。

表3.22 リソース特性用のフレーバーメタデータ

キー	説明
trait:<trait_name>	コンピュートノードの特性を指定します。特性を、以下の有効な値のいずれかに設定します。 ● **required**: インスタンスをホストするために選択したコンピュートノードになければいけない特性 ● **forbidden**: インスタンスをホストするために選択したコンピュートノードにあってはいけない特性 以下に例を示します。 $ openstack flavor set --property trait:HW_CPU_X86_AVX512BW=required avx512-flavor

インスタンスのベアメタルリソースクラス

以下の表に示す属性キーを使用して、インスタンスのベアメタルリソースクラスを要求します。

表3.23 ベアメタルリソースクラス用のフレーバーメタデータ

キー	説明
resources:<resource_class_name>	この属性を使用して、値をオーバーライドする標準のベアメタルリソースクラスを指定するか、インスタンスが要求するカスタムのベアメタルリソースクラスを指定します。 オーバーライドすることができる標準のリソースクラスは **VCPU**、**MEMORY_MB**、および **DISK_GB** です。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ぐには、標準のリソースクラスの値を **0** に設定します。 カスタムリソースクラスの名前は、**CUSTOM_** で始まる必要があります。Bare Metal サービス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、すべて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。 たとえば、**--resource-class baremetal.SMALL** のノードにインスタンスをスケジュールするには、以下のフレーバーを作成します。 $ openstack flavor set \ --property resources:CUSTOM_BAREMETAL_SMALL=1 \ --property resources:VCPU=0 --property resources:MEMORY_MB=0 \ --property resources:DISK_GB=0 compute-small

第4章 コンピュートノードで CPU を設定する

クラウドユーザーは、インスタンスのスケジューリングおよび配置を設定して、最大のパフォーマンスを得ることができます。そのためには、NFV や高性能コンピューティング (HPC) などの特化されたワークロードを対象にするカスタムフレーバーを作成します。

以下の機能を使用して、最適な CPU パフォーマンスを得るためにインスタンスを調整します。

- **CPU ピニング**: 仮想 CPU を物理 CPU に固定します。
- **エミュレータースレッド**: インスタンスに関連付けられたエミュレータースレッドを物理 CPU に固定します。
- **CPU 機能フラグ**: コンピュートノード間のライブマイグレーションの互換性を向上させるために、インスタンスに適用される CPU 機能フラグの標準セットを設定します。

4.1. コンピュートノードでの CPU ピニングの設定

コンピュートノードで CPU ピニングを有効化することで、各インスタンスの CPU プロセスを専用のホスト CPU で実行するように設定することができます。インスタンスが CPU ピニングを使用する場合には、各インスタンスの仮想 CPU プロセスには、他のインスタンスの仮想 CPU プロセスが使用できない独自のホストの物理 CPU が割り当てられます。CPU ピニングが設定されたコンピュートノード上で動作するインスタンスには、NUMA トポロジーがあります。インスタンスの NUMA トポロジーの各 NUMA ノードは、ホストコンピュートノード上の NUMA ノードにマッピングされます。

専用の (ピニングされた) CPU を持つインスタンスと共有 (フローティング) の CPU を持つインスタンスを同じコンピュートノード上にスケジューリングするように、Compute のスケジューラーを設定することができます。NUMA トポロジーを持つコンピュートノード上で CPU ピニングを設定するには、以下の手順を実施する必要があります。

1. CPU ピニング用のコンピュートノードを指定する。
2. ピニングされたインスタンス仮想 CPU プロセス、フローティングのインスタンス仮想 CPU プロセス、およびホストのプロセス用にホストコアを確保するようにコンピュートノードを設定する。
3. オーバークラウドをデプロイする。
4. CPU ピニングを要求するインスタンスを起動するためのフレーバーを作成する。
5. 共有 (あるいはフローティング) の CPU を使用するインスタンスを起動するためのフレーバーを作成する。

注記

CPU ピニングを設定すると、NUMA トポロジーが要求されていない場合でも、インスタンス上に暗黙的な NUMA トポロジーが作成されます。

4.1.1. 前提条件

- コンピュートノードの NUMA トポロジーを把握している。
- コンピュートノードで **NovaReservedHugePages** を設定している。詳細は、コンピュートノードで Huge Page を設定する を参照してください。

4.1.2. CPU ピニング用コンピュートノードの指定

ピニングされた CPU を持つインスタンスのコンピュートノードを指定するには、新しいロールファイルを作成して CPU ピニングロールを設定し、CPU ピニング用のコンピュートノードにタグを付けるために使用する CPU ピニングリソースクラスを使用してベアメタルノードを設定する必要があります。

注記

以下の手順は、まだプロビジョニングされていない新しいオーバークラウドノードに適用されます。すでにプロビジョニングされている既存のオーバークラウドノードにリソースクラスを割り当てるには、スケールダウン手順を使用してノードのプロビジョニングを解除してから、スケールアップ手順を使用して新しいリソースクラスの割り当てでノードを再プロビジョニングする必要があります。詳細は、オーバークラウドノードのスケーリング を参照してください。

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. **Controller**、**Compute**、**ComputeCPUPinning** ロール、およびオーバークラウドに必要なその他のロールを含む、**roles_data_cpu_pinning.yaml** という名前の新しいロールデータファイルを生成します。

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_cpu_pinning.yaml \
 Compute:ComputeCPUPinning Compute Controller
```

4. **roles_data_cpu_pinning.yaml** を開き、以下のパラメーターおよびセクションを編集または追加します。

セクション/パラメーター	現在の値	新しい値
ロールのコメント	**Role: Compute**	**Role: ComputeCPUPinning**
ロール名	**Compute**	**name: ComputeCPUPinning**
description	**Basic Compute Node role**	**CPU Pinning Compute Node role**
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputepinning-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-cpu-pinning.yaml**

5. オーバークラウド用の CPU ピニングコンピュートノードをノード定義のテンプレート **node.json** または **node.yaml** に追加して、そのノードを登録します。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** ガイドの オーバークラウドのノードの登録 を参照してください。

6. ノードのハードウェアを検査します。

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

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの ベアメタルノードハードウェアのインベントリーの作成 を参照してください。

7. CPU ピニング用に指定する各ベアメタルノードに、カスタムの CPU ピニングリソースクラスをタグ付けします。

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.CPU-PINNING <node>
```

<node> をベアメタルノードの ID に置き換えてください。

8. ノード定義ファイル **overcloud-baremetal-deploy.yaml** に **ComputeCPUPinning** ロールを追加し、予測ノード配置、リソースクラス、ネットワークトポロジー、またはノードに割り当てるその他の属性を定義します。

```
- name: Controller
  count: 3
- name: Compute
  count: 3
- name: ComputeCPUPinning
  count: 1
  defaults:
    resource_class: baremetal.CPU-PINNING
    network_config:
      template: /home/stack/templates/nic-config/myRoleTopology.j2 (1)
```

(1) 既存のネットワークトポロジーを再利用するか、ロール用の新しいカスタムネットワークインターフェイステンプレートを作成できます。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** ガイドの カスタムネットワークインターフェイステンプレート を参照してください。**network_config** プロパティーを使用してネットワーク定義を定義しない場合、デフォルトのネットワーク定義が使用されます。

ノード定義ファイルでノード属性を設定するために使用できるプロパティーについて詳しくは、ベアメタルノードのプロビジョニング属性 を参照してください。ノード定義ファイルの例は、ノード定義ファイルの例 を参照してください。

9. プロビジョニングコマンドを実行して、ロールの新しいノードをプロビジョニングします。

```
(undercloud)$ openstack overcloud node provision \
--stack <stack> \
[--network-config \]
--output /home/stack/templates/overcloud-baremetal-deployed.yaml \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- **<stack>** を、ベアメタルノードがプロビジョニングされるスタックの名前に置き換えます。指定しない場合、デフォルトは **overcloud** です。
- **--network-config** オプションの引数を含めて、**cli-overcloud-node-network-config.yaml** Ansible Playbook にネットワーク定義を提供します。**network_config** プロパティーを使用してネットワーク定義を定義しない場合、デフォルトのネットワーク定義が使用されます。

10. 別のターミナルでプロビジョニングの進捗をモニタリングします。プロビジョニングが成功すると、ノードの状態が **available** から **active** に変わります。

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

11. **--network-config** オプションを指定してプロビジョニングコマンドを実行しなかった場合は、**network-environment.yaml** ファイルで **<Role>NetworkConfigTemplate** パラメーターを設定して、NIC テンプレートファイルを指すようにします。

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputeCPUPinningNetworkConfigTemplate: /home/stack/templates/nic-
configs/<cpu_pinning_net_top>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

<cpu_pinning_net_top> を **ComputeCPUPinning** ロールのネットワークトポロジーが含まれるファイルの名前に置き換えます。たとえば、デフォルトのネットワークトポロジーを使用する場合は **compute.yaml** です。

4.1.3. CPU ピニング用コンピュートノードの設定

ノードの NUMA トポロジーに基づいて、コンピュートノードでの CPU ピニングを設定します。効率を高めるために、全 NUMA ノードにわたって、CPU コアの一部をホストのプロセス用に確保します。残りの CPU コアをインスタンスの管理に割り当てます。

以下の手順では、以下の NUMA トポロジー (8 つの CPU コアを 2 つの NUMA ノードに分散) を使用して、CPU ピニングの設定方法を説明します。

表4.1 NUMA トポロジーの例

NUMA ノード 0		NUMA ノード 1	
コア 0	コア 1	コア 2	コア 3
コア 4	コア 5	コア 6	コア 7

以下の手順では、コア 0 および 4 をホストのプロセス用に、コア 1、3、5、および 7 を CPU ピニングが必要なインスタンス用に、そしてコア 2 および 6 を CPU ピニングが不要なフローティングインスタンス用に、それぞれ確保します。

手順

1. ピニングされたインスタンス、フローティングのインスタンス、およびホストプロセス用にコアを確保するようにコンピュートノードを設定する環境ファイルを作成します (例: **cpu_pinning.yaml**)。

2. NUMA 対応コンピュートノードに NUMA トポロジーが設定されたインスタンスをスケジュールするには、Compute 環境ファイルの **NovaSchedulerEnabledFilters** パラメーターに **NUMATopologyFilter** がなければ、このフィルターを追加します。

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - NUMATopologyFilter
```

NUMATopologyFilter の詳細は、Compute scheduler filters を参照してください。

3. 専用のインスタンス用に物理 CPU コアを確保するには、以下の設定を **cpu_pinning.yaml** に追加します。

```
parameter_defaults:
  ComputeCPUPinningParameters:
    NovaComputeCpuDedicatedSet: 1,3,5,7
```

4. 共有のインスタンス用に物理 CPU コアを確保するには、以下の設定を **cpu_pinning.yaml** に追加します。

```
parameter_defaults:
  ComputeCPUPinningParameters:
    ...
    NovaComputeCpuSharedSet: 2,6
```

5. file-backed メモリーを使用していない場合は、ホストプロセス用に予約する RAM の容量を指定します。

```
parameter_defaults:
  ComputeCPUPinningParameters:
    ...
    NovaReservedHugePages: <ram>
```

<ram> を、確保するメモリー容量 (MB 単位) に置き換えます。

6. インスタンス用に確保した CPU コアでホストプロセスが実行されないようにするには、**IsolCpusList** パラメーターに、インスタンス用に確保した CPU コアを設定します。

```
parameter_defaults:
  ComputeCPUPinningParameters:
    ...
    IsolCpusList: 1-3,5-7
```

コンマ区切りの CPU インデックスのリストまたは範囲を使用して、**IsolCpusList** パラメーターの値を指定します。

7. 新しい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] \
  -r /home/stack/templates/roles_data_cpu_pinning.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/cpu_pinning.yaml \
  -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
  -e /home/stack/templates/node-info.yaml
```

4.1.4. インスタンス用の専用 CPU フレーバーの作成

クラウドユーザーが専用の CPU を持つインスタンスを作成できるようにするには、インスタンス起動用の専用 CPU ポリシーが設定されたフレーバーを作成します。

前提条件

- ホストで同時マルチスレッド (SMT) が有効である。
- コンピュートノードが CPU ピニングを許可するように設定されている。詳しくは、コンピュートノードでの CPU ピニングの設定 を参照してください。

手順

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

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

2. CPU ピニングを要求するインスタンス用のフレーバーを作成します。

```
(overcloud)$ openstack flavor create --ram <size_mb> \
 --disk <size_gb> --vcpus <no_reserved_vcpus> pinned_cpus
```

3. ピニングされた CPU を要求するには、フレーバーの **hw:cpu_policy** 属性を **dedicated** に設定します。

```
(overcloud)$ openstack flavor set \
 --property hw:cpu_policy=dedicated pinned_cpus
```

4. file-backed メモリーを使用していない場合は、フレーバーの **hw:mem_page_size** プロパティーでNUMA 対応メモリー割り当てを有効に設定します。

```
(overcloud)$ openstack flavor set \
 --property hw:mem_page_size=<page_size> pinned_cpus
```

- **<page_size>** は、次に示す有効な値のいずれかに置き換えます。
  - **large**: ホストでサポートされる最大のページサイズを選択します。x86_64 システムでは 2 MB または 1 GB です。
  - **small**: (デフォルト) ホストでサポートされる最小のページサイズを選択します。X86_64 システムでは、4 kB (通常のページ) です。
  - **any** : イメージに設定された **hw_mem_page_size** を使用してページサイズを選択します。ページサイズがイメージで指定されていない場合は、libvirt ドライバーで決定される、利用可能な最大のページサイズを選択します。

- **<pagesize>**: ワークロードに具体的な要件がある場合、ページサイズを明示的に設定します。ページサイズには整数値を使用し、kB またはその他の標準単位で指定します。(例: 4kB、2MB、2048、1GB)。

注記

hw:mem_page_size を **small** または **any** に設定するには、インスタンスではないプロセス用に各 NUMA ノードで予約するメモリーページの量を設定しておく必要があります。詳細は、コンピュートノードで Huge Page を設定する を参照してください。

5. それぞれの仮想 CPU をスレッドシブリングに配置するには、フレーバーの **hw:cpu_thread_policy** 属性を **require** に設定します。

```
(overcloud)$ openstack flavor set \
 --property hw:cpu_thread_policy=require pinned_cpus
```

注記

- ホストに SMT アーキテクチャーがない場合や、スレッドシブリングが利用可能な CPU コアが十分にない場合には、スケジューリングが失敗します。これを回避するには、**hw:cpu_thread_policy** を **require** ではなく **prefer** に設定します。**prefer** ポリシーは、スレッドシブリングが利用可能な場合に使用されるデフォルトのポリシーです。
- **hw:cpu_thread_policy=isolate** を使用する場合は、SMT を無効にするか、SMT をサポートしないプラットフォームを使用する必要があります。

検証

1. フレーバーにより専用の CPU を持つインスタンスが作成されることを確認するには、新しいフレーバーを使用してインスタンスを起動します。

```
(overcloud)$ openstack server create --flavor pinned_cpus \
 --image <image> pinned_cpu_instance
```

4.1.5. インスタンス用の共有 CPU フレーバーの作成

クラウドユーザーが共有の (あるいはフローティング) CPU を使用するインスタンスを作成できるようにするには、インスタンス起動用の共有 CPU ポリシーが設定されたフレーバーを作成します。

前提条件

- コンピュートノードが共有 CPU 用に物理 CPU コアを確保するように設定されている。詳しくは、コンピュートノードでの CPU ピニングの設定 を参照してください。

手順

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

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

2. CPU ピニングを要求しないインスタンス用のフレーバーを作成します。

```
(overcloud)$ openstack flavor create --ram <size_mb> \
 --disk <size_gb> --vcpus <no_reserved_vcpus> floating_cpus
```

3. フローティング CPU を要求するには、フレーバーの **hw:cpu_policy** 属性を **shared** に設定します。

```
(overcloud)$ openstack flavor set \
 --property hw:cpu_policy=shared floating_cpus
```

4. file-backed メモリーを使用していない場合は、フレーバーの **hw:mem_page_size** プロパティーでNUMA 対応メモリー割り当てを有効に設定します。

```
(overcloud)$ openstack flavor set \
 --property hw:mem_page_size=<page_size> pinned_cpus
```

- **<page_size>** は、次に示す有効な値のいずれかに置き換えます。
  - **large**: ホストでサポートされる最大のページサイズを選択します。x86_64 システムでは 2 MB または 1 GB です。
  - **small**: (デフォルト) ホストでサポートされる最小のページサイズを選択します。X86_64 システムでは、4 kB (通常のページ) です。
  - **any** : イメージに設定された **hw_mem_page_size** を使用してページサイズを選択します。ページサイズがイメージで指定されていない場合は、libvirt ドライバーで決定される、利用可能な最大のページサイズを選択します。
  - **<pagesize>**: ワークロードに具体的な要件がある場合、ページサイズを明示的に設定します。ページサイズには整数値を使用し、kB またはその他の標準単位で指定します。(例: 4kB、2MB、2048、1GB)。

注記

hw:mem_page_size を **small** または **any** に設定するには、インスタンスではないプロセス用に各 NUMA ノードで予約するメモリーページの量を設定しておく必要があります。詳細は、コンピュートノードで Huge Page を設定する を参照してください。

4.1.6. インスタンス用の混合 CPU フレーバーの作成

クラウドユーザーが専用 CPU と共有 CPU の組み合わせを持つインスタンスを作成できるようにするには、インスタンス起動用の混合 CPU ポリシーが設定されたフレーバーを作成します。

手順

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

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

2. 専用 CPU と共有 CPU の組み合わせを要求するインスタンス用のフレーバーを作成します。

```
(overcloud)$ openstack flavor create --ram <size_mb> \
 --disk <size_gb> --vcpus <number_of_reserved_vcpus> \
 --property hw:cpu_policy=mixed mixed_CPUs_flavor
```

3. どの CPU を専用または共有にする必要があるかを指定します。

```
(overcloud)$ openstack flavor set \
 --property hw:cpu_dedicated_mask=<CPU_number> \
 mixed_CPUs_flavor
```

- **<CPU_number>** を、専用または共有する必要がある CPU に置き換えます。
  - 専用 CPU を指定するには、CPU 番号または CPU 範囲を指定します。たとえば、CPU 2 と 3 を専用に指定し、残りのすべての CPU を共有に指定するには、プロパティーを **2-3** に設定します。
  - 共有 CPU を指定するには、CPU 番号または CPU 範囲の前にキャレット (^) を付けます。たとえば、CPU 0 と 1 を共有に指定し、残りのすべての CPU を専用に指定するには、プロパティーを **^0-1** に設定します。

4. file-backed メモリーを使用していない場合は、フレーバーの **hw:mem_page_size** プロパティーでNUMA 対応メモリー割り当てを有効に設定します。

```
(overcloud)$ openstack flavor set \
 --property hw:mem_page_size=<page_size> pinned_cpus
```

- **<page_size>** は、次に示す有効な値のいずれかに置き換えます。
  - **large**: ホストでサポートされる最大のページサイズを選択します。x86_64 システムでは 2 MB または 1 GB です。
  - **small**: (デフォルト) ホストでサポートされる最小のページサイズを選択します。X86_64 システムでは、4 kB (通常のページ) です。
  - **any** : イメージに設定された **hw_mem_page_size** を使用してページサイズを選択します。ページサイズがイメージで指定されていない場合は、libvirt ドライバーで決定される、利用可能な最大のページサイズを選択します。
  - **<pagesize>**: ワークロードに具体的な要件がある場合、ページサイズを明示的に設定します。ページサイズには整数値を使用し、kB またはその他の標準単位で指定します。(例: 4kB、2MB、2048、1GB)。

注記

hw:mem_page_size を **small** または **any** に設定するには、インスタンスではないプロセス用に各 NUMA ノードで予約するメモリーページの量を設定しておく必要があります。詳細は、コンピュートノードで Huge Page を設定する を参照してください。

4.1.7. 同時マルチスレッド (SMT) 対応のコンピュートノードでの CPU ピニングの設定

コンピュートノードが同時マルチスレッド (SMT) をサポートする場合、スレッドシブリングを専用または共有セットのいずれかにグルーピングします。スレッドシブリングは共通のハードウェアを共有するため、あるスレッドシブリング上で動作しているプロセスが、他のスレッドシブリングのパフォーマンスに影響を与える可能性があります。

たとえば、ホストは、SMT 対応のデュアルコア CPU に 4 つの論理 CPU コア (0、1、2、および 3) を認識します。この 4 つの CPU に対して、スレッドシブリングのペアが 2 つあります。

- スレッドシブリング 1: 論理 CPU コア 0 および 2
- スレッドシブリング 2: 論理 CPU コア 1 および 3

このシナリオでは、論理 CPU コア 0 および 1 を専用として、2 および 3 を共有として割り当てないでください。そうではなく、0 および 2 を専用として、1 および 3 を共有として割り当てます。

/sys/devices/system/cpu/cpuN/topology/thread_siblings_list のファイル。**N** は論理 CPU 番号で、スレッドペアが含まれます。以下のコマンドを使用して、スレッドシブリングである論理 CPU コアを特定できます。

```
# grep -H . /sys/devices/system/cpu/cpu*/topology/thread_siblings_list | sort -n -t ':' -k 2 -u
```

以下の出力は、論理 CPU コア 0 と論理 CPU コア 2 が同じコア上のスレッドであることを示しています。

```
/sys/devices/system/cpu/cpu0/topology/thread_siblings_list:0,2
/sys/devices/system/cpu/cpu2/topology/thread_siblings_list:1,3
```

4.1.8. 関連情報

- NUMA ノードのトポロジーについての理解

4.2. エミュレータースレッドの設定

コンピュートノードには、エミュレータースレッドと呼ばれる各インスタンスのハイパーバイザーとリンクするオーバーヘッドタスクがあります。デフォルトでは、エミュレータースレッドはインスタンスと同じ CPU で実行され、インスタンスのパフォーマンスに影響を及ぼします。

エミュレータースレッドポリシーを設定して、インスタンスが使用する CPU とは別の CPU でエミュレータースレッドを実行することができます。

注記

- パケットロスを避けるために、NFV デプロイメントでは絶対に仮想 CPU のプリエンプションを行わないでください。

前提条件

- CPU ピニングが有効になっている。

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. Compute 環境ファイルを開きます。
3. CPU ピニングを必要とするインスタンス用に物理 CPU コアを確保するには、Compute 環境ファイルで **NovaComputeCpuDedicatedSet** パラメーターを設定します。たとえば、以下の設定では、32 コア CPU を持つコンピュートノードに専用の CPU を設定します。

```
parameter_defaults:
  ...
  NovaComputeCpuDedicatedSet: 2-15,18-31
```

```
...
```

詳しくは、コンピュートノードでの CPU ピニングの設定 を参照してください。

4. エミュレータースレッド用に物理 CPU コアを確保するには、Compute 環境ファイルで **NovaComputeCpuSharedSet** パラメーターを設定します。たとえば、以下の設定では、32 コア CPU を持つコンピュートノードに共有の CPU を設定します。

```
parameter_defaults:
  ...
  NovaComputeCpuSharedSet: 0,1,16,17
  ...
```

注記

Compute スケジューラーは、共有 (またはフローティング) CPU 上で動作するインスタンス用にも共有セット内の CPU を使用します。詳しくは、コンピュートノードでの CPU ピニングの設定を参照してください。

5. **NovaSchedulerEnabledFilters** パラメーターにまだ **NUMATopologyFilter** がなければ、この Compute スケジュールフィルターを追加します。

6. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

7. インスタンスのエミュレータースレッドを **NovaComputeCpuSharedSet** を使用して設定した共有 CPU から選択した専用の CPU 上で実行するフレーバーを設定します。

```
(overcloud)$ openstack flavor set --property hw:cpu_policy=dedicated \
 --property hw:emulator_threads_policy=share \
 dedicated_emulator_threads
```

hw:emulator_threads_policy の設定オプションについての詳しい情報は、Flavor metadata の Emulator threads policy を参照してください。

4.3. インスタンスの CPU 機能フラグの設定

ホストコンピュートノードの設定を変更してコンピュートノードをリブートすることなく、インスタンスの CPU 機能フラグを有効または無効にすることができます。インスタンスに適用される CPU 機能フラグの標準的なセットを設定することで、コンピュートノード間でライブマイグレーションの互換性を実現するのに役立ちます。また、特定の CPU モデルにおいてインスタンスのセキュリティーやパフォーマンスに悪影響を与えるフラグを無効にしたり、セキュリティーやパフォーマンスの問題を軽減するフラグを有効したりして、インスタンスのパフォーマンスおよびセキュリティーを管理するのにも役立ちます。

4.3.1. 前提条件

- CPU モデルおよび機能フラグが、ホストコンピュートノードのハードウェアおよびソフトウェアでサポートされる必要があります。

- ホストが対応しているハードウェアを確認するには、コンピュートノードで以下のコマンドを入力します。

```
$ cat /proc/cpuinfo
```

- ホストが対応している CPU モデルを確認するには、コンピュートノードで以下のコマンドを入力します。

```
$ sudo podman exec -it nova_libvirt virsh cpu-models <arch>
```

<arch> をアーキテクチャーの名前に置き換えます (例: **x86_64**)。

4.3.2. インスタンスの CPU 機能フラグの設定

Compute サービスを設定し、特定の仮想 CPU モデルのインスタンスに CPU 機能フラグを適用します。

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. Compute 環境ファイルを開きます。
4. インスタンスの CPU モードを設定します。

```
parameter_defaults:
  ComputeParameters:
    NovaLibvirtCPUMode: <cpu_mode>
```

<cpu_mode> をコンピュートノード上の各インスタンスの CPU モードに置き換えます。以下の有効な値のいずれかに設定します。

- **host-model**:(デフォルト) ホストコンピュートノードの CPU モデルを使用します。この CPU モードを使用して、重要な CPU フラグをインスタンスに自動的に追加し、セキュリティー上の欠陥の軽減策を提供します。
- **custom**: 各インスタンスが使用する特定の CPU モデルを設定するのに使用します。

注記

CPU モードを **host-passthrough** に設定すると、コンピュートノードでホストされるインスタンスにそのコンピュートノードと同じ CPU モデルおよび機能フラグを使用することができます。

5. (オプション) **NovaLibvirtCPUMode** を **custom** に設定した場合は、カスタマイズするインスタンス CPU モデルを設定します。

```
parameter_defaults:
  ComputeParameters:
```

```
    NovaLibvirtCPUMode: 'custom'
    NovaLibvirtCPUModels: <cpu_model>
```

<cpu_model> を、ホストがサポートする CPU モデルのコンマ区切りリストに置き換えます。CPU モデルを順にリスト表示します。この際、より一般的で高度ではない CPU モデルは最初にリストに配置され、より機能が充実した CPU モデルが最後に置かれます (例: **SandyBridge,IvyBridge,Haswell,Broadwell**)。モデル名のリストは、**/usr/share/libvirt/cpu_map.xml** を参照してください。または、ホストコンピュートノードで以下のコマンドを入力します。

```
$ sudo podman exec -it nova_libvirt virsh cpu-models <arch>
```

<arch> をコンピュートノードのアーキテクチャー名に置き換えてください (例: **x86_64**)。

6. 指定の CPU モデルのインスタンスの CPU 機能フラグを設定します。

```
parameter_defaults:
  ComputeParameters:
    ...
    NovaLibvirtCPUModelExtraFlags: <cpu_feature_flags>
```

<cpu_feature_flags> を、有効または無効にする機能フラグのコンマ区切りリストに置き換えます。フラグを有効にするには各フラグの前に+を付け、無効にするには-を付けます。接頭辞が指定されていない場合、フラグが有効になります。特定の CPU モデルで利用可能な機能フラグのリストは、**/usr/share/libvirt/cpu_map/*.xml** を参照してください。

以下の例では、**IvyBridge** および **Cascadelake-Server** モデルの CPU 機能フラグ **pcid** および **ssbd** を有効にし、機能フラグ **mtrr** を無効にします。

```
parameter_defaults:
  ComputeParameters:
    NovaLibvirtCPUMode: 'custom'
    NovaLibvirtCPUModels: 'IvyBridge','Cascadelake-Server'
    NovaLibvirtCPUModelExtraFlags: 'pcid,+ssbd,-mtrr'
```

7. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

第5章 コンピュートノードでメモリーを設定する

クラウドユーザーは、インスタンスのスケジューリングおよび配置を設定して、最大のパフォーマンスを得ることができます。そのためには、NFV や高性能コンピューティング (HPC) などの特化されたワークロードを対象にするカスタムフレーバーを作成します。

以下の機能を使用して、最適なメモリーパフォーマンスを得るためにインスタンスを調整します。

- **Overallocation**: 仮想 RAM と物理 RAM の割り当て比率を調整します。
- **Swap**: メモリーのオーバーコミットを処理するために、割り当てられたスワップサイズを調整します。
- **Huge pages**: 通常のメモリー (4 KB ページ) とヒュージページ (2 MB または 1 GB ページ) の両方について、インスタンスのメモリー割り当てポリシーを調整します。
- **File-backed memory**: コンピュートノードのメモリー容量を拡張するために使用します。
- **SEV**: クラウドユーザーが、メモリー暗号化を使用するインスタンスを作成できるようにするために使用します。

5.1. オーバーコミットのためのメモリー設定

メモリーのオーバーコミットを使用する場合 (**NovaRAMAllocationRatio** >= 1.0)、割り当て率をサポートするのに十分なスワップ領域を確保してオーバークラウドをデプロイする必要があります。

注記

NovaRAMAllocationRatio パラメーターが **1 より小さい値** に設定されている場合は、RHEL でのスワップサイズの推奨事項に従ってください。詳細は、RHEL **ストレージデバイスの管理**の システムの推奨スワップ領域 を参照してください。

前提条件

- ノードに必要なスワップサイズが計算されている。詳細は、スワップサイズの計算 を参照してください。

手順

1. **/usr/share/openstack-tripleo-heat-templates/environments/enable-swap.yaml** ファイルを環境ファイルのディレクトリーにコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/enable-swap.yaml
/home/stack/templates/enable-swap.yaml
```

2. 以下のパラメーターを **enable-swap.yaml** ファイルに追加して、スワップサイズを設定します。

```
parameter_defaults:
  swap_size_megabytes: <swap size in MB>
  swap_path: <full path to location of swap, default: /swap>
```

3. その他の環境ファイルと共に **enable_swap.yaml** 環境ファイルをスタックに追加して、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/enable-swap.yaml
```

5.2. コンピュートノード上で確保するホストメモリーの計算

ホストのプロセス用に確保する RAM 容量の合計を判断するには、以下の各項目に十分なメモリーを割り当てる必要があります。

- ホスト上で実行されるリソース (たとえば、OSD は 3 GB のメモリーを消費します)
- ホストインスタンスに必要なエミュレーターのオーバーヘッド
- 各インスタンスのハイパーバイザー

メモリーへの追加要求を計算したら、以下の式を使用して各ノードのホストプロセス用に確保するメモリーの容量を決定します。

```
NovaReservedHostMemory = total_RAM - ( (vm_no * (avg_instance_size + overhead)) + (resource1 *
resource_ram) + (resourcen * resource_ram))
```

- **vm_no** は、インスタンスの数に置き換えてください。
- **avg_instance_size** は、各インスタンスが使用できるメモリーの平均容量に置き換えてください。
- **overhead** は、各インスタンスに必要なハイパーバイザーのオーバーヘッドに置き換えてください。
- **resource1** および **<resourcen>** までのすべてのリソースを、ノード上のリソース種別の数に置き換えてください。
- **resource_ram** は、この種別の各リソースに必要な RAM の容量に置き換えてください。

5.3. スワップサイズの計算

割り当てるスワップサイズは、メモリーのオーバーコミットを処理するのに十分な容量でなければなりません。以下の式を使用して、ノードに必要なスワップサイズを計算することができます。

- overcommit_ratio = **NovaRAMAllocationRatio** - 1
- 最小スワップサイズ (MB) = **(total_RAM * overcommit_ratio) + RHEL_min_swap**
- 推奨 (最大) スワップサイズ (MB) = **total_RAM * (overcommit_ratio + percentage_of_RAM_to_use_for_swap)**

percentage_of_RAM_to_use_for_swap 変数は、QEMU のオーバーヘッドおよびオペレーティングシステムまたはホストのサービスが消費するその他のリソースに対応するバッファーです。

たとえば、RAM 容量の合計が 64 GB で **NovaRAMAllocationRatio** が **1** に設定されている場合、利用可能な RAM の 25% をスワップに使用するには、

- 推奨 (最大) スワップサイズ = 64000 MB * (0 + 0.25) = 16000 MB

NovaReservedHostMemory の値を計算する方法は、コンピュートノード上で確保するホストメモリーの計算 を参照してください。

RHEL_min_swap の値を決定する方法は、RHEL の**ストレージデバイスの管理**の システムの推奨スワップ領域 を参照してください。

5.4. コンピュートノードでの HUGE PAGE の設定

クラウド管理者は、インスタンスが Huge Page を要求できるようにコンピュートノードを設定することができます。

注記

Huge Page を設定すると、NUMA トポロジーが要求されていない場合でも、インスタンス上に暗黙的な NUMA トポロジーが作成されます。

手順

1. Compute 環境ファイルを開きます。
2. インスタンスではないプロセス用に各 NUMA ノードで確保するヒュージページのメモリー量を設定します。

```
parameter_defaults:
  ComputeParameters:
    NovaReservedHugePages: ["node:0,size:1GB,count:1","node:1,size:1GB,count:1"]
```

   - 各ノードの **size** の値を、割り当てられたヒュージページのサイズに置き換えます。以下の有効な値のいずれかに設定します。
     - 2048 (2 MB 用)
     - 1 GB
   - 各ノードの **count** の値を、NUMA ノードごとに OVS が使用するヒュージページの数に置き換えます。たとえば、Open vSwitch が 4096 のソケットメモリーを使用する場合、この属性を 2 に設定します。
3. コンピュートノードで Huge Page を設定します。

```
parameter_defaults:
  ComputeParameters:
    ...
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32"
```

   注記

   複数の Huge Page サイズを設定する場合、初回の起動時に Huge Page フォルダーもマウントする必要があります。詳細は、初回起動時に複数の Huge Page フォルダーをマウントする を参照してください。

4. (オプション) インスタンスが 1 GB のヒュージページを割り当てるのを許可するには、CPU 機能フラグ **NovaLibvirtCPUModelExtraFlags** を設定して **pdpe1gb** を指定します。

```
parameter_defaults:
  ComputeParameters:
    NovaLibvirtCPUMode: 'custom'
    NovaLibvirtCPUModels: 'Haswell-noTSX'
    NovaLibvirtCPUModelExtraFlags: 'vmx, pdpe1gb'
```

注記

- インスタンスが 2 MB のヒュージページしか要求しない場合、CPU 機能フラグを設定する必要はありません。
- ホストが 1 GB ヒュージページの割り当てをサポートする場合に限り、インスタンスに 1 GB のヒュージページを割り当てることができます。
- **NovaLibvirtCPUMode** が **host-model** または **custom** に設定されている場合にのみ、**NovaLibvirtCPUModelExtraFlags** を **pdpe1gb** に設定する必要があります。
- ホストが **pdpe1gb** をサポートし、**NovaLibvirtCPUMode** に **host-passthrough** が使用される場合、**NovaLibvirtCPUModelExtraFlags** に **pdpe1gb** を設定する必要はありません。**pdpe1gb** フラグは Opteron_G4 および Opteron_G5 CPU モデルにのみ含まれ、QEMU がサポートする Intel CPU モデルには含まれません。
- Microarchitectural Data Sampling (MDS) 等の CPU ハードウェアの問題を軽減するには、他の CPU フラグを設定しなければならない場合があります。詳しくは、RHOS Mitigation for MDS ("Microarchitectural Data Sampling") Security Flaws を参照してください。

5. Meltdown に対する保護の適用後にパフォーマンスが失われないようにするには、CPU 機能フラグ **NovaLibvirtCPUModelExtraFlags** を設定して **+pcid** を指定します。

```
parameter_defaults:
  ComputeParameters:
    NovaLibvirtCPUMode: 'custom'
    NovaLibvirtCPUModels: 'Haswell-noTSX'
    NovaLibvirtCPUModelExtraFlags: 'vmx, pdpe1gb, +pcid'
```

ヒント

詳しくは、Reducing the performance impact of **Meltdown** CVE fixes for OpenStack guests with "PCID" CPU feature flag を参照してください。

6. **NovaSchedulerEnabledFilters** パラメーターにまだ **NUMATopologyFilter** がなければ、このフィルターを追加します。

7. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

5.4.1. インスタンス用のヒュージページフレーバーの作成

クラウドユーザーがヒュージページを使用するインスタンスを作成できるようにするには、インスタンス起動用の **hw:mem_page_size** 追加スペックキーが設定されたフレーバーを作成します。

前提条件

- コンピュートノードがヒュージページに対応する設定である。詳細は、コンピュートノードで Huge Page を設定する を参照してください。

手順

1. ヒュージページを要求するインスタンス用のフレーバーを作成します。

```
$ openstack flavor create --ram <size_mb> --disk <size_gb> \
 --vcpus <no_reserved_vcpus> huge_pages
```

2. ヒュージページを要求するには、フレーバーの **hw:mem_page_size** 属性を必要なサイズに設定します。

```
$ openstack flavor set huge_pages --property hw:mem_page_size=<page_size>
```

   - **<page_size>** は、次に示す有効な値のいずれかに置き換えます。
     - **large**: ホストでサポートされる最大のページサイズを選択します。x86_64 システムでは 2 MB または 1 GB です。
     - **small**: (デフォルト) ホストでサポートされる最小のページサイズを選択します。X86_64 システムでは、4 kB (通常のページ) です。
     - **any** : イメージに設定された **hw_mem_page_size** を使用してページサイズを選択します。ページサイズがイメージで指定されていない場合は、libvirt ドライバーで決定される、利用可能な最大のページサイズを選択します。
     - **<pagesize>**: ワークロードに具体的な要件がある場合、ページサイズを明示的に設定します。ページサイズには整数値を使用し、kB またはその他の標準単位で指定します。(例: 4kB、2MB、2048、1GB)。

3. フレーバーによりヒュージページを使用するインスタンスが作成されることを確認するには、新しいフレーバーを使用してインスタンスを起動します。

```
$ openstack server create --flavor huge_pages \
 --image <image> huge_pages_instance
```

   Compute スケジューラーは、インスタンスのメモリーをサポートするのに十分なサイズの空きヒュージページを持つホストを特定します。スケジューラーが十分なページを持つホストおよび NUMA ノードを検出できない場合、リクエストは失敗して **NoValidHost** エラーが報告されます。

5.4.2. 初回起動時に複数の Huge Page フォルダーをマウント

Compute サービス (nova) を設定して、初回起動プロセスの一環として複数のページサイズを処理することができます。初回起動プロセスでは、初めてノードを起動する際に heat テンプレート設定をすべてのノードに追加します。これ以降は (たとえば、オーバークラウドスタックの更新時)、これらのテンプレートを追加してもこれらのスクリプトは実行されません。

手順

1. Huge Page フォルダーのマウントを作成するためにスクリプトを実行する初回起動テンプレートファイル **hugepages.yaml** を作成します。**OS::TripleO::MultipartMime** リソースタイプを使用して、この設定スクリプトを送信することができます。

```
heat_template_version: <version>

description: >
  Huge pages configuration

resources:
  userdata:
    type: OS::Heat::MultipartMime
    properties:
      parts:
      - config: {get_resource: hugepages_config}

  hugepages_config:
    type: OS::Heat::SoftwareConfig
    properties:
      config: |
        #!/bin/bash
        hostname | grep -qiE 'co?mp' || exit 0
        systemctl mask dev-hugepages.mount || true
        for pagesize in 2M 1G;do
          if ! [ -d "/dev/hugepages${pagesize}" ]; then
            mkdir -p "/dev/hugepages${pagesize}"
            cat << EOF > /etc/systemd/system/dev-hugepages${pagesize}.mount
        [Unit]
        Description=${pagesize} Huge Pages File System
        Documentation=https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt
        Documentation=https://www.freedesktop.org/wiki/Software/systemd/APIFileSystems
        DefaultDependencies=no
        Before=sysinit.target
        ConditionPathExists=/sys/kernel/mm/hugepages
        ConditionCapability=CAP_SYS_ADMIN
        ConditionVirtualization=!private-users

        [Mount]
        What=hugetlbfs
        Where=/dev/hugepages${pagesize}
        Type=hugetlbfs
        Options=pagesize=${pagesize}

        [Install]
        WantedBy = sysinit.target
        EOF
          fi
        done
        systemctl daemon-reload
        for pagesize in 2M 1G;do
          systemctl enable --now dev-hugepages${pagesize}.mount
        done
```

```
outputs:
  OS::stack_id:
    value: {get_resource: userdata}
```

このテンプレートの **config** スクリプトは、以下のタスクを実行します。

a. **'co?mp'** に一致するホスト名を指定することで、ホストをフィルタリングして、huge page フォルダーのマウントを作成します。必要に応じて、特定のコンピュートの filter grep パターンを更新できます。

b. デフォルトの **dev-hugepages.mount systemd** ユニットファイルをマスクし、ページサイズを使用して新規マウントを作成できるようにします。

c. フォルダーが最初に作成されていることを確認します。

d. **pagesize** ごとに **systemd** マウントユニットを作成します。

e. 最初のループ後に **systemd daemon-reload** を実行し、新たに作成されたユニットファイルを追加します。

f. 2M および 1G のページサイズのマウントを有効にします。このループを更新して、必要に応じて追加のページサイズを含めることができます。

2. オプション: **/dev** フォルダーは、自動的に **nova_compute** および **nova_libvirt** のコンテナーにバインドマウントされます。Huge Page マウントに別の宛先を使用した場合は、そのマウントを **nova_compute** および **nova_libvirt** コンテナーに渡す必要があります。

```
parameter_defaults
  NovaComputeOptVolumes:
    - /opt/dev:/opt/dev
  NovaLibvirtOptVolumes:
    - /opt/dev:/opt/dev
```

3. **~/templates/firstboot.yaml** 環境ファイルの **OS::TripleO::NodeUserData** リソースタイプとして heat テンプレートを登録します。

```
resource_registry:
  OS::TripleO::NodeUserData: ./hugepages.yaml
```

重要

NodeUserData リソースを登録することができるのは、1 つのリソースにつき 1 つの heat テンプレートだけです。別の heat テンプレートに登録すると、使用する heat テンプレートがそのテンプレートに変わります。

4. 初回起動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/firstboot.yaml \
  ...
```

5.5. インスタンスにファイルベースのメモリーを使用するコンピュートノードの設定

ファイルベースのメモリーを使用して、コンピュートノードのメモリー容量を拡張することができます。この場合、libvirt メモリーバッキングディレクトリー内のファイルをインスタンスのメモリーとして割り当てます。インスタンスのメモリーとして使用できるホストディスクの容量、およびインスタンスメモリーファイルのディスク上の場所を設定することができます。

Compute サービスは、ファイルベースのメモリーに設定された容量をシステムメモリーの合計容量として Placement サービスに報告します。これにより、コンピュートノードは通常システムメモリー内で対応できるよりも多くのインスタンスをホストすることができます。

インスタンスにファイルベースのメモリーを使用するには、コンピュートノードでファイルベースのメモリーを有効にする必要があります。

制限

- ファイルベースのメモリーが有効なコンピュートノードとファイルベースのメモリーが有効ではないコンピュートノード間で、インスタンスのライブマイグレーションを行うことはできません。
- ファイルベースのメモリーとヒュージページとの間に互換性はありません。ファイルベースのメモリーが有効なコンピュートノード上で、ヒュージページを使用するインスタンスを起動することはできません。ホストアグリゲートを使用して、ヒュージページを使用するインスタンスがファイルベースのメモリーが有効なコンピュートノードに配置されないようにします。
- ファイルベースのメモリーとメモリーのオーバーコミットとの間に互換性はありません。
- **NovaReservedHostMemory** を使用してホストのプロセス用にメモリーを確保することはできません。ファイルベースのメモリーが使用されている場合、確保されるメモリーはファイルベースのメモリー用に用意されないディスク領域を表します。ファイルベースのメモリーは、キャッシュメモリーとして使用される RAM と共に、システムメモリーの合計として Placement サービスに報告されます。

前提条件

- ノードおよびノードが追加されたすべてのホストアグリゲートで、**NovaRAMAllocationRatio** を 1.0 に設定する必要があります。
- **NovaReservedHostMemory** を 0 に設定する必要があります。

手順

1. Compute 環境ファイルを開きます。
2. Compute 環境ファイルに以下のパラメーターを追加して、インスタンスの RAM 用に利用可能なホストディスク容量を MiB 単位で設定します。

```
parameter_defaults:
  NovaLibvirtFileBackedMemory: 102400
```

3. オプション: メモリーバッキングファイルを保存するディレクトリーを設定するには、Compute 環境ファイルに **QemuMemoryBackingDir** パラメーターを設定します。設定しなければ、メモリーバッキングディレクトリーはデフォルトの **/var/lib/libvirt/qemu/ram/** に設定されます。

注記

デフォルトのディレクトリー (**/var/lib/libvirt/qemu/ram/**) またはそれより上の階層のディレクトリーに、バッキングストアを配置する必要があります。

バッキングストアのホストディスクを変更することもできます。詳細は、メモリーバッキングディレクトリーのホストディスクの変更 を参照してください。

4. 更新内容を Compute 環境ファイルに保存します。

5. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

5.5.1. メモリーバッキングディレクトリーのホストディスクの変更

デフォルトのプライマリーディスクから別のディスクに、メモリーバッキングディレクトリーを変更することができます。

手順

1. 代替のバッキングデバイスにファイルシステムを作成します。たとえば、**/dev/sdb** に **ext4** ファイルシステムを作成するには、以下のコマンドを入力します。

```
# mkfs.ext4 /dev/sdb
```

2. バッキングデバイスをマウントします。たとえば、**/dev/sdb** をデフォルトの libvirt メモリーバッキングディレクトリーにマウントするには、以下のコマンドを入力します。

```
# mount /dev/sdb /var/lib/libvirt/qemu/ram
```

注記

マウントポイントは、**QemuMemoryBackingDir** パラメーターの値と一致する必要があります。

5.6. インスタンスのメモリーを暗号化するための AMD SEV コンピュートノードの設定

クラウドユーザーは、メモリーの暗号化が有効な SEV 対応コンピュートノード上で動作するインスタンスを作成することができます。

この機能は、2nd Gen AMD EPYC™ 7002 Series (Rome) から利用できます。

クラウドユーザーがメモリーの暗号化を使用するインスタンスを作成できるようにするには、以下のタスクを実施する必要があります。

1. メモリーの暗号化用に AMD SEV コンピュートノードを指定する。

2. メモリーの暗号化用にコンピュートノードを設定する。

3. オーバークラウドをデプロイする。

4. メモリーを暗号化してインスタンスを起動するためのフレーバーまたはイメージを作成する。

ヒント

AMD SEV ハードウェアが制限されている場合は、ホストアグリゲートを設定して AMD SEV コンピュートノードでのスケジューリングを最適化することもできます。メモリーの暗号化を要求するインスタンスのみを AMD SEV コンピュートノードにスケジュールするには、AMD SEV ハードウェアを持つコンピュートノードのホストアグリゲートを作成し、Compute スケジューラーがメモリーの暗号化を要求するインスタンスのみをホストアグリゲートに配置するように設定します。詳細は、Creating and managing host aggregates および Filtering by isolating host aggregates を参照してください。

5.6.1. Secure Encrypted Virtualization (SEV)

AMD が提供する Secure Encrypted Virtualization (SEV) は、動作中の仮想マシンインスタンスが使用している DRAM のデータを保護します。SEV は、各インスタンスのメモリーを一意の鍵で暗号化します。

SEV は、不揮発性メモリーテクノロジー (NVDIMM) を使用する際にセキュリティーを強化します。ハードドライブと同様に、NVDIMM チップはデータが保存されたままシステムから物理的に取り外すことができるためです。暗号化しないと、機密データ、パスワード、またはシークレットキー等の保存された情報が危険にさらされる可能性があります。

詳細は、**AMD Secure Encrypted Virtualization (SEV)** のドキュメントを参照してください。

メモリー暗号化を使用する場合のインスタンスの制限

- メモリー暗号化を使用するインスタンスのライブマイグレーションや、インスタンスを一時停止および再開することはできません。
- PCI パススルーを使用して、メモリーの暗号化を使用するインスタンス上のデバイスに直接アクセスすることはできません。
- kernel-4.18.0-115.el8 (RHEL-8.1.0) 以前の Red Hat Enterprise Linux (RHEL) カーネルでメモリー暗号化を使用するインスタンスのブートディスクとして **virtio-blk** を使用することはできません。

  注記

  virtio-scsi または **SATA** をブートディスクとして使用することができます。また、ブートディスク以外の用途に **virtio-blk** を使用することができます。

- 暗号化されたインスタンスで実行されているオペレーティングシステムは、SEV をサポートしている必要があります。詳細は、Red Hat ナレッジベースのソリューション Enabling AMD Secure Encrypted Virtualization in RHEL 8 を参照してください。
- SEV をサポートするマシンでは、暗号鍵を格納するためのメモリーコントローラーのスロット数に制限があります。動作中のメモリーが暗号化された各インスタンスは、これらのスロットの 1 つを使用します。したがって、同時に実行できるメモリー暗号化インスタンスの数は、メモリーコントローラーのスロット数に制限されます。たとえば、1st Gen AMD EPYC™ 7001 Series (Naples) の場合、制限は 16 で、2nd Gen AMD EPYC™ 7002 Series (Rome) では上限は 255 です。

- メモリー暗号化を使用するインスタンスの RAM ページの固定Compute サービスはこれらのページをスワップすることができないため、メモリーが暗号化されたインスタンスをホストするコンピュートノードでメモリーをオーバーコミットすることはできません。
- NUMA ノードが複数あるインスタンスでは、メモリーの暗号化を使用することはできません。

5.6.2. メモリー暗号化用 AMD SEV コンピュートノードの指定

メモリーの暗号化を使用するインスタンス用に AMD SEV コンピュートノードを指定するには、AMD SEV ロールを設定するための新規ロールファイルを作成し、メモリーの暗号化のためにコンピュートノードをタグ付けするための AMD SEV リソースクラスを持つベアメタルノードを設定する必要があります。

注記

以下の手順は、まだプロビジョニングされていない新しいオーバークラウドノードに適用されます。すでにプロビジョニングされている既存のオーバークラウドノードにリソースクラスを割り当てるには、スケールダウン手順を使用してノードのプロビジョニングを解除してから、スケールアップ手順を使用して新しいリソースクラスの割り当てでノードを再プロビジョニングする必要があります。詳細は、オーバークラウドノードのスケーリング を参照してください。

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. オーバークラウドに必要なその他のロールに加えて **ComputeAMDSEV** ロールが含まれる新しいロールデータファイルを生成します。以下の例では、ロールデータファイル **roles_data_amd_sev.yaml** を生成します。これには、**Controller** および **ComputeAMDSEV** ロールが含まれます。

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_amd_sev.yaml \
 Compute:ComputeAMDSEV Controller
```

4. **roles_data_amd_sev.yaml** を開き、以下のパラメーターおよびセクションを編集または追加します。

セクション/パラメーター	現在の値	新しい値
ロールのコメント	**Role: Compute**	**Role: ComputeAMDSEV**
ロール名	**Compute**	**name: ComputeAMDSEV**
description	**Basic Compute Node role**	**AMD SEV Compute Node role**

セクション/パラメーター	現在の値	新しい値
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputeamdsev-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-amd-sev.yaml**

5. オーバークラウド用の AMD SEV コンピュートノードをノード定義のテンプレート **node.json** または **node.yaml** に追加して、そのノードを登録します。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** ガイドの オーバークラウドのノードの登録 を参照してください。

6. ノードのハードウェアを検査します。

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

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの ベアメタルノードハードウェアのインベントリーの作成 を参照してください。

7. メモリーの暗号化用に指定する各ベアメタルノードに、カスタムの AMD SEV リソースクラスをタグ付けします。

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.AMD-SEV <node>
```

<node> は、ベアメタルノードの名前または ID に置き換えます。

8. ノード定義ファイル **overcloud-baremetal-deploy.yaml** に **ComputeAMDSEV** ロールを追加し、予測ノード配置、リソースクラス、ネットワークトポロジー、またはノードに割り当てるその他の属性を定義します。

```
- name: Controller
  count: 3
- name: Compute
  count: 3
- name: ComputeAMDSEV
  count: 1
  defaults:
    resource_class: baremetal.AMD-SEV
    network_config:
      template: /home/stack/templates/nic-config/myRoleTopology.j2 (1)
```

(1) 既存のネットワークトポロジーを再利用するか、ロール用の新しいカスタムネットワークインターフェイステンプレートを作成できます。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** ガイドの カスタムネットワークインターフェイステンプレート を参照してください。**network_config** プロパティーを使用してネットワーク定義を定義しない場合、デフォルトのネットワーク定義が使用されます。

ノード定義ファイルでノード属性を設定するために使用できるプロパティーについて詳しくは、ベアメタルノードのプロビジョニング属性 を参照してください。ノード定義ファイルの例は、ノード定義ファイルの例 を参照してください。

9. プロビジョニングコマンドを実行して、ロールの新しいノードをプロビジョニングします。

```
(undercloud)$ openstack overcloud node provision \
--stack <stack> \
[--network-config \]
--output /home/stack/templates/overcloud-baremetal-deployed.yaml \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- **<stack>** を、ベアメタルノードがプロビジョニングされるスタックの名前に置き換えます。指定しない場合、デフォルトは **overcloud** です。
- **--network-config** オプションの引数を含めて、**cli-overcloud-node-network-config.yaml** Ansible Playbook にネットワーク定義を提供します。**network_config** プロパティーを使用してネットワーク定義を定義しない場合、デフォルトのネットワーク定義が使用されます。

10. 別のターミナルでプロビジョニングの進捗をモニタリングします。プロビジョニングが成功すると、ノードの状態が **available** から **active** に変わります。

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

11. **--network-config** オプションを指定してプロビジョニングコマンドを実行しなかった場合は、**network-environment.yaml** ファイルで **<Role>NetworkConfigTemplate** パラメーターを設定して、NIC テンプレートファイルを指すようにします。

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputeAMDSEVNetworkConfigTemplate: /home/stack/templates/nic-
configs/<amd_sev_net_top>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

<amd_sev_net_top> を **ComputeAMDSEV** ロールのネットワークトポロジーが含まれるファイルの名前に置き換えます。たとえば、デフォルトのネットワークトポロジーを使用する場合は **compute.yaml** です。

5.6.3. メモリー暗号化用 AMD SEV コンピュートノードの設定

クラウドユーザーがメモリーの暗号化を使用するインスタンスを作成できるようにするには、AMD SEV ハードウェアを持つコンピュートノードを設定する必要があります。

注記

RHOSP OSP17.0 以降では、Q35 がデフォルトのマシンタイプです。Q35 マシンタイプは PCIe ポートを使用します。heat パラメーター **NovaLibvirtNumPciePorts** を設定すると、PCIe ポートデバイスの数を管理できます。PCIe ポートに接続できるデバイスの数は、以前のバージョンで実行されているインスタンスよりも少なくなります。より多くのデバイスを使用する場合は、**hw_disk_bus=scsi** または **hw_scsi_model=virtio-scsi** イメージ属性を使用する必要があります。詳細は、仮想ハードウェアのメタデータプロパティー を 参照してください。

前提条件

- デプロイメントには、SEV に対応する AMD ハードウェア (AMD EPYC CPU 等) 上で実行されるコンピュートノードが含まれている必要があります。以下のコマンドを使用して、デプロイメントが SEV に対応しているかどうか判断することができます。

```
$ lscpu | grep sev
```

手順

1. Compute 環境ファイルを開きます。
2. オプション: AMD SEV コンピュートノードが同時にホストできるメモリーが暗号化されたインスタンス数の最大値を指定するには、以下の設定を Compute 環境ファイルに追加します。

```
parameter_defaults:
  ComputeAMDSEVExtraConfig:
    nova::config::nova_config:
      libvirt/num_memory_encrypted_guests:
        value: 15
```

注記

libvirt/num_memory_encrypted_guests パラメーターのデフォルト値は **none** です。カスタム値を設定しない場合、AMD SEV コンピュートノードは、ノードが同時にホストできるメモリーが暗号化されたインスタンスの数に制限を設けません。代わりに、ハードウェアが、AMD SEV コンピュートノードが同時にホストできるメモリーが暗号化されたインスタンス数の最大値を決定します。この場合、メモリーが暗号化されたインスタンスの一部が起動に失敗する可能性があります。

3. (オプション) デフォルトでは、すべての x86_64 イメージが q35 マシン種別を使用するように指定するには、以下の設定を Compute 環境ファイルに追加します。

```
parameter_defaults:
  ComputeAMDSEVParameters:
    NovaHWMachineType: x86_64=q35
```

このパラメーターの値を指定する場合、すべての AMD SEV インスタンスイメージで **hw_machine_type** 属性を **q35** に設定する必要はありません。

4. AMD SEV コンピュートノードがホストレベルのサービスが機能するのに十分なメモリーが確保するようにするには、考え得る AMD SEV インスタンスごとに 16 MB を追加します。

```
parameter_defaults:
  ComputeAMDSEVParameters:
    ...
    NovaReservedHostMemory: <libvirt/num_memory_encrypted_guests * 16>
```

5. AMD SEV コンピュートノード用のカーネルパラメーターを設定します。

```
parameter_defaults:
  ComputeAMDSEVParameters:
    ...
```

```
KernelArgs: "hugepagesz=1GB hugepages=32 default_hugepagesz=1GB
mem_encrypt=on kvm_amd.sev=1"
```

注記

KernelArgs パラメーターをロールの設定に初めて追加すると、オーバークラウドノードが自動的に再起動されます。必要に応じて、ノードの自動再起動を無効にし、代わりに各オーバークラウドのデプロイ後にノードの再起動を手動で実行できます。詳細は、**KernelArgs** を定義するための手動でのノード再起動の設定を参照してください。

6. 更新内容を Compute 環境ファイルに保存します。

7. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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] \
 -r /home/stack/templates/roles_data_amd_sev.yaml \
 -e /home/stack/templates/network-environment.yaml \
 -e /home/stack/templates/<compute_environment_file>.yaml \
 -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
 -e /home/stack/templates/node-info.yaml
```

5.6.4. メモリー暗号化用のイメージの作成

オーバークラウドに AMD SEV コンピュートノードが含まれる場合、AMD SEV インスタンスイメージを作成することができます。クラウドユーザーはこのイメージを使用して、メモリーが暗号化されたインスタンスを起動することができます。

注記

RHOSP OSP17.0 以降では、Q35 がデフォルトのマシンタイプです。Q35 マシンタイプは PCIe ポートを使用します。heat パラメーター **NovaLibvirtNumPciePorts** を設定すると、PCIe ポートデバイスの数を管理できます。PCIe ポートに接続できるデバイスの数は、以前のバージョンで実行されているインスタンスよりも少なくなります。より多くのデバイスを使用する場合は、**hw_disk_bus=scsi** または **hw_scsi_model=virtio-scsi** イメージ属性を使用する必要があります。詳細は、仮想ハードウェアのメタデータプロパティー を 参照してください。

手順

1. メモリー暗号化用の新規イメージの作成

```
(overcloud)$ openstack image create ...  \
 --property hw_firmware_type=uefi amd-sev-image
```

注記

既存のイメージを使用する場合、イメージの **hw_firmware_type** 属性が **uefi** に設定されている必要があります。

2. (オプション) イメージに属性 **hw_mem_encryption=True** を追加して、イメージで AMD SEV のメモリー暗号化を有効にします。

```
(overcloud)$ openstack image set \
 --property hw_mem_encryption=True amd-sev-image
```

ヒント

フレーバーでメモリー暗号化を有効にすることができます。詳細は、Creating a flavor for memory encryption を参照してください。

3. オプション: コンピュートノード設定のマシン種別がまだ **q35** に設定されていない場合には、そのように設定します。

```
(overcloud)$ openstack image set \
 --property hw_machine_type=q35 amd-sev-image
```

4. オプション: SEV 対応ホストアグリゲートでメモリーが暗号化されたインスタンスをスケジュールするには、イメージの追加スペックに以下の特性を追加します。

```
(overcloud)$ openstack image set \
 --property trait:HW_CPU_X86_AMD_SEV=required amd-sev-image
```

ヒント

フレーバーでこの特性を指定することもできます。詳細は、Creating a flavor for memory encryption を参照してください。

5.6.5. メモリー暗号化用のフレーバーの作成

オーバークラウドに AMD SEV コンピュートノードが含まれる場合、1 つまたは複数の AMD SEV フレーバーを作成することができます。クラウドユーザーはこのイメージを使用して、メモリーが暗号化されたインスタンスを起動することができます。

注記

AMD SEV フレーバーは、**hw_mem_encryption** 属性がイメージで設定されていない場合にのみ必要です。

手順

1. メモリー暗号化用のフレーバーを作成します。

```
(overcloud)$ openstack flavor create --vcpus 1 --ram 512 --disk 2 \
 --property hw:mem_encryption=True m1.small-amd-sev
```

2. SEV 対応ホストアグリゲートでメモリーが暗号化されたインスタンスをスケジュールするには、フレーバーの追加スペックに以下の特性を追加します。

```
(overcloud)$ openstack flavor set \
 --property trait:HW_CPU_X86_AMD_SEV=required m1.small-amd-sev
```

5.6.6. メモリーが暗号化されたインスタンスの起動

メモリーの暗号化を有効にして AMD SEV コンピュートノードでインスタンスを起動できることを確認するには、メモリー暗号化フレーバーまたはイメージを使用してインスタンスを作成します。

手順

1. AMD SEV フレーバーまたはイメージを使用してインスタンスを作成します。以下の例では、メモリー暗号化用のフレーバーの作成 で作成したフレーバーおよび メモリー暗号化用のイメージの作成 で作成したイメージを使用してインスタンスを作成します。

```
(overcloud)$ openstack server create --flavor m1.small-amd-sev \
 --image amd-sev-image amd-sev-instance
```

2. クラウドユーザーとしてインスタンスにログインします。

3. インスタンスがメモリーの暗号化を使用していることを確認するには、インスタンスから以下のコマンドを入力します。

```
$ dmesg | grep -i sev
AMD Secure Encrypted Virtualization (SEV) active
```

第6章 COMPUTE サービスのストレージの設定

Compute サービスが Image (glance) サービスからコピーしてコンピュートノード上でローカルにキャッシュしたベースイメージから、インスタンスを作成します。インスタンスのバックエンドであるインスタンスディスクも、ベースイメージに基づいています。

Compute サービスを設定して、一時インスタンスのディスクデータをホストのコンピュートノード上にローカルに保存するか、NFS 共有または Ceph クラスターでリモートで保存することができます。あるいは、Compute サービスを設定して、Block Storage (Cinder) サービスが提供する永続ストレージにインスタンスディスクデータを保存することもできます。

環境のイメージキャッシュを設定し、インスタンスディスクのパフォーマンスおよびセキュリティーを設定することができます。Image サービス (glance) がバックエンドとして Red Hat Ceph RADOS Block Device (RBD) を使用する場合に、Image サービス API を使用せずに RBD イメージリポジトリーから直接イメージをダウンロードするように Compute サービスを設定することもできます。

6.1. イメージキャッシュの設定オプション

以下の表で詳細を説明するパラメーターを使用して、Compute サービスがどのようにコンピュートノードでのイメージのキャッシュを実装して管理するかを設定します。

表6.1 Compute (nova) サービスのイメージキャッシュパラメーター

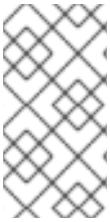
設定方法	パラメーター	説明
Puppet	**nova::compute::image_cache::manager_interval**	コンピュートノードでのベースイメージのキャッシュを管理するイメージキャッシュマネージャーの実行間で待機する時間を秒単位で指定します。**nova::compute::image_cache::remove_unused_base_images** が **True** に設定されている場合、Compute サービスはこの期間を使用して使用されていないキャッシュイメージの自動削除を実行します。 デフォルトのメトリック間隔である 60 秒で実行するには、**0** に設定します (推奨されません)。イメージキャッシュマネージャーを無効にするには、**-1** に設定します。 デフォルト: **2400**

設定方法	パラメーター	説明
Puppet	**nova::compute::image_cache::precache_concurrency**	イメージを並行して事前キャッシュできるコンピュートノードの数の最大値を指定します。 **注記** ● このパラメーターに大きな数値を設定すると、事前キャッシュの処理が遅くなり、Image サービスで DDoS が発生する場合があります。 ● このパラメーターに小さな数値を設定すると、Image サービスの負荷は軽減されますが、事前キャッシュがより連続的な操作で実行されるため、完了までの実行時間が長くなる場合があります。 デフォルト: **1**
Puppet	**nova::compute::image_cache::remove_unused_base_images**	**manager_interval** を使用して設定された間隔で使用されていないベースイメージをキャッシュから自動的に削除するには、**True** に設定します。**NovaImageCacheTTL** を使用して指定された期間アクセスされなかった場合、イメージは使用されていないと定義されます。 デフォルト: **True**
Puppet	**nova::compute::image_cache::remove_unused_resized_minimum_age_seconds**	未使用のサイズ変更されたベースイメージをキャッシュから削除する最短の期間を指定します (秒単位)。未使用のサイズ変更されたベースイメージは、この期間削除されません。無効にする場合は **undef** に設定します。 デフォルト: **3600**

設定方法	パラメーター	説明
Puppet	**nova::compute::image_cache::subdirectory_name**	キャッシュされたイメージを保存するフォルダーの名前を、**$instances_path** との相対パスで指定します。 デフォルト: **_base**
heat	**NovaImageCacheTTL**	コンピュートノード上のどのインスタンスもイメージを使用しなくなった場合に、Compute サービスがイメージをキャッシュし続ける期間を秒単位で指定します。Compute サービスは、コンピュートノードにキャッシュされたイメージのうち、ここで設定したライフタイムを過ぎたイメージを、再度必要になるまでキャッシュディレクトリーから削除します。 デフォルト: 86400 (24 時間)

6.2. インスタンスの一時ストレージ属性の設定オプション

以下の表で詳細を説明するパラメーターを使用して、インスタンスが使用する一時ストレージのパフォーマンスおよびセキュリティーを設定します。



注記

Red Hat OpenStack Platform (RHOSP) は、インスタンスディスクの LVM イメージ種別をサポートしません。したがって、インスタンスの削除時に一時ディスクを消去する **[libvirt]/volume_clear** 設定オプションは、インスタンスディスクイメージ種別が LVM の場合にのみ適用されるため、サポートされません。

表6.2 Compute (nova) サービスのインスタンス一時ストレージパラメーター

設定方法	パラメーター	説明
Puppet	**nova::compute::default_ephemeral_format**	新規の一時ボリュームに使用されるデフォルトの形式を指定します。以下の有効な値のいずれかに設定します。 ● **ext2** ● **ext3** ● **ext4** **ext4** 形式では、ext3 に比べ、サイズの大きい新規ディスクを初期化する時間が大幅に短縮されます。 デフォルト: **ext4**

設定方法	パラメーター	説明
Puppet	**nova::compute::force_raw_images**	raw 以外の形式でキャッシュされたベースイメージを raw 形式に変換するには、**True** に設定します。raw イメージ形式は、qcow2 等の他のイメージ形式よりも多くの領域を使用します。raw 以外のイメージ形式は、圧縮により多くの CPU パワーを使用します。**False** に設定すると、CPU のボトルネックを防ぐために、Compute サービスは圧縮時にベースイメージからすべての圧縮を削除します。システムの I/O 処理が遅い場合や空き容量が少ない場合は、入力の帯域幅を削減するために **False** に設定します。 デフォルト: **True**
Puppet	**nova::compute::use_cow_images**	インスタンスのディスクに qcow2 形式の CoW (Copy on Write) イメージを使用するには、**True** に設定します。CoW を使用する場合には、バッキングストアとホストのキャッシュによっては、各インスタンスが独自のコピー上で稼働することで、並行処理が改善される場合があります。 raw 形式を使用するには、**False** に設定します。raw 形式は、ディスクイメージの共通の部分により多くの領域を使用します。 デフォルト: **True**
Puppet	**nova::compute::libvirt::preallocate_images**	インスタンスディスクに対する事前割り当てモードを指定します。以下の有効な値のいずれかに設定します。 ● **none**: インスタンスの起動時にはストレージがプロビジョニングされません。 ● **space**: インスタンスのディスクイメージで **fallocate(1)** を実行することで、Compute サービスはインスタンスの起動時にストレージを完全に割り当てます。これにより CPU のオーバーヘッドおよびファイルの断片化が軽減され、I/O パフォーマンスが向上し、要求されたディスク領域を確保するのに役立ちます。 デフォルト: **none**

設定方法	パラメーター	説明
hieradata override	**DEFAULT/resize_fs_using_block_device**	ブロックデバイスを使用してイメージにアクセスすることで、ベースイメージのサイズを直接変更できるようにするには、**True** に設定します。これは、cloud-init のバージョンがより古いイメージ (それ自体ではサイズの変更ができない) でのみ必要です。 このパラメーターによりイメージの直接マウントが可能になるため (セキュリティー上の理由により無効にされる場合がある)、このパラメーターはデフォルトでは有効化されていません。 デフォルト: **False**

設定方法	パラメーター	説明
hieradata override	**[libvirt]/images_type**	インスタンスのディスクに使用するイメージ種別を指定します。以下の有効な値のいずれかに設定します。 ● **raw** ● **qcow2** ● **flat** ● **rbd** ● **default** **注記** RHOSP は、インスタンスディスクの LVM イメージ種別をサポートしません。 **default** 以外の有効な値に設定すると、イメージ種別は **use_cow_images** の設定よりも優先されます。**default** が指定されると、**use_cow_images** の設定によりイメージ種別が決定されます。 ● **use_cow_images** が **True**(デフォルト) に設定されると、イメージ種別は **qcow2** になります。 ● **use_cow_images** を **False** に設定すると、イメージ種別は **Flat** になります。 デフォルト値は **NovaEnableRbdBackend** の設定により決定されます。 ● **NovaEnableRbdBackend: False** デフォルト: **default** ● **NovaEnableRbdBackend: True** デフォルト: **rbd**

6.3. 1 つのインスタンスにアタッチすることのできる最大ストレージデバイス数の設定

デフォルトでは、1 つのインスタンスにアタッチすることのできるストレージデバイスの数に制限はありません。多数のディスクデバイスをインスタンスにアタッチすると、インスタンスのパフォーマンスが低下する可能性があります。お使いの環境がサポートすることのできる限度に基づいて、インスタンスにアタッチできるデバイスの最大数を調整できます。インスタンスがサポートするストレージディスクの数は、ディスクが使用するバスにより異なります。たとえば、IDE ディスクバスでは、アタッチされるデバイスは 4 つに制限されます。マシン種別が Q35 のインスタンスには、最大 500 のディスクデバイスをアタッチできます。

注記

RHOSP OSP17.0 以降では、Q35 がデフォルトのマシンタイプです。Q35 マシンタイプは PCIe ポートを使用します。heat パラメーター **NovaLibvirtNumPciePorts** を設定すると、PCIe ポートデバイスの数を管理できます。PCIe ポートに接続できるデバイスの数は、以前のバージョンで実行されているインスタンスよりも少なくなります。より多くのデバイスを使用する場合は、**hw_disk_bus=scsi** または **hw_scsi_model=virtio-scsi** イメージ属性を使用する必要があります。詳細は、仮想ハードウェアのメタデータプロパティー を 参照してください。

警告

- アクティブなインスタンスを持つコンピュートノードで **NovaMaxDiskDevicesToAttach** パラメーターの値を変更した場合に、すでにインスタンスにアタッチされているデバイスの数よりも最大数が小さいと、再ビルドが失敗する可能性があります。たとえば、インスタンス A に 26 のデバイスがアタッチされている場合に、**NovaMaxDiskDevicesToAttach** を 20 に変更すると、インスタンス A を再ビルドする要求は失敗します。
- コールドマイグレーション時には、設定されたストレージデバイスの最大数は、移行する元のインスタンスにのみ適用されます。移動前に移行先が確認されることはありません。つまり、コンピュートノード A に 26 のディスクデバイスがアタッチされていて、コンピュートノード B の最大ディスクデバイスアタッチ数が 20 に設定されている場合に、26 のデバイスがアタッチされたインスタンスをコンピュートノード A からコンピュートノード B に移行するコールドマイグレーションの操作は成功します。ただし、これ以降、コンピュートノード B でインスタンスを再ビルドする要求は失敗します。すでにアタッチされているデバイスの数 26 が、設定された最大値の 20 を超えているためです。

注記

設定されたストレージデバイスの最大数は、退避オフロード中のインスタンスには適用されません。これらのインスタンスはコンピュートノードを持たないためです。

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. 新しい環境ファイルを作成するか、既存の環境ファイルを開きます。
4. 次の設定を環境ファイルに追加して、単一インスタンスにアタッチできるストレージデバイスの最大数の制限を設定します。

```
parameter_defaults:
```

```
...
NovaMaxDiskDevicesToAttach: <max_device_limit>
...
```

- **<max_device_limit>** を、インスタンスにアタッチできるストレージデバイスの最大数に置き換えます。

5. 更新を環境ファイルに保存します。

6. その他の環境ファイルとともに環境ファイルをスタックに追加して、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/<environment_file>.yaml
```

6.4. 共有インスタンスストレージの設定

デフォルトでは、インスタンスの起動時に、インスタンスのディスクはインスタンスディレクトリー **/var/lib/nova/instances** にファイルとして保存されます。Compute サービスの NFS ストレージバックエンドを設定して、これらのインスタンスファイルを共有 NFS ストレージに保存することができます。

前提条件

- NFSv4 以降を使用している。Red Hat OpenStack Platform (RHOSP) は、以前のバージョンの NFS をサポートしません。詳細は、Red Hat ナレッジベースのソリューションRHOS NFSv4-Only Support Notesを参照してください。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。

2. **stackrc** ファイルを取得します。

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

3. 共有インスタンスストレージを設定するための環境ファイルを作成します (例: **nfs_instance_disk_backend.yaml**)。

4. インスタンスファイル用に NFS バックエンドを設定するには、以下の設定を **nfs_instance_disk_backend.yaml** に追加します。

```
parameter_defaults:
  ...
  NovaNfsEnabled: True
  NovaNfsShare: <nfs_share>
```

<nfs_share> をインスタンスのファイルストレージ用にマウントする NFS 共有ディレクトリーに置き換えます (例: **'192.168.122.1:/export/nova'** または **'192.168.24.1:/var/nfs'**)。IPv6 を使用している場合は、二重と単一引用符の両方を使用してください (例: **'"[fdd0::1]:/export/nova"'**)。

5. オプション: NFS バックエンドストレージが有効な場合、NFS ストレージ用のデフォルトのマ

ウント SELinux コンテキストは **'context=system_u:object_r:nfs_t:s0'** です。以下のパラメーターを追加して、NFS インスタンスファイルストレージのマウントポイントのマウントオプションを変更します。

```
parameter_defaults:
  ...
  NovaNfsOptions: 'context=system_u:object_r:nfs_t:s0,<additional_nfs_mount_options>'
```

<additional_nfs_mount_options> を、NFS インスタンスファイルストレージに使用するマウントオプションのコンマ区切りリストに置き換えます。利用可能なマウントオプションの詳細は、**mount** の man ページを参照してください。

```
$ man 8 mount.
```

6. 更新を環境ファイルに保存します。

7. その他の環境ファイルと共に新しい環境ファイルをスタックに追加して、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/nfs_instance_disk_backend.yaml
```

6.5. RED HAT CEPH RADOS BLOCK DEVICE (RBD) からの直接イメージダウンロードの設定

Image サービス (glance) がバックエンドとして Red Hat Ceph RADOS Block Device (RBD) を使用し、Compute サービスがローカルのファイルベースの一時ストレージを使用する場合、Image サービス API を使用せずに RBD イメージリポジトリーから直接イメージをダウンロードするように Compute サービスを設定することができます。これにより、インスタンスのブート時にコンピュートノードイメージキャッシュにイメージをダウンロードする時間が短縮されます。これにより、インスタンスの起動時間が短縮されます。

前提条件

- Image サービスのバックエンドが、Red Hat Ceph RADOS Block Device (RBD) である。
- Compute サービスが、イメージキャッシュおよびインスタンスのディスクにローカルのファイルベースの一時ストアを使用している。

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. Compute 環境ファイルを開きます。

3. RBD バックエンドから直接イメージをダウンロードするには、以下の設定を Compute 環境ファイルに追加します。

```
parameter_defaults:
  ComputeParameters:
    NovaGlanceEnableRbdDownload: True
    NovaEnableRbdBackend: False
    ...
```

4. (オプション) Image サービスが複数の Red Hat Ceph Storage バックエンドを使用するように設定されている場合には、Compute 環境ファイルに以下の設定を追加して、イメージをダウンロードする RBD バックエンドを特定します。

```
parameter_defaults:
  ComputeParameters:
    NovaGlanceEnableRbdDownload: True
    NovaEnableRbdBackend: False
    NovaGlanceRbdDownloadMultistoreID: <rbd_backend_id>
    ...
```

<rbd_backend_id> を **GlanceMultistoreConfig** 設定のバックエンドを指定するために使用される ID(例: **rbd2_store**) に置き換えます。

5. 以下の設定を Compute 環境ファイルに追加して Image サービス RBD バックエンドを指定し、Compute サービスが Image サービス RBD バックエンドへの接続を待機する最大期間 (秒単位) を指定します。

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      glance/rbd_user:
        value: 'glance'
      glance/rbd_pool:
        value: 'images'
      glance/rbd_ceph_conf:
        value: '/etc/ceph/ceph.conf'
      glance/rbd_connect_timeout:
        value: '5'
```

6. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

7. Compute サービスが RBD から直接イメージをダウンロードすることを確認するには、インスタンスを作成してインスタンスのデバッグログで Attempting to export RBD image: のエントリーを確認します。

6.6. 関連情報

- Compute サービス (nova) の設定

第7章 インスタンスのスケジューリングと配置の設定

Compute スケジューラーサービスは、インスタンスの配置先となるコンピュートノードまたはホストアグリゲートを決定します。Compute (nova) サービスがインスタンスの起動または移動に関するリクエストを受け取ると、リクエスト、フレーバー、およびイメージで提供される仕様を使用して適切なホストを決定します。たとえば、フレーバーでは、ストレージディスクの種別や Intel CPU 拡張命令セットなど、インスタンスがホストに要求する特性を指定することができます。

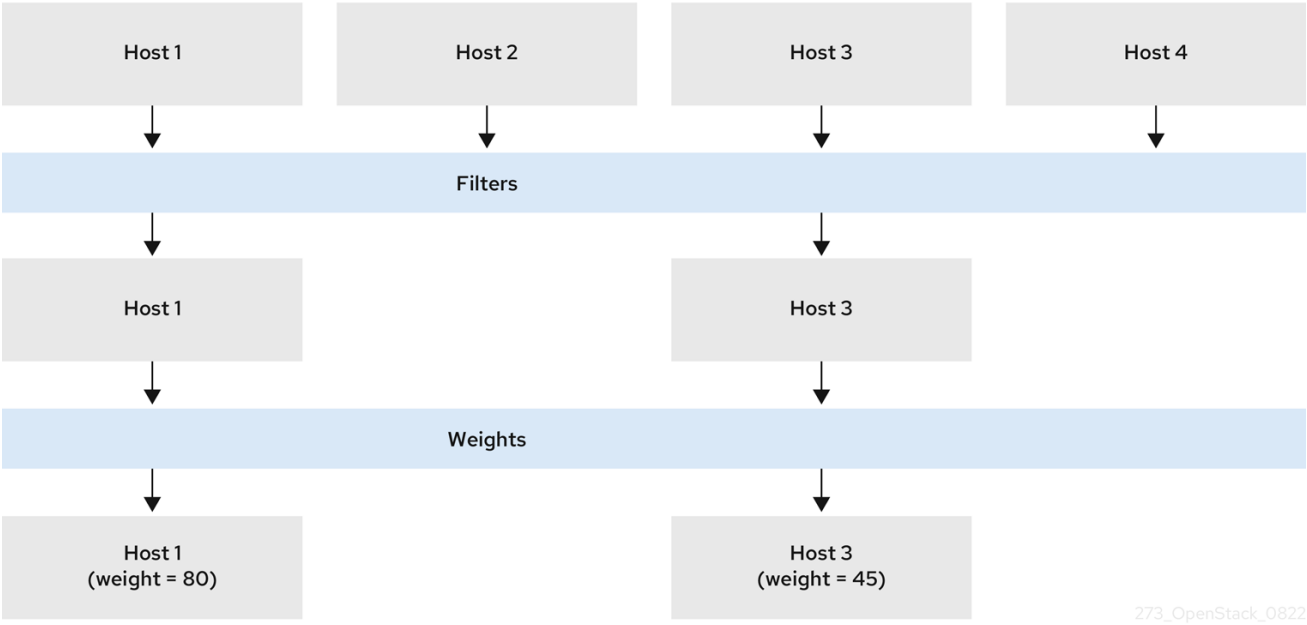
Compute スケジューラーサービスは、以下の順序で以下のコンポーネントの設定を使用して、インスタンスを起動または移動するコンピュートノードを決定します。

1. **Placement サービスのプレフィルター**: Compute スケジューラーサービスは Placement サービスを使用して、特定の属性に基づいて候補のコンピュートノードのセットを絞り込みます。たとえば、Placement サービスは無効な状態のコンピュートノードを自動的に除外します。

2. **フィルター**: Compute スケジューラーサービスは、これを使用してインスタンスを起動するコンピュートノードの初期セットを決定します。

3. **重み**: Compute スケジューラーサービスは、重み付けシステムを使用して絞り込まれたコンピュートノードの優先順位付けを行います。最も高い重みが最も優先されます。

下図では、絞り込み後、Host 1 および 3 が条件を満たしています。Host 1 の重みが最も高いため、スケジューリングで最も優先されます。



7.1. PLACEMENT サービスを使用した事前絞り込み

Compute サービス (nova) は、Placement サービスと協調してインスタンスを作成および管理します。Placement サービスは、コンピュートノード、共有ストレージプール、または IP 割り当てプールなど、リソースプロバイダーのインベントリーおよび使用状況、ならびに利用可能な仮想 CPU 数などのリソースの量的情報を追跡します。リソースの選択および消費を管理する必要があるサービスは、すべて Placement サービスを使用することができます。

Placement サービスは、リソースプロバイダーのストレージディスク特性の種別など、リソースの機能的情報とリソースプロバイダー間のマッピングも追跡します。

Placement サービスは、Placement サービスリソースプロバイダーインベントリーおよび特性に基づいて、候補のコンピュートノードセットにプレフィルターを適用します。以下の尺度に基づいてプレフィルターを作成することができます。

- サポートされるイメージ種別
- 特性
- プロジェクトまたはテナント
- アベイラビリティーゾーン

7.1.1. 要求されたイメージ種別のサポートによる絞り込み

インスタンスの起動に使用するイメージのディスク形式をサポートしないコンピュートノードを除外することができます。これは、環境の一時バックエンドに QCOW2 イメージをサポートしない Red Hat Ceph Storage が使用される場合に有用です。この機能を有効にすると、スケジューラーは QCOW2 イメージを使用するインスタンスの起動要求を Red Hat Ceph Storage ベースのコンピュートノードに送信しないようになります。

手順

1. Compute 環境ファイルを開きます。
2. インスタンスの起動に使用するイメージのディスク形式をサポートしないコンピュートノードを除外するには、Compute 環境ファイルの **NovaSchedulerQueryImageType** パラメーターを **True** に設定します。
3. 更新内容を Compute 環境ファイルに保存します。
4. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

7.1.2. リソースプロバイダー特性による絞り込み

各リソースプロバイダーには特性のセットがあります。特性は、ストレージディスクの種別や Intel CPU 拡張命令セットなど、リソースプロバイダーの機能的な要素です。

コンピュートノードは、その機能を特性として Placement サービスに報告します。インスタンスは、要求する特性またはリソースプロバイダーにあってはいけない特性を指定することができます。Compute スケジューラーは、これらの特性を使用して、インスタンスをホストするのに適したコンピュートノードまたはホストアグリゲートを特定することができます。

クラウドユーザーが特定の特性を持つホストにインスタンスを作成できるようにするには、特定の特性を要求または禁止するフレーバーを定義して、その特性を要求または禁止するイメージを作成することができます。

利用可能な特性のリストは、**os-traits** ライブラリー を参照してください。必要に応じて、カスタムの特性を作成することもできます。

関連情報

- 「カスタム特性とリソースクラスの宣言」

7.1.2.1. リソースプロバイダー特性を要求または禁止するイメージの作成

クラウドユーザーが特定の特性を持つホストでインスタンスを起動するのに使用することのできるインスタンスイメージを作成することができます。

手順

1. 新規イメージを作成します。

```
(overcloud)$ openstack image create ... trait-image
```

2. ホストまたはホストアグリゲートに必要な特性を識別します。既存の特性を選択するか、新たな特性を作成することができます。

   - 既存の特性を使用するには、既存特性のリストを表示して特性名を取得します。

```
(overcloud)$ openstack --os-placement-api-version 1.6 trait list
```

   - 新規特性を作成するには、以下のコマンドを入力します。

```
(overcloud)$ openstack --os-placement-api-version 1.6 trait \
 create CUSTOM_TRAIT_NAME
```

   カスタムの特性は接頭辞 **CUSTOM_** で始まり、A から Z までの文字、0 から 9 までの数字、およびアンダースコア_だけを使用する必要があります。

3. 各ホストの既存のリソースプロバイダー特性を収集します。

```
(overcloud)$ existing_traits=$(openstack --os-placement-api-version 1.6 resource provider
trait list -f value <host_uuid> | sed 's/^/--trait /')
```

4. 既存のリソースプロバイダー特性に、ホストまたはホストアグリゲートに必要な特性があることを確認します。

```
(overcloud)$ echo $existing_traits
```

5. 必要な特性がまだリソースプロバイダーに追加されていない場合は、既存の特性と必要な特性を各ホストのリソースプロバイダーに追加してください。

```
(overcloud)$ openstack --os-placement-api-version 1.6 \
 resource provider trait set $existing_traits \
 --trait <TRAIT_NAME> \
 <host_uuid>
```

   <TRAIT_NAME> を、リソースプロバイダーに追加する特性の名前に置き換えます。必要に応じて、**--trait** オプションを複数回使用して、さらに特性を追加することができます。

注記

このコマンドは、リソースプロバイダーの特性をすべて置き換えます。したがって、ホスト上の既存のリソースプロバイダー特性のリストを取得して、削除されないように再度設定する必要があります。

6. 要求された特性を持つホストまたはホストアグリゲートにインスタンスをスケジュールするには、イメージの追加スペックに特性を追加します。たとえば、AVX-512 をサポートするホストまたはホストアグリゲートにインスタンスをスケジュールするには、イメージの追加スペックに以下の特性を追加します。

```
(overcloud)$ openstack image set \
 --property trait:HW_CPU_X86_AVX512BW=required \
 trait-image
```

7. 禁止された特性を持つホストまたはホストアグリゲートを除外するには、イメージの追加スペックに特性を追加します。たとえば、ボリュームの複数接続をサポートするホストまたはホストアグリゲートにインスタンスがスケジュールされるのを防ぐには、イメージの追加スペックに以下の特性を追加します。

```
(overcloud)$ openstack image set \
 --property trait:COMPUTE_VOLUME_MULTI_ATTACH=forbidden \
 trait-image
```

7.1.2.2. リソースプロバイダー特性を要求または禁止するフレーバーの作成

クラウドユーザーが特定の特性を持つホストでインスタンスを起動するのに使用することのできるフレーバーを作成することができます。

手順

1. フレーバーを作成します。

```
(overcloud)$ openstack flavor create --vcpus 1 --ram 512 \
 --disk 2 trait-flavor
```

2. ホストまたはホストアグリゲートに必要な特性を識別します。既存の特性を選択するか、新たな特性を作成することができます。

   - 既存の特性を使用するには、既存特性のリストを表示して特性名を取得します。

     ```
     (overcloud)$ openstack --os-placement-api-version 1.6 trait list
     ```

   - 新規特性を作成するには、以下のコマンドを入力します。

     ```
     (overcloud)$ openstack --os-placement-api-version 1.6 trait \
      create CUSTOM_TRAIT_NAME
     ```

     カスタムの特性は接頭辞 **CUSTOM_** で始まり、A から Z までの文字、0 から 9 までの数字、およびアンダースコア_だけを使用する必要があります。

3. 各ホストの既存のリソースプロバイダー特性を収集します。

```
(overcloud)$ existing_traits=$(openstack --os-placement-api-version 1.6 resource provider
trait list -f value <host_uuid> | sed 's/^/--trait /')
```

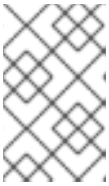
4. 既存のリソースプロバイダー特性に、ホストまたはホストアグリゲートに必要な特性があることを確認します。

```
(overcloud)$ echo $existing_traits
```

5. 必要な特性がまだリソースプロバイダーに追加されていない場合は、既存の特性と必要な特性を各ホストのリソースプロバイダーに追加してください。

```
(overcloud)$ openstack --os-placement-api-version 1.6 \
 resource provider trait set $existing_traits \
 --trait <TRAIT_NAME> \
 <host_uuid>
```

<TRAIT_NAME> を、リソースプロバイダーに追加する特性の名前に置き換えます。必要に応じて、**--trait** オプションを複数回使用して、さらに特性を追加することができます。



注記

このコマンドは、リソースプロバイダーの特性をすべて置き換えます。したがって、ホスト上の既存のリソースプロバイダー特性のリストを取得して、削除されないように再度設定する必要があります。

6. 要求された特性を持つホストまたはホストアグリゲートにインスタンスをスケジュールするには、フレーバーの追加スペックに特性を追加します。たとえば、AVX-512 をサポートするホストまたはホストアグリゲートにインスタンスをスケジュールするには、フレーバーの追加スペックに以下の特性を追加します。

```
(overcloud)$ openstack flavor set \
 --property trait:HW_CPU_X86_AVX512BW=required \
 trait-flavor
```

7. 禁止された特性を持つホストまたはホストアグリゲートを除外するには、フレーバーの追加スペックに特性を追加します。たとえば、ボリュームの複数接続をサポートするホストまたはホストアグリゲートにインスタンスがスケジュールされるのを防ぐには、フレーバーの追加スペックに以下の特性を追加します。

```
(overcloud)$ openstack flavor set \
 --property trait:COMPUTE_VOLUME_MULTI_ATTACH=forbidden \
 trait-flavor
```

7.1.3. ホストアグリゲートの分離による絞り込み

ホストアグリゲートへのスケジューリングを、フレーバーおよびイメージの特性がホストアグリゲートのメタデータと一致するインスタンスだけに制限することができます。フレーバーとイメージのメタデータの組み合わせでは、そのホストアグリゲートに属するコンピュートノードへのスケジューリングを有効にするホストアグリゲート特性をすべて要求する必要があります。

手順

1. Compute 環境ファイルを開きます。

2. ホストアグリゲートを分離してフレーバーおよびイメージの特性がアグリゲートのメタデータと一致するインスタンスだけをホストするには、Compute 環境ファイルの **NovaSchedulerEnableIsolatedAggregateFiltering** パラメーターを **True** に設定します。

3. 更新内容を Compute 環境ファイルに保存します。

4. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

5. ホストアグリゲートを分離する対象の特性を特定します。既存の特性を選択するか、新たな特性を作成することができます。

   - 既存の特性を使用するには、既存特性のリストを表示して特性名を取得します。

     ```
     (overcloud)$ openstack --os-placement-api-version 1.6 trait list
     ```

   - 新規特性を作成するには、以下のコマンドを入力します。

     ```
     (overcloud)$ openstack --os-placement-api-version 1.6 trait \
      create CUSTOM_TRAIT_NAME
     ```

     カスタムの特性は接頭辞 **CUSTOM_** で始まり、A から Z までの文字、0 から 9 までの数字、およびアンダースコア_だけを使用する必要があります。

6. 各コンピュートノードの既存のリソースプロバイダー特性を収集します。

```
(overcloud)$ existing_traits=$(openstack --os-placement-api-version 1.6 resource provider
trait list -f value <host_uuid> | sed 's/^/--trait /')
```

7. 既存のリソースプロバイダー特性で、ホストアグリゲートを分離する特性を確認します。

```
(overcloud)$ echo $existing_traits
```

8. 必要な特性がまだリソースプロバイダーに追加されていない場合は、既存の特性と必要な特性をホストアグリゲートの各コンピュートノードのリソースプロバイダーに追加してください。

```
(overcloud)$ openstack --os-placement-api-version 1.6 \
 resource provider trait set $existing_traits \
 --trait <TRAIT_NAME> \
 <host_uuid>
```

<TRAIT_NAME> を、リソースプロバイダーに追加する特性の名前に置き換えます。必要に応じて、**--trait** オプションを複数回使用して、さらに特性を追加することができます。

注記

このコマンドは、リソースプロバイダーの特性をすべて置き換えます。したがって、ホスト上の既存のリソースプロバイダー特性のリストを取得して、削除されないように再度設定する必要があります。

9. ホストアグリゲートに属する各コンピュートノードで、ステップ 6 - 8 を繰り返します。

10. 特性のメタデータ属性をホストアグリゲートに追加します。

```
(overcloud)$ openstack --os-compute-api-version 2.53 aggregate set \
 --property trait:<TRAIT_NAME>=required <aggregate_name>
```

11. フレーバーまたはイメージに特性を追加します。

```
(overcloud)$ openstack flavor set \
 --property trait:<TRAIT_NAME>=required <flavor>
(overcloud)$ openstack image set \
 --property trait:<TRAIT_NAME>=required <image>
```

7.1.4. Placement サービスを使用したアベイラビリティーゾーンによる絞り込み

Placement サービスを使用して、アベイラビリティーゾーンの要求を適用することができます。Placement サービスを使用してアベイラビリティーゾーンで絞り込むには、アベイラビリティーゾーンホストアグリゲートのメンバーシップおよび UUID と一致する配置アグリゲートが存在する必要があります。

手順

1. Compute 環境ファイルを開きます。

2. Placement サービスを使用してアベイラビリティーゾーンで絞り込むには、Compute 環境ファイルの **NovaSchedulerQueryPlacementForAvailabilityZone** パラメーターを **True** に設定します。

3. **NovaSchedulerEnabledFilters** パラメーターから **AvailabilityZoneFilter** フィルターを削除します。

4. 更新内容を Compute 環境ファイルに保存します。

5. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

関連情報

- アベイラビリティーゾーンとして使用するホストアグリゲートの作成に関する詳細は、Creating an availability zone を参照してください。

7.2. COMPUTE スケジューラーサービス用フィルターおよび重みの設定

インスタンスを起動するコンピュートノードの初期セットを決定するには、Compute スケジューラーサービス用にフィルターおよび重みを設定する必要があります。

手順

1. Compute 環境ファイルを開きます。

2. スケジューラーが使用するフィルターを **NovaSchedulerEnabledFilters** パラメーターに追加します。以下に例を示します。

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AggregateInstanceExtraSpecsFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
```

3. 各コンピュートノードの重みを計算するのに使用する属性を指定します。以下に例を示します。

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      filter_scheduler/weight_classes:
        value: nova.scheduler.weights.all_weighers
```

利用可能な属性についての詳しい情報は、Compute スケジューラーの重みを参照してください。

4. オプション: 各重み付け関数に適用する重みの乗数を設定します。たとえば、コンピュートノードの利用可能な RAM が他のデフォルトの重み付け関数よりも高い重みを持つように指定し、Compute スケジューラーが、利用可能な RAM が少ないコンピュートノードよりも、利用可能な RAM が多いコンピュートノードを優先するには、以下の設定を使用します。

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      filter_scheduler/weight_classes:
        value: nova.scheduler.weights.all_weighers
      filter_scheduler/ram_weight_multiplier:
        value: 2.0
```

ヒント

また、重みの乗数を負の値に設定することもできます。上記の例では、利用可能な RAM が多いコンピュートノードよりも利用可能な RAM が少ないノードを優先するには、**ram_weight_multiplier** を **-2.0** に設定します。

5. 更新内容を Compute 環境ファイルに保存します。

6. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

関連情報

- 利用可能な Compute スケジューラーサービスのフィルターリストは、Compute scheduler filters を参照してください。
- 利用可能な重みの設定オプションリストは、Compute scheduler weights を参照してください。

7.3. COMPUTE スケジューラーのフィルター

インスタンスをホストするのに適切なコンピュートノードを選択する際に Compute スケジューラーが適用しなければならないフィルターを指定するには、Compute 環境ファイルの **NovaSchedulerEnabledFilters** パラメーターを設定します。デフォルト設定では、以下のフィルターが適用されます。

- **AvailabilityZoneFilter**: コンピュートノードは要求されたアベイラビリティーゾーンに属していなければならない。
- **ComputeFilter**: コンピュートノードは要求に対応することができる。
- **ComputeCapabilitiesFilter**: コンピュートノードはフレーバーの追加スペックを満足する。
- **ImagePropertiesFilter**: コンピュートノードは要求されたイメージ属性を満足する。
- **ServerGroupAntiAffinityFilter**: コンピュートノードは、まだ指定されたグループに属するインスタンスをホストしていない。
- **ServerGroupAffinityFilter**: コンピュートノードは、すでに指定されたグループに属するインスタンスをホストしている。

フィルターを追加および削除することができます。利用可能なすべてのフィルターの詳細を以下の表に示します。

表7.1 Compute スケジューラーのフィルター

フィルター	説明
AggregateImagePropertiesIsolation	このフィルターを使用して、インスタンスのイメージメタデータとホストアグリゲートのメタデータを照合します。いずれかのホストアグリゲートのメタデータがイメージのメタデータと一致する場合は、そのホストアグリゲートに属するコンピュートノードはそのイメージからインスタンスを起動する候補となります。スケジューラーは、有効なイメージメタデータ属性のみを認識します。有効なイメージメタデータプロパティーの詳細は、イメージ設定パラメーター を参照してください。
AggregateInstanceExtraSpecsFilter	このフィルターを使用して、インスタンスのフレーバー追加スペックで定義された名前空間属性とホストアグリゲートのメタデータを照合します。 フレーバー **extra_specs** キーのスコープは、**aggregate_instance_extra_specs:** namespace の前に付けて定義する必要があります。 いずれかのホストアグリゲートのメタデータがフレーバー追加スペックのメタデータと一致する場合は、そのホストアグリゲートに属するコンピュートノードはそのイメージからインスタンスを起動する候補となります。

フィルター	説明
AggregateIoOpsFilter	このフィルターを使用して、ホストアグリゲートごとの **filter_scheduler/max_io_ops_per_host** 値を条件に I/O 操作でホストを絞り込みます。ホストアグリゲートごとの値が確認されない場合は、値はグローバル設定にフォールバックします。ホストが複数のアグリゲートに属し、複数の値が確認された場合、スケジューラーは最小の値を使用します。
AggregateMultiTenancyIsolation	このフィルターを使用して、プロジェクト分離ホストアグリゲートに属するコンピュートノードの可用性を、指定したプロジェクトのセットに制限します。**filter_tenant_id** メタデータキーを使用して指定したプロジェクトだけが、ホストアグリゲートに属するコンピュートノードでインスタンスを起動することができます。詳しくは、Creating a project-isolated host aggregate を参照してください。 **注記** プロジェクトが他のホストにインスタンスを配置することは可能です。これを制限するには、**NovaSchedulerPlacementAggregateRequiredForTenants** パラメーターを使用します。
AggregateNumInstancesFilter	このフィルターを使用して、アグリゲートに属する各コンピュートノードでホスト可能なインスタンスの数を制限します。**filter_scheduler/max_instances_per_host** パラメーターを使用して、ホストアグリゲートごとのインスタンスの最大数を設定することができます。ホストアグリゲートごとの値が確認されない場合は、値はグローバル設定にフォールバックします。コンピュートノードが複数のホストアグリゲートに属する場合、スケジューラーは最小の **max_instances_per_host** 値を使用します。
AggregateTypeAffinityFilter	フレーバーメタデータキーが設定されていない場合や、フレーバーアグリゲートメタデータの値に要求するフレーバーの名前が含まれる場合には、このフィルターを使用してホスト渡します。フレーバーメタデータエントリーの値は、単一のフレーバー名またはフレーバー名のコンマ区切りリストのいずれかを含む文字列です (例: **m1.nano** または **m1.nano,m1.small**)。
AllHostsFilter	このフィルターを使用して、利用可能なすべてのコンピュートノードをインスタンスのスケジューリング対象として考慮します。 **注記** このフィルターを使用しても、他のフィルターは無効になりません。
AvailabilityZoneFilter	このフィルターを使用して、インスタンスが指定するアベイラビリティーゾーンに属するコンピュートノードでインスタンスを起動します。

フィルター	説明
ComputeCapabilitiesFilter	このフィルターを使用して、インスタンスのフレーバー追加スペックで定義した名前空間属性とコンピュートノードのケイパビリティーを照合します。フレーバーの追加スペックに **capabilities:** 名前空間の接頭辞を指定する必要があります。 **ComputeCapabilitiesFilter** フィルターを使用するよりも効率的な方法は、フレーバーで CPU 特性を使用することです。これは、Placement サービスに報告されます。特性により、CPU 機能に一貫性のある名前が付けられます。詳細は、Filtering by using resource provider traits を参照してください。
ComputeFilter	このフィルターを使用して、稼働中で有効なすべてのコンピュートノードを渡します。このフィルターは常に設定されている必要があります。

DifferentHostFilter	このフィルターを使用して、特定のインスタンスセットとは異なるコンピュートノードへのインスタンスのスケジューリングを有効にします。インスタンスの起動時にこれらのインスタンスを指定するには、**--hint** 引数を使用して **different_host** およびインスタンスの UUID をキー/値のペアとして指定します。
	
	$ openstack server create --image cedef40a-ed67-4d10-800e-17455edce175 \
	  --flavor 1 --hint different_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 \
	  --hint different_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1

ImagePropertiesFilter	このフィルターを使用して、インスタンスイメージで定義された以下の属性に基づいてコンピュートノードを絞り込みます。 ● **hw_architecture**: ホストのアーキテクチャーを定義します (例: x86、ARM、および Power)。 ● **img_hv_type**: ハイパーバイザーの種別を定義します (例: KVM、QEMU、Xen、および LXC)。 ● **img_hv_requested_version**: Compute サービスが報告するハイパーバイザーのバージョンを定義します。 ● **hw_vm_mode**: ハイパーバイザーの種別を定義します (例: hvm、xen、uml、または exe)。 インスタンスに含まれる指定のイメージ属性をサポートできるコンピュートノードが、スケジューラーに渡されます。イメージプロパティーの詳細は、イメージ設定パラメーター を参照してください。

フィルター	説明

IsolatedHostsFilter	このフィルターを使用して、分離したコンピュートノード上で分離したイメージだけを使用してインスタンスをスケジュールします。また、**filter_scheduler/restrict_isolated_hosts_to_isolated_images** を設定して、分離したコンピュートノード上でのインスタンスのビルドに分離していないイメージが使用されるのを防ぐこともできます。
	
	分離されたイメージとホストのセットを指定するには、**filter_scheduler/isolated_hosts** および **filter_scheduler/isolated_images** 設定オプションを使用します。以下に例を示します。
	
	`parameter_defaults:`
	`  ComputeExtraConfig:`
	`  nova::config::nova_config:`
	`  filter_scheduler/isolated_hosts:`
	`  value: server1, server2`
	`  filter_scheduler/isolated_images:`
	`  value: 342b492c-128f-4a42-8d3a-c5088cf27d13, ebd267a6-ca86-4d6c-9a0e-bd132d6b7d09`

IoOpsFilter	このフィルターを使用して、(ホストで実行可能な高 I/O 負荷インスタンスの最大数を指定する) **filter_scheduler/max_io_ops_per_host** の設定値を超える同時 I/O 操作があるホストを除外します。

MetricsFilter	このフィルターを使用して、**metrics/weight_setting** を使用して設定されたメトリックを報告するコンピュートノードにスケジューリングを制限します。
	
	このフィルターを使用するには、Compute 環境ファイルに以下の設定を追加します。
	
	`parameter_defaults:`
	`  ComputeExtraConfig:`
	`  nova::config::nova_config:`
	`  DEFAULT/compute_monitors:`
	`  value: 'cpu.virt_driver'`
	
	デフォルトでは、Compute スケジューラーサービスは 60 秒ごとにメトリックを更新します。

NUMATopologyFilter	このフィルターを使用して、NUMA 対応コンピュートノードに NUMA トポロジーが設定されたインスタンスをスケジュールします。フレーバー **extra_specs** およびイメージ属性を使用して、インスタンスの NUMA トポロジーを指定します。このフィルターは、各ホストの NUMA セルのオーバーサブスクリプション限度を考慮して、インスタンスの NUMA トポロジーをコンピュートノードのトポロジーに一致させることを試みます。
NumInstancesFilter	このフィルターを使用して、**max_instances_per_host** オプションで指定した以上のインスタンスを実行中のコンピュートノードを除外します。

フィルター	説明
PciPassthroughFilter	このフィルターを使用して、フレーバー **extra_specs** を使用してインスタンスが要求するデバイスを持つコンピュートノードにインスタンスをスケジュールします。 (通常高価で使用が制限される) PCI デバイスを要求するインスタンス用に、そのデバイスを持つノードを確保する場合に、このフィルターを使用します。

SameHostFilter	このフィルターを使用して、特定のインスタンスセットと同じコンピュートノードへのインスタンスのスケジューリングを有効にします。インスタンスの起動時にこれらのインスタンスを指定するには、**--hint** 引数を使用して **same_host** およびインスタンスの UUID をキー/値のペアとして指定します。
	
	``
	$ openstack server create --image cedef40a-ed67-4d10-800e-17455edce175 \
	  --flavor 1 --hint same_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 \
	  --hint same_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1
	``

ServerGroupAffinityFilter	このフィルターを使用して、同じコンピュートノード上でアフィニティーサーバーグループに属するインスタンスをスケジュールします。サーバーグループを作成するには、以下のコマンドを入力します。
	
	``
	$ openstack server group create --policy affinity <group_name>
	``
	
	このグループに属するインスタンスを起動するには、**--hint** 引数を使用して **group** およびグループの UUID をキー/値のペアとして指定します。
	
	``
	$ openstack server create --image <image> \
	  --flavor <flavor> \
	  --hint group=<group_uuid> <instance_name>
	``

ServerGroupAntiAffinityFilter	このフィルターを使用して、異なるコンピュートノード上で非アフィニティーサーバーグループに属するインスタンスをスケジュールします。サーバーグループを作成するには、以下のコマンドを入力します。
	
	``
	$ openstack server group create --policy anti-affinity <group_name>
	``
	
	このグループに属するインスタンスを起動するには、**--hint** 引数を使用して **group** およびグループの UUID をキー/値のペアとして指定します。
	
	``
	$ openstack server create --image <image> \
	  --flavor <flavor> \
	  --hint group=<group_uuid> <instance_name>
	``

フィルター	説明

SimpleCIDRAffinityFilter	このフィルターを使用して、特定の IP サブネット範囲を持つコンピュートノードにインスタンスをスケジュールします。必要な範囲を指定するには、**--hint** 引数を使用してインスタンスの起動時にキー **build_near_host_ip** および **cidr** を渡します。
	`$ openstack server create --image <image> \`
	`  --flavor <flavor> \`
	`  --hint build_near_host_ip=<ip_address> \`
	`  --hint cidr=<subnet_mask> <instance_name>`

7.4. COMPUTE スケジューラーの重み

それぞれのコンピュートノードは重みを持ち、スケジューラーはこれを使用してインスタンスのスケジューリングの優先度を決定します。フィルターを適用した後、Compute スケジューラーは残った候補のコンピュートノードから最大の重みを持つコンピュートノードを選択します。

Compute スケジューラーは、以下のタスクを実行することにより、各コンピュートノードの重みを決定します。

1. スケジューラーは、各重みを 0.0 から 1.0 までの値に正規化します。
2. スケジューラーは、正規化された重みを重み付け関数の乗数で乗算します。

Compute スケジューラーは、候補のコンピュートノード全体でリソースの可用性の低い値および高い値を使用して、各リソース種別の重みの正規化を計算します。

- 最も低いリソース可用性 (minval) を持つノードには、0 が割り当てられます。
- 最も高いリソース可用性 (maxval) を持つノードには 1 が割り当てられます。
- minval と maxval 内の範囲のリソース可用性を持つノードには、以下の式を使用して正規化された重みが割り当てられます。

```
(node_resource_availability - minval) / (maxval - minval)
```

すべてのコンピュートノードが同じリソース可用性を持つ場合、それらはすべて 0 に正規化されます。

たとえば、スケジューラーは、利用可能な仮想 CPU の数がそれぞれ異なる 10 個のコンピュートノードに関して、利用可能な仮想 CPU の正規化された重みを以下のように計算します。

コンピュートノード	1	2	3	4	5	6	7	8	9	10
仮想 CPU の数	5	5	10	10	15	20	20	15	10	5
正規化された重み	0	0	0.33	0.33	0.67	1	1	0.67	0.33	0

Compute スケジューラーは、以下の式を使用してコンピュートノードの重みを計算します。

```
(w1_multiplier * norm(w1)) + (w2_multiplier * norm(w2)) + ...
```

重みに使用することのできる設定オプションの詳細を以下の表に示します。

注記

以下の表で説明するオプションと同じ名前のアグリゲートメタデータのキーを使用して、ホストアグリゲートに重みを設定することができます。ホストアグリゲートに設定すると、ホストアグリゲートの値が優先されます。

表7.2 Compute スケジューラーの重み

設定オプション	型	説明

設定オプション	型	説明
filter_scheduler/weight_classes	String	このパラメーターを使用して、各コンピュートノードの重みを計算するのに以下の属性のどれを使用するかを設定します。 ● **nova.scheduler.weights.ram.RAMWeigher**: コンピュートノードで利用可能な RAM を重み付けします。 ● **nova.scheduler.weights.cpu.CPUWeigher**: コンピュートノードで利用可能な CPU の数を重み付けします。 ● **nova.scheduler.weights.disk.DiskWeigher**: コンピュートノードで利用可能なディスクを重み付けします。 ● **nova.scheduler.weights.metrics.MetricsWeigher**: コンピュートノードのメトリックを重み付けします。 ● **nova.scheduler.weights.affinity.ServerGroupSoftAffinityWeigher**: 指定したインスタンスグループの他のノードとコンピュートノードの近接性を重み付けします。 ● **nova.scheduler.weights.affinity.ServerGroupSoftAntiAffinityWeigher**: 指定したインスタンスグループの他のノードとコンピュートノードの近接性を重み付けします。 ● **nova.scheduler.weights.compute.BuildFailureWeigher**: 直近ブート試行の失敗回数でコンピュートノードを重み付けします。 ● **nova.scheduler.weights.io_ops.IoOpsWeigher**: ワークロードでコンピュートノードを重み付けします。 ● **nova.scheduler.weights.pci.PCIWeigher**: PCI の可用性でコンピュートノードを重み付けします。 ● **nova.scheduler.weights.cross_cell.CrossCellWeigher**: 置かれているセルに基づいてコンピュートノードを重み付けします。インスタンスを移動する際に、移行元セル内にあるコンピュートノードを優先します。 ● **nova.scheduler.weights.all_weighers**: (デフォルト) 上記の重み付け関数をすべて使用します。

設定オプション	型	説明
filter_scheduler/ram_weight_multiplier	浮動小数点	このパラメーターを使用して、利用可能な RAM 容量に基づいてホストを重み付けするのに使用する重みの乗数を指定します。 利用可能な RAM 容量がより大きいホストを優先するには、正の値に設定します。この場合、インスタンスは多くのホストに分散されます。 利用可能な RAM 容量がより小さいホストを優先するには、負の値に設定します。この場合、可能な限り多くのインスタンスをホストに分担 (スタック) させた後に、使用率が低いホストをスケジューリングします。 正または負の絶対値で、他の重み付け関数と比べて{b><b}RAM の重み付け関数をどれだけ優先するかを指定します。 デフォルトは 1.0 で、スケジューラーはインスタンスをすべてのホストに均等に分散します。
filter_scheduler/disk_weight_multiplier	浮動小数点	このパラメーターを使用して、利用可能なディスク容量に基づいてホストを重み付けするのに使用する重みの乗数を指定します。 利用可能なディスク容量がより大きいホストを優先するには、正の値に設定します。この場合、インスタンスは多くのホストに分散されます。 利用可能なディスク容量がより小さいホストを優先するには、負の値に設定します。この場合、可能な限り多くのインスタンスをホストに分担 (スタック) させた後に、使用率が低いホストをスケジューリングします。 正または負の絶対値で、他の重み付け関数と比べてディスクの重み付け関数をどれだけ優先するかを指定します。 デフォルトは 1.0 で、スケジューラーはインスタンスをすべてのホストに均等に分散します。

設定オプション	型	説明
filter_scheduler/cpu_weight_multiplier	浮動小数点	このパラメーターを使用して、利用可能な仮想 CPU の数に基づいてホストを重み付けするのに使用する重みの乗数を指定します。 利用可能な仮想 CPU の数がより多いホストを優先するには、正の値に設定します。この場合、インスタンスは多くのホストに分散されます。 利用可能な仮想 CPU の数がより少ないホストを優先するには、負の値に設定します。この場合、可能な限り多くのインスタンスをホストに分担 (スタック) させた後に、使用率が低いホストをスケジューリングします。 正または負の絶対値で、他の重み付け関数と比べて仮想 CPU の重み付け関数をどれだけ優先するかを指定します。 デフォルトは 1.0 で、スケジューラーはインスタンスをすべてのホストに均等に分散します。
filter_scheduler/io_ops_weight_multiplier	浮動小数点	このパラメーターを使用して、負荷に基づいてホストを重み付けするのに使用する重みの乗数を指定します。 負荷がより軽いホストを優先するには、負の値に設定します。この場合、負荷はより多くのホストに分散されます。 負荷がより重いホストを優先するには、正の値に設定します。この場合、インスタンスはすでにビジー状態にあるホストにスケジューリングされます。 正または負の絶対値で、他の重み付け関数と比べて I/O 操作の重み付け関数をどれだけ優先するかを指定します。 デフォルトは -1.0 で、スケジューラーは負荷をより多くのホストに分散します。
filter_scheduler/build_failure_weight_multiplier	浮動小数点	このパラメーターを使用して、直近のビルド失敗回数に基づいてホストを重み付けするのに使用する重みの乗数を指定します。 ホストから報告される直近のビルド失敗回数をより重要視するには、正の値に設定します。直近ビルドに失敗したホストは、選択されにくくなります。 直近の失敗回数でコンピュートホストを重み付けするのを無効にするには、**0** に設定します。 デフォルト: 1000000.0

設定オプション	型	説明
filter_scheduler/cross_cell_move_weight_multiplier	浮動小数点	このパラメーターを使用して、セルを越えて移動する際にホストを重み付けするのに使用する重みの乗数を指定します。このオプションは、インスタンスを移動する際に、同じ移動元セル内にあるホストに加える重みを決定します。インスタンスを移行する場合、デフォルトではスケジューラーは同じ移行元セル内にあるホストを優先します。 現在インスタンスを実行中のセル内にあるホストを優先するには、正の値に設定します。現在インスタンスを実行中のセルとは別のセルにあるホストを優先するには、負の値に設定します。 デフォルト: 1000000.0
filter_scheduler/pci_weight_multiplier	正の浮動小数点	このパラメーターを使用して、ホスト上の PCI デバイスの数とインスタンスが要求する PCI デバイスの数に基づいてホストを重み付けするのに使用する重みの乗数を指定します。インスタンスが PCI デバイスを要求する場合、より多くの PCI デバイスを持つコンピュートノードにより高い重みが割り当てられます。 たとえば、ホストが 3 台利用可能で、PCI デバイスが 1 つのホストが 1 台、複数の PCI デバイスがあるホストが 1 台、PCI デバイスがないホストが 1 台の場合には、Compute スケジューラーはインスタンスの需要に基づいてこれらのホストの優先順位付けを行います。スケジューラーは、インスタンスが PCI デバイスを 1 つ要求している場合には 1 番目のホストを、複数の PCI デバイスを要求している場合には 2 番目のホストを、PCI デバイスを要求していない場合には 3 番目のホストを、それぞれ優先するはずです。 このオプションを設定して、PCI を要求しないインスタンスが PCI デバイスを持つホストのリソースを占有するのを防ぎます。 デフォルト: 1.0

設定オプション	型	説明
filter_scheduler/host_subset_size	Integer	このパラメーターを使用して、ホストを選択するサブセット (絞り込まれたホストのサブセット) のサイズを指定します。このオプションを 1 以上に設定する必要があります。値を 1 に指定した場合には、重み付け関数によって最初に返されるホストが選択されます。スケジューラーは 1 未満の値を無視し、代わりに 1 を使用します。 類似の要求を処理する複数のスケジューラープロセスが同じホストを選択して競合状態が生じるのを防ぐには、1 より大きい値に設定します。要求に最も適した N 台のホストからホストを無作為に選択することで、競合の可能性が低減されます。ただし、この値を高く設定すると、選択されるホストが特定の要求に対して最適ではない可能性が高くなります。 デフォルト: 1
filter_scheduler/soft_affinity_weight_multiplier	正の浮動小数点	このパラメーターを使用して、グループのソフトアフィニティーのホストを重み付けするのに使用する重みの乗数を指定します。 **注記** このポリシーでグループを作成する場合は、マイクロバージョンを指定する必要があります。 `$ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity <group_name>` デフォルト: 1.0

設定オプション	型	説明
filter_scheduler/soft_anti_affinity_weight_multiplier	正の浮動小数点	このパラメーターを使用して、グループのソフト非アフィニティーのホストを重み付けするのに使用する重みの乗数を指定します。 **注記** このポリシーでグループを作成する場合は、マイクロバージョンを指定する必要があります。 $ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity <group_name> デフォルト: 1.0
metrics/weight_multiplier	浮動小数点	このパラメーターを使用して、メトリックの重み付けに使用する重みの乗数を指定します。デフォルトでは **weight_multiplier=1.0** に設定されており、使用可能な全ホストの間でインスタンスを分散します。 重み全体でメトリックの影響を増大させるには、1.0 を超える数値に設定します。 重み全体でメトリックの影響を減少させるには、0.0 と 1.0 の間の数値に設定します。 メトリックの値を無視して **weight_of_unavailable** オプションの値を返すには、0.0 に設定します。 低いメトリックのホストを優先してインスタンスをホストにスタックするには、負の数値に設定します。 デフォルト: 1.0

設定オプション	型	説明
metrics/weight_setting	**metric=ratio** ペアのコンマ区切りリスト	このパラメーターを使用して、重み付けに使用するメトリック、および各メトリックの重みを計算するのに使用する比率を指定します。有効なメトリック名は以下のとおりです。 ● **cpu.frequency**: CPU の周波数 ● **cpu.user.time**: CPU のユーザーモード時間 ● **cpu.kernel.time**: CPU のカーネル時間 ● **cpu.idle.time**: CPU のアイドル時間 ● **cpu.iowait.time**: CPU の I/O 待機時間 ● **cpu.user.percent**: CPU のユーザーモード率 ● **cpu.kernel.percent**: CPU のカーネル率 ● **cpu.idle.percent**: CPU のアイドル率 ● **cpu.iowait.percent**: CPU の I/O 待機率 ● **cpu.percent**: 汎用 CPU の使用率 例: **weight_setting=cpu.user.time=1.0**
metrics/required	Boolean	このパラメーターを使用して、設定した **metrics/weight_setting** メトリックが使用できない場合の処理方法を指定します。 ● **True**: メトリックは必須です。メトリックが使用できない場合には、例外が発生します。この例外を回避するには、**NovaSchedulerEnabledFilters** の **MetricsFilter** フィルターを使用します。 ● **False**: 使用できないメトリックは、重み付け処理において負の係数として扱われます。**weight_of_unavailable** 設定オプションを使用して、戻り値を設定します。
metrics/weight_of_unavailable	浮動小数点	このパラメーターを使用して、**metrics/weight_setting** メトリックが使用できず、かつ **metrics/required=False** の場合に用いる重みを指定します。 デフォルト: -10000.0

7.5. カスタム特性とリソースクラスの宣言

管理者は、YAML ファイル **provider.yaml** でリソースのカスタムインベントリーを定義することにより、Red Hat OpenStack Platform (RHOSP) オーバークラウドノードでどのカスタム物理機能と消費可能なリソースが利用可能であるかを宣言できます。

CUSTOM_DIESEL_BACKUP_POWER、**CUSTOM_FIPS_COMPLIANT**、**CUSTOM_HPC_OPTIMIZED** などのカスタム特性を定義することで、物理ホスト機能の可用性を宣言できます。**CUSTOM_DISK_IOPS** や **CUSTOM_POWER_WATTS** などのリソースクラスを定義することで、消費可能なリソースの可用性を宣言することもできます。

注記

フレーバーメタデータを使用して、カスタムリソースとカスタム特性を要求できます。詳細は、インスタンスのベアメタルリソースクラス と インスタンスのリソース特性 を参照してください。

手順

1. **/home/stack/templates/** に **provider.yaml** というファイルを作成します。

2. リソースプロバイダーを設定するには、**provider.yaml** ファイルに次の設定を追加します。

```
meta:
  schema_version: '1.0'
providers:
  - identification:
      uuid: <node_uuid>
```

- **<node_uuid>** をノードの UUID に置き換えます (例: **'5213b75d-9260-42a6-b236-f39b0fd10561')**。あるいは、**name** プロパティーを使用してリソースプロバイダーを指定することもできます (**name: 'EXAMPLE_RESOURCE_PROVIDER')**。

3. リソースプロバイダー用に使用可能なカスタムリソースクラスを設定するには、次の設定を **provider.yaml** ファイルに追加します。

```
meta:
  schema_version: '1.0'
providers:
  - identification:
      uuid: <node_uuid>
    inventories:
      additional:
        - CUSTOM_EXAMPLE_RESOURCE_CLASS:
            total: <total_available>
            reserved: <reserved>
            min_unit: <min_unit>
            max_unit: <max_unit>
            step_size: <step_size>
            allocation_ratio: <allocation_ratio>
```

- **CUSTOM_EXAMPLE_RESOURCE_CLASS** をリソースクラスの名前に置き換えます。カスタムリソースクラスは接頭辞 CUSTOM_ で始まり、A から Z までの文字、0 から 9 までの数字、およびアンダースコア "_" だけを使用する必要があります。

- **<total_available>** は、このリソースプロバイダーで使用可能な **CUSTOM_EXAMPLE_RESOURCE_CLASS** の数に置き換えます。

- **<reserved>** は、このリソースプロバイダーで使用可能な **CUSTOM_EXAMPLE_RESOURCE_CLASS** の数に置き換えます。

- **<min_unit>** は、単一インスタンスが消費できるリソースの最小単位に置き換えます。
- **<max_unit>** は、単一インスタンスが消費できるリソースの最大単位に置き換えます。
- **<step_size>** は、このリソースプロバイダーで使用できる **CUSTOM_EXAMPLE_RESOURCE_CLASS** の数に置き換えます。
- **<allocation_ratio>** は、割り当て比率を設定する値に置き換えます。allocation_ratio が 1.0 に設定されている場合、過剰割り当ては許可されません。しかし、allocation_ration が 1.0 より大きい場合、使用可能なリソースの合計は物理的に存在するリソースよりも多くなります。

4. リソースプロバイダー用に使用可能な特性を設定するには、**provider.yaml** ファイルに次の設定を追加します。

```
meta:
  schema_version: '1.0'
providers:
  - identification:
      uuid: <node_uuid>
    inventories:
      additional:
      ...
      traits:
      additional:
        - 'CUSTOM_EXAMPLE_TRAIT'
```

- **CUSTOM_EXAMPLE_TRAIT** を特性の名前に置き換えます。カスタムの特性は接頭辞 CUSTOM_ で始まり、A から Z までの文字、0 から 9 までの数字、およびアンダースコア "_" だけを使用する必要があります。

provider.yaml ファイルの例

次の例では、リソースプロバイダーの 1 つのカスタムリソースクラスと 1 つのカスタム特性を宣言します。

```
meta:
  schema_version: 1.0
providers:
  - identification:
      uuid: $COMPUTE_NODE
    inventories:
      additional:
        CUSTOM_LLC:
          # Describing LLC on this compute node
          # max_unit indicates maximum size of single LLC
          # total indicates sum of sizes of all LLC
          total: 22 (1)
          reserved: 2 (2)
          min_unit: 1 (3)
          max_unit: 11 (4)
          step_size: 1 (5)
          allocation_ratio: 1.0 (6)
    traits:
        additional:
```

```
# Describing that this compute node enables support for
# P-state control
- CUSTOM_P_STATE_ENABLED
```

1. このハイパーバイザーには 22 ユニットの最終レベルキャッシュ (LLC) があります。
2. LLC のユニットのうち 2 つはホスト用に予約されています。
3. 4. min_unit および max_unit の値は、単一の仮想マシンが消費できるリソースのユニット数を定義します。
5. ステップサイズは消費量の増分を定義します。
6. 割り当て比率は、リソースの過剰割り当てを設定します。

5. **provider.yaml** ファイルを保存して閉じます。
6. **provider.yaml** 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/provider.yaml
```

7.6. ホストアグリゲートの作成と管理

クラウド管理者は、パフォーマンスおよび管理目的で、コンピュートのデプロイメントを論理グループに分割することができます。Red Hat OpenStack Platform (RHOSP) では、論理グループへの分割に以下のメカニズムを使用することができます。

ホストアグリゲート

ホストアグリゲートとは、ハードウェアやパフォーマンス特性などの属性に基づいてコンピュートノードを論理的なユニットにグループ化したものです。コンピュートノードを 1 つまたは複数のホストアグリゲートに割り当てることができます。
ホストアグリゲートでメタデータを設定してフレーバーおよびイメージをホストアグリゲートにマッピングし、続いてフレーバーの追加スペックまたはイメージのメタデータ属性をホストアグリゲートのメタデータにマッチングすることができます。必要なフィルターが有効な場合、Compute スケジューラーはこのメタデータを使用してインスタンスのスケジューリングを行うことができます。ホストアグリゲートで指定するメタデータは、ホストの使用をフレーバーまたはイメージで指定するメタデータと同じメタデータのインスタンスに限定します。

ホストアグリゲートのメタデータで **xxx_weight_multiplier** 設定オプションを定義することで、それぞれのホストアグリゲートに重みの乗数を設定することができます。

ホストアグリゲートを使用して、負荷分散の処理、物理的な分離または冗長性の適用、共通の属性を持つサーバーのグループ化、ハードウェアのクラス分け等を行うことができます。

ホストアグリゲートを作成する際に、ゾーン名を指定することができます。この名前は、クラウドユーザーが選択することのできるアベイラビリティーゾーンとして提示されます。

アベイラビリティーゾーン

アベイラビリティーゾーンは、ホストアグリゲートのクラウドユーザー側のビューです。クラウドユーザーは、アベイラビリティーゾーンに属するコンピュートノードやアベイラビリティーゾーンのメタデータを把握することはできません。クラウドユーザーは、アベイラビリティーゾーンの名

前を見ることしかできません。
それぞれのコンピュートノードは、1つのアベイラビリティーゾーンにしか割り当てることができません。デフォルトのアベイラビリティーゾーンを設定することができます。クラウドユーザーがゾーンを指定しない場合、インスタンスはこのアベイラビリティーゾーンにスケジューリングされます。特定の機能を持つアベイラビリティーゾーンを使用するように、クラウドユーザーに指示することができます。

7.6.1. ホストアグリゲートでのスケジューリングの有効化

特定の属性を持つホストアグリゲートにインスタンスをスケジュールするには、Compute スケジューラーの設定を更新し、ホストアグリゲートのメタデータに基づく絞り込みを有効にします。

手順

1. Compute 環境ファイルを開きます。
2. **NovaSchedulerEnabledFilters** パラメーターにまだ以下の値がなければ、値を追加します。
   - **AggregateInstanceExtraSpecsFilter**: フレーバーの追加スペックに一致するホストアグリゲートメタデータでコンピュートノードを絞り込むには、この値を追加します。

     注記

     このフィルターが想定どおりに機能するには、**extra_specs** キーに **aggregate_instance_extra_specs:** 名前空間の接頭辞を指定して、フレーバーの追加スペックのスコープを定義する必要があります。

   - **AggregateImagePropertiesIsolation**: イメージメタデータ属性に一致するホストアグリゲートメタデータでコンピュートノードを絞り込むには、この値を追加します。

     注記

     イメージメタデータ属性を使用してホストアグリゲートのメタデータを絞り込むには、ホストアグリゲートメタデータキーが有効なイメージメタデータ属性と一致する必要があります。有効なイメージメタデータ属性に関する情報は、イメージ設定パラメーター を参照してください。

   - **AvailabilityZoneFilter**: インスタンスの起動時にアベイラビリティーゾーンで絞り込むには、この値を追加します。

     注記

     Compute スケジューラーサービスのフィルター **AvailabilityZoneFilter** を使用する代わりに、Placement サービスを使用してアベイラビリティーゾーンの要求を処理することができます。詳細は、Filtering by availability zone using the Placement service を参照してください。

3. 更新内容を Compute 環境ファイルに保存します。
4. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

7.6.2. ホストアグリゲートの作成

クラウド管理者は、ホストアグリゲートを必要なだけ作成することができます。

手順

1. ホストアグリゲートを作成するには、以下のコマンドを入力します。

```
(overcloud)# openstack aggregate create <aggregate_name>
```

   <aggregate_name> をホストアグリゲートに割り当てる名前に置き換えてください。

2. ホストアグリゲートにメタデータを追加します。

```
(overcloud)# openstack aggregate set \
 --property <key=value> \
 --property <key=value> \
 <aggregate_name>
```

   - **<key=value>** はメタデータのキー/値のペアに置き換えてください。**AggregateInstanceExtraSpecsFilter** フィルターを使用している場合、キーは任意の文字列 (例: **ssd=true**) にすることができます。**AggregateImagePropertiesIsolation** フィルターを使用している場合は、キーは有効なイメージメタデータ属性と一致する必要があります。有効なイメージメタデータプロパティーの詳細は、イメージ設定パラメーター を参照してください。
   - **<aggregate_name>** をホストアグリゲートの名前に置き換えてください。

3. コンピュートノードをホストアグリゲートに追加します。

```
(overcloud)# openstack aggregate add host \
 <aggregate_name> \
 <host_name>
```

   - **<aggregate_name>** は、コンピュートノードを追加するホストアグリゲートの名前に置き換えます。
   - **<host_name>** は、ホストアグリゲートに追加するコンピュートノードの名前に置き換えてください。

4. ホストアグリゲート用のフレーバーまたはイメージを作成します。
   - フレーバーを作成します。

```
(overcloud)$ openstack flavor create \
  --ram <size_mb> \
  --disk <size_gb> \
  --vcpus <no_reserved_vcpus> \
  host-agg-flavor
```

- イメージの作成

```
(overcloud)$ openstack image create host-agg-image
```

5. フレーバーまたはイメージに、ホストアグリゲートのキー/値のペアに一致するキー/値のペアを 1 つまたは複数設定します。

   - フレーバーにキー/値のペアを設定するには、スコープ **aggregate_instance_extra_specs** を使用します。

```
(overcloud)# openstack flavor set \
 --property aggregate_instance_extra_specs:ssd=true \
 host-agg-flavor
```

   - イメージにキー/値のペアを設定するには、有効なイメージメタデータ属性をキーとして使用します。

```
(overcloud)# openstack image set \
 --property os_type=linux \
 host-agg-image
```

7.6.3. アベイラビリティーゾーンの作成

クラウド管理者は、クラウドユーザーがインスタンスを作成する際に選択できるアベイラビリティーゾーンを作成することができます。

手順

1. アベイラビリティーゾーンを作成するには、新しいアベイラビリティーゾーンホストアグリゲートを作成するか、既存のホストアグリゲートをアベイラビリティーゾーンにすることができます。

   a. 新しいアベイラビリティーゾーンホストアグリゲートを作成するには、以下のコマンドを入力します。

```
(overcloud)# openstack aggregate create \
 --zone <availability_zone> \
 <aggregate_name>
```

   - **<availability_zone>** をアベイラビリティーゾーンに割り当てる名前に置き換えてください。
   - **<aggregate_name>** をホストアグリゲートに割り当てる名前に置き換えてください。

   b. 既存のホストアグリゲートをアベイラビリティーゾーンにするには、以下のコマンドを入力します。

```
(overcloud)# openstack aggregate set --zone <availability_zone> \
  <aggregate_name>
```

   - **<availability_zone>** をアベイラビリティーゾーンに割り当てる名前に置き換えてください。
   - **<aggregate_name>** をホストアグリゲートの名前に置き換えてください。

2. オプション: アベイラビリティーゾーンにメタデータを追加します。

```
(overcloud)# openstack aggregate set --property <key=value> \
 <aggregate_name>
```

- **<key=value>** をメタデータのキー/値のペアに置き換えてください。キー/値の属性は、必要なだけ追加することができます。
- **<aggregate_name>** をアベイラビリティーゾーンホストアグリゲートの名前に置き換えてください。

3. コンピュートノードをアベイラビリティーゾーンホストアグリゲートに追加します。

```
(overcloud)# openstack aggregate add host <aggregate_name> \
 <host_name>
```

- **<aggregate_name>** は、コンピュートノードを追加するアベイラビリティーゾーンホストアグリゲートの名前に置き換えてください。
- **<host_name>** は、アベイラビリティーゾーンに追加するコンピュートノードの名前に置き換えてください。

7.6.4. ホストアグリゲートの削除

ホストアグリゲートを削除するには、まずホストアグリゲートからすべてのコンピュートノードを削除します。

手順

1. ホストアグリゲートに割り当てられたすべてのコンピュートノードのリストを表示するには、以下のコマンドを入力します。

```
(overcloud)# openstack aggregate show <aggregate_name>
```

2. ホストアグリゲートから割り当てられたすべてのコンピュートノードを削除するには、それぞれのコンピュートノードで以下のコマンドを入力します。

```
(overcloud)# openstack aggregate remove host <aggregate_name> \
 <host_name>
```

- **<aggregate_name>** は、コンピュートノードを削除するホストアグリゲートの名前に置き換えてください。
- **<host_name>** は、ホストアグリゲートから削除するコンピュートノードの名前に置き換えてください。

3. ホストアグリゲートからすべてのコンピュートノードを削除したら、以下のコマンドを入力してホストアグリゲートを削除します。

```
(overcloud)# openstack aggregate delete <aggregate_name>
```

7.6.5. プロジェクト分離ホストアグリゲートの作成

特定のプロジェクトでのみ利用可能なホストアグリゲートを作成することができます。ホストアグリゲートに割り当てたプロジェクトだけが、ホストアグリゲートでインスタンスを起動することができます。

注記

プロジェクト分離では、Placement サービスを使用して各プロジェクトのホストアグリゲートを絞り込みます。このプロセスは、**AggregateMultiTenancyIsolation** フィルターの機能に優先します。したがって、**AggregateMultiTenancyIsolation** フィルターを使用する必要はありません。

手順

1. Compute 環境ファイルを開きます。
2. プロジェクト分離ホストアグリゲートでプロジェクトインスタンスをスケジュールするには、Compute 環境ファイルの **NovaSchedulerLimitTenantsToPlacementAggregate** パラメーターを **True** に設定します。
3. オプション: ホストアグリゲートに割り当てたプロジェクトだけがクラウド上でインスタンスを作成できるようにするには、**NovaSchedulerPlacementAggregateRequiredForTenants** パラメーターを **True** に設定します。

   注記

   NovaSchedulerPlacementAggregateRequiredForTenants のデフォルト値は **False** です。このパラメーターが **False** の場合、ホストアグリゲートに割り当てられていないプロジェクトは、任意のホストアグリゲートでインスタンスを作成することができます。

4. 更新内容を Compute 環境ファイルに保存します。
5. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml \
```

6. ホストアグリゲートを作成します。
7. プロジェクト ID のリストを取得します。

```
(overcloud)# openstack project list
```

8. **filter_tenant_id<suffix>** メタデータキーを使用して、プロジェクトをホストアグリゲートに割り当てます。

```
(overcloud)# openstack aggregate set \
 --property filter_tenant_id<ID0>=<project_id0> \
 --property filter_tenant_id<ID1>=<project_id1> \
 ...
 --property filter_tenant_id<IDn>=<project_idn> \
 <aggregate_name>
```

- **<ID0>**、**<ID1>**、および **<IDn>** までのすべての ID を、作成する各プロジェクトフィルターの一意の値に置き換えてください。
- **<project_id0>**、**<project_id1>**、および **<project_idn>** までのすべてのプロジェクト ID を、ホストアグリゲートに割り当てる各プロジェクトの ID に置き換えてください。
- **<aggregate_name>** をプロジェクト分離ホストアグリゲートの名前に置き換えてください。
  たとえば、プロジェクト **78f1**、**9d3t**、および **aa29** をホストアグリゲート **project-isolated-aggregate** に割り当てるには、以下の構文を使用します。

```
(overcloud)# openstack aggregate set \
 --property filter_tenant_id0=78f1 \
 --property filter_tenant_id1=9d3t \
 --property filter_tenant_id2=aa29 \
 project-isolated-aggregate
```

ヒント

filter_tenant_id メタデータキーの接尾辞を省略することで、単一の特定プロジェクトでのみ利用可能なホストアグリゲートを作成することができます。

```
(overcloud)# openstack aggregate set \
 --property filter_tenant_id=78f1 \
 single-project-isolated-aggregate
```

関連情報

- ホストアグリゲートの作成に関する詳細は、Creating and managing host aggregates を参照してください。

第8章 PCI パススルーの設定

PCI パススルーを使用して、グラフィックカードまたはネットワークデバイス等の物理 PCI デバイスをインスタンスにアタッチすることができます。デバイスに PCI パススルーを使用する場合、インスタンスはタスクを実行するためにデバイスへの排他的アクセスを確保し、ホストはデバイスを利用することができません。

重要

ルーティング対応プロバイダーネットワークでの PCI パススルーの使用

Compute サービスは、複数のプロバイダーネットワークにまたがる単一のネットワークをサポートしません。ネットワークに複数の物理ネットワークが含まれる場合、Compute サービスは最初の物理ネットワークだけを使用します。したがって、ルーティング対応プロバイダーネットワークを使用する場合は、すべてのコンピュートノードで同じ **physical_network** 名を使用する必要があります。

VLAN またはフラットネットワークのルーティング対応プロバイダーネットワークを使用する場合は、すべてのセグメントで同じ **physical_network** 名を使用する必要があります。その後、ネットワークに複数のセグメントを作成し、そのセグメントを適切なサブネットにマッピングします。

クラウドユーザーが PCI デバイスがアタッチされたインスタンスを作成できるようにするには、以下の手順を実施する必要があります。

1. PCI パススルー用のコンピュートノードを指定する。
2. 必要な PCI デバイスを持つ PCI パススルー用のコンピュートノードを設定する。
3. オーバークラウドをデプロイする。
4. PCI デバイスがアタッチされたインスタンスを起動するためのフレーバーを作成する。

前提条件

- 必要な PCI デバイスを持つコンピュートノード

8.1. PCI パススルー用コンピュートノードの指定

物理 PCI デバイスが接続されたインスタンスのコンピュートノードを指定するには、新しいロールファイルを作成して PCI パススルーロールを設定し、PCI パススルーのコンピュートノードにタグを付けるために使用する PCI パススルーリソースクラスを使用してベアメタルノードを設定する必要があります。

注記

以下の手順は、まだプロビジョニングされていない新しいオーバークラウドノードに適用されます。すでにプロビジョニングされている既存のオーバークラウドノードにリソースクラスを割り当てるには、スケールダウン手順を使用してノードのプロビジョニングを解除してから、スケールアップ手順を使用して新しいリソースクラスの割り当てでノードを再プロビジョニングする必要があります。詳細は、オーバークラウドノードのスケーリング を参照してください。

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. **Controller**、**Compute**、および **ComputePCI** ロールを含む、**roles_data_pci_passthrough.yaml** という名前の新しいロールデータファイルを、オーバークラウドに必要なその他のロールとともに生成します。

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_pci_passthrough.yaml \
 Compute:ComputePCI Compute Controller
```

4. **roles_data_pci_passthrough.yaml** を開き、以下のパラメーターおよびセクションを編集または追加します。

セクション/パラメーター	現在の値	新しい値
ロールのコメント	**Role: Compute**	**Role: ComputePCI**
ロール名	**Compute**	**name: ComputePCI**
description	**Basic Compute Node role**	**PCI パススルー用コンピュートノードロール**
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputepci-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-pci-passthrough.yaml**

5. オーバークラウドの PCI パススルー用コンピュートノードをノード定義のテンプレート **node.json** または **node.yaml** に追加して、そのノードを登録します。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** ガイドの オーバークラウドのノードの登録 を参照してください。

6. ノードのハードウェアを検査します。

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

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの ベアメタルノードハードウェアのインベントリーの作成 を参照してください。

7. PCI パススルー用に指定する各ベアメタルノードに、カスタムの PCI パススルーリソースクラスをタグ付けします。

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.PCI-PASSTHROUGH <node>
```

<node> をベアメタルノードの ID に置き換えてください。

8. ノード定義ファイル **overcloud-baremetal-deploy.yaml** に **ComputePCI** ロールを追加し、予測ノード配置、リソースクラス、ネットワークトポロジー、またはノードに割り当てたいその他の属性を定義します。

```
- name: Controller
  count: 3
- name: Compute
  count: 3
- name: ComputePCI
  count: 1
  defaults:
    resource_class: baremetal.PCI-PASSTHROUGH
    network_config:
      template: /home/stack/templates/nic-config/myRoleTopology.j2 (1)
```

(1) 既存のネットワークトポロジーを再利用するか、ロール用の新しいカスタムネットワークインターフェイステンプレートを作成できます。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** ガイドの カスタムネットワークインターフェイステンプレート を参照してください。**network_config** プロパティーを使用してネットワーク定義を定義しない場合、デフォルトのネットワーク定義が使用されます。

ノード定義ファイルでノード属性を設定するために使用できるプロパティーについて詳しくは、ベアメタルノードのプロビジョニング属性 を参照してください。ノード定義ファイルの例は、ノード定義ファイルの例 を参照してください。

9. プロビジョニングコマンドを実行して、ロールの新しいノードをプロビジョニングします。

```
(undercloud)$ openstack overcloud node provision \
--stack <stack> \
[--network-config \]
--output /home/stack/templates/overcloud-baremetal-deployed.yaml \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- **<stack>** を、ベアメタルノードがプロビジョニングされるスタックの名前に置き換えます。指定しない場合、デフォルトは **overcloud** です。
- **--network-config** オプションの引数を含めて、**cli-overcloud-node-network-config.yaml** Ansible Playbook にネットワーク定義を提供します。**network_config** プロパティーを使用してネットワーク定義を定義しない場合、デフォルトのネットワーク定義が使用されます。

10. 別のターミナルでプロビジョニングの進捗をモニタリングします。プロビジョニングが成功すると、ノードの状態が **available** から **active** に変わります。

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

11. **--network-config** オプションを指定してプロビジョニングコマンドを実行しなかった場合は、**network-environment.yaml** ファイルで **<Role>NetworkConfigTemplate** パラメーターを設定して、NIC テンプレートファイルを指すようにします。

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputePCINetworkConfigTemplate: /home/stack/templates/nic-
```

```
configs/<pci_passthrough_net_top>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

<pci_passthrough_net_top> を **ComputePCI** ロールのネットワークトポロジーを含むファイルの名前に置き換えます。たとえば、デフォルトのネットワークトポロジーを使用するには、**compute.yaml** のようにします。

8.2. PCI パススルー用コンピュートノードの設定

クラウドユーザーが PCI デバイスがアタッチされたインスタンスを作成できるようにするには、PCI デバイスを持つコンピュートノードとコントローラーノードの両方を設定する必要があります。

手順

1. PCI パススルー用にオーバークラウド上のコントローラーノードを設定するには、環境ファイル (例: **pci_passthru_controller.yaml**) を作成します。

2. **pci_passthrough_controller.yaml** の **NovaSchedulerEnabledFilters** パラメーターに **PciPassthroughFilter** を追加します。

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - NUMATopologyFilter
```

3. コントローラーノード上のデバイスの PCI エイリアスを指定するには、以下の設定を **pci_passthrough_controller.yaml** に追加します。

```
parameter_defaults:
  ...
  ControllerExtraConfig:
    nova::pci::aliases:
      - name: "a1"
        product_id: "1572"
        vendor_id: "8086"
        device_type: "type-PF"
```

device_type フィールドの設定に関する詳細は、PCI パススルーデバイス種別フィールド を参照してください。

注記

nova-api サービスが **Controller** 以外のロールで実行されている場合は、**ControllerExtraConfig** を **<Role>ExtraConfig** の形式でユーザーロールに置き換えます。

4. (オプション): PCI パススルーデバイスにデフォルトの NUMA アフィニティーポリシーを設定するには、ステップ 3 の **numa_policy** 設定に **nova::pci::aliases:** を追加します。

```
parameter_defaults:
  ...
  ControllerExtraConfig:
    nova::pci::aliases:
      - name: "a1"
        product_id: "1572"
        vendor_id: "8086"
        device_type: "type-PF"
        numa_policy: "preferred"
```

5. PCI パススルー用にオーバークラウド上のコンピュートノードを設定するには、環境ファイル (例: **pci_passthrough_compute.yaml**) を作成します。

6. コンピュートノード上のデバイスの利用可能な PCI を指定するには、**vendor_id** および **product_id** オプションを使用して、インスタンスへのパススルーに使用できる PCI デバイスのプールに、一致するすべての PCI デバイスを追加します。たとえば、すべての Intel® Ethernet Controller X710 デバイスをインスタンスへのパススルーに使用できる PCI デバイスのプールに追加するには、以下の設定を **pci_passthrough_compute.yaml** に追加します。

```
parameter_defaults:
  ...
  ComputePCIParameters:
    NovaPCIPassthrough:
      - vendor_id: "8086"
        product_id: "1572"
```

NovaPCIPassthrough の設定方法の詳細は、**NovaPCIPassthrough** 設定のガイドライン を参照してください。

7. インスタンスの移行およびサイズ変更の操作を行うために、コンピュートノードの PCI エイリアスのコピーを作成する必要があります。PCI パススルー用コンピュートノード上のデバイスの PCI エイリアスを指定するには、以下の設定を **pci_passthrough_compute.yaml** に追加します。

```
parameter_defaults:
  ...
  ComputePCIExtraConfig:
    nova::pci::aliases:
      - name: "a1"
        product_id: "1572"
        vendor_id: "8086"
        device_type: "type-PF"
```

注記

コンピュートノードのエイリアスは、コントローラーノードのエイリアスと同じでなければなりません。したがって、**pci_passthrough_controller.yaml** の **nova::pci::aliases** に **numa_affinity** を追加した場合は、**pci_passthrough_compute.yaml** の **nova::pci::aliases** にも追加する必要があります。

8. PCI パススルーをサポートするためにコンピュートノードのサーバー BIOS で IOMMU を有効にするには、**pci_passthrough_compute.yaml** に **KernelArgs** パラメーターを追加します。たとえば、Intel IOMMU を有効にするには、以下の **KernalArgs** 設定を使用します。

```
parameter_defaults:
  ...
  ComputePCIParameters:
    KernelArgs: "intel_iommu=on iommu=pt"
```

AMD IOMMU を有効にするには、**KernelArgs** を **"amd_iommu=on iommu=pt"** に設定します。

注記

KernelArgs パラメーターをロールの設定に初めて追加すると、オーバークラウドノードが自動的に再起動されます。必要に応じて、ノードの自動再起動を無効にし、代わりに各オーバークラウドのデプロイ後にノードの再起動を手動で実行できます。詳細は、**KernelArgs** を定義するための手動でのノード再起動の設定を参照してください。

9. その他の環境ファイルと共にこれらのカスタム環境ファイルをスタックに追加して、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] \
  -r /home/stack/templates/roles_data_pci_passthrough.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/pci_passthrough_controller.yaml \
  -e /home/stack/templates/pci_passthrough_compute.yaml \
  -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
  -e /home/stack/templates/node-info.yaml
```

10. クラウドユーザーが PCI デバイスを要求するのに使用できるフレーバーを作成および設定します。以下の例では、ステップ 7 で定義したエイリアスを使用して、それぞれベンダー ID および製品 ID が **8086** および **1572** の 2 つのデバイスを要求します。

```
(overcloud)$ openstack flavor set \
 --property "pci_passthrough:alias"="a1:2" device_passthrough
```

11. (オプション) フレーバーまたはイメージに NUMA アフィニティーポリシーの属性キーを追加して、PCI パススルーデバイスのデフォルト NUMA アフィニティーポリシーをオーバーライドすることができます。

- フレーバーを使用してデフォルトの NUMA アフィニティーポリシーをオーバーライドするには、**hw:pci_numa_affinity_policy** 属性キーを追加します。

```
(overcloud)$ openstack flavor set \
 --property "hw:pci_numa_affinity_policy"="required" \
 device_passthrough
```

  hw:pci_numa_affinity_policy の有効な値についての詳しい情報は、フレーバーのメタデータ を参照してください。

- イメージを使用してデフォルトの NUMA アフィニティーポリシーをオーバーライドするには、**hw_pci_numa_affinity_policy** 属性キーを追加します。

```
(overcloud)$ openstack image set \
 --property hw_pci_numa_affinity_policy=required \
 device_passthrough_image
```

注記

イメージとフレーバーの両方で NUMA アフィニティーポリシーを設定する場合には、属性の値が一致している必要があります。フレーバーの設定は、イメージおよびデフォルトの設定よりも優先されます。したがって、イメージの NUMA アフィニティーポリシーの設定は、フレーバーで属性が設定されていない場合に限り効果を持ちます。

検証

1. PCI パススルーデバイスを設定してインスタンスを作成します。

```
$ openstack server create --flavor device_passthrough \
 --image <image> --wait test-pci
```

2. クラウドユーザーとしてインスタンスにログインします。詳細は、インスタンスへの接続 を参照してください。

3. インスタンスが PCI デバイスにアクセスできることを確認するには、インスタンスから以下のコマンドを入力します。

```
$ lspci -nn | grep <device_name>
```

8.3. PCI パススルーデバイス種別フィールド

Compute サービスでは、デバイスが報告する機能に応じて、PCI デバイスは 3 つの種別のいずれかに分類されます。**device_type** フィールドに設定することのできる有効な値を、以下のリストに示します。

type-PF

デバイスは、SR-IOV をサポートする親またはルートデバイスです。SR-IOV を完全にサポートするデバイスをパススルーするには、このデバイス種別を指定します。

type-VF

デバイスは、SR-IOV をサポートするデバイスの子デバイスです。

type-PCI

デバイスは SR-IOV をサポートしません。**device_type** フィールドが設定されていない場合は、これがデフォルトのデバイス種別です。

注記

コンピュートノードとコントローラーノードの **device_type** を、同じ値に設定する必要があります。

8.4. NOVAPCIPASSTHROUGH 設定のガイドライン

- NIC のデバイス名は変更される可能性があるため、PCI パススルーを設定する場合は **devname** パラメーターを使用しないでください。代わりに、**vendor_id** と **product_id** の方が安定しているため使用するか、NIC の **address** を使用してください。

- 特定の Physical Function (PF) をパススルーするには、PCI アドレスが各デバイスに固有であるので、**address** パラメーターを使用できます。または、**product_id** パラメーターを使用して PF をパススルーすることもできますが、同じ種別の PF が複数ある場合には、PF の **address** も指定する必要があります。

- すべての Virtual Function (VF) をパススルーするには、PCI パススルーに使用する VF の **product_id** および **vendor_id** のみを指定します。NIC の分割に SRIOV を使用し、VF 上で OVS を実行している場合は、VF の **address** も指定する必要があります。

- PF の VF のパススルーだけを設定し、PF そのもののパススルーは設定しない場合は、**address** パラメーターを使用して PF の PCI アドレスを指定し、 **product_id** を使用して VF の 製品 ID を指定することができます。

address パラメーターの設定

address パラメーターは、デバイスの PCI アドレスを指定します。String または **dict** マッピングのいずれかを使用して、**address** パラメーターの値を設定できます。

文字列形式

文字列を使用してアドレスを指定する場合は、以下の例のようにワイルドカード (*) を含めることができます。

```
NovaPCIPassthrough:
 -
   address: "*:0a:00.*"
   physical_network: physnet1
```

ディクショナリー形式

ディクショナリー形式を使用してアドレスを指定する場合は、以下の例のように正規表現構文を含めることができます。

```
NovaPCIPassthrough:
 -
   address:
     domain: ".*"
     bus: "02"
     slot: "01"
     function: "[0-2]"
   physical_network: net1
```

注記

Compute サービスは、**アドレス** フィールドの設定を次の最大値に制限します。

- domain - 0xFFFF
- bus - 0xFF
- slot - 0x1F
- function - 0x7

Compute サービスは、16 ビットアドレスドメインを持つ PCI デバイスをサポートします。Compute サービスは、アドレスドメインが 32 ビットの PCI デバイスを無視します。

第9章 VDPA ポートを使用するインスタンスを有効にするための VDPA コンピュートノードの設定

VIRTIO データパスアクセラレーション (VDPA) は、VIRTIO を介したワイヤースピードのデータ転送を提供します。VDPA デバイスは、SR-IOV 仮想機能 (VF) に対する VIRTIO 抽象化を提供します。これにより、インスタンスでベンダー固有のドライバーなしで VF を使用できます。

注記

NIC を VDPA インターフェイスとして使用する場合は、VDPA インターフェイス専用にする必要があります。NIC の物理機能 (PF) を **switchdev** モードで設定し、ハードウェアオフロード OVS を使用して PF を管理する必要があるため、NIC を他の接続に使用することはできません。

クラウドユーザーが VDPA ポートを使用するインスタンスを作成できるようにするには、次のタスクを完了します。

1. オプション: VDPA のコンピュートノードを指定します。
2. 必要な VDPA ドライバーを持つ VDPA のコンピュートノードを設定します。
3. オーバークラウドをデプロイする。

ヒント

VDPA ハードウェアが制限されている場合は、ホストアグリゲートを設定して VDPA コンピュートノードでのスケジューリングを最適化することもできます。VDPA コンピュートノードで VDPA を要求するインスタンスのみをスケジュールするには、VDPA ハードウェアを備えたコンピュートノードのホストアグリゲートを作成し、VDPA インスタンスのみをホストアグリゲートに配置するようにコンピュートスケジューラーを設定します。詳細は、ホストアグリゲートの分離による絞り込み および ホストアグリゲートの作成と管理 を参照してください。

前提条件

- コンピュートノードには、必要な VDPA デバイスとドライバーがあります。
- Compute ノードには Mellanox NIC があります。
- オーバークラウドは OVS ハードウェアオフロード用に設定されています。詳細は、OVS ハードウェアオフロードの設定 を参照してください。
- オーバークラウドは ML2/OVN を使用するように設定されています。

9.1. VDPA 用コンピュートノードの指定

VIRTIO データパスアクセラレーション (VDPA) インターフェイスを要求するインスタンスのコンピュートノードを指定するには、新しいロールファイルを作成して VDPA ロールを設定し、VDPA リソースクラスを使用してベアメタルノードを設定し、VDPA のコンピュートノードにタグを付けます。

注記

以下の手順は、まだプロビジョニングされていない新しいオーバークラウドノードに適用されます。すでにプロビジョニングされている既存のオーバークラウドノードにリソースクラスを割り当てるには、オーバークラウドをスケールダウンしてノードのプロビジョニングを解除してから、オーバークラウドをスケールアップして、新しいリソースクラスの割り当てでノードを再プロビジョニングします。詳細は、オーバークラウドノードのスケーリング を参照してください。

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。

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

3. **Controller**、**Compute**、および **ComputeVdpa** ロールを含む、**roles_data_vdpa.yaml** という名前の新しいロールデータファイルを生成します。

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_vdpa.yaml \
 ComputeVdpa Compute Controller
```

4. VDPA ロールの **roles_data_vdpa.yaml** ファイルを更新します。

```
###############################################################################
# Role: ComputeVdpa                                             #
###############################################################################
- name: ComputeVdpa
  description: |
    VDPA Compute Node role
  CountDefault: 1
  # Create external Neutron bridge
  tags:
    - compute
    - external_bridge
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Tenant:
      subnet: tenant_subnet
    Storage:
      subnet: storage_subnet
  HostnameFormatDefault: '%stackname%-computevdpa-%index%'
  deprecated_nic_config_name: compute-vdpa.yaml
```

5. オーバークラウド用の VDPA コンピュートノードをノード定義のテンプレート **node.json** または **node.yaml** に追加して、そのノードを登録します。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** ガイドの オーバークラウドのノードの登録 を参照してください。

6. ノードのハードウェアを検査します。

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

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの ベアメタルノードハードウェアのインベントリーの作成 を参照してください。

7. カスタム VDPA リソースクラスを使用して、VDPA 用に指定する各ベアメタルノードにタグを付けます。

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.VDPA <node>
```

<node> は、ベアメタルノードの名前または UUID に置き換えます。

8. ノード定義ファイル **overcloud-baremetal-deploy.yaml** に **ComputeVdpa** ロールを追加し、予測ノード配置、リソースクラス、ネットワークトポロジー、またはノードに割り当てるその他の属性を定義します。

```
- name: Controller
  count: 3
- name: Compute
  count: 3
- name: ComputeVdpa
  count: 1
    defaults:
      resource_class: baremetal.VDPA
      network_config:
        template: /home/stack/templates/nic-config/<role_topology_file>
```

- **<role_topology_file>** を、**ComputeVdpa** ロールに使用するトポロジーファイルの名前 (**vdpa_net_top.j2** など) に置き換えます。既存のネットワークトポロジーを再利用するか、ロール用の新しいカスタムネットワークインターフェイステンプレートを作成できます。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの カスタムネットワークインターフェイステンプレート を参照してください。デフォルトのネットワーク定義設定を使用するには、ロール定義に **network_config** を含めないでください。

ノード定義ファイルでノード属性を設定するために使用できるプロパティーについて詳しくは、ベアメタルノードのプロビジョニング属性 を参照してください。ノード定義ファイルの例は、ノード定義ファイルの例 を参照してください。

9. ネットワークインターフェイステンプレート **vdpa_net_top.j2** を開き、次の設定を追加して、VDPA 対応のネットワークインターフェイスを OVS ブリッジのメンバーとして指定します。

```
- type: ovs_bridge
  name: br-tenant
  members:
    - type: sriov_pf
      name: enp6s0f0
      numvfs: 8
      use_dhcp: false
      vdpa: true
      link_mode: switchdev
    - type: sriov_pf
      name: enp6s0f1
```

```
    numvfs: 8
    use_dhcp: false
    vdpa: true
    link_mode: switchdev
```

10. ロールの新しいノードをプロビジョニングします。

```
(undercloud)$ openstack overcloud node provision \
[--stack <stack>] \
[--network-config \]
--output <deployment_file> \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- オプション: **<stack>** をベアメタルノードがプロビジョニングされるスタックの名前に置き換えます。デフォルトは **overcloud** です。
- オプション: **--network-config** オプションの引数を含めて、Ansible Playbook **cli-overcloud-node-network-config.yaml** にネットワーク定義を提供します。**network_config** プロパティーを使用してノード定義ファイルにネットワーク定義を定義していない場合は、デフォルトのネットワーク定義が使用されます。
- **<deployment_file>** は、デプロイメントコマンドに含めるために生成する heat 環境ファイルの名前に置き換えます (例 **:/home/stack/templates/overcloud-baremetal-deployed.yaml)**。

11. 別のターミナルでプロビジョニングの進捗をモニタリングします。プロビジョニングが成功すると、ノードの状態が **available** から **active** に変わります。

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

12. **--network-config** オプションを指定せずにプロビジョニングコマンドを実行した場合は、**network-environment.yaml** ファイルで **<Role>NetworkConfigTemplate** パラメーターを設定して、NIC テンプレートファイルを指すようにします。

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputeVdpaNetworkConfigTemplate: /home/stack/templates/nic-
configs/<role_topology_file>
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

<role_topology_file> を、**ComputeVdpa** ロールのネットワークトポロジーを含むファイルの名前 (**vdpa_net_top.j2** など) に置き換えます。デフォルトのネットワークトポロジーを使用するには、**compute.j2** に設定します。

9.2. VDPA コンピュートノードの設定

クラウドユーザーが VIRTIO データパスアクセラレーション (VDPA) ポートを使用するインスタンスを作成できるようにするには、VDPA デバイスを持つコンピュートノードを設定します。

手順

1. VDPA コンピュートノードを設定するための新しいコンピューティング環境ファイル (**vdpa_compute.yaml** など) を作成します。

2. **PciPassthroughFilter** と **NUMATopologyFilter** を、**vdpa_compute.yaml** の **NovaSchedulerEnabledFilters** パラメーターに追加します。

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
['AvailabilityZoneFilter','ComputeFilter','ComputeCapabilitiesFilter','ImagePropertiesFilter','ServerGroupAntiAffinityFilter','ServerGroupAffinityFilter','PciPassthroughFilter','NUMATopologyFilter']
```

3. **NovaPCIPassthrough** パラメーターを **vdpa_compute.yaml** に追加して、コンピュートノード上の VDPA デバイスで使用可能な PCI を指定します。たとえば、NVIDIA® ConnectX®-6 Dx デバイスを、インスタンスへのパススルーに使用できる PCI デバイスのプールに追加するには、次の設定を **vdpa_compute.yaml** に追加します。

```
parameter_defaults:
  ...
  ComputeVdpaParameters:
    NovaPCIPassthrough:
    - vendor_id: "15b3"
      product_id: "101d"
      address: "06:00.0"
      physical_network: "tenant"
    - vendor_id: "15b3"
      product_id: "101d"
      address: "06:00.1"
      physical_network: "tenant"
```

NovaPCIPassthrough の設定方法の詳細は、**NovaPCIPassthrough** 設定のガイドライン を参照してください。

4. **KernelArgs** パラメーターを **vdpa_compute.yaml** に追加して、各コンピュートノード BIOS で入出力メモリー管理ユニット (IOMMU) を有効にします。たとえば、Intel Corporation IOMMU を有効にするには、次の **KernalArgs** 設定を使用します。

```
parameter_defaults:
  ...
  ComputeVdpaParameters:
    ...
    KernelArgs: "intel_iommu=on iommu=pt"
```

AMD IOMMU を有効にするには、**KernelArgs** を **"amd_iommu=on iommu=pt"** に設定します。

注記

KernelArgs パラメーターをロールの設定に初めて追加すると、オーバークラウドのデプロイメント中にオーバークラウドノードが自動的に再起動します。必要に応じて、ノードの自動再起動を無効にし、代わりに各オーバークラウドのデプロイ後にノードの再起動を手動で実行できます。詳細は、**KernelArgs** を定義するための手動でのノード再起動の設定 を参照してください。

5. ネットワーク環境ファイルを開き、次の設定を追加して物理ネットワークを定義します。

```
parameter_defaults:
  ...
```

```
NeutronBridgeMappings:
- <bridge_map_1>
- <bridge_map_n>
NeutronTunnelTypes: '<tunnel_types>'
NeutronNetworkType: '<network_types>'
NeutronNetworkVLANRanges:
- <network_vlan_range_1>
- <network_vlan_range_n>
```

- **<bridge_map_1>**、および **<bridge_map_n>** までのすべてのブリッジマッピングを、VDPA ブリッジに使用する論理から物理へのブリッジマッピングに置き換えます。たとえば、**tenant:br-tenant** です。
- **<tunnel_types>** を、プロジェクトネットワークのトンネルタイプのコンマ区切りリストに置き換えます。たとえば、**geneve** です。
- **<network_types>** を Networking サービス (neutron) のプロジェクトネットワークタイプのコンマ区切りリストに置き換えます。利用可能なネットワークがすべてなくなるまで、最初に指定した種別が使用されます。その後、次の種別が使用されます。たとえば、**geneve,vlan** です。
- **<network_vlan_range_1>**、および**<network_vlan_range_n>** までのすべての物理ネットワークと VLAN 範囲を、サポートする ML2 および OVN VLAN マッピング範囲に置き換えます。たとえば、**datacenter:1:1000**、**tenant:100:299**。

6. その他の環境ファイルと共にこれらのカスタム環境ファイルをスタックに追加して、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] \
  -r /home/stack/templates/roles_data_vdpa.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/vdpa_compute.yaml \
  -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
  -e /home/stack/templates/node-info.yaml
```

検証

1. VDPA デバイスでインスタンスを作成します。詳細は、**インスタンスの作成と管理** ガイドの VDPA インターフェイスを使用したインスタンスの作成 を参照してください。
2. クラウドユーザーとしてインスタンスにログインします。詳細は、**インスタンスの作成と管理** ガイドの インスタンスへの接続 を参照してください。
3. インスタンスから VDPA デバイスにアクセスできることを確認します。

```
$ openstack port show vdpa-port
```

第10章 インスタンス用の仮想 GPU の設定

インスタンスで GPU ベースのレンダリングをサポートするには、利用可能な物理 GPU デバイスおよびハイパーバイザーの種別に応じて、仮想 GPU (vGPU) リソースを定義し、管理できます。この設定を使用して、レンダリングの負荷をすべての物理 GPU デバイス間でより効果的に分割し、仮想 GPU 対応のインスタンスをスケジューリングする際の制御性を向上させることができます。

Compute (nova) サービスで仮想 GPU を有効にするには、クラウドユーザーが仮想 GPU デバイスの設定された Red Hat Enterprise Linux (RHEL) インスタンスを作成するのに使用できるフレーバーを作成します。これにより、各インスタンスは物理 GPU デバイスに対応する仮想 GPU デバイスで GPU 負荷に対応することができます。

Compute サービスは、各ホストに定義する GPU プロファイルで利用可能な仮想 GPU デバイスの数を追跡します。Compute サービスはフレーバーに基づいてこれらのホストにインスタンスをスケジュールし、デバイスをアタッチし、使用状況を継続的に監視します。インスタンスが削除されると、Compute サービスは仮想 GPU デバイスを利用可能なプールに戻します。

重要

Red Hat では、サポート例外を要求せずに RHOSP での NVIDIA 仮想 GPU の使用を有効にしています。ただし、Red Hat は、NVIDIA 仮想 GPU ドライバーのテクニカルサポートを提供しません。NVIDIA 仮想 GPU ドライバーは、NVIDIA により提供され、サポートされます。NVIDIA 仮想 GPU ソフトウェアの NVIDIA Enterprise サポートを取得するには、NVIDIA 認定サポートサービスサブスクリプションが必要です。サポートされるコンポーネントで問題を再現できない NVIDIA 仮想 GPU の使用から生じる問題については、以下のサポートポリシーが適用されます。

- サードパーティーコンポーネントが問題に関与していないと Red Hat が考える場合は、通常の サポート対象範囲 および Red Hat SLA が適用されます。
- サードパーティーコンポーネントが問題に関与していると Red Hat が考える場合は、お客様は Red Hat のサードパーティーサポートおよび認定ポリシー に従って NVIDIA に問い合わせを依頼されます。詳細は、ナレッジベースの記事Obtaining Support from NVIDIA を参照してください。

10.1. サポートされる設定および制限

サポートされる GPU カード

サポートされる NVIDIA GPU カードのリストについては、NVIDIA の Web サイトで Virtual GPU Software Supported Products を参照してください。

仮想 GPU デバイスを使用する際の制限

- 各インスタンスが使用できる仮想 GPU のリソースは 1 つだけです。
- ホスト間の vGPU インスタンスのライブマイグレーションはサポートされていません。
- vGPU インスタンスの退避はサポートされていません。
- 仮想 GPU インスタンスをホストするコンピュートノードをリブートする必要がある場合、仮想 GPU は自動的に再作成されたインスタンスに再割り当てされません。コンピュートノードをリブートする前にインスタンスのコールドマイグレーションを行うか、リブート後に各仮想 GPU を正しいインスタンスに手動で割り当てる必要があります。各仮想 GPU を手動で割り当てるに

は、リブートする前にコンピュートノードで実行される各仮想 GPU インスタンスのインスタンス XML から **mdev** UUID を取得する必要があります。以下のコマンドを使用して、各インスタンスの **mdev** UUID を検出することができます。

```
# virsh dumpxml <instance_name> | grep mdev
```

<instance_name> を、Compute API への **/servers** リクエストで返される libvirt インスタンス名 (**OS-EXT-SRV-ATTR:instance_name**) に置き換えます。

- libvirt の制限により、仮想 GPU 対応インスタンスでの休止操作はサポートされていません。代わりに、インスタンスのスナップショット作成またはシェルブ処理が可能です。
- デフォルトでは、コンピュートホストの仮想 GPU の種別は API ユーザーに公開されません。コンピューティングホスト上の vGPU タイプを API ユーザーに公開するには、リソースプロバイダーの特性を設定し、その特性を必要とするフレーバーを作成する必要があります。詳細は、カスタム vGPU リソースプロバイダー特性の作成 を参照してください。また、vGPU タイプが 1 つしかない場合は、ホストをホストアグリゲートに追加することでアクセスを許可できます。詳細は、Creating and managing host aggregates を参照してください。
- NVIDIA アクセラレーターハードウェアを使用する場合は、NVIDIA ライセンス要件に従う必要があります。たとえば、NVIDIA vGPU GRID にはライセンスサーバーが必要です。NVIDIA のライセンス要件の詳細は、NVIDIA の Web サイトで Virtual GPU License Server Release Notes を参照してください。

10.2. コンピュートノードでの仮想 GPU の設定

クラウドユーザーが仮想 GPU (vGPU) を使用するインスタンスを作成できるようにするには、物理 GPU を持つコンピュートノードを設定する必要があります。

1. vGPU 用のコンピュートノードを指定する。
2. 仮想 GPU 用のコンピュートノードを設定する。
3. オーバークラウドをデプロイする。
4. オプション: vGPU タイプのカスタム特性を作成します。
5. オプション: カスタム GPU インスタンスイメージを作成します。
6. vGPU を持つインスタンスを起動するための vGPU フレーバーを作成します。

ヒント

GPU ハードウェアが制限されている場合は、ホストアグリゲートを設定して vGPU コンピュートノードでのスケジューリングを最適化することもできます。仮想 GPU を要求するインスタンスのみを仮想 GPU コンピュートノードにスケジュールするには、仮想 GPU が設定されたコンピュートノードのホストアグリゲートを作成し、Compute スケジューラーが仮想 GPU インスタンスのみをホストアグリゲートに配置するように設定します。詳細は、Creating and managing host aggregates および Filtering by isolating host aggregates を参照してください。

注記

NVIDIA GRID vGPU を使用するには、NVIDIA GRID ライセンス要件に従う共に、セルフホストライセンスサーバーの URL が必要です。詳細は、Virtual GPU License Server Release Notes の Web ページを参照してください。

10.2.1. 前提条件

- NVIDIA の Web サイトから、GPU デバイスに対応する NVIDIA GRID ホストドライバー RPM パッケージをダウンロードしている。必要なドライバーを確認するには、NVIDIA ドライバーダウンロードポータル を参照してください。ポータルからドライバーをダウンロードするには、NVIDIA カスタマーとして登録されている必要があります。
- NVIDIA GRID ホストドライバーがインストールされているカスタムオーバークラウドイメージをビルドしている。

10.2.2. 仮想 GPU 用コンピュートノードの指定

vGPU ワークロードのコンピュートノードを指定するには、新しいロールファイルを作成して vGPU ロールを設定し、GPU 対応のコンピュートノードのタグ付けに使用する GPU リソースクラスを使用してベアメタルノードを設定する必要があります。

注記

以下の手順は、まだプロビジョニングされていない新しいオーバークラウドノードに適用されます。すでにプロビジョニングされている既存のオーバークラウドノードにリソースクラスを割り当てるには、スケールダウン手順を使用してノードのプロビジョニングを解除してから、スケールアップ手順を使用して新しいリソースクラスの割り当てでノードを再プロビジョニングする必要があります。詳細は、オーバークラウドノードのスケーリング を参照してください。

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. **Controller**、**Compute**、**ComputeGpu** ロールを含む、**roles_data_gpu.yaml** という名前の新しいロールデータファイルを、オーバークラウドに必要なその他のロールとともに生成します。

```
(undercloud)$ openstack overcloud roles \
  generate -o /home/stack/templates/roles_data_gpu.yaml \
  Compute:ComputeGpu Compute Controller
```

4. **roles_data_gpu.yaml** を開き、以下のパラメーターおよびセクションを編集または追加します。

セクション/パラメーター	現在の値	新しい値
ロールのコメント	**Role: Compute**	**Role: ComputeGpu**
ロール名	**Compute**	**ComputeGpu**
description	**Basic Compute Node role**	**GPU Compute Node role**
HostnameFormatDefault	**-compute-**	**-computegpu-**

セクション/パラメーター	現在の値	新しい値
deprecated_nic_config_name	**compute.yaml**	**compute-gpu.yaml**

5. オーバークラウド用の GPU 対応コンピュートノードをノード定義のテンプレート **node.json** または **node.yaml** に追加して、そのノードを登録します。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** ガイドの オーバークラウドのノードの登録 を参照してください。

6. ノードのハードウェアを検査します。

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

詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**ガイドの ベアメタルノードハードウェアのインベントリーの作成 を参照してください。

7. GPU 負荷用に指定する各ベアメタルノードに、カスタムの GPU リソースクラスをタグ付けします。

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.GPU <node>
```

<node> をベアメタルノードの ID に置き換えてください。

8. ノード定義ファイル **overcloud-baremetal-deploy.yaml** に **ComputeGpu** ロールを追加し、予測ノード配置、リソースクラス、ネットワークトポロジー、またはノードに割り当てるその他の属性を定義します。

```
- name: Controller
  count: 3
- name: Compute
  count: 3
- name: ComputeGpu
  count: 1
  defaults:
    resource_class: baremetal.GPU
    network_config:
      template: /home/stack/templates/nic-config/myRoleTopology.j2 (1)
```

(1) 既存のネットワークトポロジーを再利用するか、ロール用の新しいカスタムネットワークインターフェイステンプレートを作成できます。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** ガイドの カスタムネットワークインターフェイステンプレート を参照してください。**network_config** プロパティーを使用してネットワーク定義を定義しない場合、デフォルトのネットワーク定義が使用されます。

ノード定義ファイルでノード属性を設定するために使用できるプロパティーについて詳しくは、ベアメタルノードのプロビジョニング属性 を参照してください。ノード定義ファイルの例は、ノード定義ファイルの例 を参照してください。

9. プロビジョニングコマンドを実行して、ロールの新しいノードをプロビジョニングします。

```
(undercloud)$ openstack overcloud node provision \
--stack <stack> \
[--network-config \]
--output /home/stack/templates/overcloud-baremetal-deployed.yaml \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- **<stack>** を、ベアメタルノードがプロビジョニングされるスタックの名前に置き換えます。指定しない場合、デフォルトは **overcloud** です。
- **--network-config** オプションの引数を含めて、**cli-overcloud-node-network-config.yaml** Ansible Playbook にネットワーク定義を提供します。**network_config** プロパティーを使用してネットワーク定義を定義しない場合、デフォルトのネットワーク定義が使用されます。

10. 別のターミナルでプロビジョニングの進捗をモニタリングします。プロビジョニングが成功すると、ノードの状態が **available** から **active** に変わります。

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

11. **--network-config** オプションを指定してプロビジョニングコマンドを実行しなかった場合は、**network-environment.yaml** ファイルで **<Role>NetworkConfigTemplate** パラメーターを設定して、NIC テンプレートファイルを指すようにします。

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputeGpuNetworkConfigTemplate: /home/stack/templates/nic-configs/<gpu_net_top>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

<gpu_net_top> を **ComputeGpu** ロールのネットワークトポロジーが含まれるファイルの名前に置き換えます。たとえば、デフォルトのネットワークトポロジーを使用する場合は **compute.yaml** です。

10.2.3. 仮想 GPU 用コンピュートノードの設定およびオーバークラウドのデプロイ

環境内の物理 GPU デバイスに対応する仮想 GPU の種別を取得して割り当て、仮想 GPU 用コンピュートノードを設定するための環境ファイルを準備する必要があります。

手順

1. Red Hat Enterprise Linux と NVIDIA GRID ドライバーを一時コンピュートノードにインストールし、そのノードを起動します。
2. 仮想 GPU は、仲介デバイス、または **mdev** タイプのデバイスです。各コンピュートノード上の各 **mdev** デバイスの PCI アドレスを取得します。

```
$ ls /sys/class/mdev_bus/
```

PCI アドレスは、デバイスドライバーのディレクトリー名として使用されます (例: **0000:84:00.0)**。

3. 各コンピュートノードで利用可能な各 pGPU デバイスでサポートされている **mdev** タイプを確認して、利用可能な vGPU タイプを見つけます。

```
$ ls /sys/class/mdev_bus/<mdev_device>/mdev_supported_types
```

- **<mdev_device>** を、**mdev** デバイスの PCI アドレス (**0000:84:00.0** など) に置き換えます。
  たとえば、次のコンピュートノードには 4 つの pGPU があり、各 pGPU は 11 個の同じ vGPU タイプをサポートしています。

```
[root@overcloud-computegpu-0 ~]# ls
/sys/class/mdev_bus/0000:84:00.0/mdev_supported_types:
nvidia-35  nvidia-36  nvidia-37  nvidia-38  nvidia-39  nvidia-40  nvidia-41  nvidia-42  nvidia-43
nvidia-44  nvidia-45
[root@overcloud-computegpu-0 ~]# ls
/sys/class/mdev_bus/0000:85:00.0/mdev_supported_types:
nvidia-35  nvidia-36  nvidia-37  nvidia-38  nvidia-39  nvidia-40  nvidia-41  nvidia-42  nvidia-43
nvidia-44  nvidia-45
[root@overcloud-computegpu-0 ~]# ls
/sys/class/mdev_bus/0000:86:00.0/mdev_supported_types:
nvidia-35  nvidia-36  nvidia-37  nvidia-38  nvidia-39  nvidia-40  nvidia-41  nvidia-42  nvidia-43
nvidia-44  nvidia-45
[root@overcloud-computegpu-0 ~]# ls
/sys/class/mdev_bus/0000:87:00.0/mdev_supported_types:
nvidia-35  nvidia-36  nvidia-37  nvidia-38  nvidia-39  nvidia-40  nvidia-41  nvidia-42  nvidia-43
nvidia-44  nvidia-45
```

4. **gpu.yaml** ファイルを作成して、各 GPU デバイスがサポートする vGPU タイプを指定します。

```
parameter_defaults:
  ComputeGpuExtraConfig:
    nova::compute::vgpu::enabled_vgpu_types:
      - nvidia-35
      - nvidia-36
```

5. オプション: 複数の vGPU タイプを設定するには、サポートされる vGPU タイプを pGPU にマップします。

```
parameter_defaults:
  ComputeGpuExtraConfig:
    nova::compute::vgpu::enabled_vgpu_types:
      - nvidia-35
      - nvidia-36
    NovaVGPUTypesDeviceAddressesMapping: {'vgpu_<vgpu_type>': ['<pci_address>',
'<pci_address>'],'vgpu_<vgpu_type>': ['<pci_address>', '<pci_address>']}
```

- **<vgpu_type>** を vGPU タイプの名前に置き換えて、vGPU グループのラベルを作成します (例: **vgpu_nvidia-35**)。追加の vGPU タイプをマップするには、**vgpu_<vgpu_type>** 定義のコンマ区切りリストを使用します。

- **<pci_address>** を、vGPU タイプをサポートする pGPU デバイスの PCI アドレス (**0000:84:00.0** など) に置き換えます。**<pci_address>** 定義のコンマ区切りリストを使用して、vGPU グループを追加の pGPU にマップします。
  以下に例を示します。

  NovaVGPUTypesDeviceAddressesMapping: {'vgpu_nvidia-35': ['0000:84:00.0', '0000:85:00.0'],'vgpu_nvidia-36': ['0000:86:00.0']}

  - **nvidia-35** vGPU タイプは、PCI アドレス **0000:84:00.0** および **0000:85:00.0** にある pGPU でサポートされています。

- **nvidia-36** vGPU タイプは、PCI アドレス **0000:86:00.0** にある pGPU でのみサポートされています。

6. 更新内容を Compute 環境ファイルに保存します。

7. その他の環境ファイルと共に新しいロールファイルおよび環境ファイルをスタックに追加して、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] \
  -r /home/stack/templates/roles_data_gpu.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/gpu.yaml \
  -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
  -e /home/stack/templates/node-info.yaml
```

10.3. カスタム VGPU リソースプロバイダー特性の作成

RHOSP 環境がサポートする vGPU タイプごとにカスタムリソースプロバイダーの特性を作成できます。その後、クラウドユーザーがそれらのカスタム特性を持つホストでインスタンスを起動するのに使用できるフレーバーを作成できます。カスタム特性は大文字で定義し、接頭辞 **CUSTOM_** で始まる必要があります。リソースプロバイダーの特性の詳細は、リソースプロバイダー特性による絞り込み を参照してください。

手順

1. 新しい特性を作成します。

```
(overcloud)$ openstack --os-placement-api-version 1.6 trait \
 create CUSTOM_<TRAIT_NAME>
```

- **<TRAIT_NAME>** を特性の名前に置き換えます。名前には、A から Z までの文字、0 から 9 までの数字、およびアンダースコア "_" だけを使用する必要があります。

2. 各ホストの既存のリソースプロバイダー特性を収集します。

```
(overcloud)$ existing_traits=$(openstack --os-placement-api-version 1.6 resource provider
trait list -f value <host_uuid> | sed 's/^/--trait /')
```

3. 既存のリソースプロバイダー特性に、ホストまたはホストアグリゲートに必要な特性があることを確認します。

```
(overcloud)$ echo $existing_traits
```

4. 必要な特性がまだリソースプロバイダーに追加されていない場合は、既存の特性と必要な特性を各ホストのリソースプロバイダーに追加してください。

```
(overcloud)$ openstack --os-placement-api-version 1.6 \
 resource provider trait set $existing_traits \
 --trait CUSTOM_<TRAIT_NAME> \
 <host_uuid>
```

- **<TRAIT_NAME>** を、リソースプロバイダーに追加する特性の名前に置き換えます。必要に応じて、**--trait** オプションを複数回使用して、さらに特性を追加することができます。

注記

このコマンドは、リソースプロバイダーの特性をすべて置き換えます。したがって、ホスト上の既存のリソースプロバイダー特性のリストを取得して、削除されないように再度設定する必要があります。

10.4. カスタム GPU インスタンスイメージの作成

クラウドユーザーが仮想 GPU (vGPU) を使用するインスタンスを作成できるようにするには、インスタンス起動用のカスタムの仮想 GPU 対応イメージを作成します。NVIDIA GRID ゲストドライバーおよびライセンスファイルを使用してカスタムの仮想 GPU 対応インスタンスイメージを作成するには、以下の手順を使用します。

前提条件

- GPU 対応のコンピュートノードと共にオーバークラウドを設定およびデプロイしている。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. source コマンドで **overcloudrc** 認証情報ファイルを読み込みます。

```
$ source ~/overcloudrc
```

3. 仮想 GPU インスタンスが必要とするハードウェアおよびソフトウェアプロファイルでインスタンスを作成します。

```
(overcloud)$ openstack server create --flavor <flavor> \
 --image <image> temp_vgpu_instance
```

   - **<flavor>** を、仮想 GPU インスタンスが必要とするハードウェアプロファイルを持つフレーバーの名前または ID に置き換えてください。仮想 GPU フレーバー作成に関する詳細は、Creating a vGPU flavor for instances を参照してください。
   - **<image>** を、仮想 GPU 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名前または ID に置き換えてください。RHEL クラウドイメージのダウンロードについて、詳細は **イメージの作成と管理** の RHEL KVM または RHOSP 互換イメージの作成 を参照してください。
4. cloud-user としてインスタンスにログインします。
5. NVIDIA のガイダンス (Licensing an NVIDIA vGPU on Linux by Using a Configuration File) に従って、インスタンス上に **gridd.conf** NVIDIA GRID ライセンスファイルを作成します。
6. インスタンスに GPU ドライバーをインストールします。NVIDIA ドライバーのインストールについての詳細は、Installing the NVIDIA vGPU Software Graphics Driver on Linux を参照してください。

注記

hw_video_model イメージ属性を使用して GPU ドライバーの種別を定義します。仮想 GPU インスタンスのエミュレートされた GPU を無効にする場合は、**none** を選択します。サポートされているドライバーについての詳しい情報は、イメージ設定パラメーター を参照してください。

7. インスタンスのイメージスナップショットを作成します。

```
(overcloud)$ openstack server image create \
 --name vgpu_image temp_vgpu_instance
```

8. オプション: インスタンスを削除します。

10.5. インスタンス用の仮想 GPU フレーバーの作成

クラウドユーザーが GPU 負荷用のインスタンスを作成できるようにするには、仮想 GPU インスタンスを起動するための GPU フレーバーを作成し、仮想 GPU のリソースをそのフレーバーに割り当てます。

前提条件

- GPU 対応コンピュートノードと共にオーバークラウドを設定およびデプロイしている。

手順

1. NVIDIA GPU フレーバーを作成します。以下に例を示します。

```
(overcloud)$ openstack flavor create --vcpus 6 \
 --ram 8192 --disk 100 m1.small-gpu
```

2. vGPU リソースをフレーバーに割り当てます。

```
(overcloud)$ openstack flavor set m1.small-gpu \
 --property "resources:VGPU=1"
```

注記

各インスタンスに割り当てられる仮想 GPU は 1 つだけです。

3. オプション: 特定の vGPU タイプ用のフレーバーをカスタマイズするには、必要な特性をフレーバーに追加します。

```
(overcloud)$ openstack flavor set m1.small-gpu \
 --property trait:CUSTOM_NVIDIA_11=required
```

各 vGPU タイプのカスタムリソースプロバイダー特性を作成する方法については、カスタム vGPU リソースプロバイダー特性の作成 を参照してください。

10.6. 仮想 GPU インスタンスの起動

GPU 負荷用の GPU 対応インスタンスを作成することができます。

手順

1. GPU フレーバーおよびイメージを使用して、インスタンスを作成します。以下に例を示します。

```
(overcloud)$ openstack server create --flavor m1.small-gpu \
 --image vgpu_image --security-group web --nic net-id=internal0 \
 --key-name lambda vgpu-instance
```

2. cloud-user としてインスタンスにログインします。

3. インスタンスが GPU にアクセスできることを確認するには、インスタンスから以下のコマンドを入力します。

```
$ lspci -nn | grep <gpu_name>
```

10.7. GPU デバイスの PCI パススルーの有効化

PCI パススルーを使用して、グラフィックカード等の物理 PCI デバイスをインスタンスに接続することができます。デバイスに PCI パススルーを使用する場合、インスタンスはタスクを実行するためにデバイスへの排他的アクセスを確保し、ホストはデバイスを利用することができません。

前提条件

- **pciutils** パッケージが PCI カードを持つ物理サーバーにインストールされている。
- GPU デバイスのドライバーが、デバイスをパススルーするインスタンスにインストールされている必要があります。したがって、必要な GPU ドライバーがインストールされたカスタムのインスタンスイメージを作成している必要があります。GPU ドライバーがインストールされたカスタムのインスタンスイメージを作成する方法についての詳細は、Creating a custom GPU instance image を参照してください。

手順

1. 各パススルーデバイス種別のベンダー ID および製品 ID を確認するには、PCI カードを持つ物理サーバーで以下のコマンドを入力します。

```
# lspci -nn | grep -i <gpu_name>
```

たとえば、NVIDIA GPU のベンダーおよび製品 ID を確認するには、以下のコマンドを入力します。

```
# lspci -nn | grep -i nvidia
3b:00.0 3D controller [0302]: NVIDIA Corporation TU104GL [Tesla T4] [10de:1eb8] (rev a1)
d8:00.0 3D controller [0302]: NVIDIA Corporation TU104GL [Tesla T4] [10de:1db4] (rev a1)
```

2. 各 PCI デバイスに Single Root I/O Virtualization (SR-IOV) 機能があるかどうかを確認するには、PCI カードを持つ物理サーバーで以下のコマンドを入力します。

```
# lspci -v -s 3b:00.0
3b:00.0 3D controller: NVIDIA Corporation TU104GL [Tesla T4] (rev a1)
        ...
        Capabilities: [bcc] Single Root I/O Virtualization (SR-IOV)
        ...
```

3. PCI パススルー用にオーバークラウド上のコントローラーノードを設定するには、環境ファイル (例: **pci_passthru_controller.yaml**) を作成します。

4. **pci_passthru_controller.yaml** の **NovaSchedulerEnabledFilters** パラメーターに **PciPassthroughFilter** を追加します。

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - NUMATopologyFilter
```

5. コントローラーノード上のデバイスの PCI エイリアスを指定するには、以下の設定を **pci_passthru_controller.yaml** に追加します。

   - PCI デバイスに SR-IOV 機能がある場合:

```
ControllerExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
      device_type: "type-PF"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
      device_type: "type-PF"
```

   - PCI デバイスに SR-IOV 機能がない場合:

```
ControllerExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
```

   device_type フィールドの設定に関する詳細は、PCI passthrough device type field を参照してください。

注記

nova-api サービスが Controller 以外のロールで実行されている場合は、**ControllerExtraConfig** を **<Role>ExtraConfig** の形式でユーザーロールに置き換えます。

6. PCI パススルー用にオーバークラウド上のコンピュートノードを設定するには、環境ファイル (例: **pci_passthru_compute.yaml**) を作成します。

7. コンピュートノード上のデバイスで利用可能な PCI を指定するには、以下の設定を **pci_passthru_compute.yaml** に追加します。

```
parameter_defaults:
  NovaPCIPassthrough:
    - vendor_id: "10de"
      product_id: "1eb8"
```

8. インスタンスの移行およびサイズ変更の操作を行うために、コンピュートノードの PCI エイリアスのコピーを作成する必要があります。コンピュートノード上のデバイスの PCI エイリアスを指定するには、以下の設定を **pci_passthru_compute.yaml** に追加します。

   - PCI デバイスに SR-IOV 機能がある場合:

```
ComputeExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
      device_type: "type-PF"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
      device_type: "type-PF"
```

   - PCI デバイスに SR-IOV 機能がない場合:

```
ComputeExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
```

注記

コンピュートノードのエイリアスは、コントローラーノードのエイリアスと同じでなければなりません。

9. PCI パススルーをサポートするためにコンピュートノードのサーバー BIOS で IOMMU を有効にするには、**pci_passthru_compute.yaml** に **KernelArgs** パラメーターを追加します。

```
parameter_defaults:
  ...
  ComputeParameters:
    KernelArgs: "intel_iommu=on iommu=pt"
```

注記

KernelArgs パラメーターをロールの設定に初めて追加すると、オーバークラウドノードが自動的に再起動されます。必要に応じて、ノードの自動再起動を無効にし、代わりに各オーバークラウドのデプロイ後にノードの再起動を手動で実行できます。詳細は、**KernelArgs** を定義するための手動でのノード再起動の設定を参照してください。

10. その他の環境ファイルと共にこれらのカスタム環境ファイルをスタックに追加して、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/pci_passthru_controller.yaml \
  -e /home/stack/templates/pci_passthru_compute.yaml
```

11. PCI デバイスを要求するためのフレーバーを設定します。以下の例では、それぞれベンダー ID および製品 ID が **10de** および **13f2** の 2 つのデバイスをリクエストします。

```
# openstack flavor set m1.large \
 --property "pci_passthrough:alias"="t4:2"
```

検証

1. PCI パススルーデバイスを設定してインスタンスを作成します。

```
# openstack server create --flavor m1.large \
 --image <custom_gpu> --wait test-pci
```

<custom_gpu> を、必要な GPU ドライバーがインストールされたカスタムインスタンスイメージの名前に置き換えます。

2. クラウドユーザーとしてインスタンスにログインします。詳細は、インスタンスへの接続 を参照してください。

3. インスタンスが GPU にアクセスできることを確認するには、インスタンスから以下のコマンドを入力します。

```
$ lspci -nn | grep <gpu_name>
```

4. NVIDIA System Management Interface のステータスを確認するには、インスタンスから以下のコマンドを入力します。

```
$ nvidia-smi
```

出力例:

```
-----------------------------------------------------------------------------
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================
====|
```

```
|   0  Tesla T4            Off  | 00000000:01:00.0 Off |                    0 |
| N/A   43C    P0    20W /  70W |      0MiB / 15109MiB |      0%      Default |
-----------------------------------------------------------------------------

-----------------------------------------------------------------------------
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================
==|
|  No running processes found                                                 |
-----------------------------------------------------------------------------
```

第11章 インスタンスへのメタデータの追加

Compute (nova) サービスは、メタデータを使用してインスタンスの起動時に設定情報を渡します。インスタンスは、コンフィグドライブまたはメタデータサービスを使用してメタデータにアクセスすることができます。

コンフィグドライブ

コンフィグドライブは、インスタンスのブート時にアタッチすることのできる特別なドライブです。コンフィグドライブは読み取り専用ドライブとしてインスタンスに提示されます。インスタンスはこのドライブをマウントしてそこからファイルを読み取り、通常メタデータサービスから利用する情報を取得することができます。

メタデータサービス

Compute サービスは、REST API としてメタデータサービスを提供します。これを使用して、インスタンス固有のデータを取得することができます。インスタンスは、**169.254.169.254** または **fe80::a9fe:a9fe** からこのサービスにアクセスします。

11.1. インスタンスメタデータの種別

クラウドユーザー、クラウド管理者、および Compute サービスは、メタデータをインスタンスに渡すことができます。

クラウドユーザーが提供するデータ

クラウドユーザーは、インスタンスがブート時に実行するシェルスクリプトなど、インスタンスを起動する際に使用する追加データを指定することができます。クラウドユーザーは、インスタンスを作成または更新する際に、ユーザーデータ機能を使用し、キー/値のペアを必要な属性として渡すことで、データをインスタンスに渡すことができます。

クラウド管理者が提供するデータ

RHOSP 管理者は、ベンダーデータ機能を使用してデータをインスタンスに渡します。Compute サービスの提供するベンダーデータモジュール **StaticJSON** および **DynamicJSON** により、管理者はメタデータをインスタンスに渡すことができます。

- **StaticJSON**:(デフォルト) 全インスタンスで共通のメタデータに使用します。
- **DynamicJSON**: 各インスタンスで異なるメタデータに使用します。このモジュールは外部の REST サービスにリクエストを行い、インスタンスに追加するメタデータを決定します。

ベンダーデータの設定は、インスタンスの以下の読み取り専用ファイルのいずれかにあります。

- **/openstack/{version}/vendor_data.json**
- **/openstack/{version}/vendor_data2.json**

Compute サービスが提供するデータ

Compute サービスはメタデータサービスの内部実装を使用して、要求されたインスタンスのホスト名やインスタンスが属するアベイラビリティーゾーン等の情報をインスタンスに渡します。この操作はデフォルトで実施され、クラウドユーザーまたは管理者が設定を行う必要はありません。

11.2. 全インスタンスへのコンフィグドライブの追加

管理者は Compute サービスを設定し、常にインスタンス用のコンフィグドライブを作成し、コンフィグドライブにデプロイメント固有のメタデータを設定することができます。たとえば、以下の理由によりコンフィグドライブを使用する場合があります。

- デプロイメントにおいて、インスタンスへの IP アドレスの割り当てに DHCP を使用しない場合に、ネットワーク設定を渡すため。インスタンスのネットワーク設定を行う前に、コンフィグドライブを通じてインスタンスの IP アドレス設定を渡すことができます。インスタンスは、コンフィグドライブをマウントして設定にアクセスすることができます。

- Active Directory ポストブートにインスタンスを登録するのに使用される暗号化トークン等、インスタンスを起動するユーザーがアクセスできないデータをインスタンスに渡すため。

- ローカルにキャッシュされたディスク読み取りを作成し、インスタンスのリクエストの負荷を管理するため。これにより、ファクトのチェックインおよびビルドのために定期的にメタデータサーバーにアクセスするインスタンスの影響が軽減されます。

ISO 9660 または VFAT ファイルシステムをマウントできるインスタンスのオペレーティングシステムは、すべてコンフィグドライブを使用することができます。

手順

1. Compute 環境ファイルを開きます。

2. インスタンスの起動時に常にコンフィグドライブをアタッチするには、以下のパラメーターを **True** に設定します。

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::force_config_drive: 'true'
```

3. (オプション) コンフィグドライブの形式をデフォルト値の **iso9660** から **vfat** に変更するには、設定に **config_drive_format** パラメーターを追加します。

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::force_config_drive: 'true'
    nova::compute::config_drive_format: vfat
```

4. 更新内容を Compute 環境ファイルに保存します。

5. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml \
```

検証

1. インスタンスを作成します。

```
(overcloud)$ openstack server create --flavor m1.tiny \
 --image cirros test-config-drive-instance
```

2. インスタンスにログインします。

3. コンフィグドライブをマウントします。

   - インスタンスの OS が **udev** を使用する場合:

```
# mkdir -p /mnt/config
# mount /dev/disk/by-label/config-2 /mnt/config
```

- インスタンスの OS が **udev** を使用しない場合は、まずコンフィグドライブに対応するブロックデバイスを特定する必要があります。

```
# blkid -t LABEL="config-2" -odevice
/dev/vdb
# mkdir -p /mnt/config
# mount /dev/vdb /mnt/config
```

4. マウントされたコンフィグドライブディレクトリー **mnt/config/openstack/{version}/** で、メタデータのファイルを検査します。

11.3. インスタンスへの動的メタデータの追加

デプロイメントを設定してインスタンス固有のメタデータを作成し、そのインスタンスが JSON ファイルを使用してメタデータを利用できるようにすることができます。

ヒント

アンダークラウド上で動的メタデータを使用して、director を Red Hat Identity Management (IdM) サーバーと統合することができます。IdM サーバーは認証局として使用することができ、オーバークラウドで SSL/TLS が有効な場合にオーバークラウドの証明書を管理することができます。詳細は、**Red Hat OpenStack Platform の強化** の Ansible を使用した TLS-e の実装 を参照してください。

手順

1. Compute 環境ファイルを開きます。
2. ベンダーデータプロバイダーモジュールに **DynamicJSON** を追加します。

```
parameter_defaults:
  ControllerExtraConfig:
   nova::vendordata::vendordata_providers:
     - DynamicJSON
```

3. メタデータを生成するためにアクセスする REST サービスを指定します。必要な数だけ目的の REST サービスを指定することができます。以下に例を示します。

```
parameter_defaults:
   ControllerExtraConfig:
   nova::vendordata::vendordata_providers:
     - DynamicJSON
     nova::vendordata::vendordata_dynamic_targets:
       "target1@http://127.0.0.1:125"
     nova::vendordata::vendordata_dynamic_targets:
       "target2@http://127.0.0.1:126"
```

Compute サービスは設定されたターゲットサービスから取得したメタデータが含まれる JSON ファイル **vendordata2.json** を生成し、それをコンフィグドライブディレクトリーに保存します。

注記

ターゲットサービスに同じ名前を複数回使用しないでください。

4. 更新内容を Compute 環境ファイルに保存します。

5. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

第12章 KERNELARGS を定義するための手動でのノード再起動の設定

オーバークラウドのデプロイメントに **KernelArgs** の初回設定が含まれる場合に、オーバークラウドノードは自動的に再起動されます。**KernelArgs** をすでに運用中のデプロイメントに追加する場合には、ノードを再起動することで、既存のワークロードに対して問題となる可能性があります。デプロイメントの更新時にノードの自動再起動を無効にし、代わりに各オーバークラウドのデプロイメント後にノードの再起動を手動で実行できます。

注記

自動再起動を無効にしてから新しいコンピュートノードをデプロイに追加すると、新しいノードは初期プロビジョニング中に再起動されません。**KernelArgs** の設定は再起動後にのみ適用されるため、デプロイメントエラーが発生する可能性があります。

12.1. KERNELARGS を定義するための手動でのノード再起動の設定

KernelArgs を初めて設定するときにノードの自動再起動を無効にし、代わりにノードを手動で再起動できます。

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. カスタム環境ファイル (**kernelargs_manual_reboot.yaml** など) で **KernelArgsDeferReboot** ロールパラメーターを有効にします。

```
parameter_defaults:
  <Role>Parameters:
    KernelArgsDeferReboot: True
```

4. その他の環境ファイルと共にこれらのカスタム環境ファイルをスタックに追加して、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/kernelargs_manual_reboot.yaml
```

5. コンピュートノードのリストを取得して、再起動するノードのホスト名を特定します。

```
(undercloud)$ source ~/overcloudrc
(overcloud)$ openstack compute service list
```

6. 再起動するコンピュートノードで Compute サービスを無効にして、Compute スケジューラーが新しいインスタンスをノードに割り当てないようにします。

```
(overcloud)$ openstack compute service set <node> nova-compute --disable
```

<node> を、Compute サービスを無効にするノードのホスト名に置き換えます。

7. 移行するコンピュートノードでホストされているインスタンスのリストを取得します。

```
(overcloud)$ openstack server list --host <node_UUID> --all-projects
```

8. インスタンスを別のコンピュートノードに移行します。インスタンスの移行については、コンピュートノード間での仮想マシンインスタンスの移行 を参照してください。

9. 再起動するノードにログインします。

10. ノードをリブートします。

```
[tripleo-admin@overcloud-compute-0 ~]$ sudo reboot
```

11. ノードがブートするまで待ちます。

12. コンピュートノードを再度有効にします。

```
(overcloud)$ openstack compute service set <node_UUID>  nova-compute --enable
```

13. コンピュートノードが有効であることを確認します。

```
(overcloud)$ openstack compute service list
```

第13章 インスタンスのセキュリティーを設定する

クラウド管理者は、クラウド上で実行されるインスタンスに対して、次のセキュリティー機能を設定できます。

- **UEFI Secure boot** プロパティーキー **os:secure_boot** を有効にして、UEFI Secure Boot フレーバーを作成できます。クラウドユーザーは、このフレーバーを使用して、UEFI Secure Boot で保護されたインスタンスを作成できます。詳細は、UEFI Secure Boot を参照してください。
- **VNC console security**: VNC プロキシーサービスへの受信クライアントの接続用に適用する許可される TLS 暗号と最小プロトコルバージョンを設定することで、インスタンスの VNC コンソールへの接続を保護できます。詳細は、インスタンスの VNC コンソールへの接続を保護する を参照してください。
- **Emulated virtual Trusted Platform Module (vTPM)**: クラウド管理者は、エミュレートされた Virtual Trusted Platform Module (vTPM) デバイスを持つインスタンスを作成できる機能をクラウドユーザーに提供できます。詳細は、インスタンスにエミュレートされた Trusted Platform Module (TPM) デバイスを提供するためのコンピュートノード設定 を参照してください。
- **SEV**: クラウドユーザーが、メモリー暗号化を使用するインスタンスを作成できるようにするために使用します。詳しくは、Configuring AMD SEV Compute nodes to provide memory encryption for instances を参照してください。

13.1. インスタンスの VNC コンソールへの接続のセキュリティー保護

VNC プロキシーサービスへの受信クライアントの接続用に適用する許可される TLS 暗号と最小プロトコルバージョンを設定することで、インスタンスの VNC コンソールへの接続をセキュアにすることができます。

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. Compute 環境ファイルを開きます。
4. インスタンスへの VNC コンソール接続に使用する最小プロトコルバージョンを設定します。

```
parameter_defaults:
  ...
  NovaVNCProxySSLMinimumVersion: <version>
```

<version> を、許可される最小の SSL/TLS プロトコル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。

- **default**: 基礎となるシステム OpenSSL のデフォルトを使用します。
- **tlsv1_1**: 新しいバージョンをサポートしていないクライアントがある場合に使用します。

注記

TLS 1.0 および TLS 1.1 は RHEL 8 で廃止され、RHEL 9 ではサポートされていません。

- **tlsv1_2**: インスタンスへの VNC コンソール接続に使用する SSL/TLS 暗号を設定する場合に使用します。
- **tlsv1_3**: TLSv1.3 の標準暗号ライブラリーを使用する場合に使用します。**NovaVNCProxySSLCiphers** パラメーターの設定は無視されます。

5. 使用可能な最小の SSL/TLS プロトコルバージョンを **tlsv1_2** に設定する場合は、インスタンスへの VNC コンソール接続に使用する SSL/TLS 暗号を設定します。

```
parameter_defaults:
  NovaVNCProxySSLCiphers: <ciphers>
```

<ciphers> を、許可する暗号スイートのコロン区切りリストに置き換えます。**openssl** から利用可能な暗号のリストを取得します。

6. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
```

13.2. エミュレートされた TRUSTED PLATFORM MODULE (TPM) デバイスをインスタンスに提供するためのコンピュートノードの設定

クラウド管理者は、エミュレートされた Virtual Trusted Platform Module (vTPM) デバイスを持つインスタンスを作成できる機能をクラウドユーザーに提供できます。

クラウドユーザーが vTPM デバイスを持つインスタンスを作成できるようにするには、次のタスクを実行する必要があります。

1. vTPM デバイスを持つンスタンスのサポートを有効にし、オーバークラウドをデプロイします。
2. vTPM デバイスを持つインスタンスを起動するためのフレーバーまたはイメージを作成します。

前提条件

- Key Manager サービス (barbican) が、vTPM キーを保存するために RHOSP デプロイメントに含まれている。Key Manager サービスを使用したシークレットの管理については、Key Manager サービスを使用したシークレットの管理 を参照してください。

vTPM デバイスを持つインスタンスの制限事項

- vTPM デバイスを持つインスタンスをライブマイグレーションまたは退避することはできません。
- vTPM デバイスを持つインスタンスをレスキューまたは退避することはできません。

- インスタンスには Q35 マシン種別が必要です。

13.2.1. vTPM デバイスを持つインスタンスのサポートの有効化

クラウドユーザーが vTPM デバイスを持つインスタンスを作成できるようにするには、インスタンスに対して vTPM デバイス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。

手順

1. Compute 環境ファイルを開きます。
2. vTPM デバイスのサポートを有効にします。

```
parameter_defaults:
  ComputeParameters:
    ...
    NovaEnableVTPM: True
```

3. 更新内容を Compute 環境ファイルに保存します。
4. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、オーバークラウドをデプロイします。

```
(undercloud)$ openstack overcloud deploy --templates \
 -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
 -e /home/stack/templates/node-info.yaml \
 -e [your environment files] \
 -e /home/stack/templates/<compute_environment_file>.yaml
```

13.2.2. vTPM デバイス用のイメージの作成

オーバークラウドで vTPM デバイスを持つインスタンスの作成を有効にしたら、クラウドユーザーが vTPM デバイスを持つインスタンスを起動するために使用できる vTPM デバイスのインスタンスイメージを作成できます。

注記

フレーバーとイメージの両方で TPM デバイスモデルが指定されていて、2 つの値が一致しない場合、スケジューリングが失敗します。

手順

1. vTPM デバイスの新しいイメージを作成します。

```
(overcloud)$ openstack image create ...  \
 --property hw_tpm_version=2.0 vtpm-image
```

注記

TPM バージョン **1.2** はサポートされていません。

2. オプション: 使用する TPM モデルを指定します。

```
(overcloud)$ openstack image set \
 --property hw_tpm_model=<tpm_model> \
 vtpm-image
```

- **<tpm_model>** を、使用する TPM デバイスのモデルに置き換えます。以下の有効な値のいずれかに設定します。
  - **tpm-tis**: (デフォルト) TPM インターフェイス仕様。
  - **tpm-crb**: コマンド応答バッファー。

> **注記**
>
> **hw_tpm_version** プロパティーが設定されていない場合、Compute サービスは **hw_tpm_model** プロパティーの設定を無視します。

検証

1. vTPM イメージを使用してインスタンスを作成します。

```
(overcloud)$ openstack server create --flavor m1.small \
 --image vtpm-image vtpm-instance
```

2. クラウドユーザーとしてインスタンスにログインします。
3. インスタンスが vTPM デバイスにアクセスできることを確認するには、インスタンスから次のコマンドを入力します。

```
$ dmesg | grep -i tpm
```

13.2.3. vTPM デバイス用のフレーバーの作成

オーバークラウドで vTPM デバイスを持つインスタンスの作成を有効にしたら、クラウドユーザーが vTPM デバイスを持つインスタンスを起動するために使用できる 1 つ以上の vTPM デバイスフレーバーを作成できます。

> **注記**
>
> vTPM デバイスフレーバーは、イメージに **hw_tpm_model** プロパティーと **hw_tpm_version** プロパティーが設定されていない場合にのみ必要です。フレーバーとイメージの両方で TPM デバイスモデルが指定されていて、2 つの値が一致しない場合、スケジューリングが失敗します。

手順

1. vTPM デバイス用のフレーバーを作成します。

```
(overcloud)$ openstack flavor create --vcpus 1 --ram 512 --disk 2  \
 --property hw:tpm_version=2.0 \
 vtpm-flavor
```

注記

TPM バージョン **1.2** はサポートされていません。

2. オプション: 使用する TPM モデルを指定します。

```
(overcloud)$ openstack flavor set \
 --property hw:tpm_model=<tpm_model> \
 vtpm-flavor
```

- **<tpm_model>** を、使用する TPM デバイスのモデルに置き換えます。以下の有効な値のいずれかに設定します。
  - **tpm-tis**: (デフォルト) TPM インターフェイス仕様。
  - **tpm-crb**: コマンド応答バッファー。TPM バージョン 2.0 とのみ互換性があります。

注記

hw:tpm_version プロパティーが設定されていない場合、Compute サービスは **hw:tpm_model** プロパティーの設定を無視します。

検証

1. vTPM フレーバーを使用してインスタンスを作成します。

```
(overcloud)$ openstack server create --flavor vtpm-flavor \
 --image rhel-image vtpm-instance
```

2. クラウドユーザーとしてインスタンスにログインします。
3. インスタンスが vTPM デバイスにアクセスできることを確認するには、インスタンスから次のコマンドを入力します。

```
$ dmesg | grep -i tpm
```

第14章 データベースのクリーニング

Compute サービスには管理ツール **nova-manage** が含まれています。このツールを使用して、データベーススキーマの適用、アップグレード中のオンラインデータ移行の実行、データベースの管理およびクリーンアップ等の、デプロイメント、アップグレード、クリーンアップ、およびメンテナンス関連のタスクを実行することができます。

director は、cron を使用してオーバークラウドでの以下のデータベース管理タスクを自動化します。

- 削除された行を実稼働テーブルからシャドウテーブルに移動して、削除されたインスタンスレコードをアーカイブする。
- アーカイブ処理が完了した後に、シャドウテーブルから削除された行をパージする。

14.1. データベース管理の設定

cron ジョブは、デフォルト設定を使用してデータベース管理タスクを実行します。デフォルトでは、データベースをアーカイブする cron ジョブは毎日 00:01 に実行され、データベースをパージする cron ジョブは 毎日 05:00 に実行されます。共にジッターは 0 秒から 3600 秒の間です。必要に応じて、これらの設定は heat パラメーターを使用して変更することができます。

手順

1. Compute 環境ファイルを開きます。
2. 追加または変更する cron ジョブを制御する heat パラメーターを追加します。たとえば、シャドウテーブルをアーカイブ直後にパージするには、次のパラメーターを True に設定します。

   ```
   parameter_defaults:
     ...
     NovaCronArchiveDeleteRowsPurge: True
   ```

   データベースの cron ジョブを管理する heat パラメーターの完全リストは、Configuration options for the Compute service automated database management を参照してください。
3. 更新内容を Compute 環境ファイルに保存します。
4. その他の環境ファイルと共に Compute 環境ファイルをスタックに追加して、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/<compute_environment_file>.yaml
   ```

14.2. COMPUTE サービスのデータベース自動管理用設定オプション

以下の heat パラメーターを使用して、データベースを管理する自動 cron ジョブを有効化および変更します。

表14.1 Compute (nova) サービスの cron パラメーター

パラメーター	説明
NovaCronArchiveDeleteAllCells	すべてのセルから削除されたインスタンスレコードをアーカイブするには、このパラメーターを True に設定します。 デフォルト: **True**
NovaCronArchiveDeleteRowsAge	このパラメーターを使用して、削除されたインスタンスレコードを経過時間 (日数単位) に基づいてアーカイブします。 シャドウテーブル内の本日以前のデータをアーカイブするには、**0** に設定します。 デフォルト: **90**
NovaCronArchiveDeleteRowsDestination	このパラメーターを使用して、削除されたインスタンスレコードを記録するファイルを設定します。 デフォルト: **/var/log/nova/nova-rowsflush.log**
NovaCronArchiveDeleteRowsHour	このパラメーターを使用して、削除されたインスタンスレコードを別のテーブルに移動する cron コマンドを実行する時刻の時間部分を設定します。 デフォルト: **0**
NovaCronArchiveDeleteRowsMaxDelay	このパラメーターを使用して、削除されたインスタンスレコードを別のテーブルに移動するまでの最大遅延時間 (秒単位) を設定します。 デフォルト: **3600**
NovaCronArchiveDeleteRowsMaxRows	このパラメーターを使用して、別のテーブルに移動することのできる削除インスタンスレコード数の最大値を設定します。 デフォルト: **1000**
NovaCronArchiveDeleteRowsMinute	このパラメーターを使用して、削除されたインスタンスレコードを別のテーブルに移動する cron コマンドを実行する時刻の分部分を設定します。 デフォルト: **1**
NovaCronArchiveDeleteRowsMonthday	このパラメーターを使用して、削除されたインスタンスレコードを別のテーブルに移動する cron コマンドを実行する日にちを設定します。 デフォルト: ***** (毎日)

パラメーター	説明
NovaCronArchiveDeleteRowsMonth	このパラメーターを使用して、削除されたインスタンスレコードを別のテーブルに移動する cron コマンドを実行する月を設定します。 デフォルト: ***** (毎月)
NovaCronArchiveDeleteRowsPurge	スケジュールされたアーカイブ処理の直後にシャドウテーブルをパージするには、このパラメーターを True に設定します。 デフォルト: **False**
NovaCronArchiveDeleteRowsUntilComplete	すべてのレコードを移動するまで削除されたインスタンスレコードを別のテーブルに移動し続けるには、このパラメーターを True に設定します。 デフォルト: **True**
NovaCronArchiveDeleteRowsUser	このパラメーターを使用して、削除されたインスタンスレコードをアーカイブする crontab の所有権を持ち、crontab が使用するログファイルにアクセスできるユーザーを設定します。 デフォルト: **nova**
NovaCronArchiveDeleteRowsWeekday	このパラメーターを使用して、削除されたインスタンスレコードを別のテーブルに移動する cron コマンドを実行する曜日を設定します。 デフォルト: ***** (毎日)
NovaCronPurgeShadowTablesAge	このパラメーターを使用して、シャドウテーブルを経過時間 (日数単位) に基づいてパージします。 本日以前のシャドウテーブルをパージするには、**0** に設定します。 デフォルト: **14**
NovaCronPurgeShadowTablesAllCells	すべてのセルからシャドウテーブルをパージするには、このパラメーターを True に設定します。 デフォルト: **True**
NovaCronPurgeShadowTablesDestination	このパラメーターを使用して、パージされたシャドウテーブルを記録するファイルを設定します。 デフォルト: **/var/log/nova/nova-rowspurge.log**

パラメーター	説明
NovaCronPurgeShadowTablesHour	このパラメーターを使用して、シャドウテーブルをパージする cron コマンドを実行する時刻の時間部分を設定します。 デフォルト: **5**
NovaCronPurgeShadowTablesMaxDelay	このパラメーターを使用して、シャドウテーブルをパージするまでの最大遅延時間 (秒単位) を設定します。 デフォルト: **3600**
NovaCronPurgeShadowTablesMinute	このパラメーターを使用して、シャドウテーブルをパージする cron コマンドを実行する時刻の分部分を設定します。 デフォルト: **0**
NovaCronPurgeShadowTablesMonth	このパラメーターを使用して、シャドウテーブルをパージする cron コマンドを実行する月を設定します。 デフォルト: ***** (毎月)
NovaCronPurgeShadowTablesMonthday	このパラメーターを使用して、シャドウテーブルをパージする cron コマンドを実行する日にちを設定します。 デフォルト: ***** (毎日)
NovaCronPurgeShadowTablesUser	このパラメーターを使用して、シャドウテーブルをパージする crontab の所有権を持ち、crontab が使用するログファイルにアクセスできるユーザーを設定します。 デフォルト: **nova**
NovaCronPurgeShadowTablesVerbose	このパラメーターを使用して、パージされたシャドウテーブルに関するログファイルの詳細ロギングを有効にします。 デフォルト: **False**
NovaCronPurgeShadowTablesWeekday	このパラメーターを使用して、シャドウテーブルをパージする cron コマンドを実行する曜日を設定します。 デフォルト: ***** (毎日)

第15章 コンピュートノード間の仮想マシンインスタンスの移行

メンテナンスを実行する場合やワークロードのリバランスを行う場合、あるいは障害が発生した/障害が発生しつつあるノードを置き換える場合に、あるコンピュートノードからオーバークラウド内の別のコンピュートノードにインスタンスを移行しなければならない場合があります。

コンピュートノードのメンテナンス

ハードウェアのメンテナンスや修理、カーネルのアップグレードおよびソフトウェアの更新を行うなどの理由により、コンピュートノードを一時的に停止する必要がある場合、コンピュートノード上で実行中のインスタンスを別のコンピュートノードに移行することができます。

障害が発生しつつあるコンピュートノード

コンピュートノードで障害が発生する可能性があり、ノードのサービスまたは置き換えが必要な場合、障害が発生しつつあるコンピュートノードから正常なコンピュートノードにインスタンスを移行することができます。

障害が発生したコンピュートノード

コンピュートノードですでに障害が発生している場合には、インスタンスを退避させることができます。同じ名前、UUID、ネットワークアドレス、およびコンピュートノードに障害が発生する前にインスタンスに割り当てられていたその他すべてのリソースを使用して、元のイメージから別のコンピュートノードにインスタンスを再ビルドすることができます。

ワークロードのリバランス

ワークロードをリバランスするために、1つまたは複数のインスタンスを別のコンピュートノードに移行することができます。たとえば、コンピュートノード上のインスタンスを1つにまとめて電力を節約する、他のネットワークリソースに物理的に近いコンピュートノードにインスタンスを移行してレイテンシーを低減する、インスタンスを全コンピュートノードに分散してホットスポットをなくし復元力を向上させる、等が可能です。

director は、すべてのコンピュートノードがセキュアな移行を提供するように設定します。すべてのコンピュートノードには、移行プロセス中それぞれのホストのユーザーが他のコンピュートノードにアクセスできるように、共有 SSH キーも必要です。director は、**OS::TripleO::Services::NovaCompute** コンポーザブルサービスを使用してこのキーを作成します。このコンポーザブルサービスは、すべての Compute ロールにデフォルトで含まれているメインのサービスの1つです。詳細は、**Red Hat OpenStack Platform デプロイメントのカスタマイズ** ガイドの コンポーザブルサービスとカスタムロール を参照してください。

注記

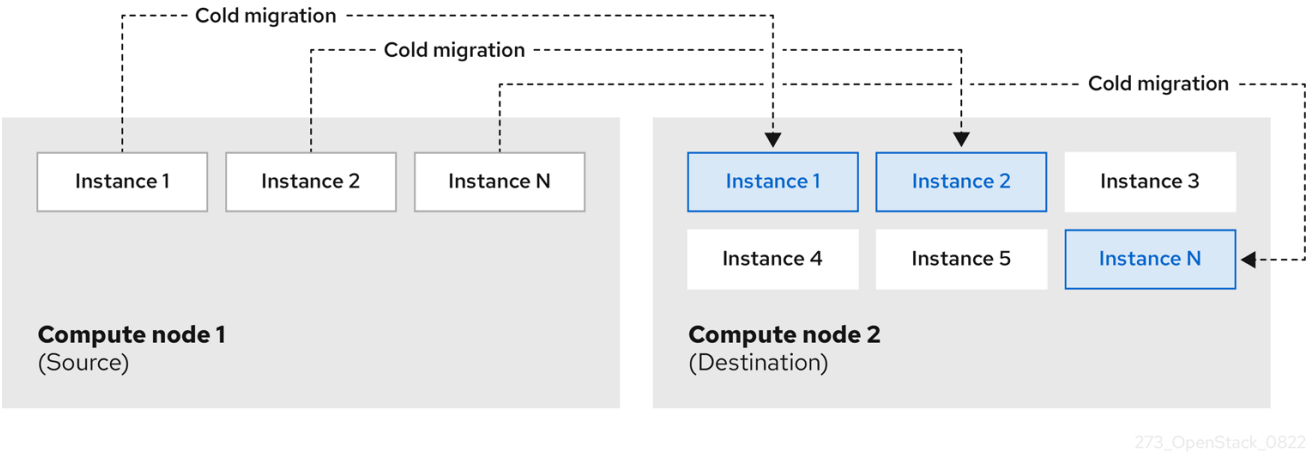
機能しているコンピュートノードがあり、バックアップ目的でインスタンスのコピーを作成する場合、またはインスタンスを別の環境にコピーする場合は、**director を使用した Red Hat OpenStack Platform のインストールおよび管理** ガイドの オーバークラウドへの仮想マシンのインポート の手順に従ってください。

15.1. 移行の種別

Red Hat OpenStack Platform (RHOSP) では、以下の移行種別がサポートされています。

コールドマイグレーション

コールドマイグレーション (あるいは、非ライブマイグレーション) では、動作中のインスタンスをシャットダウンしてから、移行元コンピュートノードから移行先コンピュートノードに移行します。



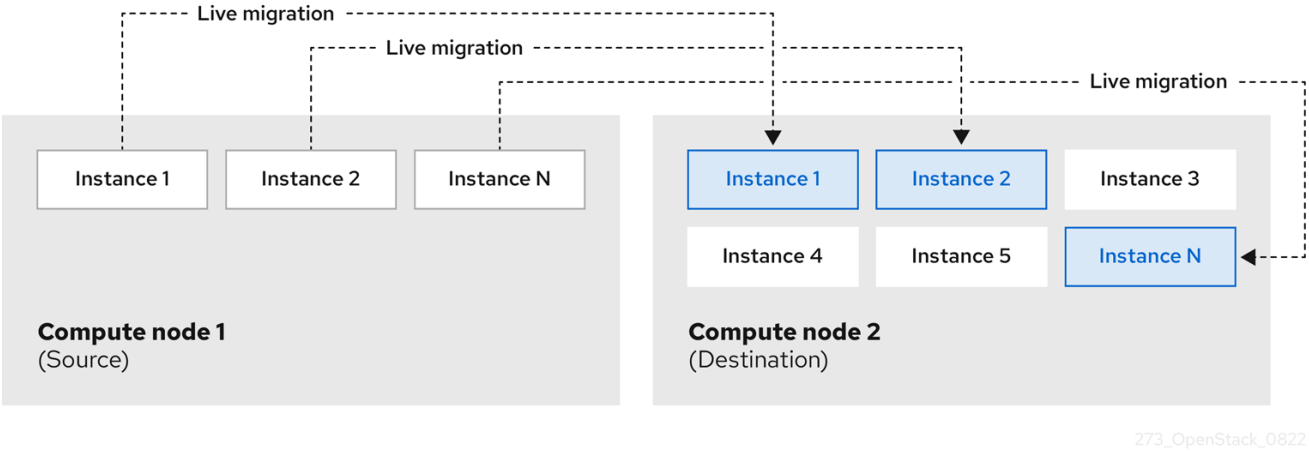
コールドマイグレーションでは、インスタンスに多少のダウンタイムが発生します。移行したインスタンスは、引き続き同じボリュームおよび IP アドレスにアクセスすることができます。

注記

コールドマイグレーションを行うためには、移行元および移行先両方のコンピュートノードが動作状態になければなりません。

ライブマイグレーション

ライブマイグレーションでは、インスタンスをシャットダウンせずに、動作状態を維持しながら移行元コンピュートノードから移行先コンピュートノードに移行します。



インスタンスのライブマイグレーションを行う場合、ダウンタイムはほとんど発生しません。ただし、ライブマイグレーションは、移行操作中のパフォーマンスに影響を及ぼします。したがって、移行中のインスタンスは重要なパスから除外する必要があります。

重要

ライブマイグレーションは、移動されるワークロードのパフォーマンスに影響を与えます。Red Hat は、ライブマイグレーション中のパケット損失、ネットワーク遅延、メモリー遅延の増加、またはネットワーク帯域幅、メモリー帯域幅、ストレージ IO、または CPU パフォーマンスの低下をサポートしません。

注記

ライブマイグレーションを行うためには、移行元および移行先両方のコンピュートノードが動作状態になければなりません。

状況によっては、インスタンスのライブマイグレーションを行うことができない場合があります。詳細は、移行の制約 を参照してください。

退避

コンピュートノードですでに障害が発生しているためインスタンスを移行する必要がある場合、インスタンスを退避させることができます。

15.2. 移行の制約

移行の制約は通常、ブロックマイグレーション、設定ディスク、またはいずれかのインスタンスがコンピュートノード上の物理ハードウェアにアクセスする場合に生じます。

CPU に関する制約

移行元および移行先コンピュートノードの CPU アーキテクチャーは、同一であることが必須です。たとえば、Red Hat では、**ppc64le** CPU から **x86_64** CPU へのインスタンスの移行をサポートしません。

異なる CPU モデル間の移行はサポートされていません。CPU ホストパススルーを使用するインスタンス等の場合には、移行元および移行先コンピュートノードの CPU は、完全に同一でなければなりません。すべてのケースで、移行先ノードの CPU 機能は、移行元ノードの CPU 機能の上位セットであることが必須です。

メモリーに関する制約

移行先コンピュートノードでは、十分な RAM が利用可能でなければなりません。メモリーのオーバーサブスクリプションが、移行失敗の原因となる可能性があります。

ブロックマイグレーションに関する制約

インスタンスの使用するディスクがコンピュートノード上にローカルに格納されている場合、その移行には、共有ストレージ (Red Hat Ceph Storage 等) を使用するボリュームベースのインスタンスよりもはるかに長い時間がかかります。このレイテンシーは、OpenStack Compute (nova) がコンピュートノード間でローカルディスクをブロックごとに移行するために発生します。この処理は、デフォルトではコントロールプレーンネットワークを通じて行われます。これとは対照的に、Red Hat Ceph Storage 等の共有ストレージを使用するボリュームベースのインスタンスでは、ボリュームを移行する必要がありません。それぞれのコンピュートノードが、共有ストレージにアクセスできるためです。

注記

大量の RAM を消費するローカルディスクまたはインスタンスの移行により、コントロールプレーンネットワークに輻輳が生じ、コントロールプレーンネットワークを使用する他のシステム (RabbitMQ 等) のパフォーマンスに悪影響を及ぼす場合があります。

読み取り専用ドライブの移行に関する制約

ドライブの移行は、ドライブに読み取りおよび書き込み両方の機能がある場合に限りサポートされます。たとえば、OpenStack Compute (nova) は CD-ROM ドライブまたは読み取り専用のコンフィグドライブを移行することはできません。ただし、OpenStack Compute (nova) は、**vfat** 等のドライブ形式を持つコンフィグドライブなど、読み取りおよび書き込み両方の機能を持つドライブを移行することができます。

ライブマイグレーションに関する制約

インスタンスのライブマイグレーションでは、さらに制約が生じる場合があります。

重要

ライブマイグレーションは、移動されるワークロードのパフォーマンスに影響を与えます。Red Hat は、ライブマイグレーション中のパケット損失、ネットワーク遅延、メモリー遅延の増加、またはネットワーク帯域幅、メモリー帯域幅、ストレージ IO、または CPU パフォーマンスの低下をサポートしません。

移行中に新しい操作を行うことができない

移行元および移行先ノードのインスタンスのコピー間で状態の整合性を確保するために、RHOSP ではライブマイグレーション中の新規操作を拒否する必要があります。拒否しないと、ライブマイグレーションでメモリーの状態を複製する前にメモリーへの書き込みが行われた場合、ライブマイグレーションに長い時間がかかる状況や、永久に完了しない状況が発生する可能性があります。

NUMA を使用した CPU ピニング

Compute 設定の **NovaSchedulerEnabledFilters** パラメーターには、**AggregateInstanceExtraSpecsFilter** および **NUMATopologyFilter** の値が含まれている必要があります。

マルチセルクラウド

マルチセルクラウドでは、同じセル内の別のホストにインスタンスのライブマイグレーションを行うことができますが、セル全体では移行できません。

フローティングインスタンス

フローティングインスタンスのライブマイグレーションを行う場合、移行先コンピュートノードの **NovaComputeCpuSharedSet** の設定と移行元コンピュートノードの **NovaComputeCpuSharedSet** の設定が異なると、移行先コンピュートノードでは、インスタンスは共有の (ピニングされていない) インスタンス用に設定された CPU には割り当てられません。したがって、フローティングインスタンスのライブマイグレーションを行う必要がある場合は、専用の (ピニングされた) インスタンスおよび共有の (ピニングされていない) インスタンスに関して、すべてのコンピュートノードに同じ CPU マッピングを設定する必要があります。あるいは、共有のインスタンスにホストアグリゲートを使用します。

移行先コンピュートノードの容量

移行先コンピュートノードには、移行するインスタンスをホストするのに十分な空き容量が必要です。

SR-IOV ライブマイグレーション

SR-IOV ベースのネットワークインターフェイスを使用するインスタンスは、ライブマイグレーションが可能です。ダイレクトモードの SR-IOV ネットワークインターフェイスを持つインスタンスのライブマイグレーションでは、ネットワークのダウンタイムが発生します。これは、移行時に、ダイレクトモードのインターフェイスの接続を解除し再び接続する必要があるためです。

ML2/OVS デプロイメントでのライブマイグレーション

ライブマイグレーションプロセス中に、移行先ホストで仮想マシンの一時停止が解除されると、メタデータサーバープロキシーがまだ生成されていないため、メタデータサービスが利用できない可能性があります。この利用できない状態は長く続きません。すぐにサービスは利用可能になり、ライブマイグレーションは成功します。

ライブマイグレーションの妨げとなる制約

以下の機能を使用するインスタンスのライブマイグレーションを行うことはできません。

PCI パススルー

QEMU/KVM ハイパーバイザーでは、コンピュートノード上の PCI デバイスをインスタンスにアタッチすることができます。PCI パススルーを使用すると、インスタンスは PCI デバイスに排他的

にアクセスすることができ、これらのデバイスがインスタンスのオペレーティングシステムに物理的に接続されているかのように表示され、動作します。ただし、PCI パススルーには物理デバイスへの直接アクセスが必要なため、QEMU/KVM は PCI パススルーを使用するインスタンスのライブマイグレーションをサポートしません。

ポートリソースの要求

最小帯域幅を確保する QoS ポリシーなど、リソース要求が設定されたポートを使用するインスタンスのライブマイグレーションを行うことはできません。ポートにリソース要求があるかどうかを確認するには、以下のコマンドを使用します。

```
$ openstack port show <port_name/port_id>
```

15.3. 移行の準備

1 つまたは複数のインスタンスを移行する前に、コンピュートノード名および移行するインスタンスの ID を把握する必要があります。

手順

1. 移行元コンピュートノードのホスト名および移行先コンピュートノードのホスト名を特定します。

```
(undercloud)$ source ~/overcloudrc
(overcloud)$ openstack compute service list
```

2. 移行元コンピュートノード上のインスタンスをリスト表示し、移行するインスタンスの ID を特定します。

```
(overcloud)$ openstack server list --host <source> --all-projects
```

<source> を移行元コンピュートノードの名前または ID に置き換えてください。

3. オプション: ノードのメンテナンスを行うためにインスタンスを移行元コンピュートノードから移行する場合、ノードを無効にして、メンテナンス中にスケジューラーがノードに新規インスタンスを割り当てるのを防ぐ必要があります。

```
(overcloud)$ openstack compute service set <source> nova-compute --disable
```

<source> を移行元コンピュートノードのホスト名に置き換えてください。

これで移行を行う準備が整いました。Cold migrating an instance または Live migrating an instance で詳しく説明されている必須手順に従います。

15.4. インスタンスのコールドマイグレーション

インスタンスのコールドマイグレーションでは、インスタンスを停止して別のコンピュートノードに移動します。コールドマイグレーションは、PCI パススルーを使用するインスタンスの移行など、ライブマイグレーションでは対応することのできない移行シナリオに対応します。移行先コンピュートノードは、スケジューラーにより自動的に選択されます。詳細は、移行の制約 を参照してください。

手順

1. インスタンスのコールドマイグレーションを行うには、以下のコマンドを入力してインスタンスの電源をオフにして移動します。

```
(overcloud)$ openstack server migrate <instance> --wait
```

   - **<instance>** を移行するインスタンスの名前または ID に置き換えてください。
   - ローカルに確保されたボリュームを移行する場合には、**--block-migration** フラグを指定します。

2. 移行が完了するまで待ちます。インスタンスの移行が完了するのを待つ間、移行のステータスを確認することができます。詳細は、Checking migration status を参照してください。

3. インスタンスのステータスを確認します。

```
(overcloud)$ openstack server list --all-projects
```

   ステータスが VERIFY_RESIZE と表示される場合は、移行を確認する、または元に戻す必要があることを示しています。

   - 予想どおりに機能している場合は、移行を確認します。

```
(overcloud)$ openstack server resize --confirm <instance>
```

     <instance> を移行するインスタンスの名前または ID に置き換えてください。ステータスが ACTIVE と表示される場合は、インスタンスを使用する準備が整っていることを示しています。

   - 予想どおりに機能していない場合は、移行を元に戻します。

```
(overcloud)$ openstack server resize --revert <instance>
```

     <instance> をインスタンスの名前または ID に置き換えてください。

4. インスタンスを再起動します。

```
(overcloud)$ openstack server start <instance>
```

   <instance> をインスタンスの名前または ID に置き換えてください。

5. オプション: メンテナンスのために移行元コンピュートノードを無効にした場合は、新規インスタンスがノードに割り当てられるようにノードを再度有効にする必要があります。

```
(overcloud)$ openstack compute service set <source> nova-compute --enable
```

   <source> を移行元コンピュートノードのホスト名に置き換えてください。

15.5. インスタンスのライブマイグレーション

ライブマイグレーションでは、ダウンタイムを最小限に抑えて、インスタンスを移行元コンピュートノードから移行先コンピュートノードに移動します。ライブマイグレーションがすべてのインスタンスに適しているとは限りません。詳細は、移行の制約 を参照してください。

手順

1. インスタンスのライブマイグレーションを行うには、インスタンスおよび移行先コンピュートノードを指定します。

```
(overcloud)$ openstack server migrate <instance> --live-migration [--host <dest>] --wait
```

- **<instance>** をインスタンスの名前または ID に置き換えてください。
- **<dest>** を移行先コンピュートノードの名前または ID に置き換えてください。

注記

openstack server migrate コマンドは、共有ストレージを持つインスタンスの移行が対象です。これがデフォルトの設定です。ローカルに確保されたボリュームを移行するには、**--block-migration** フラグを指定します。

```
(overcloud)$ openstack server migrate <instance> --live-migration [--host
<dest>] --wait --block-migration
```

2. インスタンスが移行されていることを確認します。

```
(overcloud)$ openstack server show <instance>

+----------------------+-------------------------------------+
| Field                | Value                               |
+----------------------+-------------------------------------+
| ...                  | ...                                 |
| status               | MIGRATING                           |
| ...                  | ...                                 |
+----------------------+-------------------------------------+
```

3. 移行が完了するまで待ちます。インスタンスの移行が完了するのを待つ間、移行のステータスを確認することができます。詳細は、Checking migration status を参照してください。

4. インスタンスのステータスをチェックして、移行が成功したかどうかを確認します。

```
(overcloud)$ openstack server list --host <dest> --all-projects
```

<dest> を移行先コンピュートノードの名前または ID に置き換えてください。

5. オプション: メンテナンスのために移行元コンピュートノードを無効にした場合は、新規インスタンスがノードに割り当てられるようにノードを再度有効にする必要があります。

```
(overcloud)$ openstack compute service set <source> nova-compute --enable
```

<source> を移行元コンピュートノードのホスト名に置き換えてください。

15.6. 移行ステータスの確認

移行が完了するまでに、さまざまな移行状態を遷移します。正常な移行では、通常、移行状態は以下のように遷移します。

1. **Queued:** Compute サービスはインスタンス移行の要求を受け入れ、移行は保留中です。

2. **Preparing:** Compute サービスはインスタンス移行の準備中です。
3. **Running:** Compute サービスはインスタンスを移行中です。
4. **Post-migrating:** Compute サービスはインスタンスを移行先コンピュートノードにビルドし、移行元コンピュートノードのリソースを解放しています。
5. **Completed:** Compute サービスはインスタンスの移行を完了し、移行元コンピュートノードのリソース解放を終了しています。

手順

1. インスタンスの移行 ID のリストを取得します。

```
$ openstack server migration list --server <instance>
+----+-------------+-----------  (...)
| Id | Source Node | Dest Node | (...)
+----+-------------+-----------+ (...)
| 2  | -           | -         | (...)
+----+-------------+-----------+ (...)
```

<instance> をインスタンスの名前または ID に置き換えてください。

2. 移行のステータスを表示します。

```
$ openstack server migration show <instance> <migration_id>
```

- **<instance>** をインスタンスの名前または ID に置き換えてください。
- **<migration_id>** を移行の ID に置き換えてください。
  openstack server migration show コマンドを実行すると、次の出力例が返されます。

```
+------------------------+--------------------------------------+
| Property               | Value                                |
+------------------------+--------------------------------------+
| created_at             | 2017-03-08T02:53:06.000000           |
| dest_compute           | controller                           |
| dest_host              | -                                    |
| dest_node              | -                                    |
| disk_processed_bytes   | 0                                    |
| disk_remaining_bytes   | 0                                    |
| disk_total_bytes       | 0                                    |
| id                     | 2                                    |
| memory_processed_bytes | 65502513                             |
| memory_remaining_bytes | 786427904                            |
| memory_total_bytes     | 1091379200                           |
| server_uuid            | d1df1b5a-70c4-4fed-98b7-423362f2c47c |
| source_compute         | compute2                             |
| source_node            | -                                    |
| status                 | running                              |
| updated_at             | 2017-03-08T02:53:47.000000           |
+------------------------+--------------------------------------+
```

ヒント

Compute サービスは、コピーする残りのメモリーのバイト数によって移行の進捗を測定します。時間が経過してもこの数字が減少しない場合、移行を完了することができず、Compute サービスは移行を中止する可能性があります。

インスタンスの移行に長い時間がかったり、エラーが発生したりする場合があります。詳細は、Troubleshooting migration を参照してください。

15.7. インスタンスの退避

インスタンスを障害の発生したコンピュートノードまたはシャットダウンしたコンピュートノードから同じ環境内の新しいホストに移動する場合、インスタンスを退避させることができます。

退避プロセスにより元のインスタンスは破棄され、元のイメージ、インスタンス名、UUID、ネットワークアドレス、およびインスタンスに割り当てられていたその他すべてのリソースを使用して、別のコンピュートノードに元のインスタンスが再ビルドされます。

インスタンスが共有ストレージを使用する場合、インスタンスのルートディスクは退避プロセス中に再ビルドされません。移行先コンピュートノードが引き続きこのディスクにアクセス可能なためです。インスタンスが共有ストレージを使用しない場合は、インスタンスのルートディスクも移行先コンピュートノードに再ビルドされます。

注記

- コンピュートノードがフェンシングされ、API が報告するコンピュートノードの状態が down または forced-down である場合に限り、退避を行うことができます。コンピュートノードが down または forced-down と報告されない場合、**evacuate** コマンドは失敗します。
- クラウド管理者でなければ、退避を行うことはできません。

15.7.1. 単一のインスタンスの退避

インスタンスを一度に 1 つずつ退避させることができます。

手順

1. インスタンスが実行されていないことを確認します。

```
(overcloud)$ openstack server list --host <node> --all-projects
```

   - **<node>** をインスタンスをホストするコンピュートノードの名前または UUID に置き換えます。

2. ホストコンピュートノードのフェンシングまたはシャットダウンを確認します。

```
(overcloud)[stack@director ~]$ openstack baremetal node show <node>
```

   - **<node>** を退避するインスタンスをホストするコンピュートノードの名前または UUID に置き換えます。退避を実行するには、コンピュートノードのステータスが **down** または **forced-down** である必要があります。

3. コンピュートノードを無効にします。

```
(overcloud)[stack@director ~]$ openstack compute service set \
 <node> nova-compute --disable --disable-reason <disable_host_reason>
```

- **<node>** をインスタンスの退避元となるコンピュートノードの名前に置き換えてください。
- **<disable_host_reason>** をコンピュートノードを無効にした理由の詳細に置き換えます。

4. インスタンスを退避させます。

```
(overcloud)[stack@director ~]$ nova evacuate [--password <pass>] <instance> [<dest>]
```

- オプション: **<pass>** を、退避されたインスタンスへのアクセスに必要な管理パスワードに置き換えます。パスワードを指定しなかった場合には、無作為に生成され、退避の完了時に出力されます。

注記

パスワードは、一時インスタンスディスクがローカルのハイパーバイザーディスクに保存されている場合にのみ変更されます。インスタンスが共有ストレージでホストされる場合、または Block Storage ボリュームが割り当てられている場合はパスワードは変更されず、パスワードが変更されなかったことを通知するエラーメッセージは表示されません。

- **<instance>** を退避させるインスタンスの名前または ID に置き換えてください。
- オプション: **<dest>** をインスタンスの退避先となるコンピュートノードの名前に置き換えます。退避先コンピュートノードを指定しなかった場合には、Compute スケジューラーがノードを選択します。退避先に指定可能なコンピュートノードを確認するには、以下のコマンドを使用します。

```
(overcloud)[stack@director ~]$ openstack hypervisor list
```

5. オプション: コンピューティングノードは、復元されたら有効化します。

```
(overcloud)[stack@director ~]$ openstack compute service set \
 <node> nova-compute --enable
```

- **<node>** を、有効化するコンピューティングノードの名前に置き換えます。

15.7.2. ホスト上の全インスタンスの退避

指定したコンピュートノード上の全インスタンスを退避させることができます。

手順

1. 退避させるインスタンスが実行されていないことを確認します。

```
(overcloud)$ openstack server list --host <node> --all-projects
```

- **<node>** を、退避するインスタンスをホストするコンピュートノードの名前または UUID に置き換えます。

2. ホストコンピュートノードのフェンシングまたはシャットダウンを確認します。

–

```
(overcloud)[stack@director ~]$ openstack baremetal node show <node>
```

- **<node>** を、退避するインスタンスをホストするコンピュートノードの名前または UUID に置き換えます。退避を実行するには、コンピュートノードのステータスが **down** または **forced-down** である必要があります。

3. コンピュートノードを無効にします。

```
(overcloud)[stack@director ~]$ openstack compute service set \
 <node> nova-compute --disable --disable-reason <disable_host_reason>
```

- **<node>** をインスタンスの退避元となるコンピュートノードの名前に置き換えます。
- **<disable_host_reason>** をコンピュートノードを無効にした理由の詳細に置き換えます。

4. 指定したコンピュートノード上の全インスタンスを退避させます。

```
(overcloud)[stack@director ~]$ nova host-evacuate [--target_host <dest>] <node>
```

- **<dest>** をインスタンスの退避先となるコンピュートノードの名前に置き換えます。退避先を指定しなかった場合には、Compute スケジューラーがノードを選択します。退避先に指定可能なコンピュートノードを確認するには、以下のコマンドを使用します。

```
(overcloud)[stack@director ~]$ openstack hypervisor list
```

- **<node>** をインスタンスの退避元となるコンピュートノードの名前に置き換えます。

5. オプション: コンピューティングノードは、復元されたら有効化します。

```
(overcloud)[stack@director ~]$ openstack compute service set \
 <node> nova-compute --enable
```

- **<node>** を、有効化するコンピューティングノードの名前に置き換えます。

15.8. 移行に関するトラブルシューティング

インスタンスの移行時に、以下の問題が発生する可能性があります。

- 移行プロセスでエラーが生じる。
- 移行プロセスが終了しない。
- 移行後にインスタンスのパフォーマンスが低下する。

15.8.1. 移行中のエラー

以下の問題が発生すると、移行操作が **error** 状態に遷移します。

- 実行しているクラスターに異なるバージョンの Red Hat OpenStack Platform (RHOSP) が存在する。
- 指定したインスタンス ID が見つからない。
- 移行を試みているインスタンスが **error** 状態にある。

- Compute サービスが停止している。
- 競合状態が発生する。
- ライブマイグレーションが **failed** 状態に移行する。

ライブマイグレーションが **failed** 状態に移行すると、通常は **error** 状態になります。**failed** 状態の原因となる可能性のある典型的な問題を以下に示します。

- 移行先コンピュートホストが利用可能な状態にない。
- スケジューラーの例外が発生する。
- コンピューティングリソースが不十分なため、再ビルドプロセスに失敗する。
- サーバーグループの確認に失敗する。
- 移行先コンピュートノードへの移行が完了する前に、移行元コンピュートノードのインスタンスが削除される。

15.8.2. ライブマイグレーションのスタック

ライブマイグレーションが完了せず、永久に **running** 状態のままになる可能性があります。ライブマイグレーションが永久に完了しない一般的な理由は、Compute サービスがインスタンスの変更を移行先コンピュートノードに複製するより早く、クライアントのリクエストにより移行元コンピュートノード上で実行中のインスタンスに変更が生じることです。

この状況に対処するには、以下のいずれかの方法を使用します。

- ライブマイグレーションを中止する。
- ライブマイグレーションを強制的に完了させる。

ライブマイグレーションの中止

移行プロセスがインスタンスの状態の変化を移行先ノードにコピーするより早くインスタンスの状態が変化する状況で、インスタンスの動作を一時的に中断したくない場合には、ライブマイグレーションを中止することができます。

手順

1. インスタンスの移行のリストを取得します。

```
$ openstack server migration list --server <instance>
```

   <instance> をインスタンスの名前または ID に置き換えてください。

2. ライブマイグレーションを中止します。

```
$ openstack server migration abort <instance> <migration_id>
```

   - **<instance>** をインスタンスの名前または ID に置き換えてください。
   - **<migration_id>** を移行の ID に置き換えてください。

ライブマイグレーション完了の強制

移行プロセスがインスタンスの状態の変化を移行先ノードにコピーするより早くインスタンスの状態が変化する状況で、インスタンスの動作を一時的に中断して移行を強制的に完了させたい場合には、ライブマイグレーションの手順を強制的に完了させることができます。

重要

ライブマイグレーションを強制的に完了させると、かなりのダウンタイムが発生する可能性があります。

手順

1. インスタンスの移行のリストを取得します。

```
$ openstack server migration list --server <instance>
```

<instance> をインスタンスの名前または ID に置き換えてください。

2. ライブマイグレーションを強制的に完了させます。

```
$ openstack server migration force complete <instance> <migration_id>
```

- **<instance>** をインスタンスの名前または ID に置き換えてください。
- **<migration_id>** を移行の ID に置き換えてください。

15.8.3. 移行後のインスタンスパフォーマンスの低下

NUMA トポロジーを使用するインスタンスの場合、移行元および移行先コンピュートノードの NUMA トポロジーおよび設定は同一でなければなりません。移行先コンピュートノードの NUMA トポロジーでは、十分なリソースが利用可能でなければなりません。移行元および移行先コンピュートノード間で NUMA 設定が同一でない場合、ライブマイグレーションは成功するがインスタンスのパフォーマンスが低下する可能性があります。たとえば、移行元コンピュートノードは NIC 1 を NUMA ノード 0 にマッピングするが、移行先コンピュートノードは NIC 1 を NUMA ノード 5 にマッピングする場合、移行後にインスタンスはバス内の最初の CPU からのネットワークトラフィックを NUMA ノード 5 の別の CPU にルーティングし、トラフィックを NIC 1 にルーティングする可能性があります。その結果、予想されたとおりに動作はしますが、パフォーマンスが低下します。同様に、移行元コンピュートノードの NUMA ノード 0 では十分な CPU および RAM が利用可能だが、移行先コンピュートノードの NUMA ノード 0 にリソースの一部を使用するインスタンスがすでに存在する場合、インスタンスは正しく動作するがパフォーマンスが低下する可能性があります。詳細は、移行の制約 を参照してください。