Red Hat OpenStack Platform 15

インスタンス＆イメージガイド

インスタンスとイメージの管理

Last Updated: 2022-09-15

Red Hat OpenStack Platform 15 インスタンス＆イメージガイド

インスタンスとイメージの管理

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Instances_and_Images_Guide.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

インスタンス＆イメージガイドには、Red Hat OpenStack Platform 環境におけるインスタンスおよびイメージの管理の手順を記載しています。

目次

前書き 5

第1章 IMAGE サービス 6
1.1. IMAGE サービスについての理解 6
1.1.1. イメージの署名および検証 6
1.1.2. イメージの変換 7
1.1.3. イメージのイントロスペクション 7
1.1.4. 相互運用可能なイメージのインポート 8
1.1.5. Image サービスのキャッシュ機能を使用したスケーラビリティーの向上 8
1.2. イメージの管理 8
1.2.1. イメージの作成 9
1.2.1.1. Red Hat OpenStack Platform における KVM ゲストイメージの使用 9
1.2.1.2. Red Hat Enterprise Linux または Windows のカスタムイメージの作成 9
1.2.1.2.1. Red Hat Enterprise Linux 7 イメージの作成 10
1.2.1.2.2. Red Hat Enterprise Linux 6 イメージの作成 15
1.2.1.2.3. Windows イメージの作成 19
1.2.1.3. libosinfo の使用 21
1.2.2. イメージのアップロード 21
1.2.3. イメージの更新 22
1.2.4. イメージのインポート 23
1.2.4.1. リモート URI からのインポート 23
1.2.4.2. ローカルボリュームからのインポート 23
1.2.5. イメージの削除 24
1.2.6. イメージの表示または非表示 24
1.2.7. 非表示にしたイメージの表示 24
1.2.8. イメージ変換の有効化 25
1.2.9. RAW 形式へのイメージの変換 25
1.2.9.1. RAW および ISO だけを受け入れる Image サービスの設定 25
1.2.10. RAW 形式でのイメージの保存 26

第2章 COMPUTE (NOVA) サービスの設定 27
2.1. オーバーコミットのためのメモリー設定 28
2.2. 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 28
2.3. スワップサイズの計算 29

第3章 OPENSTACK COMPUTE 用のストレージの設定 30
3.1. アーキテクチャーの概要 30
3.2. 設定 31

第4章 仮想マシンインスタンス 34
4.1. インスタンスの管理 34
4.1.1. コンポーネントの追加 34
4.1.2. インスタンスの起動 34
4.1.2.1. インスタンスの起動オプション 35
4.1.3. インスタンスの更新 (アクションメニュー) 37
4.1.4. インスタンスのリサイズ 38
4.1.5. インスタンスへの接続 39
4.1.5.1. Dashboard を使用したインスタンスのコンソールへのアクセス 39
4.1.5.2. VNC コンソールへの直接接続 40
4.1.6. インスタンスの使用状況の表示 41
4.1.7. インスタンスの削除 41
4.1.8. 複数インスタンスの同時管理 41

4.2. インスタンスのセキュリティーの管理 42
4.2.1. キーペアの管理 42
4.2.1.1. キーペアの作成 42
4.2.1.2. キーペアのインポート 42
4.2.1.3. キーペアの削除 43
4.2.2. セキュリティーグループの作成 43
4.2.3. Floating IP アドレスの作成、割り当て、および解放 43
4.2.3.1. プロジェクトへの Floating IP アドレスの確保 43
4.2.3.2. Floating IP の割り当て 43
4.2.3.3. Floating IP の解放 44
4.2.4. インスタンスへのログイン 44
4.2.5. インスタンスへの admin パスワード挿入 45
4.3. フレーバーの管理 46
4.3.1. 設定パーミッションの更新 47
4.3.2. フレーバーの作成 47
4.3.3. 一般属性の更新 48
4.3.4. フレーバーのメタデータの更新 48
4.3.4.1. メタデータの表示 48
4.3.4.2. メタデータの追加 49
4.4. ホストアグリゲートの管理 54
4.4.1. ホストアグリゲートのスケジューリングの有効化 54
4.4.2. アベイラビリティーゾーンまたはホストアグリゲートの表示 55
4.4.3. ホストアグリゲートの追加 55
4.4.4. ホストアグリゲートの更新 55
4.4.5. ホストアグリゲートの削除 56
4.5. ホストのスケジュール 57
4.5.1. スケジューリングフィルターの設定 57
4.5.2. スケジューリングの重みの設定 60
4.5.2.1. ホストの重みのオプション設定 61
4.5.3. Placement サービスの特性の設定 62
4.5.3.1. Placement サービスの特性としての libvirt 仮想化ドライバー機能 63
4.5.3.2. Placement サービス特性を使用したリソースプロバイダー要求の指定 63
4.5.4. PCI デバイスがアタッチされている NUMA ノードの確保 64
4.5.5. 専用の物理 CPU で実行されるエミュレータースレッドの設定 65
4.6. インスタンスのスナップショットの管理 66
4.6.1. インスタンスのスナップショットの作成 66
4.6.2. スナップショットの管理 67
4.6.3. スナップショットの状態へのインスタンスの再ビルド 68
4.6.4. 一貫性のあるスナップショット 68
4.7. インスタンスのレスキューモードの使用 68
4.7.1. レスキューモードのインスタンス用イメージの準備 69
4.7.1.1. ext4 ファイルシステムを使用している場合のレスキューイメージ 69
4.7.2. OpenStack Image サービスへのレスキューイメージの追加 69
4.7.3. レスキューモードでのインスタンスの起動 70
4.7.4. インスタンスのアンレスキュー 70

第5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 72
5.1. 移行の種別 72
5.2. 移行の制約 74
5.3. 移行の準備 76
5.4. DPDK インスタンス向けの追加準備作業 76
5.5. インスタンスのコールドマイグレーション 78
5.6. インスタンスのライブマイグレーション 79

5.7. 移行ステータスの確認 81
5.8. インスタンスの退避 82
5.8.1. 単一のインスタンスの退避 82
5.8.2. ホスト上の全インスタンスの退避 83
5.8.3. 共有ストレージの設定 84
5.9. 移行に関するトラブルシューティング 85
5.9.1. 移行中のエラー 85
5.9.2. ライブマイグレーションのスタック 86
5.9.3. 移行後のインスタンスパフォーマンスの低下 87

第6章 インスタンス用のコンフィグドライブの設定 88
6.1. コンフィグドライブのオプション 88
6.2. コンフィグドライブの使用 88

第7章 パフォーマンス改善のためのコンピュートノードの設定 89
7.1. NUMA ノードを使用する CPU ピニングの設定 89
7.1.1. コンピュートノードの設定 90
7.1.2. スケジューラーの設定 91
7.1.3. アグリゲートとフレーバーの設定 91
7.2. コンピュートノードでのヒュージページの設定 93
7.2.1. インスタンスへのヒュージページの割り当て 95

第8章 ゲストインスタンス用の仮想 GPU の設定 97
8.1. サポートされる構成および制限 97
8.2. NVIDIA GRID VGPU のデプロイ 97
8.2.1. カスタム GPU オーバークラウドイメージのビルド 98
8.2.2. 仮想 GPU ロール、プロファイル、およびフレーバーの設定 100
8.2.3. 設定ファイルの準備とオーバークラウドのデプロイ 102
8.2.4. カスタム GPU ゲストイメージのビルド 104
8.2.5. インスタンスの仮想 GPU プロファイルの作成 106
8.2.6. 仮想 GPU インスタンスの起動およびテスト 107

第9章 REAL-TIME COMPUTE の設定 109
9.1. REAL-TIME 用コンピュートノードの準備 109
9.2. REAL-TIME COMPUTE ロールのデプロイメント 112
9.3. デプロイメントの例およびテストシナリオ 115
9.4. REAL-TIME インスタンスの起動およびチューニング 115

付録A イメージの設定パラメーター 118

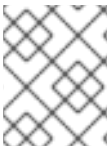
付録B インスタンスの起動ウィザードの有効化 129

前書き

Red Hat OpenStack Platform (RHOSP) は、Red Hat Enterprise Linux 上にプライベートまたはパブリックの Infrastructure-as-a-Service (IaaS) クラウドを構築するための基盤を提供します。これにより、スケーラビリティーが極めて高く、耐障害性に優れたプラットフォームをクラウド対応のワークロード開発にご利用いただくことができます。

本ガイドでは、イメージとインスタンスの作成/管理手順について説明します。また、Red Hat OpenStack Platform のインスタンス用ストレージの設定手順も記載しています。

OpenStack Dashboard またはコマンドラインクライアントを使用してクラウドの管理を行うことができます。大半の手順は、これらのいずれかの方法を使用することができますが、一部の高度な手順はコマンドラインのみで実行可能となっています。本ガイドでは、可能な場合には Dashboard を使用する手順を記載しています。



注記

Red Hat OpenStack Platform の全ドキュメントスイートは、Red Hat OpenStack Platform の製品ドキュメント を参照し てください。

第1章 IMAGE サービス

本章では、Red Hat OpenStack Platform でイメージとストレージを管理するための手順を説明します。

仮想マシンのイメージとは、起動可能なオペレーティングシステムがインストールされた仮想ディスクを含むファイルです。仮想マシンのイメージは、複数の形式をサポートしています。以下は、Red Hat OpenStack Platform で利用可能な形式です。

- **RAW**: 非構造化のディスクイメージ形式
- **QCOW2**: QEMU エミュレーターでサポートされているディスク形式。この形式には、QEMU 1.1 以降が必要な QCOW2v3 (QCOW3 と呼ばれる場合があります) が含まれます。
- **ISO**: ディスク上のデータをセクター単位でコピーし、バイナリーファイルに格納した形式
- **AKI**: Amazon Kernel Image
- **AMI**: Amazon Machine Image
- **ARI**: Amazon RAMDisk Image
- **VDI**: VirtualBox の仮想マシンモニターおよび QEMU エミュレーターでサポートされているディスク形式
- **VHD**: VMware、VirtualBox などの仮想マシンモニターで使用されている一般的なディスク形式
- **VMDK**: 数多くの一般的な仮想マシンモニターでサポートされているディスク形式

通常、仮想マシンイメージの形式に **ISO** は考慮されませんが、ISO にはオペレーティングシステムがインストール済みのブート可能なファイルシステムが含まれているので、他の形式の仮想マシンイメージファイルと同様に扱うことができます。

公式の Red Hat Enterprise Linux クラウドイメージをダウンロードするには、お使いのアカウントに有効な Red Hat Enterprise Linux サブスクリプションが必要です。

- Red Hat Enterprise Linux 8 KVM Guest Image
- Red Hat Enterprise Linux 7 KVM Guest Image
- Red Hat Enterprise Linux 6 KVM Guest Image

カスタマーポータルにログインしていない場合には、Red Hat アカウントの認証情報を入力するように求められます。

1.1. IMAGE サービスについての理解

OpenStack Image サービス (glance) は、以下のような注目すべき機能を提供しています。

1.1.1. イメージの署名および検証

イメージの署名および検証により、デプロイ担当者がイメージに署名して、その署名と公開鍵の証明書をイメージのプロパティーとして保存できるようにすることで、イメージの整合性と信頼性を保護します。

この機能を活用すると、以下が可能になります。

- 秘密鍵を使用してイメージに署名し、そのイメージ、署名、公開鍵の証明書 (検証メタデータ) への参照をアップロードすることができます。Image サービスは、署名が有効かどうかを検証します。
- Compute サービスでイメージを作成し、Compute サービスがそのイメージに署名し、イメージや検証メタデータをアップロードすることができます。Image サービスは、この場合も、署名が有効であるかどうかを検証します。
- Compute サービスで署名済みのイメージを要求することができます。Image サービスは、イメージと検証メタデータを提供します。これにより、Compute サービスはイメージを起動する前に検証することができます。

イメージの署名および検証に関する詳しい情報は、『**Manage Secrets with OpenStack Key Manager**』の「Validate Glance images」の章を参照してください。

1.1.2. イメージの変換

イメージの変換は、イメージのインポート中にタスク API を呼び出して、イメージを変換します。

インポートのワークフローの一環として、プラグインがイメージの変換機能を提供します。このプラグインは、デプロイ担当者の設定に基づいて、アクティブ化/非アクティブ化することができます。そのため、デプロイ担当者は、デプロイメントに希望のイメージ形式を指定する必要があります。

内部では、Image サービスが特定の形式でイメージのビットを受信します。これらのビットは、一時的な場所に保管されます。次にプラグインが起動されて、イメージを対象のフォーマットに変換し、最終的な保管場所に移動します。タスクが終了すると、一時的な場所は削除されます。このため、Image サービスでは最初にアップロードした形式は保持されません。

イメージの変換についての詳細は、「イメージ変換の有効化」を参照してください。

注記

イメージの変換は、イメージをインポートする時にだけトリガーされます。イメージのアップロード時には実行されません。以下に例を示します。

```
$ glance image-create-via-import \
    --disk-format qcow2 \
    --container-format bare \
    --name NAME \
    --visibility public \
    --import-method web-download \
    --uri http://server/image.qcow2
```

1.1.3. イメージのイントロスペクション

イメージ形式はすべて、イメージ自体の中にメタデータセットが埋め込まれた状態で提供されます。たとえば、ストリーム最適化 **VMDK** には、以下のようなパラメーターが含まれます。

```
$ head -20 so-disk.vmdk

# Disk DescriptorFile
version=1
CID=d5a0bce5
parentCID=ffffffff
createType="streamOptimized"
```

```
# Extent description
RDONLY 209714 SPARSE "generated-stream.vmdk"

# The Disk Data Base
#DDB

ddb.adapterType = "buslogic"
ddb.geometry.cylinders = "102"
ddb.geometry.heads = "64"
ddb.geometry.sectors = "32"
ddb.virtualHWVersion = "4"
```

この **vmdk** をイントロスペクションすることにより、**disk_type** が **streamOptimized** で、**adapter_type** が **buslogic** であることを簡単に確認することができます。これらのメタデータパラメーターは、イメージのコンシューマーに役立ちます。Compute では、**streamOptimized** ディスクをインスタンス化するワークフローは、**flat** ディスクをインスタンス化するワークフローとは完全に異なります。この新機能により、メタデータの抽出が可能となります。イメージのイントロスペクションは、イメージのインポート中に、タスク API を呼び出すことによって実行できます。管理者は、メタデータの設定をオーバーライドすることができます。

1.1.4. 相互運用可能なイメージのインポート

OpenStack Image サービスでは、相互運用可能なイメージインポートワークフローを使用して、イメージをインポートするための 2 つの方法が提供されています。

- URI からイメージをインポートする **web-download** (デフォルト)
- ローカルファイルシステムからインポートする **glance-direct**

1.1.5. Image サービスのキャッシュ機能を使用したスケーラビリティーの向上

glance-api キャッシュメカニズムを使用して、ローカルマシンにイメージのコピーを保存し、イメージを自動的に取得してスケーラビリティーを向上させます。Image サービスのキャッシュ機能を使用することで、複数のホスト上で glance-api を実行することができます。つまり、同じイメージをバックエンドストレージから何度も取得する必要はありません。Image サービスのキャッシュ機能は、Image サービスの動作には一切影響を与えません。

TripleO heat テンプレートを使用して Image サービスのキャッシュ機能を設定するには、以下の手順を実施します。

手順

1. 環境ファイルの **GlanceCacheEnabled** パラメーターの値を **true** に設定します。これにより、**glance-api.conf** Heat テンプレートの **flavor** の値が自動的に **keystone+cachemanagement** に設定されます。

```
parameter_defaults:
  GlanceCacheEnabled: true
```

2. オーバークラウドを再デプロイする際に、**openstack overcloud deploy** コマンドにその環境ファイルを追加します。

1.2. イメージの管理

OpenStack Image サービス (glance) は、ディスクおよびサーバーイメージの検出、登録、および配信のサービスを提供します。サーバーイメージのコピーやスナップショットを作成して直ちに保管する機能を提供します。保管したイメージは、テンプレートとして使用し、新規サーバーを迅速に稼働させるのに使用することができます。これはサーバーのオペレーティングシステムをインストールしてサービスを個別に設定するよりも一貫性の高い方法です。

1.2.1. イメージの作成

本項では、Red Hat Enterprise Linux 7、Red Hat Enterprise Linux 6 または Windows の ISO ファイルを使用して、QCOW2 形式の OpenStack 互換イメージを手動で作成する手順について説明します。

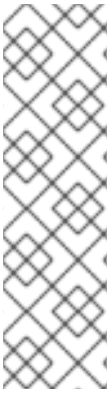
1.2.1.1. Red Hat OpenStack Platform における KVM ゲストイメージの使用

すでに準備済みの RHEL KVM ゲスト QCOW2 イメージを使用することができます。

- Red Hat Enterprise Linux 8 KVM Guest Image
- Red Hat Enterprise Linux 7 KVM Guest Image
- Red Hat Enterprise Linux 6 KVM Guest Image

これらのイメージは、**cloud-init** を使用して設定されます。適切に機能させるには、ec2 互換のメタデータサービスを利用して SSH キーをプロビジョニングする必要があります。

準備済みの Windows KVM ゲスト QCOW2 イメージはありません。



注記

KVM ゲストイメージでは、以下の点に注意してください。

- KVM ゲストイメージでは **root** アカウントが無効になっていますが、**cloud-user** という名前の特別なユーザーに **sudo** アクセスが許可されています。
- このイメージには **root** パスワードは設定されていません。

root パスワードは、**/etc/shadow** で 2 番目のフィールドに **!!** と記載することによりロックされます。

OpenStack インスタンスでは、OpenStack Dashboard またはコマンドラインから ssh キーペアを生成し、その鍵の組み合わせを使用して、インスタンスに対して root として SSH 公開認証を実行することを推奨します。

インスタンスの起動時には、この公開鍵がインスタンスに挿入されます。続いて、キーペア作成時にダウンロードされた秘密鍵を使用して認証を行うことができます。

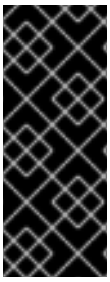
キーペアを使用しない場合には、「インスタンスへの **admin** パスワード挿入」の手順を使用して設定されている **admin** パスワードを使用することができます。

Red Hat Enterprise Linux または Windows のカスタムイメージを作成する場合は、「Red Hat Enterprise Linux 7 イメージの作成」、「Red Hat Enterprise Linux 6 イメージの作成」 、または「Windows イメージの作成」を参照してください。

1.2.1.2. Red Hat Enterprise Linux または Windows のカスタムイメージの作成

前提条件

- イメージを作成する Linux ホストマシン。これは、Linux パッケージをインストール/実行することのできる任意のマシンです。
- libvirt、virt-manager (**dnf groupinstall -y @virtualization** のコマンドを実行)。ゲストオペレーティングシステムを作成するのに必要な全パッケージがインストールされます。
- Libguestfs ツール (**dnf install -y libguestfs-tools-c** のコマンドを実行)。仮想マシンイメージにアクセスして変更を行うためのツールセットがインストールされます。
- Red Hat Enterprise Linux 7 もしくは 6 ISO ファイル (RHEL 7.2 Binary DVD もしくは RHEL 6.8 Binary DVD を参照)、または Windows ISO ファイル。Windows ISO ファイルがない場合には、Microsoft TechNet Evaluation Center に移動して評価版イメージをダウンロードしてください。
- **キックスタート** ファイルを編集する必要がある場合にはテキストエディター (RHEL のみ)



重要

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

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

注記

以下の手順では、プロンプトに **[root@host]#** と表示されているコマンドはすべて、お使いのホストマシンで実行する必要があります。

1.2.1.2.1. Red Hat Enterprise Linux 7 イメージの作成

本項では、Red Hat Enterprise Linux 7 の ISO ファイルを使用して、QCOW2 形式の OpenStack 互換イメージを手動で作成する手順について説明します。

1. 以下に示したように **virt-install** でインストールを開始します。

```
[root@host]# qemu-img create -f qcow2 rhel7.qcow2 8G
[root@host]# virt-install --virt-type kvm --name rhel7 --ram 2048 \
--cdrom /tmp/rhel-server-7.2-x86_64-dvd.iso \
--disk rhel7.qcow2,format=qcow2 \
--network=bridge:virbr0 --graphics vnc,listen=0.0.0.0 \
--noautoconsole --os-type=linux --os-variant=rhel7
```

このコマンドによりインスタンスが起動し、インストールプロセスが開始されます。

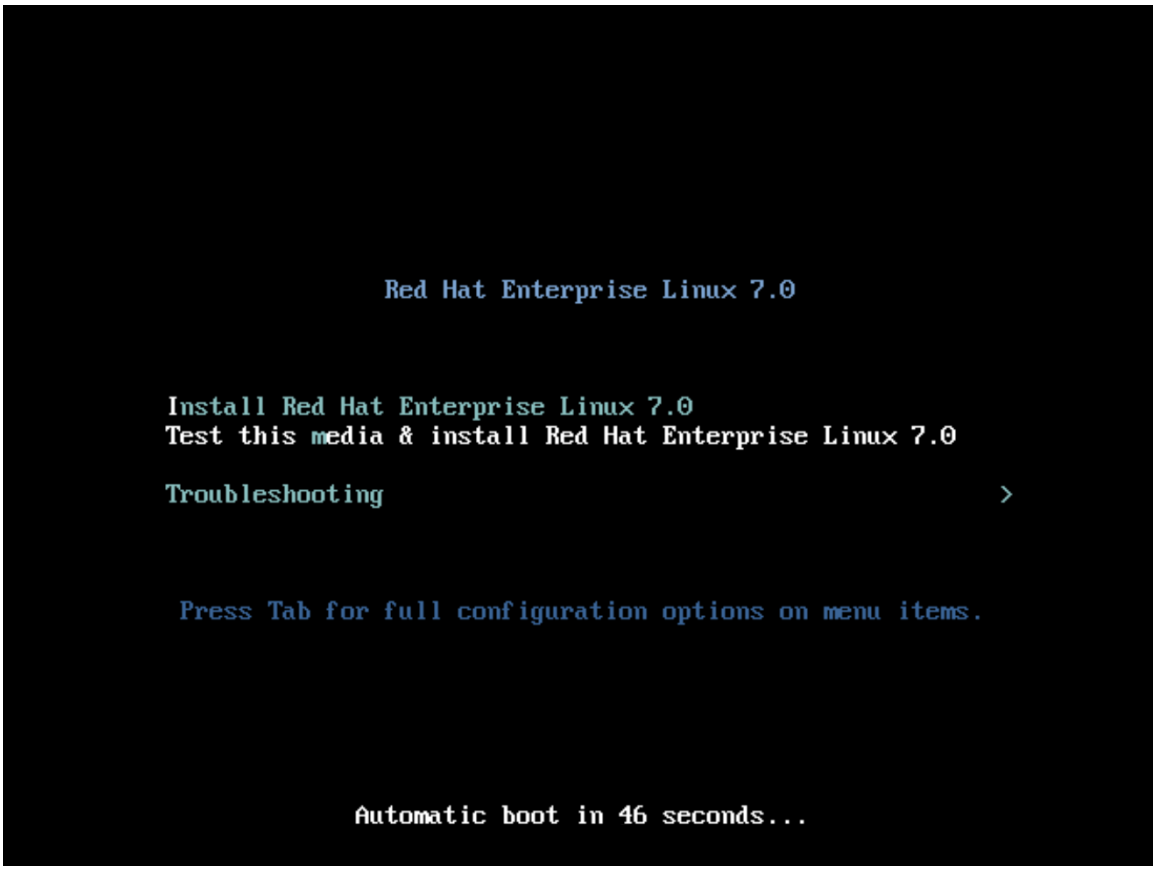
注記

インスタンスが自動的に起動しない場合には、**virt-viewer** のコマンドを実行して、コンソールを確認します。

```
[root@host]# virt-viewer rhel7
```

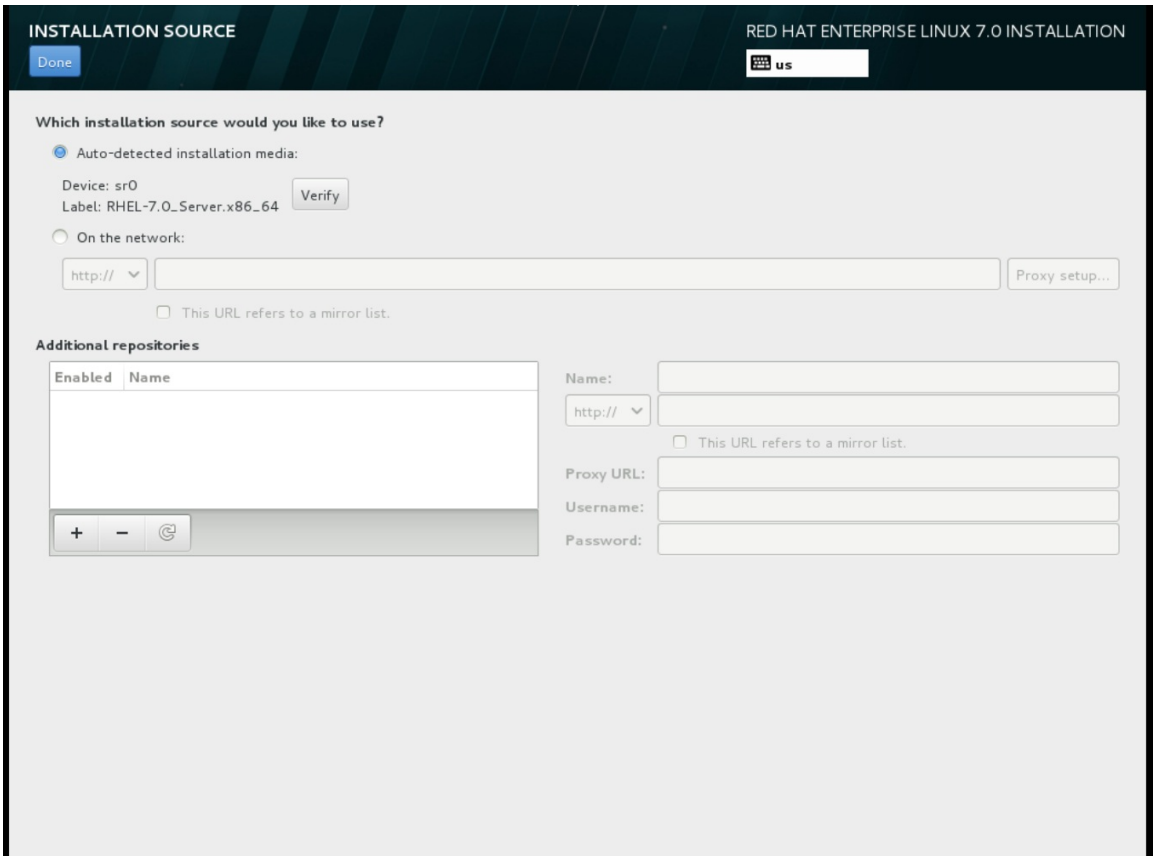
2. 以下の手順に従って、仮想マシンを設定します。

a. インストーラーの初期起動メニューで、**Install Red Hat Enterprise Linux 7.X** のオプションを選択します。

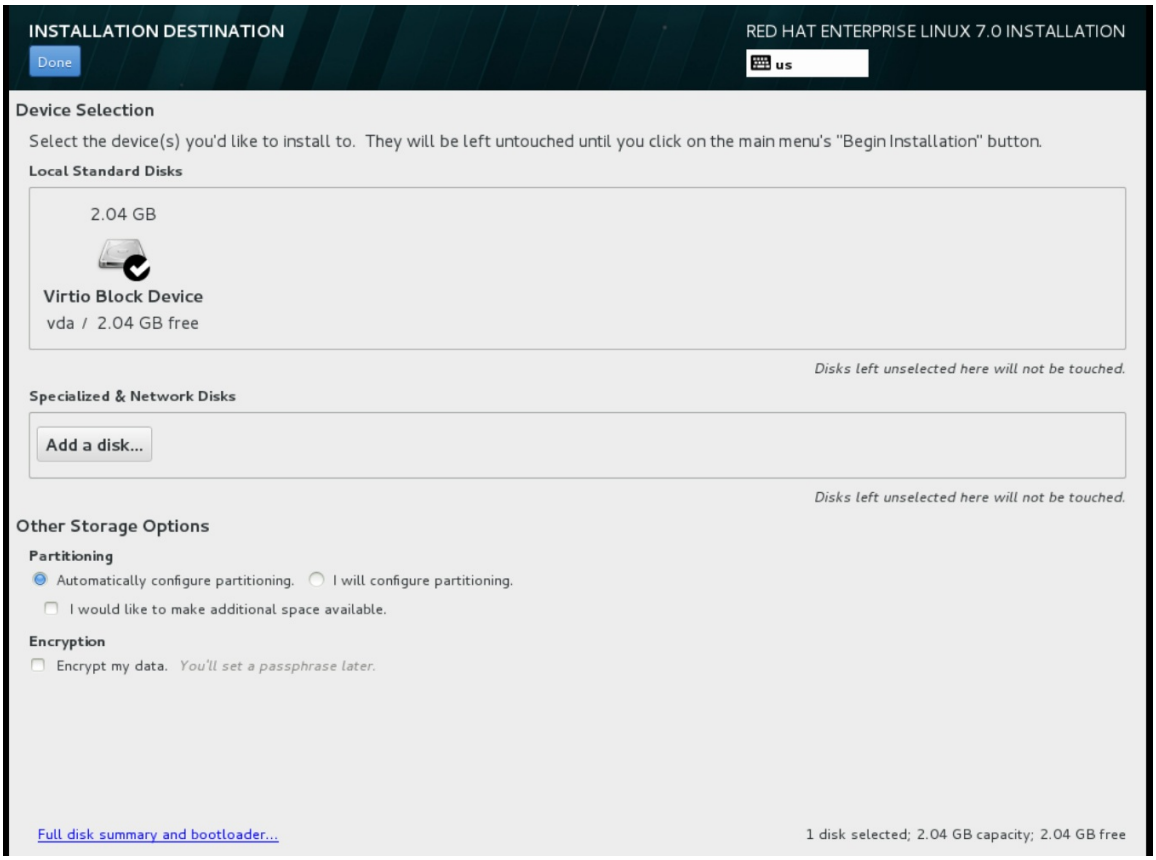


b. 適切な **言語** および **キーボード** オプションを選択します。

c. インストールに使用するデバイスタイプを尋ねるプロンプトが表示されたら、**自動検出したインストールメディア** を選択します。



d. インストール先を尋ねるプロンプトが表示されたら、**ローカルの標準ディスク** を選択します。



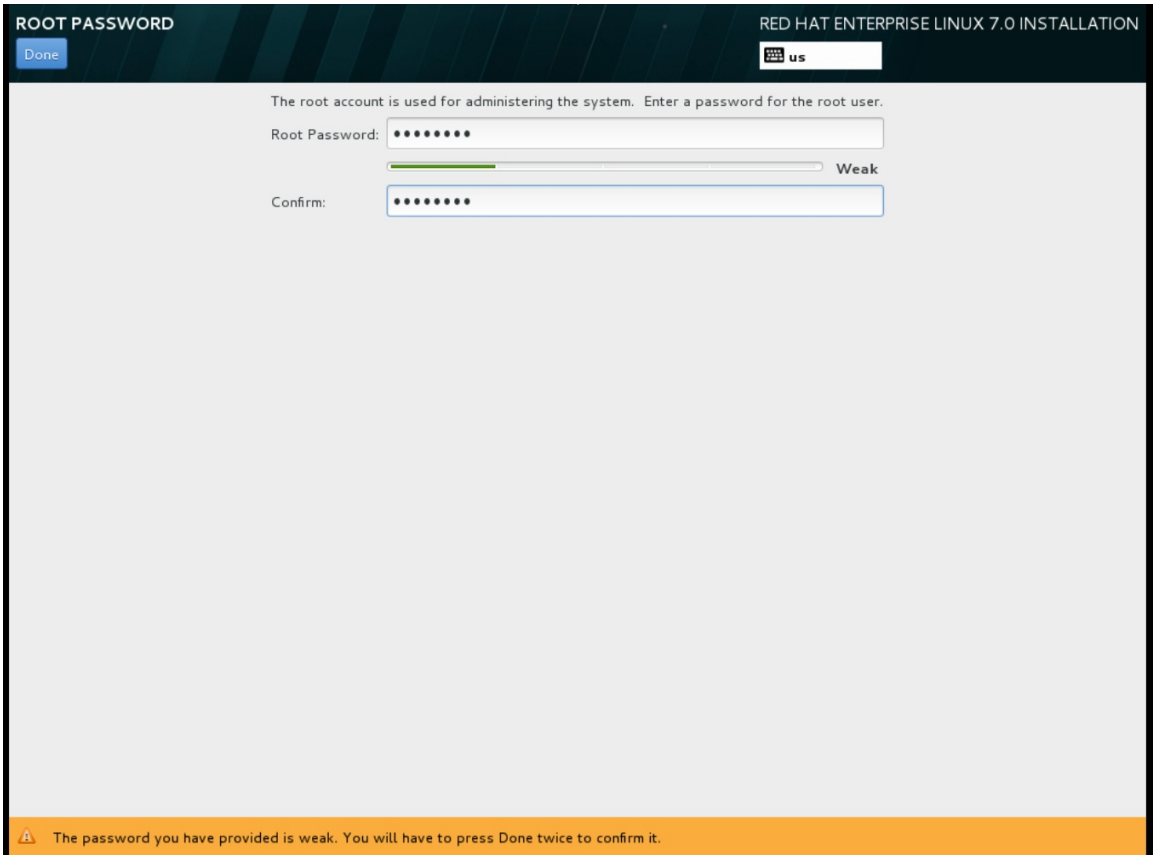
その他のストレージタイプオプションには、**自動構成のパーティション構成** を選択します。

e. ソフトウェアのオプションには、**最小限のインストール** を選択します。

f. ネットワークとホスト名の設定では、イーサネットに **eth0** を選択し、デバイスの **ホスト名** を指定します。デフォルトのホスト名は **localhost.localdomain** です。

g. **root** パスワードを選択します。

g. **root** パスワードを選択します。



インストールプロセスが完了すると、**完了しました!** の画面が表示されます。

3. インストールが完了した後には、インスタンスを再起動して、root ユーザーとしてログインします。

4. **/etc/sysconfig/network-scripts/ifcfg-eth0** ファイルを編集して、以下の値のみが記載されている状態にします。

```
TYPE=Ethernet
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=dhcp
NM_CONTROLLED=no
```

5. マシンを再起動します。

6. コンテンツ配信ネットワークにマシンを登録します。

```
# sudo subscription-manager register
# sudo subscription-manager attach --pool=Valid-Pool-Number-123456
# sudo subscription-manager repos --enable=rhel-7-server-rpms
```

7. システムを更新します。

```
# dnf -y update
```

8. **cloud-init** パッケージをインストールします。

```
# dnf install -y cloud-utils-growpart cloud-init
```

9. **/etc/cloud/cloud.cfg** 設定ファイルを編集して、**cloud_init_modules** の下に以下を追加します。

```
- resolv-conf
```

resolv-conf オプションは、インスタンスの初回起動時に **resolv.conf** を自動的に設定します。このファイルには、**nameservers**、**domain**、その他のオプションなどのインスタンスに関連した情報が記載されています。

10. **/etc/sysconfig/network** に以下の行を追加し、EC2 メタデータサービスへのアクセスで問題が発生するのを回避します。

```
NOZEROCONF=yes
```

11. コンソールメッセージが Dashboard の **ログ** タブおよび **nova console-log** の出力に表示されるようにするには、以下のブートオプションを **/etc/default/grub** ファイルに追記します。

```
GRUB_CMDLINE_LINUX_DEFAULT="console=tty0 console=ttyS0,115200n8"
```

grub2-mkconfig コマンドを実行します。

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

以下のような出力が表示されます。

```
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.10.0-229.7.2.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0-229.7.2.el7.x86_64.img
Found linux image: /boot/vmlinuz-3.10.0-121.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0-121.el7.x86_64.img
Found linux image: /boot/vmlinuz-0-rescue-b82a3044fb384a3f9aeacf883474428b
Found initrd image: /boot/initramfs-0-rescue-b82a3044fb384a3f9aeacf883474428b.img
done
```

12. 仮想マシンの登録を解除して、作成されるイメージをベースにクローン作成される全インスタンスに同じサブスクリプション情報が含まれないようにします。

```
# subscription-manager repos --disable=*
# subscription-manager unregister
# dnf clean all
```

13. インスタンスの電源をオフにします。

```
# poweroff
```

14. **virt-sysprep** コマンドでイメージのリセットおよびクリーニングをして、問題なくインスタンスの作成に使用できるようにします。

```
[root@host]# virt-sysprep -d rhel7
```

15. **virt-sparsify** コマンドを使用してイメージのサイズを縮小します。このコマンドにより、ディスクイメージ内の空き容量は、ホスト内の空き容量に戻ります。

```
[root@host]# virt-sparsify --compress /tmp/rhel7.qcow2 rhel7-cloud.qcow2
```

このコマンドを実行すると、その場所に **rhel7-cloud.qcow2** ファイルが作成されます。

rhel7-cloud.qcow2 イメージファイルを Image サービスにアップロードする準備が整いました。Dashboard を使用して OpenStack デプロイメントにこのイメージをアップロードする方法については、「イメージのアップロード」を参照してください。

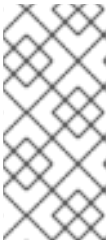
1.2.1.2.2. Red Hat Enterprise Linux 6 イメージの作成

本項では、Red Hat Enterprise Linux 6 の ISO ファイルを使用して、QCOW2 形式の OpenStack 互換イメージを手動で作成する手順について説明します。

1. **virt-install** でインストールを開始します。

```
[root@host]# qemu-img create -f qcow2 rhel6.qcow2 4G
[root@host]# virt-install --connect=qemu:///system --network=bridge:virbr0 \
--name=rhel6 --os-type linux --os-variant rhel6 \
--disk path=rhel6.qcow2,format=qcow2,size=10,cache=none \
--ram 4096 --vcpus=2 --check-cpu --accelerate \
--hvm --cdrom=rhel-server-6.8-x86_64-dvd.iso
```

このコマンドによりインスタンスが起動し、インストールプロセスが開始されます。



注記

インスタンスが自動的に起動しない場合には、**virt-viewer** のコマンドを実行して、コンソールを確認します。

```
[root@host]# virt-viewer rhel6
```

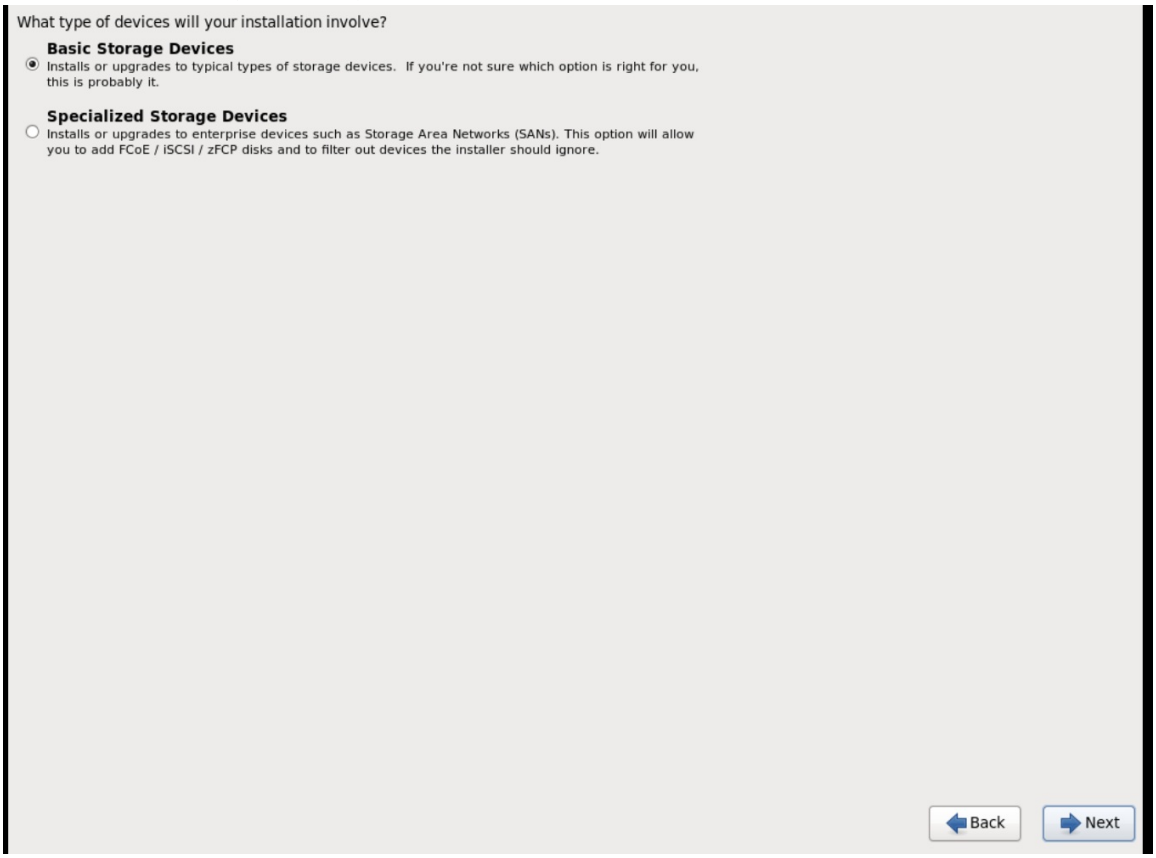
2. 仮想マシンを以下のように設定します。

   a. インストーラーの初期起動メニューで、**Install or upgrade an existing system**のオプションを選択します。

インストールのプロンプトに従って順に進みます。デフォルト値を受け入れます。
インストーラーは、ディスクの有無を確認して、インストール前にインストールメディアのテストを行うかどうかを決定するように促します。テストを実行するには **OK** を、テストを行わずに続行するには **Skip** を選択します。

b. 適切な **言語** および **キーボード** オプションを選択します。

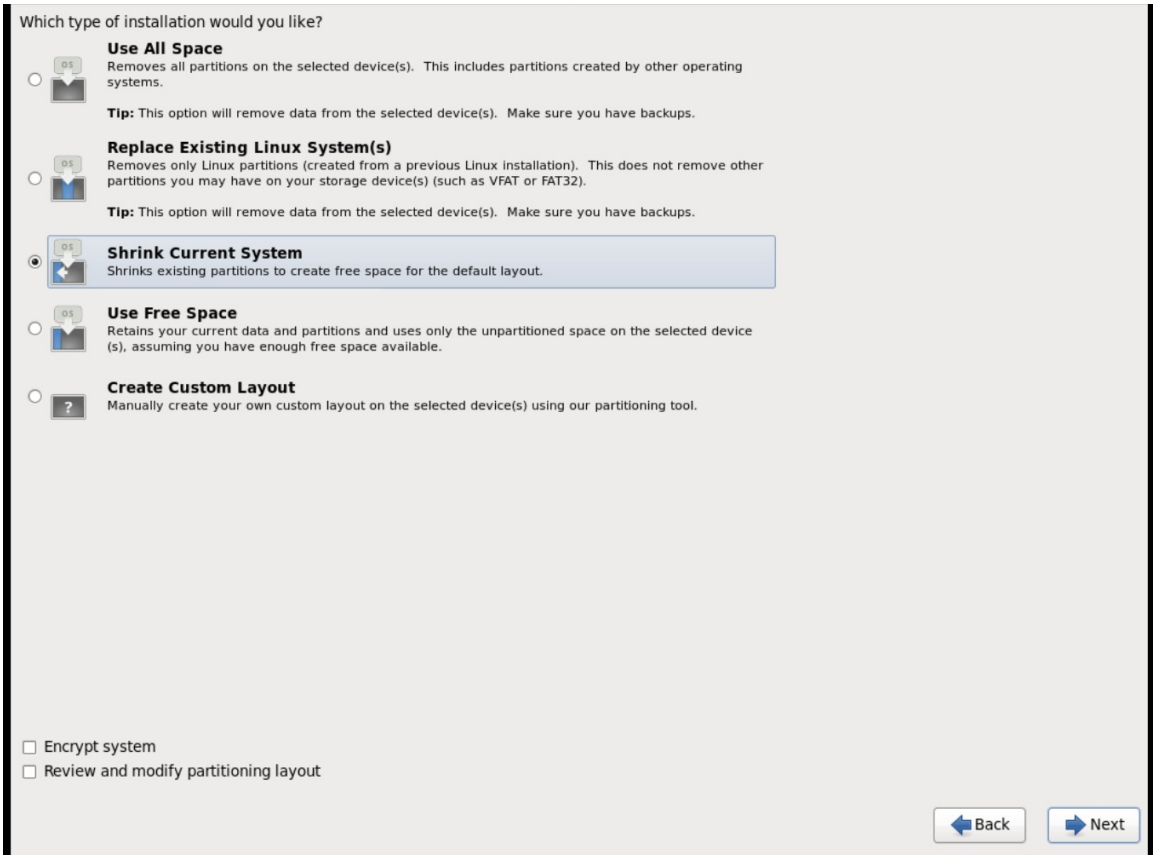
c. インストールに使用するデバイスタイプを尋ねるプロンプトが表示されたら、**基本ストレージデバイス** を選択します。



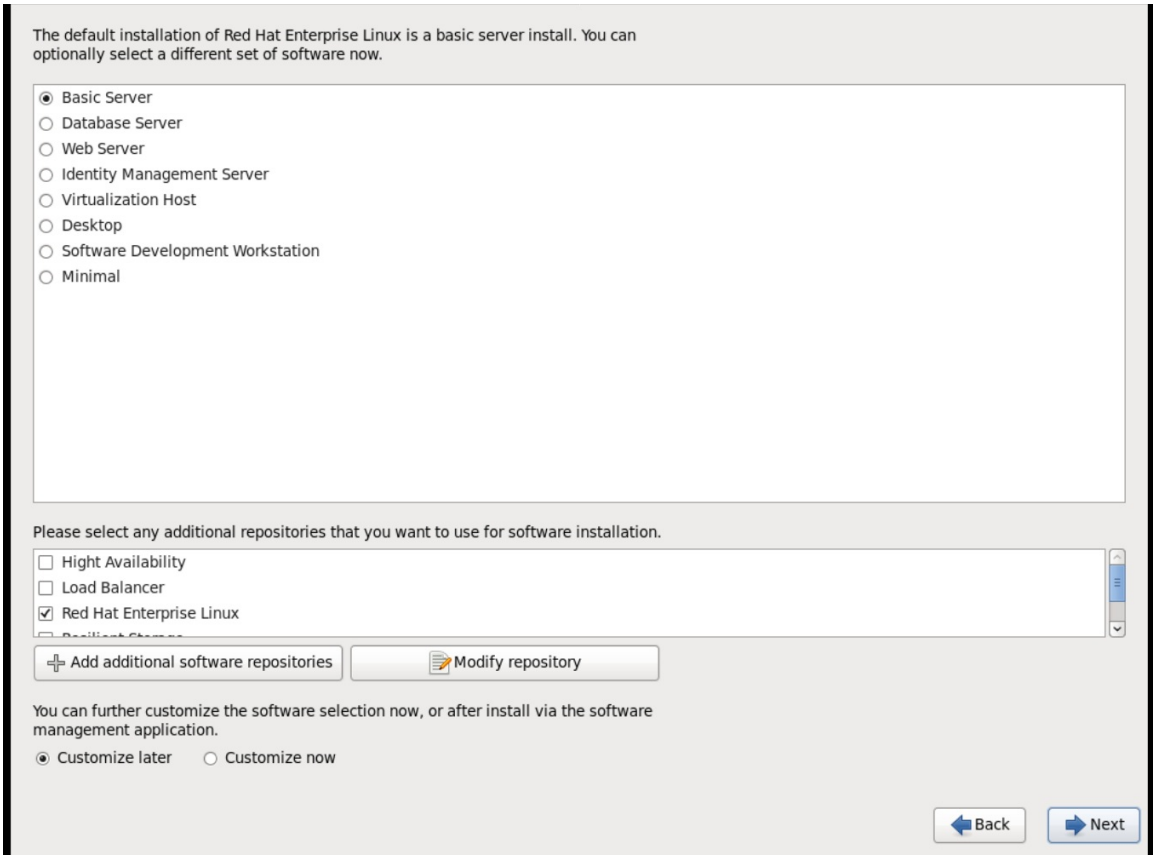
d. デバイスの **ホスト名** を指定します。デフォルトのホスト名は **localhost.localdomain** です。

e. **タイムゾーン** と **root** パスワードを指定します。

f. ディスクの空き容量に応じて、インストールのタイプを選択します。



g. SSH サーバーをインストールする **基本サーバー** インストールを選択します。



h. インストールプロセスが完了し、**おめでとうございます。Red Hat Enterprise Linux のインストールが完了しました。**の画面が表示されます。

3. インスタンスを再起動して、**root** ユーザーとしてログインします。

4. **/etc/sysconfig/network-scripts/ifcfg-eth0** ファイルを編集して、以下の値のみが記載されている状態にします。

```
TYPE=Ethernet
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=dhcp
NM_CONTROLLED=no
```

5. マシンを再起動します。

6. コンテンツ配信ネットワークにマシンを登録します。

```
# sudo subscription-manager register
# sudo subscription-manager attach --pool=Valid-Pool-Number-123456
# sudo subscription-manager repos --enable=rhel-6-server-rpms
```

7. システムを更新します。

```
# dnf -y update
```

8. **cloud-init** パッケージをインストールします。

```
# dnf install -y cloud-utils-growpart cloud-init
```

9. **/etc/cloud/cloud.cfg** 設定ファイルを編集して、**cloud_init_modules** の下に以下を追加します。

```
- resolv-conf
```

resolv-conf オプションは、インスタンスの初回起動時に **resolv.conf** 設定ファイルを自動的に設定します。このファイルには、**nameservers**、**domain**、その他のオプションなどのインスタンスに関連した情報が記載されています。

10. ネットワークの問題が発生するのを防ぐために、以下のように **/etc/udev/rules.d/75-persistent-net-generator.rules** ファイルを作成します。

```
# echo "#" > /etc/udev/rules.d/75-persistent-net-generator.rules
```

これにより、**/etc/udev/rules.d/70-persistent-net.rules** ファイルが作成されるのを防ぎます。**/etc/udev/rules.d/70-persistent-net.rules** が作成されてしまうと、スナップショットからのブート時にネットワークが適切に機能しなくなる可能性があります (ネットワークインターフェースが「eth0」ではなく「eth1」として作成され、IP アドレスが割り当てられません)。

11. **/etc/sysconfig/network** に以下の行を追加し、EC2 メタデータサービスへのアクセスで問題が発生するのを回避します。

```
NOZEROCONF=yes
```

12. コンソールメッセージが Dashboard の **ログ** タブおよび **nova console-log** の出力に表示されるようにするには、以下のブートオプションを **/etc/grub.conf** ファイルに追記します。

```
console=tty0 console=ttyS0,115200n8
```

13. 仮想マシンの登録を解除して、作成されるイメージをベースにクローン作成されるインスタンスすべてに同じサブスクリプション情報が含まれないようにします。

```
# subscription-manager repos --disable=*
# subscription-manager unregister
# dnf clean all
```

14. インスタンスの電源をオフにします。

```
# poweroff
```

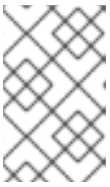
15. **virt-sysprep** コマンドでイメージのリセットおよびクリーニングをして、問題なくインスタンスの作成に使用できるようにします。

```
[root@host]# virt-sysprep -d rhel6
```

16. **virt-sparsify** コマンドを使用してイメージのサイズを縮小します。このコマンドにより、ディスクイメージ内の空き容量は、ホスト内の空き容量に戻ります。

```
[root@host]# virt-sparsify --compress rhel6.qcow2 rhel6-cloud.qcow2
```

このコマンドを実行すると、その場所に新しい **rhel6-cloud.qcow2** ファイルが作成されます。



注記

インスタンスに適用されているフレーバーのディスクスペースに応じて、イメージをベースとするインスタンスのパーティションを手動でリサイズする必要があります。

rhel6-cloud.qcow2 イメージファイルを Image サービスにアップロードする準備が整いました。Dashboard を使用して OpenStack デプロイメントにこのイメージをアップロードする方法については、「イメージのアップロード」を参照してください。

1.2.1.2.3. Windows イメージの作成

本項では、Windows の ISO ファイルを使用して、QCOW2 形式の OpenStack 互換イメージを手動で作成する手順について説明します。

1. 以下に示したように **virt-install** でインストールを開始します。

```
[root@host]# virt-install --name=name \
--disk size=size \
--cdrom=path \
--os-type=windows \
--network=bridge:virbr0 \
--graphics spice \
--ram=RAM
```

以下のように **virt-install** パラメーターの値を置き換えます。

- **name**: Windows ゲストに指定する必要のある名前
- **size**: ディスクのサイズ (GB)
- **path**: Windows のインストール ISO ファイルへのパス
- **RAM**: 要求するメモリー容量 (MB)

注記

--os-type=windows パラメーターにより、Windows ゲストのクロックが正しく設定され、Hyper-V エンライトメント機能が有効化されるようになります。

デフォルトでは、**virt-install** は **/var/lib/libvirt/images/name.qcow2** としてゲストイメージを保存します。ゲストイメージを別の場所に保存するには、以下のように **--disk** のオプションを変更してください。

```
--disk path=filename,size=size
```

filename は、ゲストイメージを保存する必要のあるファイル名 (およびオプションでそのパス) に置き換えてください。たとえば、**path=win8.qcow2,size=8** は現在の作業ディレクトリーに **win8.qcow2** という名前の 8 GB のファイルを作成します。

ヒント

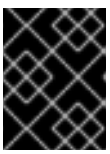
ゲストが自動的に起動しない場合には、**virt-viewer** のコマンドを実行して、コンソールを確認します。

```
[root@host]# virt-viewer name
```

2. Windows システムのインストールは、本書の対象範囲外となります。Windows のインストール方法に関する説明は、適切な Microsoft のドキュメントを参照してください。

3. 新規インストールした Windows システムで仮想化ハードウェアを使用できるようにするには、Windows システムに **virtio ドライバー** をインストールする必要がある場合があります。これには、まずホストシステムで **virtio-win** パッケージをインストールします。このパッケージには virtio ISO イメージが含まれており、そのイメージを CD-ROM ドライブとして Windows ゲストにアタッチします。「KVM 準仮想化 (virtio) ドライバー」(**『仮想化の導入および管理ガイド』**) を参照し、**virtio-win** パッケージのインストール、ゲストへの virtio ISO イメージの追加、virtio ドライバーのインストール方法を確認してください。

4. Windows システムで Cloudbase-Init をダウンロード、実行して、設定を完了します。Cloudbase-Init のインストールの最後に、**Run Sysprep** と **Shutdown** チェックボックスを選択します。**Sysprep** ツールは、特定の Microsoft サービスで使用する OS ID を生成して、ゲストを一意にします。



重要

Red Hat は Cloudbase-Init に関するテクニカルサポートは提供しません。問題が発生した場合には、contact Cloudbase Solutions からお問い合わせください。

Windows システムがシャットダウンしたら、**name.qcow2** イメージファイルを Image サービスにアップロードすることができます。Dashboard またはコマンドラインを使用して OpenStack デプロイメントにこのイメージをアップロードする方法については、「イメージのアップロード」を参照してください。

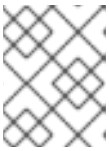
1.2.1.3. libosinfo の使用

Image サービス (glance) はイメージ用に **libosinfo** のデータを処理して、インスタンスに最適な仮想ハードウェアをより簡単に設定できるようにすることができます。これは、libosinfo 形式のオペレーティングシステム名前を glance イメージに追加することによって行うことができます。

1. 以下の例では、ID **654dbfd5-5c01-411f-8599-a27bd344d79b** のイメージが **rhel7.2** という libosinfo の値を使用するように指定します。

```
$ openstack image set 654dbfd5-5c01-411f-8599-a27bd344d79b --property
os_name=rhel7.2
```

その結果、Compute は、**654dbfd5-5c01-411f-8599-a27bd344d79b** のイメージを使用してインスタンスがビルドされる際に必ず、**rhel7.2** 向けに最適化された仮想ハードウェアを提供するようになります。



注記

libosinfo の値の完全な一覧は、libosinfo プロジェクトを参照してください (https://gitlab.com/libosinfo/osinfo-db/tree/master/data/os)。

1.2.2. イメージのアップロード

1. Dashboard で **プロジェクト** > **コンピュート** > **イメージ**を選択します。
2. **イメージの作成** をクリックします。
3. 各フィールドに値を入力し、完了したら **イメージの作成** をクリックします。

表1.1 イメージのオプション

フィールド	説明
名前	イメージの名前。そのプロジェクト内で一意な名前にする必要があります。
説明	イメージを識別するための簡単な説明
イメージソース	イメージソース: **イメージの場所** または **イメージファイル**。ここで選択したオプションに応じて次のフィールドが表示されます。
イメージの場所またはイメージファイル	• イメージの場所の URL を指定するには、**イメージの場所** オプションを選択します。 • ローカルディスクからイメージをアップロードするには、**イメージファイル** オプションを選択します。
形式	イメージの形式 (例: qcow2)

フィールド	説明
アーキテクチャー	イメージのアーキテクチャー。たとえば 32 ビットのアーキテクチャーには i686、64 ビットのアーキテクチャーには x86_64 を使用します。
最小ディスク (GB)	イメージのブートに必要な最小のディスクサイズ。このフィールドに値が指定されていない場合には、デフォルト値は 0 です (最小値なし)。
最小メモリー (MB)	イメージのブートに必要な最小のメモリーサイズ。このフィールドに値が指定されていない場合には、デフォルト値は 0 です (最小値なし)。
パブリック	このチェックボックスを選択した場合には、プロジェクトにアクセスできる全ユーザーにイメージが公開されます。
保護	このチェックボックスを選択した場合には、特定のパーミッションのあるユーザーのみがこのイメージを削除できるようになります。

イメージが正常にアップロードされたら、イメージのステータスは、イメージが使用可能であることを示す **active** に変わります。Image サービスは、アップロードの開始時に使用した Identity サービストークンのライフタイムよりもアップロードの所要時間が長くかかる大容量のイメージも処理することができる点に注意してください。これは、アップロードが完了してイメージのステータスが更新される際に、新しいトークンを取得して使用できるように、Identity サービスは最初に Identity サービスとのトラストを作成するためです。

注記

glance image-create コマンドに **property** のオプションを指定して実行する方法でイメージをアップロードすることもできます。コマンドラインで操作を行った方が、より多くの値を使用することができます。完全なリストは、「イメージの設定パラメーター」を参照してください。

1.2.3. イメージの更新

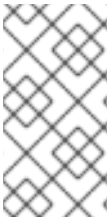
1. Dashboard で **プロジェクト > コンピュート > イメージ**を選択します。
2. ドロップダウンリストから **イメージの編集** をクリックします。

注記

イメージの編集 オプションは、**admin** ユーザーとしてログインした場合にのみ使用することができます。**demo** ユーザーとしてログインした場合には、**インスタンスの起動** または **ボリュームの作成** のオプションを使用することができます。

3. フィールドを更新して、終了したら **イメージの更新** をクリックします。次の値を更新することができます (名前、説明、カーネル ID、RAM ディスク ID、アーキテクチャー、形式、最小ディスク、最小メモリー、パブリック、保護)。
4. ドロップダウンメニューをクリックして **メタデータの更新** オプションを選択します。

5. 左のコラムから右のコラムに項目を追加して、メタデータを指定します。左のコラムには、Image サービスのメタデータカタログからのメタデータの定義が含まれています。**その他** を選択して、任意のキーを使用してメタデータを追加し、完了したら **保存** をクリックします。



注記

glance image-update コマンドに **property** オプションを指定して実行する方法でイメージを更新することもできます。コマンドラインで操作を行った方が、より多くの値を使用することができます。完全なリストは、「イメージの設定パラメーター」を参照してください。

1.2.4. イメージのインポート

URI からイメージをインポートする **web-download** と ローカルファイルシステムからイメージをインポートする **glance-direct** を使用して、Image サービス (glance) にイメージをインポートすることができます。デフォルトでは、両方のオプションが有効です。

インポートの方法は、クラウド管理者によって設定されます。**glance import-info** コマンドを実行して、利用可能なインポートオプションを一覧表示します。

1.2.4.1. リモート URI からのインポート

web-download メソッドを使用して、リモートの URI からイメージをコピーすることができます。

1. イメージを作成して、インポートするイメージの URI を指定します。

```
glance image-create --uri <URI>
```

2. **glance image-show <image-ID>** コマンドで、イメージの可用性をモニタリングすることができます。ここで ID は、イメージの作成中に提供された ID に置き換えてください。

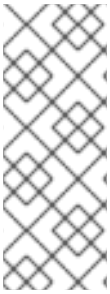
Image サービスの web-download メソッドでは、2 段階のプロセスでインポートを実施します。まず、イメージのレコードを作成します。次に、指定した URI からイメージを取得します。このメソッドは、Image API v1 で使用されていた非推奨となった **copy-from** メソッドよりもセキュアにイメージをインポートする方法です。

『Advanced Overcloud Customization』で説明するように、URI にオプションのブラックリストおよびホワイトリストによる絞り込みを適用することができます。

『オーバークラウドの高度なカスタマイズ』で説明するように、Image Property Injection プラグインにより、イメージにメタデータ属性を注入することができます。注入されたこれらの属性により、イメージインスタンスを起動するコンピュートノードが決定されます。

1.2.4.2. ローカルボリュームからのインポート

glance-direct メソッドは、イメージレコードを作成し、それによりイメージ ID が生成されます。ローカルボリュームからイメージがサービスにアップロードされるとステージングエリアに保管され、設定されているチェックに合格した後にアクティブとなります。高可用性 (HA) 構成で使用される場合には、**glance-direct** メソッドには共通のステージングエリアが必要です。



注記

HA 環境では、**glance-direct** メソッドを使用したイメージのアップロードは、共通のステージエリアがない場合には失敗します。HA の Active/Active 環境では、API コールは複数の glance コントローラーに分散されます。ダウンロード API コールは、イメージをアップロードする API コールとは別のコントローラーに送信することが可能です。ステージングエリアの設定に関する詳しい情報は、『オーバークラウドの **高度なカスタマイズ**』 の「 ストレージの設定 」セクションを参照してください。

glance-direct メソッドは、3 つの異なるコールを使用して、イメージをインポートします。

- **glance image-create**
- **glance image-stage**
- **glance image-import**

glance image-create-via-import コマンドを使用すると、これらの 3 つのコールを 1 つのコマンドで実行することができます。以下の例では、大文字の語句は適切なオプションに置き換える必要があります。

```
glance image-create-via-import --container-format FORMAT --disk-format DISKFORMAT --name
NAME --file /PATH/TO/IMAGE
```

イメージがステージングエリアからバックエンドの場所に移動すると、そのイメージはリストされます。ただし、イメージがアクティブになるには、多少時間がかかる場合があります。

glance image-show <image-ID> コマンドで、イメージの可用性をモニタリングすることができます。ここで ID は、イメージの作成中に提供された ID に置き換えてください。

1.2.5. イメージの削除

1. Dashboard で **プロジェクト > コンピュート > イメージ**を選択します。
2. 削除するイメージを選択し、**イメージの削除** ボタンをクリックします。

1.2.6. イメージの表示または非表示

ユーザーに表示される通常の一覧からパブリックイメージを非表示にすることができます。たとえば、廃止された CentOS 7 イメージを非表示にし、最新バージョンだけを表示してユーザーエクスペリエンスをシンプル化することができます。ユーザーは、非表示のイメージを検出して使用することができます。

イメージを非表示にするには、以下をコマンドを実行します。

```
glance image-update <image-id> --hidden 'true'
```

非表示のイメージを作成するには、**glance image-create** コマンドに **--hidden** 引数を追加します。

イメージの非表示を解除するには、以下のコマンドを実行します。

```
glance image-update <image-id> --hidden 'false'
```

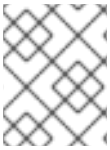
1.2.7. 非表示にしたイメージの表示

非表示にしたイメージを一覧表示するには、以下のコマンドを実行します。

```
glance image-list --hidden 'true'
```

1.2.8. イメージ変換の有効化

GlanceImageImportPlugins パラメーターを有効にすると、QCOW2 イメージをアップロードすることができ、Image サービスはそのイメージを RAW に変換します。



注記

Red Hat Ceph Storage RBD を使用してイメージを保存して Nova インスタンスをブートすると、イメージの変換は自動的に有効になります。

イメージの変換を有効にするには、以下のパラメーター値が含まれる環境ファイルを作成し、**-e** オプションを使用して新しい環境ファイルを **openstack overcloud deploy** コマンドに追加します。

```
parameter_defaults:
  GlanceImageImportPlugins:'image_conversion'
```

1.2.9. RAW 形式へのイメージの変換

Red Hat Ceph Storage は QCOW2 イメージを保管することはできますが、そのイメージを使用して仮想マシン (VM) のディスクをホストすることはできません。

アップロードした QCOW2 イメージから仮想マシンを作成する場合には、コンピュートノードはイメージをダウンロードして RAW に変換し、それを Ceph にアップロードし直してからでないと使用することができません。このプロセスは仮想マシンの作成時間に影響を及ぼします (特に、並行して仮想マシンを作成する場合)。

たとえば、複数の仮想マシンを同時に作成する場合には、Ceph クラスターへの変換済みイメージのアップロードが、すでに実行中の負荷に影響を及ぼす可能性があります。IOPS のこれらの負荷に対するリソースがアップロードプロセスにより枯渇し、ストレージの反応が遅くなる場合があります。

Ceph において仮想マシンをより効率的に起動するには (一時バックエンドまたはボリュームからの起動)、glance イメージの形式を RAW にする必要があります。

イメージを RAW に変換すると、イメージサイズが元の QCOW2 イメージファイルより大きくなる場合があります。最終的な RAW イメージのサイズを確認するには、変換前に以下のコマンドを実行します。

```
qemu-img info <image>.qcow2
```

イメージを QCOW2 から RAW 形式に変換するには、以下のコマンドを実行します。

```
qemu-img convert -p -f qcow2 -O raw <original qcow2 image>.qcow2 <new raw image>.raw
```

1.2.9.1. RAW および ISO だけを受け入れる Image サービスの設定

オプションとして、Image サービスが RAW および ISO イメージ形式だけを受け入れるように設定するには、以下の設定を含む追加の環境ファイルを使用してデプロイを行います。

```
parameter_defaults:
  ExtraConfig:
    glance::config::api_config:
      image_format/disk_formats:
        value: "raw,iso"
```

1.2.10. RAW 形式でのイメージの保存

以前に作成したイメージを RAW 形式で保存するには、**GlanceImageImportPlugins** パラメーターを有効にして以下のコマンドを実行します。

```
$ glance image-create-via-import \
   --disk-format qcow2 \
   --container-format bare \
   --name NAME \
   --visibility public \
   --import-method web-download \
   --uri http://server/image.qcow2
```

- **--name**: **NAME** をイメージ名に置き換えます。この名前が **glance image-list** に表示されます。
- **--uri**: **http://server/image.qcow2** を QCOW2 イメージの場所およびファイル名に置き換えます。

注記

このコマンド例では、イメージレコードを作成し、**web-download** メソッドを使用してそのイメージレコードをインポートします。glance-api は、インポートプロセス中に **--uri** で定義した場所からイメージをダウンロードします。**web-download** が利用できない場合、**glanceclient** はイメージデータを自動的にダウンロードすることができません。利用可能なイメージのインポート方法を一覧表示するには、**glance import-info** コマンドを実行します。

第2章 COMPUTE (NOVA) サービスの設定

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、以下の設定方法を使用します。

- **Heat パラメーター**: 詳細は、**『オーバークラウドのパラメーター』の「Compute(nova) パラメーター」セクションを参照し** てください。以下に例を示します。

```
parameter_defaults:
  NovaSchedulerDefaultFilters:
AggregateInstanceExtraSpecsFilter,RetryFilter,ComputeFilter,ComputeCapabilitiesFilter,Image
PropertiesFilter
  NovaNfsEnabled: true
  NovaNfsShare: '192.0.2.254:/export/nova'
  NovaNfsOptions: 'context=system_u:object_r:nfs_t:s0'
  NovaNfsVersion: '4.2'
```

- **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。

- **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 **disk_allocation_ratio** が定義されます。

```
parameter_defaults:
    ComputeExtraConfig:
        nova::config::nova_config:
            DEFAULT/disk_allocation_ratio:
                value: '2.0'
```

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そのパラメーターを使用する必要があります。Puppet パラメーターは存在するが heat パラメーターが存在しない場合は、手動で上書きする方法ではなく Puppet パラメーターを使用する必要があります。手動で上書きする方法は、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のみ使用してください。

ヒント

特定 の設定をカスタマイズする際に heat パラメーターまたは Puppet パラメーターが利用可能かどうかを判断するには、「変更 するパラメーターの特定」のガイダンスに従ってください。

オーバークラウドサービスの設定に関する詳しい情報は、『オーバークラウドの **高度なカスタマイズ**』の「 パラメーター 」を参照してください。

2.1. 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。詳しくは、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。

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。詳しくは、「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. この設定を適用するには、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、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 \
```

2.2. 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。

- ノード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。

- ホスト上のインスタンスを視覚化するの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) + (resource _n_ * 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** は、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。

2.3. 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。

第3章 OPENSTACK COMPUTE 用のストレージの設定

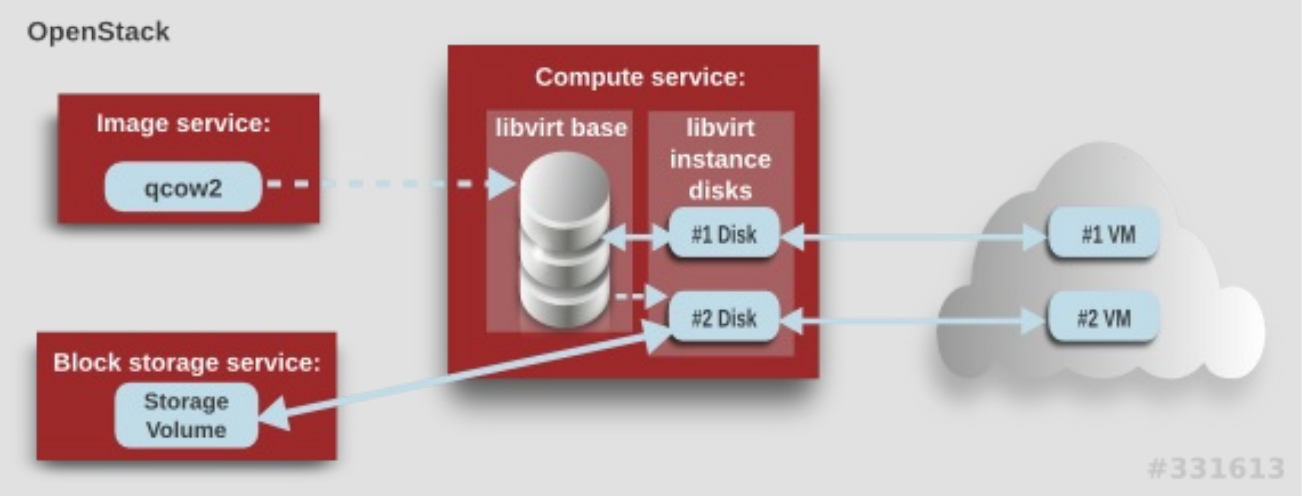
本章では、OpenStack Compute (nova) のイメージのバックエンドストレージのアーキテクチャーについて説明し、基本的な設定オプションを記載します。

3.1. アーキテクチャーの概要

Red Hat OpenStack Platform では、OpenStack Compute サービスは KVM ハイパーバイザーを使用してコンピュートのワークロードを実行します。**libvirt** ドライバーが KVM とのすべての対話を処理し、仮想マシンが作成できるようにします。

コンピュートには、2 種類の **libvirt** ストレージを考慮する必要があります。

- Image サービスのイメージのコピーをキャッシュ済み/フォーマット済みのベースイメージ
- **libvirt** ベースを使用して作成され、仮想マシンインスタンスのバックエンドとなるインスタンスディスク。インスタンスディスクデータは、コンピュートの一時ストレージ (**libvirt** ベースを使用) または永続ストレージ (例: Block Storage を使用) のいずれかに保存されます。



Compute は、以下の手順で仮想マシンインスタンスを作成します。

1. Image サービスのバッキングイメージを **libvirt** ベースとしてキャッシュします。
2. ベースイメージを Raw 形式に変換します (設定されている場合)。
3. 仮想マシンのフレーバーの仕様に一致するようにベースイメージのサイズを調節します。
4. ベースイメージを使用して libvirt インスタンスディスクを作成します。

上図では、#1 のインスタンスディスクは一時ストレージを使用し、#2 のディスクは Block Storage ボリュームを使用します。

一時ストレージとは、インスタンスで追加で利用可能な、フォーマットされていない空のディスクのことです。このストレージの値は、インスタンスのフレーバーにより定義されます。ユーザーが指定した値は、フレーバーで定義した一時ストレージの値以下でなければなりません。デフォルト値は **0** です。0 を指定すると、一時ストレージが作成されません。

一時ディスクは、外付けのハードドライブや USB ドライブと同じ方法で表示されます。一時ディスクはブロックデバイスとして利用でき、**lsblk** コマンドを使用して確認することができます。ブロックデバイスとして通常使用するように、フォーマット、マウント、使用が可能です。アタッチ先のインスタンス以外では、このディスクの保存や参照をする方法はありません。

ブロックストレージボリュームは、実行中のインスタンスがどのような状態であっても、インスタンスを利用できる一時ストレージです。

3.2. 設定

Compute (nova) の設定ファイルをカスタマイズして、仮想ディスクのパフォーマンスチューニングおよびセキュリティーを設定することができます。Compute は、『オーバークラウドのパラメーター』の「Compute(nova) パラメーター」セクションで説明するパラメーターを使用して、カスタム環境ファイルおよび heat テンプレートで設定します。生成された設定は、**/var/lib/config-data/puppet-generated/<nova_container>/etc/nova/nova.conf** ファイルに保存されます。詳細を以下の表に示します。

表3.1 Compute のイメージのパラメーター

セクション	パラメーター	説明	デフォルト
[DEFAULT]	**force_raw_images**	**non-raw** でキャッシュしたベースイメージを **raw** (ブール型) に変換するかどうかを設定します。non-raw イメージが Raw に変換された場合には、コンピュートは以下の操作を行います。 ● セキュリティー上問題がある可能性があるバッキングファイルを無効にします。 ● 既存の圧縮を削除して、CPU のボトルネックを回避します。 ベースを Raw に変換すると、ハイパーバイザーが直接使用可能なイメージの容量よりも、容量が多く使用されます (例: qcow2 イメージ)。I/O が遅いシステムや空き容量が少ないシステムを使用している場合は、**false** に指定して、圧縮の際のCPU 要件を軽減することで入力の帯域幅を最小限に抑えます。 Raw ベースイメージは常に **libvirt_images_type=lvm** と合わせて使用されます。	**true**
[DEFAULT]	**use_cow_images**	**libvirt** インスタンスディスク (ブール型) に CoW (Copy of Write) イメージを使用するかどうかを設定します。 ● **false**: Raw 形式が使用されます。CoW を使用しない場合には、ディスクイメージの共通の部分により多くの容量が使用されます。 ● **true**: cqow2 形式が使用されます。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、各仮想マシン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可能性があります。	true

セクション	パラメーター	説明	デフォルト
[DEFAULT]	**preallocate_images**	**libvirt** インスタンスディスクに対する事前割り当てモード。値は以下のとおりです。 ● **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**: インスタンスの開始時には、(**fallocate** を使用して) ストレージが完全に割り当てられます。領域および I/O パフォーマンスの両方を確保しやすくします。 CoW インスタンスディスクを使用しない場合でも、各仮想マシンが取得するコピーはスパースであるため、仮想マシンが ENOSPC でランタイムに予期せず失敗する可能性があります。インスタンスディスクに **fallocate(1)** を実行すると、コンピュートは即時に、ファイルシステム内でイメージに領域を効率的に割り当てます (サポートされている場合)。ファイルシステムではランタイム時にブロックを動的に割り当てる必要がないため、ランタイムのパフォーマンスも向上されるはずです (CPU オーバーヘッドの削減、より重要な点としてファイルの断片化の軽減)。	none
[DEFAULT]	**resize_fs_using_block_device**	ブロックデバイス (ブール型) 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調節することができるかどうかを設定します。これは、(それ自体ではサイズ調節ができないため) **cloud-init** のバージョンがより古いイメージでのみ必要です。 このパラメーターにより、セキュリティーの関係上、無効にされる可能性のあるイメージを直接マウントできるため、デフォルトでは有効化されていません。	**false**
[DEFAULT]	**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。値は、**ext2**、**ext3**、**ext4** のいずれかを使用できます。**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。また、**guest_format** の設定オプションを使用することで、インスタンスごとの設定を優先させることも可能です。	**ext4**

セクション	パラメーター	説明	デフォルト
[DEFAULT]	**image_cache_manager_interval**	libvirt コンピュートノードにキャッシュするベースに影響を与える、イメージキャッシュマネージャーを次に実行するまでの待機時間 (秒数)。この時間は、未使用のキャッシュされたイメージを自動削除する際にも使用されます (**remove_unused_base_images** と **remove_unused_original_minimum_age_seconds** 参照)。	**2400**
[DEFAULT]	**remove_unused_base_images**	未使用のベースイメージを自動的に削除できるようにするかどうかを設定します (**image_cache_manager_interval** の秒の間隔でチェック)。イメージは、**remove_unused_original_minimum_age_seconds** (秒) の期間アクセスされなかった場合に **unused** と定義されます。	**true**
[DEFAULT]	**remove_unused_original_minimum_age_seconds**	未使用となったベースイメージが **libvirt** キャッシュから削除されるまでの期間を設定します (**remove_unused_base_images** 参照)。	**86400**
[**libvirt**]	**images_type**	**libvirt** インスタンスディスクに使用するイメージ種別 (**use_cow_images** は廃止予定)。使用可能な値は、**raw**、**qcow2**、**lvm**、**rbd**、**default** のいずれかです。**default** が指定されている場合は、**use_cow_images** パラメーターに使用された値が使用されます。	**default**

第4章 仮想マシンインスタンス

OpenStack Compute は、仮想マシンをオンデマンドで提供する中核的なコンポーネントです。Compute は、認証には Identity サービス、イメージ (インスタンスの起動に使用する) には Image サービス、ユーザー/管理者用のインターフェースには Dashboard サービスと対話します。

Red Hat OpenStack Platform により、クラウド内の仮想マシンインスタンスを容易に管理することができます。Compute サービスはインスタンスの作成、スケジューリング、管理を行い、この機能をその他の OpenStackコンポーネントに公開します。本章では、これらの手順に加えて、キーペア、セキュリティーグループ、ホストアグリゲート、フレーバーなどのコンポーネントを追加する手順について説明します。OpenStack では、**インスタンス** という用語は、仮想マシンインスタンスの意味で使用されます。

4.1. インスタンスの管理

インスタンスを作成する前には、その他の特定の OpenStack コンポーネント (例: ネットワーク、キーペア、イメージ、ブートソースとなるボリュームなど) をそのインスタンスが利用できる状態にしておく必要があります。

本項では、これらのコンポーネントを追加して、インスタンスを作成/管理する手順について説明します。インスタンスの管理には、更新、ログイン、使用状況の確認、リサイズ、削除などの操作が含まれます。

4.1.1. コンポーネントの追加

以下の各項の手順に従って、ネットワーク、キーペアを作成し、イメージまたはボリュームソースをアップロードします。これらのコンポーネントは、インスタンスの作成に使用され、デフォルトでは提供されません。また、新規セキュリティーグループ作成して、ユーザーが SSH アクセスできるようにする必要があります。

1. Dashboard で **プロジェクト** を選択します。
2. ネットワーク & **gt**; ネットワーク を選択し、新規インスタンスを接続することができるプライベートネットワークが存在していることを確認してください（ネットワークの作成方法については、『ネットワーク **ガイド**』の「ネットワークの 作成 」セクションを参照してください）。
3. **コンピュート > アクセスとセキュリティー > キーペア**を選択して、キーペアが存在していることを確認します (キーペアの作成方法については、「キーペアの作成」を参照)。
4. ブートソースに使用可能なイメージまたはボリュームのいずれかがあることを確認してください。
   - ブートソースのイメージを表示するには、**イメージ** タブを選択します (イメージを作成する場合は「イメージの作成」を参照してください)。
   - ブートソースのボリュームを表示するには、ボリューム タブを選択します（ボリュームを作成する場合は **『ストレージガイド』**の「ボリューム の作成 」を参照してください）。
5. **コンピュート > アクセスとセキュリティー >セキュリティーグループ を選択し、セキュリティーグループルールが作成済みであることを確認します（セキュリティーグループの作成については、『ユーザーおよびアイデンティティー管理ガイド』の「 プロジェクトのセキュリティー管理 」を参照してください）。**

4.1.2. インスタンスの起動

Dashboard から 1つまたは複数のインスタンスを起動します。

注記

デフォルトでは、インスタンスの起動にはインスタンスの起動フォームが使用されます。ただし、必要なステップを簡単に実行することのできるインスタンスの起動ウィザードを有効にすることも可能です。詳細は、「付録B *インスタンスの起動ウィザードの有効化*」を参照してください。

1. Dashboard で プロジェクト > コンピュート > インスタンスを選択します。
2. インスタンスの起動 をクリックします。
3. 各フィールド (「*」の付いたフィールドは必須です) に設定値を入力し、起動 をクリックします。

指定したオプションに基づいて、1つまたは複数のインスタンスが作成され、起動されます。

4.1.2.1. インスタンスの起動オプション

以下の表には、インスタンスの起動フォームを使用して新規インスタンスを起動する際に利用可能なオプションをまとめています。インスタンスの起動ウィザードでも同じオプションが利用できます。

表4.1インスタンスの起動フォームのオプション

タブ	フィールド	説明
プロジェクトおよびユーザー	プロジェクト	ドロップダウンリストからプロジェクトを選択します。
	ユーザー	ドロップダウンリストからユーザーを選択します。
詳細	アベイラビリティーゾーン	ゾーンとは、インスタンスが配置されるクラウドリソースの論理グループです。不明な場合にはデフォルトのゾーンを使用してください (詳しくは「ホストアグリゲートの管理」を参照)。
	インスタンス名	インスタンスを識別するための名前
	フレーバー	フレーバーは、インスタンスに提供されるリソースを決定します (例: メモリー)。デフォルトのフレーバーの割り当ておよび新規フレーバー作成に関する情報は、「フレーバーの管理」を参照してください。
	インスタンス数	ここに記載のパラメーターで作成するインスタンスの数。「1」が事前に設定されています。

タブ	フィールド	説明
	インスタンスのブートソース	選択した項目に応じて異なる新規フィールドが表示され、ソースを選択することができます。 ● イメージソースは OpenStack との互換性がある必要があります (「イメージの管理」を参照)。 ● ボリュームまたはボリュームソースを選択した場合、そのソースはイメージを使用してフォーマットする必要があります（『**ストレージガイド**』の「ボ リュームの基本的な使用方法と設定 」を参照）。
アクセスとセキュリティー	キーペア	指定したキーペアがインスタンスに挿入され、SSH を使用したインスタンスへのリモートアクセスに使用されます (直接のログイン情報や静的キーペアが提供されない場合)。通常は 1 プロジェクトあたり 1 つのキーペアが作成されます。
	セキュリティーグループ	セキュリティーグループには、インスタンスのネットワークトラフィックの種別と方向をフィルタリングするファイアウォールルールが含まれています（グループの設定についての詳しい説明は、『**ユーザーおよびアイデンティティー 管理ガイド』の「プロジェクトのセキュリティー 管理**」を参照してください）。
ネットワーク	選択済みネットワーク	ネットワークは、少なくとも 1 つ選択する必要があります。インスタンスは通常プライベートネットワークに割り当てられ、その後に Floating IP アドレスが割り当てられて外部アクセスが可能になります。
作成後	カスタマイズスクリプトの入力方法	インスタンスのブート後に実行されるコマンドセットまたはスクリプトファイルを指定することができます (例: インスタンスのホスト名やユーザーパスワードの設定など)。**直接入力** を選択した場合には、スクリプトデータフィールドにコマンドを書き込みます。それ以外の場合には、スクリプトファイルを指定してください。 **注記** **#cloud-config** で開始するスクリプトは、cloud-config 構文を使用するものとして解釈されます (この構文についての情報は、http://cloudinit.readthedocs.org/en/latest/topics/examples.html を参照してください)。
高度な設定	ディスクパーティション	デフォルトでは、インスタンスは単一のパーティションとして作成されて、必要に応じて動的にリサイズされます。ただし、パーティションを手動で設定する方法を選択することも可能です。

タブ	フィールド	説明
	コンフィグドライブ	このオプションを選択した場合には、OpenStack はメタデータを読み取り専用の設定ドライブに書き込みます。このドライブはインスタンスのブート時に (Compute のメタデータサービスの代わりに) インスタンスに接続されます。インスタンスがブートした後には、このドライブをマウントしてコンテンツを表示することができます (これにより、ユーザーがファイルをインスタンスに提供することが可能となります)。

4.1.3. インスタンスの更新 (アクションメニュー)

インスタンスを更新するには、プロジェクト > コンピュート > インスタンスを選択してから、そのインスタンスに対して実行するアクションを アクション コラムで選択します。これらのアクションにより、数多くの方法でインスタンスを操作することができます。

表4.2 インスタンスの更新オプション

アクション	説明
スナップショットの作成	スナップショットは、実行中のインスタンスのディスクの状態を保存します。スナップショットは、インスタンスの移行やバックアップコピーの保存などの目的で作成することができます。
Floating IP の割り当て/割り当て解除	外部のネットワークとの通信および外部ユーザーによるアクセスを可能にするには、インスタンスを Floating IP アドレス (外部) に割り当てる必要があります。外部サブネットには、外部アドレスの数が限定されているため、使用していないアドレスは割り当て解除することを推奨します。
インスタンスの編集	インスタンスの名前を更新して、セキュリティーグループを割り当てます。
セキュリティーグループの編集	利用可能なセキュリティーグループの一覧を使用して、インスタンスにセキュリティーグループを追加/削除します（グループの設定についての詳しい説明は、『**ユーザーおよびアイデンティティー管理ガイド**』の「プロジェクトのセキュリティー管理」を参照してください）。
コンソール	ブラウザーでインスタンスのコンソールを表示します (インスタンスに容易にアクセスすることができます)。
ログの参照	インスタンスのコンソールログの最新のセクションを表示します。このログを開いた後に「すべてのログの表示」をクリックすると、ログ全体を参照することができます。

アクション	説明
インスタンスの一時停止/再開	インスタンスを即時に一時停止します (操作を確認するメッセージは表示されません)。インスタンスの状態はメモリー (RAM) に保存されます。
インスタンスの休止/再開	インスタンスを即時に休止します (操作を確認するメッセージは表示されません)。ハイバネートと同様に、インスタンスの状態はディスクに保存されます。
インスタンスのリサイズ	インスタンスのリサイズのウィンドウが表示されます (「インスタンスのリサイズ」を参照)。
ソフトリブート	インスタンスを正常に停止して再起動します。ソフトリブートは、全プロセスを正常にシャットダウンしてから、インスタンスを再起動するように試みます。
ハードリブート	インスタンスを停止して再起動します。ハードリブートは、実質的にはインスタンスの **電源** をオフにしてから再びオンにします。
インスタンスのシャットダウン	インスタンスを正常に停止します。
インスタンスの再ビルド	イメージおよびディスクパーティションのオプションを使用してイメージを再ビルドします (インスタンスをシャットダウンして、イメージを再作成して、再起動します)。このオプションは、オペレーティングシステムに問題が発生した場合に、インスタンスを終了して一からやり直すよりも簡単です。
インスタンスの終了	インスタンスを完全に破棄します (操作を確認するメッセージが表示されます)。

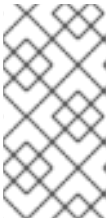
外部の IP アドレスを作成して確保することができます。「Floating IP アドレスの作成、割り当て、および解放」を参照してください。

4.1.4. インスタンスのリサイズ

インスタンスのリサイズ (メモリーまたは CPU 数) を行うには、適切な容量のあるインスタンスで新規フレーバーを選択する必要があります。サイズを大きくする場合には、ホストに十分な容量があることをあらかじめ確認することを忘れないようにしてください。

1. **各ホストに SSH 鍵認証を設定してホスト間の通信が可能な状態にし、Compute が SSH を使用してディスクを他のホストに移動できるようにします (例: 複数のコンピュートノードが同じ SSH 鍵を共有することが可能です)。**

2. **元のホストでリサイズを有効にするには、Compute 環境ファイルの allow_resize_to_same_host パラメーターを「True」に設定します。**



注記

allow_resize_to_same_host パラメーターによって、同じホスト上のインスタンスはリサイズされません。このパラメーターが全コンピュートノードで「True」に指定されている場合でも、スケジューラーは同じホスト上のインスタンスのリサイズを強制しません。これは、想定されている動作です。

3. **Dashboard で プロジェクト > コンピュート > インスタンスを選択します。**
4. **インスタンスの アクション コラムの矢印をクリックして、インスタンスのリサイズ を選択します。**
5. **新しいフレーバー フィールドで新規フレーバーを選択します。**
6. **起動時にインスタンスのパーティション分割を手動で行うには、以下の手順で設定します (これにより、ビルドタイムが短縮されます)。**
   a. **高度な設定 を選択します。**
   b. **ディスクパーティション フィールドで、手動 を選択します。**
7. **リサイズ をクリックします。**

4.1.5. インスタンスへの接続

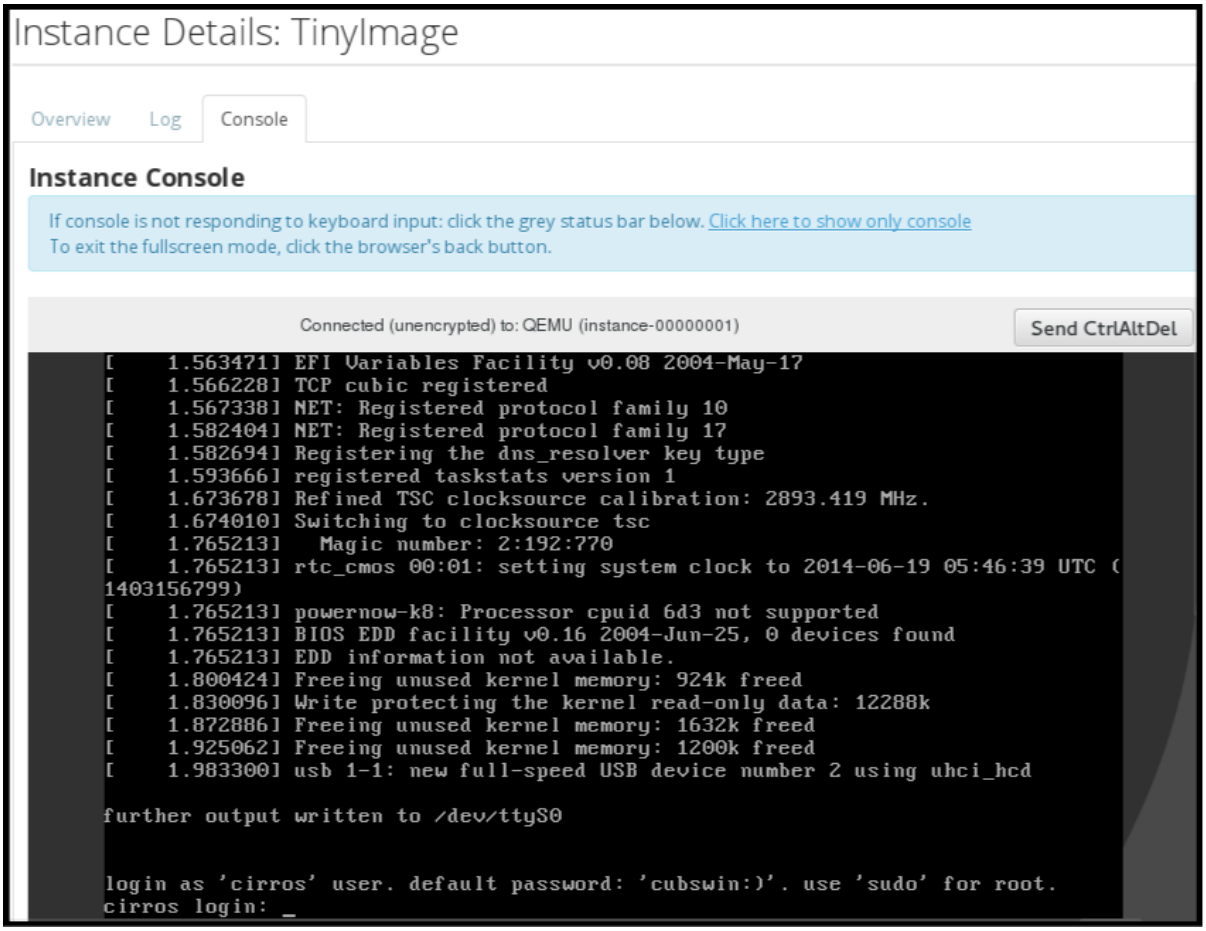
本項では、Dashboard またはコマンドラインインターフェースを使用して、インスタンスのコンソールにアクセスする複数の方法について説明します。また、インスタンスのシリアルポートに直接接続して、ネットワーク接続が失敗しても、デバッグすることが可能です。

4.1.5.1. Dashboard を使用したインスタンスのコンソールへのアクセス

コンソールを使用すると、Dashboard 内でインスタンスに直接アクセスすることができます。

1. **Dashboard で コンピュート > インスタンス を選択します。**

2. **インスタンスの ドロップダウンメニュー をクリックして、コンソール を選択します。**



3. **イメージのユーザー名とパスワードを使用してログインします (例: CirrOS イメージでは「cirros」と「cubswin:)」を使用します)。**

4.1.5.2. VNC コンソールへの直接接続

nova get-vnc-console コマンドで返された URL を使用すると、インスタンスの VNC コンソールに直接アクセスすることができます。

ブラウザー

ブラウザーの URL を取得するには、以下のコマンドを実行します。

```
$ nova get-vnc-console INSTANCE_ID novnc
```

Java クライアント

Java クライアントの URL を取得するには、以下のコマンドを実行します。

```
$ nova get-vnc-console INSTANCE_ID xvpvnc
```

注記

「nova-xvpvncviewer」は、Java クライアントの最も簡単な例を提供します。クライアントをダウンロードするには、以下のコマンドを実行します。

```
# git clone https://github.com/cloudbuilders/nova-xvpvncviewer
# cd nova-xvpvncviewer/viewer
# make
```

インスタンスの Java クライアント URL を使用してビューアーを実行します。

```
# java -jar VncViewer.jar URL
```

このツールは、お客様の便宜のためのみに提供されており、Red Hat では正式にサポートされていません。

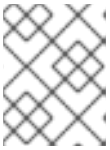
4.1.6. インスタンスの使用状況の表示

以下のような使用状況統計が提供されます。

- プロジェクト別
  プロジェクト別の使用状況を確認するには、プロジェクト > コンピュート > 概要を選択します。全プロジェクトインスタンスの使用状況の概要が即時に表示されます。

  使用状況を照会する期間を指定して 送信 ボタンをクリックすると、特定の期間の統計を表示することもできます。

- ハイパーバイザー別
  管理者としてログインしている場合には、全プロジェクトの情報を表示することができます。管理 > システムをクリックしてからタブを 1つ選択します。たとえば、リソース使用状況 タブでは、特定の期間のレポートを確認することができます。また、ハイパーバイザー をクリックすると、現在の仮想 CPU、メモリー、ディスクの統計を確認することができます。



注記

仮想 CPU 使用量 の値 (y 中 x 使用中) には、全仮想マシンの仮想 CPU の合計数 (x) とハイパーバイザーのコアの合計数 (y) が反映されます。

4.1.7. インスタンスの削除

1. Dashboard で プロジェクト > コンピュート > インスタンスを選択して、対象のインスタンスにチェックを付けます。
2. インスタンスの削除 をクリックします。

注記

インスタンスを削除しても、接続されていたボリュームは削除されません。この操作は別途実行する必要があります（『ストレージガイド』の「ボリュームの 削除 」を参照）。

4.1.8. 複数インスタンスの同時管理

同時に複数のインスタンスを起動する必要がある場合には (例: コンピュートまたはコントローラーのメンテナンスでダウンしている場合など)、プロジェクト > コンピュート > インスタンスから簡単に起動できます。

1. 起動するインスタンスの最初のコラムにあるチェックボックスをクリックします。全インスタンスを選択するには、表の最初の行のチェックボックスをクリックします。
2. 表の上にある その他のアクション をクリックして インスタンスの起動 を選択します。

同様に、適切なアクションを選択して、複数のインスタンスを終了またはソフトリブートすることができます。

4.2. インスタンスのセキュリティーの管理

適切なセキュリティーグループ (ファイアウォールのルールセット) およびキーペア (SSH を介したユーザーのアクセスの有効化) を割り当てることによってインスタンスへのアクセスを管理することができます。また、インスタンスに Floating IP アドレスを割り当てて外部ネットワークへのアクセスを有効にすることができます。以下の各項では、キーペア、セキュリティーグループ、Floating IP アドレスの作成/管理方法と SSH を使用したログインの方法について説明します。また、インスタンスに admin パスワードを挿入する手順についても記載しています。

セキュリティーグループの管理に関する情報は、『ユーザーおよびアイデンティティー 管理ガイド』の「プロジェクト のセキュリティー管理」を 参照してください。

4.2.1. キーペアの管理

キーペアにより、インスタンスへ SSH でアクセスすることができます。キーペアの生成時には毎回、証明書がローカルマシンにダウンロードされ、ユーザーに配布できます。通常は、プロジェクトごとにキーペアが 1 つ作成されます (そのキーペアは、複数のインスタンスに使用されます)。

既存のキーペアを OpenStack にインポートすることも可能です。

4.2.1.1. キーペアの作成

1. Dashboard で プロジェクト > コンピュート > アクセスとセキュリティーを選択します。
2. キーペア タブで キーペアの作成 をクリックします。
3. キーペア名 フィールドに名前を指定し、キーペアの作成 をクリックします。

キーペアが作成されると、ブラウザーを介してキーペアファイルが自動的にダウンロードされます。後ほど外部のマシンから接続できるように、このファイルを保存します。また、コマンドラインの SSH 接続には、以下のコマンドを実行して、このファイルを SSH にロードすることができます。

```
# ssh-add ~/.ssh/os-key.pem
```

4.2.1.2. キーペアのインポート

1. Dashboard で プロジェクト > コンピュート > アクセスとセキュリティーを選択します。
2. キーペア タブで キーペアのインポート をクリックします。
3. キーペア名 のフィールドに名前を指定し、公開鍵の内容をコピーして、公開鍵 のフィールドにペーストします。

4. キーペアのインポート をクリックします。

4.2.1.3. キーペアの削除

1. Dashboard で プロジェクト > コンピュート > アクセスとセキュリティーを選択します。
2. キーペア タブでそのキーの キーペアの削除 ボタンをクリックします。

4.2.2. セキュリティーグループの作成

セキュリティーグループとは、プロジェクトのインスタンスに割り当て可能な IP フィルターのルールセットで、インスタンスへのネットワークのアクセス権限を定義します。セキュリティーグループはプロジェクト別になっており、プロジェクトメンバーは自分のセキュリティーグループのデフォルトルールを編集して新規ルールセットを追加することができます。

1. Dashboard で プロジェクト タブを選択して、コンピュート > アクセスとセキュリティーをクリックします。
2. セキュリティーグループ タブで、+ セキュリティーグループの作成 をクリックします。
3. セキュリティーグループに名前と説明を指定して、セキュリティーグループの作成 をクリックします。

プロジェクトセキュリティーの管理に関する情報は、『ユーザーおよびアイデンティティー 管理 ガイド』の「プロジェクトのセキュリティー管理」を参照 してください。

4.2.3. Floating IP アドレスの作成、割り当て、および解放

デフォルトでは、インスタンスを最初に作成する際に、そのインスタンスに内部 IP アドレスが割り当てられます。ただし、Floating IP アドレス (外部アドレス) を作成して割り当てることによりパブリックネットワークを介したアクセスを有効にすることができます。インスタンスに割り当てられている IP アドレスは、インスタンスの状態に関わらず変更することができます。

プロジェクトには、使用できる Floating IP アドレスの範囲が限定されているので (デフォルトの上限は 50)、必要がなくなったアドレスは、再利用できるように解放することを推奨します。Floating IP アドレスは、既存の Floating IP プールからのみ確保することができます。『ネットワークガイド』 の「Floating IP アドレスプールの作成 」を参照してください。

4.2.3.1. プロジェクトへの Floating IP アドレスの確保

1. Dashboard で プロジェクト > コンピュート > アクセスとセキュリティーを選択します。
2. Floating IP タブで Floating IP の確保 をクリックします。
3. プール のフィールドで、IP アドレスを確保するネットワークを選択します。
4. IP の確保 をクリックします。

4.2.3.2. Floating IP の割り当て

1. Dashboard で プロジェクト > コンピュート > アクセスとセキュリティーを選択します。
2. Floating IP タブでアドレスの割り当て ボタンをクリックします。
3. IP アドレスフィールドで割り当てるアドレスを選択します。

注記

割り当てることのできるアドレスがない場合には、+ボタンをクリックして新規アドレスを作成することができます。

4. IP を割り当てるポート フィールドで割り当て先となるインスタンスを選択します。1つのインスタンスに割り当てることができる Floating IP アドレスは1つのみです。

5. 割り当て をクリックします。

4.2.3.3. Floating IP の解放

1. Dashboard で プロジェクト > コンピュート > アクセスとセキュリティーを選択します。

2. Floating IP タブで、アドレスの割り当て/割り当て解除 ボタンの横にある矢印メニューをクリックします。

3. Floating IP の解放 を選択します。

4.2.4. インスタンスへのログイン

前提条件

- インスタンスのセキュリティーグループには SSH ルールが設定されているようにしてください（『ユーザーおよびアイデンティティー 管理ガイド』の「プロジェクト のセキュリティー管理」を参照してください）。
- インスタンスに Floating IP アドレス (外部アドレス) が割り当てられていることを確認します (「Floating IP アドレスの作成、割り当て、および解放」を参照)。
- インスタンスのキーペアの証明書を取得します。証明書は、キーペアの作成時にダウンロードされます。キーペアを自分で作成しなかった場合には、管理者に問い合わせてください (「キーペアの管理」を参照)。

最初に、キーペアのファイルを SSH に読み込み、次に名前を指定せずに ssh を使用します。

1. 生成したキーペアの証明書のアクセス権を変更します。

```
$ chmod 600 os-key.pem
```

2. **ssh-agent** がすでに実行されているかどうかを確認します。

```
# ps -ef | grep ssh-agent
```

3. 実行されていない場合には、次のコマンドで起動します。

```
# eval `ssh-agent`
```

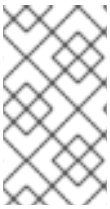
4. ローカルマシンで、キーペアの証明書を SSH に読み込みます。以下に例を示します。

```
$ ssh-add ~/.ssh/os-key.pem
```

5. これで、イメージにより提供されるユーザーで、ファイルに SSH アクセスできるようになりました。

以下のコマンドの例は、Red Hat Enterprise Linux のゲストイメージに **cloud-user** として SSH アクセスする方法を示しています。

```
$ ssh cloud-user@192.0.2.24
```



注記

証明書を直接使用することも可能です。以下に例を示します。

```
$ ssh -i /myDir/os-key.pem cloud-user@192.0.2.24
```

4.2.5. インスタンスへの admin パスワード挿入

以下の手順に従って、**admin (root)** パスワードを挿入することができます。

1. **/etc/openstack-dashboard/local_settings** ファイルで、**change_set_password** パラメーターの値を **True** に設定します。

   ```
   can_set_password: True
   ```

2. Compute 環境ファイルの **inject_password** パラメーターを「True」に設定します。

   ```
   inject_password=true
   ```

3. Compute サービスを再起動します。

   ```
   # service nova-compute restart
   ```

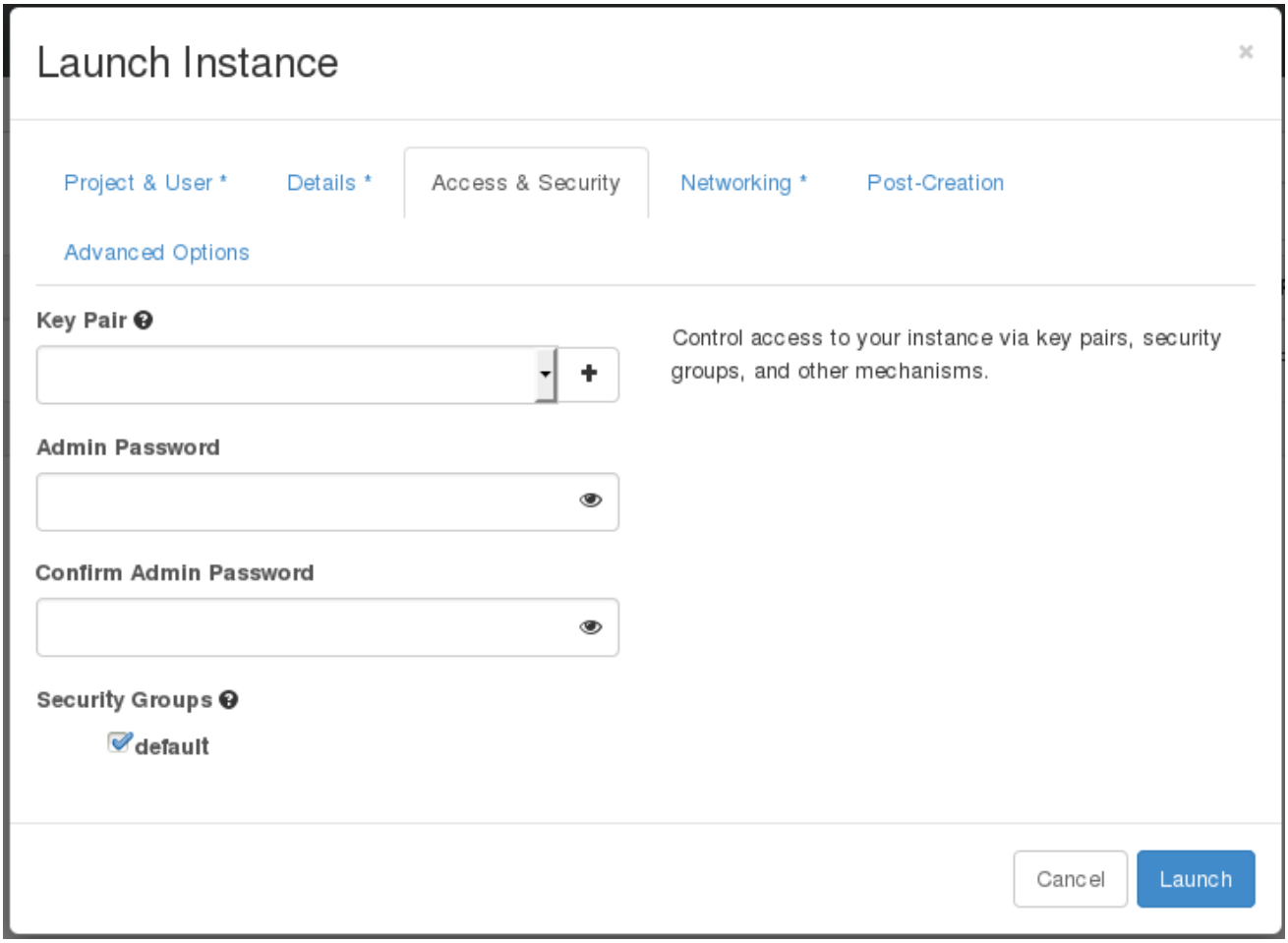
nova boot コマンドを使用して、新規インスタンスを起動する際には、コマンドの出力に **adminPass** パラメーターが表示されます。このパスワードを使用して、インスタンスに **root** ユーザーとしてログインすることができます。

Compute サービスは、**/etc/shadow** ファイル内のパスワード値を **root** ユーザー用に上書きします。以下の手順は、KVM ゲストイメージの **root** アカウントをアクティブ化するのにも使用することが可能です。KVM ゲストイメージの使用方法についての詳しい情報は、「Red Hat OpenStack Platform における KVM ゲストイメージの使用」を参照してください。

Dashboard からカスタムパスワードを設定することも可能です。これを有効にするには、**can_set_password** パラメーターを **true** に設定した後に、以下のコマンドを実行します。

```
# systemctl restart httpd.service
```

新規追加された **admin** パスワードフィールドは以下のように表示されます。



上記のフィールドは、インスタンスの起動/再ビルド時に使用することができます。

4.3. フレーバーの管理

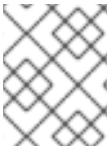
作成する各インスタンスには、インスタンスのサイズや容量を決定するためのフレーバー (リソースのテンプレート) を指定します。また、フレーバーを使用して、セカンダリー一時ストレージやスワップディスク、使用率を制限するためのメタデータ、特別なプロジェクトへのアクセスを指定することも可能です (デフォルトのフレーバーにはこのような追加の属性は一切定義されていません)。

表4.3 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.tiny	1	512 MB	1 GB
m1.small	1	2048 MB	20 GB
m1.medium	2	4096 MB	40 GB
m1.large	4	8192 MB	80 GB
m1.xlarge	8	16384 MB	160 GB

エンドユーザーの大半は、デフォルトのフレーバーを使用できます。ただし、特化したフレーバーを作成/管理することができます。たとえば、以下の操作を行うことができます。

- 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



注記

イメージのプロパティーを使用して設定した動作は、フレーバーを使用して設定した動作よりも優先されます。詳しい説明は、「イメージの管理」を参照してください。

4.3.1. 設定パーミッションの更新

デフォルトでは、フレーバーの作成およびフレーバーの完全リストの表示ができるのは管理者のみです (「管理 > システム > フレーバー」を選択)。全ユーザーがフレーバーを設定できるようにするには、/etc/nova/policy.json ファイル (nova-api サーバー) で以下の値を指定します。

"compute_extension:flavormanage": "",

4.3.2. フレーバーの作成

1. Dashboard に管理ユーザーとしてログインして 管理 > システム > フレーバーを選択します。
2. フレーバーの作成 をクリックして、以下のフィールドに入力します。

表4.4 フレーバーのオプション

タブ	フィールド	説明
フレーバー情報	名前	一意な名前
	ID	一意な ID。デフォルト値は **auto** で、UUID4 値を生成しますが、整数または UUID4 値 を手動で指定することもできます。
	仮想 CPU	仮想 CPU 数
	メモリー (MB)	メモリー (メガバイト単位)
	ルートディスク (GB)	一時ディスクのサイズ (ギガバイト単位)。ネイティブイメージサイズを使用するには **0** を指定します。このディスクは、Instance Boot Source=Boot from Volume と指定されている場合には使用されません。

タブ	フィールド	説明
	一時ディスク (GB)	インスタンスで利用可能なセカンダリー一時ディスクのサイズ (ギガバイト単位)。このディスクは、インスタンスの削除時に破棄されます。 デフォルト値は **0** です。この値を指定すると、一時ディスクは作成されません。
	スワップディスク (MB)	スワップディスクのサイズ (メガバイト単位)
フレーバーアクセス権	選択済みのプロジェクト	そのフレーバーを使用することができるプロジェクト。プロジェクトが選択されていない場合には、全プロジェクトにアクセスが提供されます (**Public=Yes**)。

3. **フレーバーの作成** をクリックします。

4.3.3. 一般属性の更新

1. Dashboard に管理ユーザーとしてログインして **管理** > **システム** > **フレーバー**を選択します。
2. **対象のフレーバーの フレーバーの編集** ボタンをクリックします。
3. **値を更新して、保存** をクリックします。

4.3.4. フレーバーのメタデータの更新

一般属性の編集に加えて、フレーバーにメタデータ (extra_specs) を追加することが可能です。メタデータは、インスタンスの使用方法を微調整するのに役立ちます。たとえば、最大許容帯域幅やディスクの書き込みを設定する場合などです。

- **事前定義済みのキーにより、ハードウェアサポートやクォータが決定されます。事前定義済みのキーは、使用するハイパーバイザーによって限定されます (libvirt の場合は 表4.5「Libvirt のメタデータ」を参照してください)。**
- **事前定義済みおよびユーザー定義のキーはいずれも、インスタンスのスケジューリングを決定します。たとえば、SpecialComp=True と指定すると、このフレーバーを使用するインスタンスはすべてメタデータのキーと値の組み合わせが同じホストアグリゲートでのみ実行可能となります (「ホストアグリゲートの管理」を参照)。**

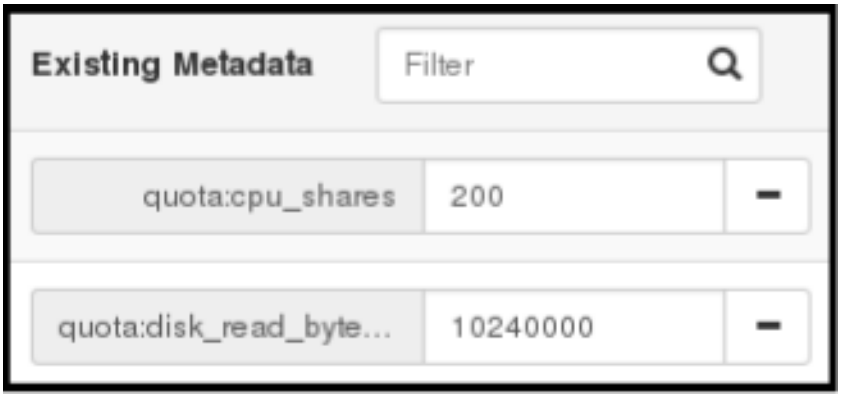
4.3.4.1. メタデータの表示

1. Dashboard に管理ユーザーとしてログインして **管理** > **システム** > **フレーバー**を選択します。
2. **フレーバーの メタデータ リンク (はい または いいえ) をクリックします。現在の値はすべて右側の 選択済みのメタデータ の下に一覧表示されます。**

4.3.4.2. メタデータの追加

キーと値 のペアを使用してフレーバーのメタデータを指定します。

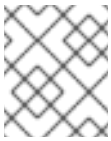
1. Dashboard に管理ユーザーとしてログインして 管理 > システム > フレーバーを選択します。
2. フレーバーの メタデータ リンク (はい または いいえ) をクリックします。現在の値はすべて右側の 選択済みのメタデータ の下に一覧表示されます。
3. 利用可能なメタデータ で その他 のフィールドをクリックして、追加するキーを指定します (表4.5「Libvirt のメタデータ」を参照)。
4. 「+」ボタンをクリックします。選択済みのメタデータ の下に新しいキーが表示されるようになりました。
5. 右側のフィールドにキーの値を入力します。



6. キーと値のペアの追加が終了したら 保存 をクリックします。

表4.5 Libvirt のメタデータ

キー	説明
hw:action	インスタンスごとにサポート制限を設定するアクション。有効なアクションは以下のとおりです。 • **cpu_max_sockets**: サポートされている最大の CPU ソケット数 • **cpu_max_cores**: サポートされている最大の CPU コア数 • **cpu_max_threads**: サポートされている最大の CPU スレッド数 • **cpu_sockets**: 推奨される CPU ソケット数 • **cpu_cores**: 推奨される CPU コア数 • **cpu_threads**: 推奨される CPU スレッド数 • **serial_port_count**: 1 インスタンスあたりの最大シリアルポート数 例: **hw:cpu_max_sockets=2**

キー	説明
hw:NUMA_def	インスタンスの NUMA トポロジーの定義。RAM 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には、NUMA トポロジーを定義することでホストが NUMA を効果的に使用してゲスト OS のパフォーマンスを向上することができます。フレーバーで定義された NUMA の定義は、イメージの定義をオーバーライドします。有効な定義は以下のとおりです。 ● **numa_nodes**: インスタンスに公開する NUMA ノードの数。イメージの NUMA 設定が上書きされるようにするには **1** を指定します。 ● **numa_cpus.0**: 仮想 CPU N-M を NUMA ノード 0 へマッピング (コンマ区切りの一覧) ● **numa_cpus.1**: 仮想 CPU N-M を NUMA ノード 1 へマッピング (コンマ区切りの一覧) ● **numa_mem.0**: メモリー N MB を NUMA ノード 0 へマッピング ● **numa_mem.1**: メモリー N MB を NUMA ノード 1 へマッピング ● **numa_cpu.N** および **numa_mem.N** は、**numa_nodes** が設定されている場合のみに有効です。また、これらの定義が必要になるのは、インスタンスの NUMA ノードの CPU および RAM が対称的に割り当てられていない場合のみです (NFV ワークロードの一部には重要)。  **注記** **numa_cpu** または **numa_mem.N** の値が利用可能な値よりも多く指定されている場合には、例外が発生します。 インスタンスに 8 個の仮想 CPU、4 GB の RAM が指定されている場合の例: ● **hw:numa_nodes=2** ● **hw:numa_cpus.0=0,1,2,3,4,5** ● **hw:numa_cpus.1=6,7** ● **hw:numa_mem.0=3072** ● **hw:numa_mem.1=1024** スケジューラーは、NUMA ノードが 2 つあり、そのうちの 1 つのノードで 6 つの CPU および 3072 MB または 3 GB のメモリーを実行し、別のノードで 2 つの CPU および 1024 MB または 1 GB のメモリーを実行できるホストを検索します。ホストに 8 つの CPU および 4 GB のメモリーを実行できる NUMA ノードが 1 つある場合は、有効な一致とは見なされません。

キー	説明
hw:watchdog_action	インスタンスのウォッチドッグデバイスを使用して、インスタンスに何らかの理由でエラー (またはハング) が発生した場合にアクションをトリガーすることができます。有効なアクションは以下のとおりです。 ● **disabled**: デバイスは接続されません (デフォルト値)。 ● **pause**: インスタンスを一時停止します。 ● **poweroff**: インスタンスを強制終了します。 ● **reset**: インスタンスを強制リセットします。 ● **none**: ウォッチドッグを有効化しますが、インスタンスにエラーが発生してもアクションは実行しません。 例: **hw:watchdog_action=poweroff**

hw:pci_numa_affinity_policy	このパラメーターを使用して、PCI パススルーデバイスおよび SR-IOV インターフェースの NUMA アフィニティーポリシーを指定することができます。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) 以下のいずれかの場合に、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。
	例: **hw:pci_numa_affinity_policy=required**

キー	説明
hw_rng:action	イメージプロパティーを使用して乱数生成器をインスタンスに追加することができます (Red Hat OpenStack Platform ドキュメントの『Command-Line Interface Reference』で **hw_rng_model** を参照してください)。 このデバイスを追加した場合の有効なアクションは以下のとおりです。 ● **allowed**: **True** に指定すると、デバイスが有効化され、**False** に指定すると無効化されます。デフォルトではデバイスは無効となっています。 ● **rate_bytes**: エントロピープールを満たすために、インスタンスのカーネルが rate_period (整数) の間隔でホストから読み取ることのできる最大のバイト数 ● **rate_period**: 秒単位で示した読み取り期間 (整数) 例: **hw_rng:allowed=True**
hw_video:ram_max_mb	ビデオデバイスの最大許容 RAM (MB 単位) 例: **hw:ram_max_mb=64**
quota:option	インスタンスの制限を強制します。有効なオプションは以下のとおりです。 ● **cpu_period**: **cpu_quota** を強制する時間 (マイクロ秒単位)。指定した **cpu_period** 内では、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。値は [1000, 1000000] の範囲内で指定する必要があります。0 は **値なし** を意味します。 ● **cpu_quota: 各 cpu_period における仮想 CPU の最大許容帯域幅 (マイクロ秒単位)。** この値は、[1000, 18446744073709551] の範囲内で指定する必要があります。0 は **値なし** を意味し、負の値は仮想 CPU が制御されないことを意味します。**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。 ● **cpu_shares**: ドメインの CPU 時間の共有。この値は、同じドメイン内の他のマシンに対する重み付けがされている場合にのみに有意となります。つまり、200 のフレーバーを使用するインスタンスには、100 のインスタンスのマシン時間の 2 倍の時間が割り当てられることになります。 ● **disk_read_bytes_sec**: 最大のディスク読み取り速度 (バイト毎秒単位) ● **disk_read_iops_sec**: 1 秒あたりの最大の読み取り I/O 操作回数 ● **disk_write_bytes_sec**: 最大のディスク書き込み速度 (バイト毎秒単位) ● **disk_write_iops_sec**: 1 秒あたりの最大の書き込み I/O 操作回数 ● **disk_total_bytes_sec**: 総スループットの上限 (バイト毎秒単位) ● **disk_total_iops_sec**: 1 秒あたりの最大の総 I/O 操作数 ● **vif_inbound_average**: 受信トラフィックの指定平均値 ● **vif_inbound_burst**: **vif_inbound_peak** の速度で受信可能なトラフィックの最大量

キー	説明
	● **vif_inbound_peak**: 受信トラフィックの最大受信速度 ● **vif_outbound_average**: 送信トラフィックの指定平均値 ● **vif_outbound_burst**: **vif_outbound_peak** の速度で送信可能なトラフィックの最大量 ● **vif_outbound_peak**: 送信トラフィックの最大送信速度 例: **quota:vif_inbound_average=10240** さらに、VMware ドライバーは、CPU、メモリー、ディスク、ネットワークの上限、下限を制御する以下のクォータオプションや、テナントで利用可能なリソースの相対割り当てを制御するのに使用可能な **共有** をサポートします。 ● **cpu_limit**: 仮想マシンで利用可能な最大 CPU 周波数 (MHz 単位) ● **cpu_reservation**: 仮想マシンで利用可能な確保済みの最大 CPU リソース (MHz 単位) ● **cpu_shares_level**: 競合が発生した場合の CPU 割り当てレベル (共有)。許容値は **high**、**normal**、**low**、および **custom** です。 ● **cpu_shares_share**: 割り当て済みの CPU 共有数。**cpu_shares_level** が **custom** に設定されている場合のみ適用可能です。 ● **memory_limit**: 仮想マシンで利用可能な最大メモリー容量 (MB 単位) ● **memory_reservation**: 仮想マシンで利用可能な確保済みの最大メモリー容量 (MB 単位) ● **memory_shares_level**: 競合が発生した場合のメモリー割り当てレベル (共有)。許容値は **high**、**normal**、**low**、および **custom** です。 ● **memory_shares_share**: 割り当て済みのメモリー共有数。**memory_shares_level** が **custom** に設定されている場合のみ適用可能です。 ● **disk_io_limit**: 仮想マシンでの最大 I/O 使用率 (1 秒あたりの I/O 操作回数単位) ● **disk_io_reservation**: 仮想マシンで利用可能な確保済みの最大ディスクリソース (1 秒あたりの I/O 操作回数単位) ● **disk_io_shares_level**: 競合が発生した場合の I/O 割り当てレベル (共有)。許容値は **high**、**normal**、**low**、および **custom** です。 ● **disk_io_shares_share**: 割り当て済みの I/O 共有数。**disk_io_shares_level** が **custom** に設定されている場合のみ適用可能です。 ● **vif_limit**: 仮想ネットワークアダプターで利用可能な最大ネットワーク帯域幅 (Mbps 単位) ● **vif_reservation**: 仮想ネットワークアダプターで利用可能な確保済みの最小ネットワーク帯域幅 (Mbps 単位) ● **vif_shares_level**: 競合が発生した場合のネットワーク帯域幅の割り当てレベル (共有)。許容値は **high**、**normal**、**low**、および **custom** です。

キー	説明
	• **vif_shares_share**: 割り当て済みの帯域幅共有数。**vif_shares_level** が **custom** に設定されている場合のみ適用可能です。

4.4. ホストアグリゲートの管理

パフォーマンスおよび管理目的で、単一の Compute デプロイメントを複数の論理グループにパーティショニングすることができます。OpenStack では以下のような用語を使用しています。

- ホストアグリゲート: ホストアグリゲートは、ホストをグループ化してまとめることによって OpenStack デプロイメント内に論理ユニットを作成します。アグリゲートは、割り当てられた Compute ホストと関連付けられたメタデータです。1台のホストは複数のアグリゲートに属することが可能です。ホストアグリゲートの表示と作成ができるのは管理者のみです。
  アグリゲートのメタデータは通常、Compute のスケジューラーで使用する情報を提供します (例: 特定のフレーバーやイメージを複数のホストの1つのサブネットに制限するなど)。ホストアグリゲートで指定されるメタデータは、フレーバー内で同じメタデータが指定されているインスタンスにホストの使用を限定します。

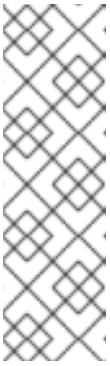
  管理者は、ホストアグリゲートを使用して、ロードバランスの処理、物理的な分離 (または冗長) の強制、共通の属性を持つサーバーのグループ化、ハードウェアクラスの分類などを行うことができます。アグリゲートの作成時には、ゾーン名を指定する必要があります。この名前がエンドユーザーに表示されます。

- アベイラビリティーゾーン: アベイラビリティーゾーンとは、ホストアグリゲートのエンドユーザーのビューです。エンドユーザーはゾーンがどのホストで構成されているかを表示したり、ゾーンのメタデータを確認したりすることはできません。ユーザーが見ることができるのはゾーン名のみです。
  一定の機能や一定のエリア内で設定された特定のゾーンを使用するようにエンドユーザーを誘導することができます。

4.4.1. ホストアグリゲートのスケジューリングの有効化

デフォルトでは、ホストアグリゲートのメタデータは、インスタンスの使用先のフィルタリングには使用されません。メタデータの使用を有効にするには、Compute のスケジューラーの設定を更新する必要があります。

1. Compute 環境ファイルを開きます。
2. **NovaSchedulerDefaultFilters** 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。
   - ホストアグリゲートのメタデータ用の **AggregateInstanceExtraSpecsFilter**



注記

同じ NovaSchedulerDefaultFilters パラメーターの値として AggregateInstanceExtraSpecsFilter および ComputeCapabilitiesFilter フィルターの両方を指定する場合には、フレーバー extra_specs の設定にスコープを定義した仕様を使用する必要があります。使用しない場合には、ComputeCapabilitiesFilter は適切なホストの選択に失敗します。これらのフィルター用にフレーバー extra_specs キーのスコープを定義するのに使用する名前空間の詳細は、表4.7「スケジューリングフィルター」 を参照してください。

- インスタンス起動時のアベイラビリティーゾーンのホスト仕様用の AvailabilityZoneFilter

3. 設定ファイルを保存します。
4. オーバークラウドをデプロイします。

4.4.2. アベイラビリティーゾーンまたはホストアグリゲートの表示

Dashboard に管理ユーザーとしてログインして 管理 > システム > ホストアグリゲートを選択します。ホストアグリゲート のセクションに現在定義済みのアグリゲートがすべてリストされます。アベイラビリティーゾーン のセクションには全ゾーンがリストされます。

4.4.3. ホストアグリゲートの追加

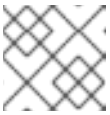
1. Dashboard に管理ユーザーとしてログインして 管理 > システム > ホストアグリゲートを選択します。ホストアグリゲート のセクションに現在定義済みのアグリゲートがすべてリストされます。
2. ホストアグリゲートの作成 をクリックします。
3. 名前 フィールドにアグリゲートの名前を入力します。この名前が アベイラビリティーゾーン フィールドでエンドユーザーに表示されます。
4. アグリゲートのホストの管理 をクリックします。
5. 「+」アイコンをクリックしてホストを選択します。
6. ホストアグリゲートの作成 をクリックします。

4.4.4. ホストアグリゲートの更新

1. Dashboard に管理ユーザーとしてログインして 管理 > システム > ホストアグリゲートを選択します。ホストアグリゲート のセクションに現在定義済みのアグリゲートがすべてリストされます。
2. インスタンスの 名前 または アベイラビリティーゾーン を更新するには、以下の手順で行います。
   - アグリゲートの ホストアグリゲートの編集 ボタンをクリックします。
   - 名前 または アベイラビリティーゾーン のフィールドを更新して、保存 をクリックします。
3. インスタンスの 割り当て済みのホスト を更新するには、以下の手順で行います。
   - アクション の下にあるアグリゲートの矢印アイコンをクリックします。

- ホストの管理 をクリックします。
- 「+」または「-」のアイコンをクリックしてホストの割り当てを変更します。
- 終了したら、保存 をクリックします。

4. インスタンスの メタデータ を更新するには、以下の手順で行います。
   - アクション の下にあるアグリゲートの矢印アイコンをクリックします。
   - メタデータの更新 ボタンをクリックします。現在の値はすべて右側の 選択済みのメタデータ の下に一覧表示されます。
   - 利用可能なメタデータ で その他 のフィールドをクリックして、追加するキーを指定します。事前に定義したキー (表4.6「ホストアグリゲートのメタデータ」を参照) を使用するか、独自のキーを追加します (このキーと全く同じキーがインスタンスのフレーバーに設定されている場合にのみ有効となります)。
   - 「+」ボタンをクリックします。選択済みのメタデータ の下に新しいキーが表示されるようになりました。

     注記

     キーを削除するには、「-」のアイコンをクリックします。

   - 保存 をクリックします。

     表4.6 ホストアグリゲートのメタデータ

キー	説明
filter_tenant_id	指定した場合には、アグリゲートはこのテナント (プロジェクト) のみをホストします。これは、Compute のスケジューラーに設定されている **AggregateMultiTenancyIsolation** フィルターによって異なります。

4.4.5. ホストアグリゲートの削除

1. Dashboard に管理ユーザーとしてログインして 管理 > システム > ホストアグリゲートを選択します。ホストアグリゲート のセクションに現在定義済みのアグリゲートがすべてリストされます。
2. 割り当てられている全ホストをアグリゲートから削除します。
   a. アクション の下にあるアグリゲートの矢印アイコンをクリックします。
   b. ホストの管理 をクリックします。
   c. 「-」アイコンをクリックして、全ホストを削除します。
   d. 終了したら、保存 をクリックします。
3. アクション の下にあるアグリゲートの矢印アイコンをクリックします。
4. このダイアログ画面と次の画面で ホストアグリゲートの削除 をクリックします。

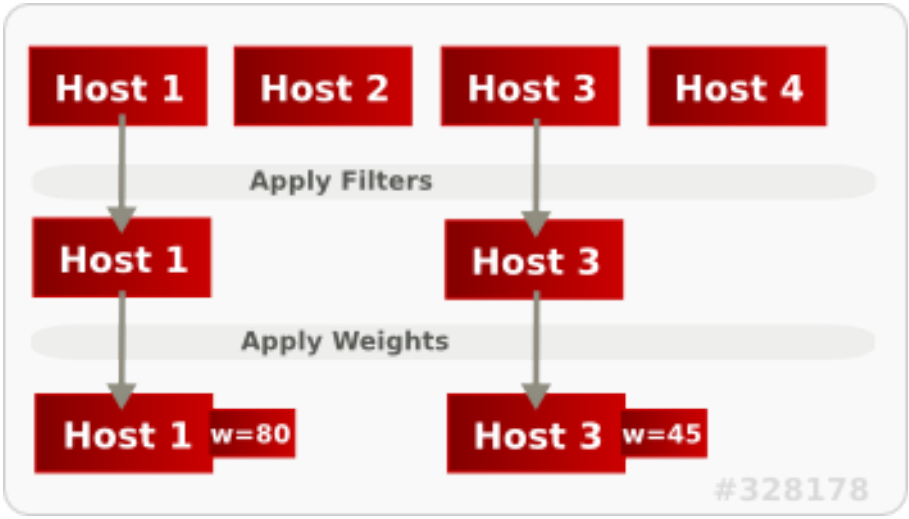
4.5. ホストのスケジュール

Compute のスケジューリングサービスは、インスタンスの配置先となるホストまたはホストアグリゲートを決定します。管理者は、設定を使用して、スケジューラーによるインスタンスの配置先の決定方法を定義することができます。たとえば、特定のグループや適切な量の RAM があるホストにスケジューリングを限定することが可能です。

以下のコンポーネントを設定することができます。

- フィルター: インスタンスの配置先候補となるホストの初期セットを決定します (「スケジューリングフィルターの設定」を参照)。
- 重み: フィルタリングの完了時に選出されたホストのセットは重み付けのシステムを使用して優先順位が決定されます。最も高い重みが最優先されます (「スケジューリングの重みの設定」を参照)。
- スケジューラーサービス: スケジューラーホスト上の /var/lib/config-data/puppet-generated/<nova_container>/etc/nova/nova.conf ファイルには数多くの設定オプションがあります。これらのオプションは、スケジューラーがタスクを実行する方法や、重み/フィルターを処理する方法を決定します。
- Placement サービス： 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必要な特性を指定します（「Placement サービスの特性の設定」を参照）。

下図では、フィルタリング後には Host 1 と Host 3 の両方が条件に適合しています。Host 1 の重みが最も高いため、スケジューリングで最優先されます。



4.5.1. スケジューリングフィルターの設定

スケジューラーが使用するフィルターを定義するには、Compute 環境ファイルの **NovaSchedulerDefaultFilters** パラメーターを使用します。フィルタを追加したり削除したりすることができます。

デフォルト設定では、スケジューラーで以下のフィルターが実行されます。

- RetryFilter
- AvailabilityZoneFilter
- ComputeFilter

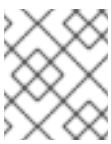
- ComputeCapabilitiesFilter
- ImagePropertiesFilter
- ServerGroupAntiAffinityFilter
- ServerGroupAffinityFilter

一部のフィルターは、以下の方法でインスタンスに渡されるパラメーターの情報を使用します。

- **nova boot** コマンド
- インスタンスのフレーバー (「フレーバーのメタデータの更新」を参照)
- インスタンスのイメージ (「付録A *イメージの設定パラメーター*」を参照)

利用可能なすべてのフィルターを一覧にして以下の表に示します。

表4.7 スケジューリングフィルター

フィルター	説明
AggregateImagePropertiesIsolation	インスタンスのイメージのメタデータが一致するホストアグリゲート内のホストのみを渡します。これは、そのインスタンスにホストアグリゲートが指定されている場合にのみ有効です。詳細は、「「イメージの作成」」を参照してください。
AggregateInstanceExtraSpecsFilter	ホストアグリゲート内のメタデータは、ホストのフレーバーのメタデータと一致する必要があります。詳細は、「「フレーバーのメタデータの更新」」を参照してください。
	このフィルターを **ComputeCapabilitiesFilter** と同じ **NovaSchedulerDefaultFilters** パラメーターで使用するには、正しい名前空間でフレーバー **extra_specs** キーをプレフィックスしてそのスコープを定義する必要があります。 • **ComputeCapabilitiesFilter** 名前空間: **"capabilities:"** • **AggregateInstanceExtraSpecsFilter** 名前空間: **"aggregate_instance_extra_specs:"**
AggregateMultiTenancyIsolation	**filter_tenant_id** を指定したホストには、そのテナント (プロジェクト) からのインスタンスのみを配置することができます。  **注記** テナン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。
AllHostsFilter	利用可能な全ホストを渡します (ただし、他のフィルターは無効化しません)。

フィルター	説明
AvailabilityZoneFilter	インスタンスに指定されているアベイラビリティーゾーンを使用してフィルタリングします。
ComputeCapabilitiesFilter	Compute のメタデータが正しく読み取られるようにします。「**:**」よりも前の部分はすべて名前空間として読み取られます。たとえば、**quota:cpu_period** では **quota** が名前空間として、**cpu_period** がキーとして使用されます。
ComputeFilter	稼働中の有効なホストのみを渡します。
DifferentHostFilter	指定されている単一または複数のホストとは別のホスト上でインスタンスをビルドできるようにします。**nova boot** の **--different_host** オプションを使用して、**different** (別の) ホストを指定します。
ImagePropertiesFilter	インスタンスのイメージプロパティーに一致するホストのみを渡します。詳細は、「「イメージの作成」」を参照してください。
IsolatedHostsFilter	**isolated_hosts** および **isolated_images** (コンマ区切りの値) を使用して指定されている分離されたイメージを実行中の分離されたホストのみを渡します。
JsonFilter	インスタンスのカスタム JSON フィルターを認識/使用します。 ● 有効な演算子: =、<、>、in、⇐、>=、not、or、and ● 認識される値: $free_ram_mb、$free_disk_mb、$total_usable_ram_mb、$vcpus_total、$vcpus_used
	このフィルターは、クエリーヒントとして **nova boot** コマンドで指定されます。以下に例を示します。 **--hint query='['>=', '$free_disk_mb', 200 * 1024]'**
MetricsFilter	メトリックが利用できないホストを除外します。
NUMATopologyFilter	NUMA トポロジーに基づいてホストを除外します。インスタンスにトポロジーが未定義の場合には、任意のホストを使用することができます。このフィルターは、NUMA トポロジーが完全に同じインスタンスとホストをマッチングするように試みます (そのホスト上ではインスタンスのスケジューリングは試みません)。また、このフィルターは、NUMA ノードの標準的なオーバーサブスクリプションの上限を確認し、それに応じて、コンピュートホストに対して制限を指定します。

フィルター	説明
PCIWeigher	重み付け関数はホスト上の PCI デバイスの数とインスタンスに要求された PCI デバイス数に基づいて重みを計算することができます。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 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う必要があります。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、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、PCI デバイスを要求していない場合には 3 番目のホストが優先されるべきです。 詳しくは、「PCI デバイスがアタッチされている NUMA ノードの確保」を参照してください。
RetryFilter	スケジュールを試みて失敗したホストを除外します。**scheduler_max_attempts** が正の値の場合に有効です (デフォルトは「3」)。
SameHostFilter	指定されている単一または複数のホストを渡します。**nova boot** に **--hint same_host** オプションを使用するインスタンスのホストを指定します。
ServerGroupAffinityFilter	特定のサーバーグループのホストのみを渡します。 • サーバーグループにアフィニティーポリシーを指定します (**nova server-group-create --policy affinity groupName**)。 • そのグループでインスタンスをビルドします (**nova boot** の **--hint group=UUID** オプション)。
ServerGroupAntiAffinityFilter	インスタンスをまだホストしていないサーバーグループ内のホストのみを渡します。 • サーバーグループにアンチアフィニティーポリシーを指定します (**nova server-group-create --policy anti-affinity groupName**)。 • そのグループでインスタンスをビルドします (**nova boot** の **--hint group=UUID** オプション)。
SimpleCIDRAffinityFilter	インスタンスの cidr および **build_new_host_ip** のヒントで指定されている IP サブネット範囲のホストのみを渡します。以下に例を示します。 **--hint build_near_host_ip=192.0.2.0 --hint cidr=/24**

4.5.2. スケジューリングの重みの設定

ホストは、スケジューリング用に重み付けすることができます。(フィルタリング後に) 重みが最大のホストが選択されます。重み付け関数にはすべて、ノードの重みを正規化した後に適用される乗数が指定されます。ノードの重みは以下のように計算されます。

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

重みのオプションは、コンピュートノードの設定ファイルで定義することができます。

4.5.2.1. ホストの重みのオプション設定

スケジューラーが使用するホストの重み付け関数は、[DEFAULT] scheduler_weight_classes のオプションで定義することができます。有効な重み付け関数は以下のとおりです。

- **nova.scheduler.weights.ram: ホストの使用可能なメモリーを重み付けします。**
- **nova.scheduler.weights.metrics: ホストのメトリックを重み付けします。**
- **nova.scheduler.weights.affinity: 特定のサーバーグループ内にある他のホストとのホストの近接性を重み付けします。**
- **nova.scheduler.weights.all_weighers: 全ホストの重み付け関数を使用します (デフォルト)。**

表4.8 ホストの重みのオプション

重み付け関数	オプション	説明
すべて	[DEFAULT] **scheduler_host_subset_size**	ホストの選択先のサブセットサイズを定義します (整数)。**1** 以上に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。1 未満の場合には無視されて、1 が使用されます (整数値)。
affinity	[default] **soft_affinity_weight_multiplier**	グループソフトアフィニティーのホストを重み付けする際に使用します。正の浮動小数点数を指定してください。これは、負の値を指定すると反対の動作が発生してしまうためです。通常、このような反対の動作は、**soft_anti_affinity_weight_multiplier** が制御します。
affinity	[default] **soft_anti_affinity_weight_multiplier**	グループソフト非アフィニティーのホストを重み付けする際に使用します。正の浮動小数点数を指定してください。これは、負の値を指定すると反対の動作が発生してしまうためです。通常、このような反対の動作は、**soft_affinity_weight_multiplier** が制御します。
metrics	[metrics] **required**	[metrics] **weight_setting** 内で使用できないメトリックの処理方法を指定します。 ● **True**: メトリックは必須です。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、**scheduler_default_filters** オプションで **MetricFilter** フィルターを使用してください。 ● **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。返される値は **weight_of_unavailable** によって設定されます。

重み付け関数	オプション	説明
metrics	[metrics] **weight_of_unavailable**	[metrics] **weight_setting** 内のメトリックが使用できない場合に重みとして使用されます。**required=False** の場合に有効です。
metrics	[metrics] **weight_multiplier**	メトリックを重み付けする乗数。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。この値が負の場合には、最も低いメトリックのホストが優先され、インスタンスが複数のホストでスタッキングされます。
metrics	[metrics] **weight_setting**	重み付けに使用されるメトリックと比率を指定します。**metric=ratio** のペアのコンマ区切りリストを使用します。有効なメトリック名は以下のとおりです。 ● **cpu.frequency**: 現在の CPU 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**
ram	[DEFAULT] **ram_weight_multiplier**	RAM の乗数 (浮動小数点)。デフォルトでは、**ram_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。この値が負の場合には、最も RAM が低いホストが優先され、インスタンスが複数のホストでスタッキングされます。

4.5.3. Placement サービスの特性の設定

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 サービスにクエリーを行うには、アンダークラウドに python3-osc-placement パッケージをインストールします。

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。

Compute (nova) サービスは、**nova-compute** および **nova-scheduler** プロセスを使用して Placement サービスと協調してインスタンスを作成します。

nova-compute

- リソースプロバイダーレコードを作成します。
- 利用可能な仮想 CPU の数など、利用可能なリソースを数量的に記述するインベントリーを設定します。
- リソースプロバイダーの性質的な要素を記述する特性を設定します。**libvirt** 仮想化ドライバーは、これらの特性を Placement サービスに報告します（詳しくは 「Placement サービスの特性としての libvirt 仮想化ドライバー機能」 を参照）。

nova-scheduler

- Placement サービスに、割り当て候補の一覧を要求します。
- インスタンスが要求する特性に基づいて、サーバーの構築先となるホストを決定します。

4.5.3.1. Placement サービスの特性としての libvirt 仮想化ドライバー機能

libvirt 仮想化ドライバーの機能を Placement サービスの特性として使用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。以下に例を示します。

- **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**

インスタンスが特定のハードウェア、仮想化、ストレージ、ネットワーク、またはデバイスの特性に要求することのできる標準化された定数のカタログについては、**os-traits** ライブラリー を参照してください。

以下の **libvirt** 仮想化ドライバーは、命令セットの種別 (SSE4、AVX、または AVX-512 等) など、ホスト CPU が提供する機能を自動的に Placement サービスに報告します。

- Libvirt QEMU (x86)
- Libvirt KVM (x86)
- Libvirt KVM (ppc64)

これらのドライバーのいずれかを使用している場合、インスタンス用にフレーバー追加スペックまたはイメージメタデータを設定して、リソースプロバイダーに特定の CPU 機能を要求することができます。

4.5.3.2. Placement サービス特性を使用したリソースプロバイダー要求の指定

以下の方法のいずれかを使用して、インスタンス用に要求するリソースプロバイダー特性を指定することができます。

- イメージメタデータを使用した特性の要求
- フレーバー追加スペックを使用した特性の要求

以下の手順例では、インスタンスは特定の CPU 種別を要求します。

前提条件

- Placement サービスのパッケージ **python3-osc-placement** がアンダークラウドにインストールされている。
- デプロイメントで、以下の **libvirt** 仮想化ドライバーのいずれかが使用されている。
  - Libvirt QEMU (x86)
  - Libvirt KVM (x86)
  - Libvirt KVM (ppc64)

手順: イメージメタデータを使用して特性を要求する

1. 新規イメージを作成するか、既存のイメージを変更して、必要な特性を設定します。

```
$ openstack image create ... $IMAGE
$ openstack image set --property trait:HW_CPU_X86_AVX512BW=required $IMAGE
```

2. そのイメージを使用してインスタンスをブートします。

```
$ openstack server create --image=$IMAGE ... $SERVER_NAME
```

結果: AVX-512 をサポートするホストにインスタンスが作成されます。

手順: フレーバー追加スペックを使用して特性を要求する

1. 新規フレーバーを作成するか、既存のフレーバーを変更して、必要な特性を設定します。

```
$ openstack flavor create ... $FLAVOR
$ openstack flavor set --property trait:HW_CPU_X86_AVX512BW=required $FLAVOR
```

2. そのフレーバーを使用してインスタンスをブートします。

```
$ openstack server create --flavor=$FLAVOR ... $SERVER_NAME
```

結果: AVX-512 をサポートするホストにインスタンスが作成されます。

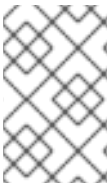
4.5.4. PCI デバイスがアタッチされている NUMA ノードの確保

Compute はフィルタースケジューラーを使用して、PCI を要求するインスタンスに対して、PCI デバイスがアタッチされたホストの優先順位付けを行います。ホストの重み付けは、**PCIWeigher** オプションが使用して、ホスト上で利用可能な PCI デバイスの数と、インスタンスによって要求されている PCI デ

バイスの数に基づいて行われます。インスタンスが PCI デバイスを要求している場合は、より多くの PCI デバイスがアタッチされたホストに、他のホストよりも高い加重値が割り当てられます。インスタンスが PCI デバイスを要求していない場合には、優先順位付けは行われません。

この機能は、以下のような場合に特に役立ちます。

- オペレーターが、PCI デバイスを要求するゲストインスタンス向けに PCI デバイスがアタッチされたノード (通常は高コストでリソースが限定される) を確保する場合
- インスタンスを起動するユーザーが、必要時に PCI デバイスを利用できるようにする場合。



注記

この値が考慮されるようにするには、Compute 環境ファイルの **NovaSchedulerDefaultFilters** パラメーターに、**PciPassthroughFilter** または **NUMATopologyFilter** どちらかの値を追加する必要があります。

pci_weight_multiplier 設定オプションは、正の値でなければなりません。

4.5.5. 専用の物理 CPU で実行されるエミュレータースレッドの設定

Compute のスケジューラーは、CPU リソースの使用状況を確認して、フレーバーの仮想 CPU (vCPU) 数に基づいてインスタンスを配置します。ゲストの代わりにホストで実行されるハイパーバイザー操作は数多くあります。たとえば、QEMU では、QEMU のメインイベントループや、非同期 I/O 操作などに使用されるスレッドがあるので、それらを考慮に入れて、別々にスケジュールする必要があります。

libvirt ドライバーは、KVM 向けの一般的な配置ポリシーを実装しています。このポリシーでは、仮想 CPU を実行しているのと同じ複数の物理 CPU (pCPU) 全体にわたって、QEMU エミュレータースレッドをフローティングさせることができます。これにより、エミュレータースレッドは、仮想 CPU 操作から借りた時間を使用することになります。ゲストに専用の仮想 CPU 割り当てが必要な場合には、エミュレータースレッドに 1 つまたは複数の物理 CPU を割り当てる必要があります。そのため、スケジューラーに対して、ゲストに関連する可能性のある他の CPU の用途を記述して、配置中にそれ計算に入れる必要があります。

注記

NFV デプロイメントでは、パケットロスを回避するために、仮想 CPU が割り込まれないようにする必要があります。

フレーバー上でエミュレータースレッドの配置ポリシーを有効にする前に、以下に示すように heat パラメーターが定義されていることを確認する必要があります。

- **NovaComputeCpuSharedSet**: このパラメーターには、エミュレータースレッドを実行するように定義された CPU の一覧を設定します。
- **NovaSchedulerDefaultFilters**: 定義するフィルターの一覧に **NUMATopologyFilter** を含めます。

注記

これらの変更を適用するには、アクティブなクラスターで heat パラメーターの値を定義または変更してから再デプロイします。

エミュレータースレッドを分離するには、以下のように設定したフレーバーを使用する必要があります。

```
# openstack flavor set FLAVOR-NAME \
--property hw:cpu_policy=dedicated \
--property hw:emulator_threads_policy=share
```

4.6. インスタンスのスナップショットの管理

インスタンスのスナップショットを使用すると、インスタンスから新規イメージを作成することができます。これは、ベースイメージのアップロードや、公開イメージを取得してローカルで使用するためにカスタマイズする際に非常に便利です。

Image サービスに直接アップロードしたイメージと、スナップショットで作成したイメージの相違点は、スナップショットで作成したイメージには Image サービスデータベースのプロパティーが追加されている点です。これらのプロパティーは **image_properties** テーブルにあり、以下のパラメーターが含まれます。

表4.9 スナップショットのオプション

名前	値
image_type	snapshot
instance_uuid	<スナップショットを作成したインスタンスの uuid>
base_image_ref	<スナップショットを作成したインスタンスのオリジナルイメージの uuid>
image_location	snapshot

スナップショットでは、指定のスナップショットをベースにして新規インスタンスを作成して、その状態にインスタンスを復元することができます。さらに、インスタンスの実行中にスナップショットを作成や復元が可能です。

デフォルトでは、スナップショットをベースとするインスタンスが起動している間は、選択したユーザーとプロジェクトがそのスナップショットにアクセスできます。

4.6.1. インスタンスのスナップショットの作成

注記

インスタンスのスナップショットをテンプレートとして使用して新規インスタンスを作成する場合には、ディスクの状態が一貫していることを確認してください。スナップショットを作成する前に、スナップショットのイメージメタデータのプロパティーを **os_require_quiesce=yes** に設定します。以下に例を示します。

```
$ glance image-update IMAGE_ID --property os_require_quiesce=yes
```

このコマンドが機能するには、ゲストに **qemu-guest-agent** パッケージがインストール済みで、メタデータプロパティーのパラメーターを **hw_qemu_guest_agent=yes** に指定してイメージを作成する必要があります。以下に例を示します。

```
$ glance image-create --name NAME \
--disk-format raw \
--container-format bare \
--file FILE_NAME \
--is-public True \
--property hw_qemu_guest_agent=yes \
--progress
```

hw_qemu_guest_agent=yes パラメーターを無条件で有効化した場合には、別のデバイスをゲストに追加してください。この設定により、PCI スロットが使用され、ゲストに割り当てることのできる他のデバイスの数が制限されます。また、これにより、Windows ゲストでは、未知のハードウェアデバイスについての警告のメッセージが表示されます。

このような理由により、**hw_qemu_guest_agent=yes** パラメーターの設定はオプションとなっており、QEMU ゲストエージェントを必要とするそれらのイメージにのみ使用すべきです。

1. Dashboard で プロジェクト > コンピュート > インスタンスを選択します。
2. スナップショットを作成するインスタンスを選択します。
3. アクション コラムで、スナップショットの作成 をクリックします。
4. スナップショットの作成 ダイアログでは、スナップショットの名前を入力して スナップショットの作成 をクリックします。
   イメージ カテゴリーには、インスタンスのスナップショットが表示されます。

スナップショットからインスタンスを起動するには、スナップショットを選択して 起動 をクリックします。

4.6.2. スナップショットの管理

1. Dashboard で プロジェクト > イメージを選択します。
2. 作成したスナップショットはすべて プロジェクト オプションの下に表示されます。
3. 作成するスナップショットごとに、ドロップダウンリストを使用して以下の機能を実行できます。

a. ボリュームの作成 オプションを使用して、ボリュームを作成してボリューム名の値、説明、イメージソース、ボリューム種別、サイズ、アベイラビリティーゾーンを入力します。詳細は、『ストレージガイド』の「ボリュームの作成」を参照してください。

b. イメージの編集 オプションを使用して、名前、説明、カーネル ID、Ramdisk ID、アーキテクチャー、形式、最小ディスク (GB)、最小メモリー (MB)、パブリックまたはプライベートを更新して、スナップショットのイメージを更新します。詳細は、「イメージの更新」を参照してください。

c. イメージの削除 オプションを使用してスナップショットを削除します。

4.6.3. スナップショットの状態へのインスタンスの再ビルド

スナップショットがベースとなっているインスタンスを削除する場合には、スナップショットにはインスタンス ID が保存されます。nova image-list コマンドを使用してこの情報を確認して、スナップショットでインスタンスを復元します。

1. Dashboard で プロジェクト > コンピュート > イメージを選択します。
2. インスタンスを復元するスナップショットを選択します。
3. アクション コラムで、インスタンスの起動 をクリックします。
4. インスタンスの起動 ダイアログで、インスタンスの名前とその他の詳細を入力して 起動 をクリックします。

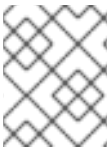
インスタンスの起動に関する詳細は、「インスタンスの起動」を参照してください。

4.6.4. 一貫性のあるスナップショット

以前のリリースでは、バックアップの一貫性を確保するには、アクティブなインスタンスのスナップショットを作成する前にファイルシステムを手動で停止 (fsfreeze) する必要がありました。

Compute の libvirt ドライバーは、QEMU ゲストーエージェント に、イメージのスナップショット作成中ファイルシステムを (fsfreeze-hook がインストールされている場合には、アプリケーションも対象) フリーズするように自動的に要求するようになりました。ファイルシステムの停止に対するサポートにより、スケジュールされた自動スナップショット作成をブロックデバイスレベルで実行できるようになりました。

この機能は、QEMU ゲストエージェント (qemu-ga) がインストール済みで、かつイメージのメタデータで有効化されている (hw_qemu_guest_agent=yes) 場合にのみ有効です。



注記

スナップショットは、実際のシステムバックアップの代わりとみなすべきではありません。

4.7. インスタンスのレスキューモードの使用

Compute では、仮想マシンをレスキューモードで再起動する方法があります。レスキューモードは、仮想マシンイメージが原因で、インスタンスがアクセス不可能な状態となっている場合に、そのインスタンスにアクセスするためのメカニズムを提供します。レスキューモードの仮想マシンは、ユーザーが仮想マシンに新規 root パスワードを使用してアクセスし、そのマシンを修復することができます。こ

の機能は、インスタンスのファイルシステムが破損した場合に役立ちます。デフォルトでは、レスキューモードのインスタンスは初期イメージから起動して、第 2 のイメージとして現在のブートディスクをアタッチします。

4.7.1. レスキューモードのインスタンス用イメージの準備

ブートディスクとレスキューモード用のディスクには同じ UUID が使用されているため、仮想マシンがレスキューモード用のディスクの代わりにブートディスクから起動されてしまう可能性があります。

この問題を回避するには、「イメージの作成」の手順に従い、レスキューイメージとして新しいイメージを作成してください。

注記

rescue イメージは、デフォルトでは glance に保管され、nova.conf で設定されていますが、レスキューを実行する際に選択することもできます。

4.7.1.1. ext4 ファイルシステムを使用している場合のレスキューイメージ

ベースイメージが ext4 ファイルシステムを使用する場合には、以下の手順を使用してそれをベースにレスキューイメージを作成できます。

1. tune2fs コマンドを使用して、UUID を無作為な値に変更します。

```
# tune2fs -U random /dev/DEVICE_NODE
```

DEVICE_NODE はルートデバイスノードに置き換えます (例:sda、vda など)。

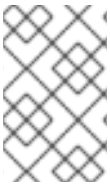
2. 新しい UUID を含む、ファイルシステムの詳細を確認します。

```
# tune2fs -l
```

3. /etc/fstab ファイルで UUID を新しい値に置き換えます。fstab にマウントされている追加のパーティションがある場合には、UUID を新しい値に置き換える必要がある場合があります。

4. /boot/grub2/grub.conf ファイルを更新し、ルートディスクの UUID パラメーターを新しい UUID に置き換えます。

5. シャットダウンして、このイメージをレスキューイメージに使用します。これにより、レスキューイメージには新たに無作為な UUID が割り当てられ、レスキューするインスタンスとの競合が発生しなくなります。



注記

XFS ファイルシステムでは、実行中の仮想マシン上のルートデバイスの UUID は変更できません。仮想マシンがレスキューモード用のディスクから起動するまで再起動を続けます。

4.7.2. OpenStack Image サービスへのレスキューイメージの追加

対象のイメージの UUID を変更したら、以下のコマンドを実行して、生成されたレスキューイメージを OpenStack Image サービスに追加します。

1. Image サービスにレスキューイメージを追加します。

```
# glance image-create --name IMAGE_NAME --disk-format qcow2 \
  --container-format bare --is-public True --file IMAGE_PATH
```

IMAGE_NAME は、イメージの名前に、IMAGE_PATH はイメージの場所に置き換えます。

2. **image-list** コマンドを使用して、インスタンスをレスキューモードで起動するのに必要な IMAGE_ID を取得します。

```
# glance image-list
```

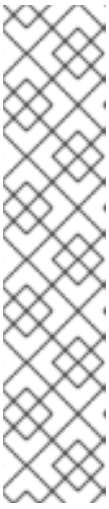
OpenStack Dashboard を使用してイメージをアップロードすることも可能です。「イメージのアップロード」を参照してください。

4.7.3. レスキューモードでのインスタンスの起動

1. デフォルトのイメージではなく、特定のイメージを使用してインスタンスをレスキューする必要があるため、**--image** パラメーターを使用します。

```
# nova rescue --image IMAGE_ID VIRTUAL_MACHINE_ID
```

IMAGE_ID は使用するイメージ ID に、VIRTUAL_MACHINE_ID はレスキューする仮想マシンの ID に置き換えます。



注記

nova rescue コマンドを使用すると、インスタンスでソフトシャットダウンを実行することができます。これにより、ゲストオペレーティングシステムは、インスタンスの電源をオフにする前に、制御されたシャットダウンを実行することができます。シャットダウンの動作は、Compute 設定ファイルの **shutdown_timeout** を使用して設定されます。この値は、ゲストオペレーティングシステムが完全にシャットダウンするまでの合計時間 (秒単位) を指定します。デフォルトのタイムアウトは 60 秒です。

このタイムアウト値は、**os_shutdown_timeout** でイメージ毎に上書きすることが可能です。これは、異なるタイプのオペレーティングシステムでクリーンにシャットダウンするために必要な時間を指定するイメージのメタデータ設定です。

2. 仮想マシンを再起動します。
3. **nova list** コマンドまたは Dashboard を使用して、コントローラーノード上で仮想マシンのステータスが RESCUE であることを確認します。
4. レスキューモード用のパスワードを使用して、新しい仮想マシンのダッシュボードにログインします。

これで、インスタンスに必要な変更を加えて、問題を修正できる状態となりました。

4.7.4. インスタンスのアンレスキュー

修正したインスタンスは **unrescue** して、ブートディスクから再起動することができます。

1. コントローラーノードで以下のコマンドを実行します。

```
# nova unrescue VIRTUAL_MACHINE_ID
```

VIRTUAL_MACHINE_ID はアンレスキューする仮想マシンの ID に置き換えます。

アンレスキューの操作が正常に完了すると、インスタンスのステータスは ACTIVE に戻ります。

第5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。詳しい情報は、『オーバークラウドの 高度なカスタマイズ』 の「コンポーザブルサービスとカスタムロール」を参照してください。

注記

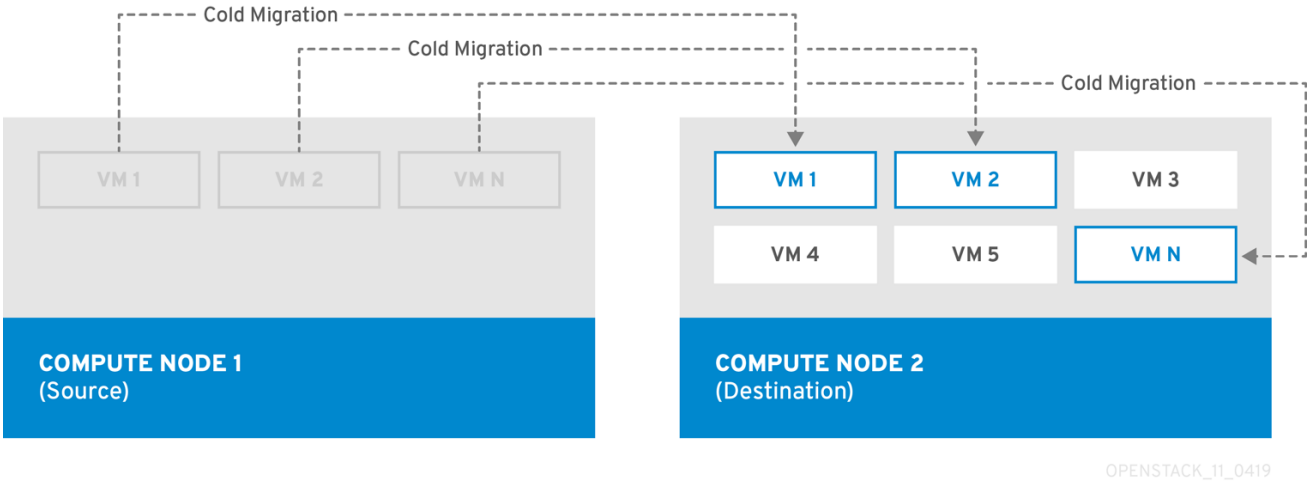
機能しているコンピュートノードがあり、バックアップの目的でインスタンスのコピーを作成する場合や、インスタンスを別の環境にコピーする場合には、『director のインストールと使用方法』 の「オーバークラウドへの仮想マシンのインポート」に記載の手順に従ってください。

5.1. 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すします。



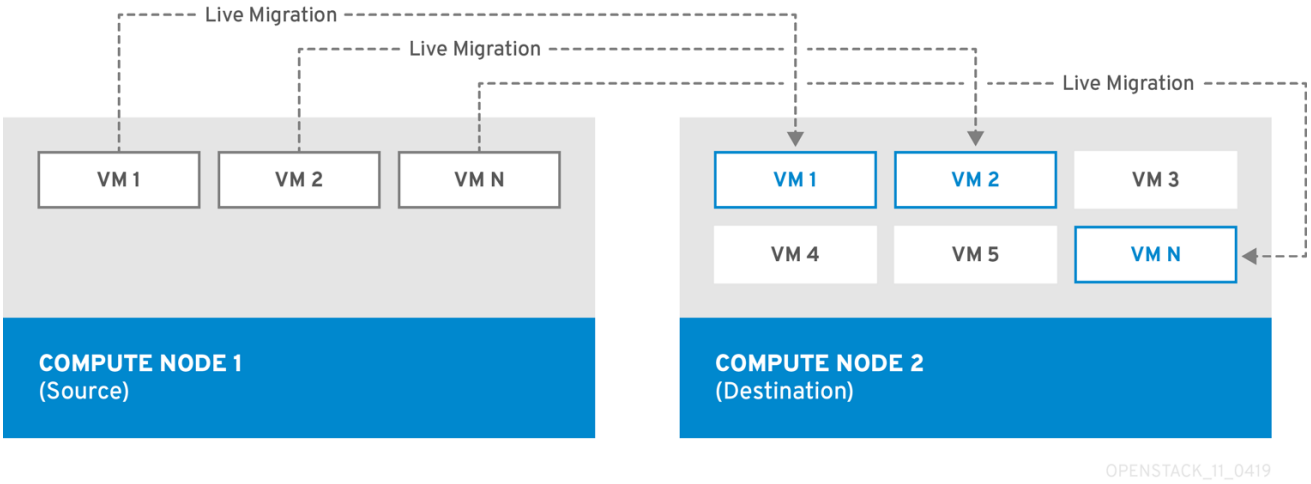
コールドマイグレーションでは、インスタンスに多少のダウンタイムが発生します。移行したインスタンスは、引き続き同じボリュームおよび IP アドレスにアクセスすることができます。

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。

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。

5.2. 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

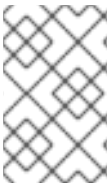
移行元および移行先コンピュートノードの CPU アーキテクチャーは、同一であることが必須です。たとえば、Red Hat では、**x86_64** CPU から **ppc64le** CPU へのインスタンスの移行をサポートしません。CPU ホストパススルーを使用するインスタンス等の場合には、移行元および移行先コンピュートノードの CPU は、完全に同一でなければなりません。すべてのケースで、移行先ノードの CPU 機能は、移行元ノードの CPU 機能の上位セットであることが必須です。CPU ピニングが使用されている場合には、さらに制約が生じます。詳細は、「ライブマイグレーションの制約」を参照してください。

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。また、NUMA トポロジーを使用するインスタンスの場合、移行先コンピュートノードの同じ NUMA ノードで十分な RAM が利用可能でなければなりません。

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。

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、CPU ピニング、ヒュージページ、および DPDK

OpenStack Compute では、環境が以下の条件を満たしている場合に、NUMA、CPU ピニング、または DPDK を使用するインスタンスのライブマイグレーションが可能です。

- 移行先コンピュートノードには、インスタンスが移行元コンピュートノードで使用するのと同じ NUMA ノードに十分な容量がなければなりません。たとえば、インスタンスが **overcloud-compute-0** で **NUMA 0** を使用している場合に、インスタンスを **overcloud-compute-1** にライブマイグレーションするには、**overcloud-compute-1** は **NUMA 0** にインスタンスをサポートするのに十分な空き容量を確保する必要があります。

- Compute の設定で **NovaEnableNUMALiveMigration** が「True」に設定されている。このパラメーターは、コンピュートホストが OVS-DPDK デプロイメント用に設定されている場合に限り、デフォルトで有効になります。

- Compute 設定の **NovaSchedulerDefaultFilters** 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。

- CPU ピニング: フレーバーで CPU ピニングが使用される場合、フレーバーは暗黙的に NUMA トポロジーをインスタンスに適用し、その CPU およびメモリーを特定のホスト CPU およびメモリーにマッピングします。シンプルな NUMA トポロジーと CPU ピニングの違いは、NUMA では CPU コアの範囲が指定されるのに対して、CPU ピニングでは特定の CPU コアが使用される点です。詳しくは、『インスタンス＆イメージガイド』の「NUMA ノードを使用する CPU ピニングの設定」を参照してください。CPU ピニング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には、移行先ホストが空で、等価なハードウェアが使用されている必要があります。

- Data Plane Development Kit (DPDK): インスタンスが DPDK を使用する場合 (Open vSwitch と **dpdk-netdev** を組み合わせて実行するインスタンスなど)、インスタンスはヒュージページも使用します。この場合、OpenStack Compute (nova) がインスタンスを NUMA ノードに固定するような NUMA トポロジーが適用されます。DPDK を使用するインスタンスを移行する場合、移行先コンピュートノードのハードウェア仕様と設定は、移行元コンピュートノードと同一でなければなりません。また、移行元コンピュートノードの NUMA トポロジーを維持するために、移行先コンピュートノードに実行中のインスタンスがあってはいけません。

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。

Single-root Input/Output Virtualization(SR-IOV)

SR-IOV Virtual Function(VF)をインスタンスに割り当てることができます。ただし、これによりライブマイグレーションが妨げられます。通常のネットワークデバイスとは異なり、SR-IOV VF ネットワークデバイスは、永続的な一意の MAC アドレスを持ちません。コンピュートノードがリブートするたびに、またはスケジューラーがインスタンスを新しいコンピュートノードに移行する際に、VF ネットワークデバイスには新しい MAC アドレスが割り当てられます。したがって、OpenStack Compute は SR-IOV 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ができません。SR-IOV を使用するインスタンスでは、コールドマイグレーションを行う必要があります。

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アドレスが必要なため、OpenStack Compute 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。

5.3. 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一覧表示し、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) $ source ~/stackrc
(undercloud) $ openstack compute service set <source> nova-compute --disable
```

<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。

NUMA、CPU ピニング、または DPDK を使用するインスタンスを移行するのでなければ、これで移行を行う準備が整いました。「インスタンスのコールドマイグレーション」または「インスタンスのライブマイグレーション」に詳細を示す必要手順に従います。

NUMA、CPU ピニング、または DPDK を使用するインスタンスを移行する場合は、移行先ノードの準備を行う必要があります。「DPDK インスタンスの準備」で詳細を示す手順を完了してください。

5.4. DPDK インスタンス向けの追加準備作業

NUMA、CPU ピニング、または DPDK を使用するインスタンスを移行する場合は、移行先ノードの準備を行う必要があります。

手順

1. NUMA、CPU ピニング、または DPDK インスタンスの移行先コンピュートノードが無効ではない場合には、これを無効にしてスケジューラーがそのノードにインスタンスを割り当てるのを防ぎます。

```
(overcloud) $ openstack compute service set <dest> nova-compute --disable
```

<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。

2. 複数の DPDK または NUMA インスタンスを移行する場合、移行元コンピュートノードから先に移行していたインスタンスを除き、移行先コンピュートノードにはインスタンスが存在しないようにしてください。

```
(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。

3. 移行先コンピュートノードに、NUMA、CPU ピニング、または DPDK インスタンスを実行するのに十分なリソースを確保するようにしてください。

```
(overcloud) $ openstack host show <dest>
$ ssh <dest>
$ numactl --hardware
$ exit
```

<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。

4. 移行元または移行先コンピュートノードの NUMA 情報を検出するには、以下のコマンドを実行します。

```
$ ssh root@overcloud-compute-n
# lscpu && lscpu | grep NUMA
# virsh nodeinfo
# virsh capabilities
# exit
```

ssh を使用して、**overcloud-compute-n** に接続します。ここで、**overcloud-compute-n** は移行元または移行先コンピュートノードです。

5. インスタンスが NUMA を使用しているかどうか不明な場合は、インスタンスのフレーバーを確認してください。

```
(overcloud) $ openstack server list -c Name -c Flavor --name <vm>
(overcloud) $ openstack flavor show <flavor>
```

- **<vm>** をインスタンスの名前または ID に置き換えてください。
- **<flavor>** をフレーバーの名前または ID に置き換えてください。
    - 表示される **properties** フィールドの **hw:mem_page_size** の値が **any** 以外 (**2MB**、**2048**、または **1GB**) の場合、インスタンスは NUMA トポロジーを持ちます。
    - **properties** フィールドに **aggregate_instance_extra_specs:pinned='true'** が含まれる場合には、インスタンスは CPU ピニングを使用しています。
    - **properties** フィールドに **hw:numa_nodes** が含まれる場合、OpenStack Compute (nova) サービスはインスタンスを特定の NUMA ノードに制限します。

6. NUMA を使用する各インスタンスについて、移行完了後に移行先コンピュートノードの NUMA トポロジーが移行元コンピュートノードの NUMA トポロジーを反映していることを確認できるように、ベースとなるコンピュートノードから NUMA トポロジーに関する情報を取得することができます。以下のコマンドを使用して、この確認を行うことができます。

- NUMA および CPU ピニングに関する詳細を確認するには、以下のコマンドを実行します。

```
$ ssh root@overcloud-compute-n
# virsh vcpuinfo <vm>
```

<vm> をインスタンスの名前に置き換えてください。

- インスタンスがどの NUMA ノードを使用しているかの詳細を確認するには、以下のコマンドを実行します。

```
$ ssh root@overcloud-compute-n
# virsh numatune <vm>
```

<vm> をインスタンスの名前に置き換えてください。

5.5. 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 <vm> --wait
```

- **<vm>** 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、「移行ステータスの確認」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 <vm>`
```

<vm> 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 <vm>`
```

<vm> 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 <vm>
```

<vm> 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) $ source ~/stackrc
(undercloud) $ 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。

6. **（オプション）DPDK を使用する移行先インスタンス用に移行先コンピュートノードを無効にした場合には、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) $ source ~/stackrc
(undercloud) $ openstack compute service set <dest> nova-compute --enable
```

<dest> を移行先コンピュートノードのホスト名に置き換えてください。

5.6. インスタンスのライブマイグレーション

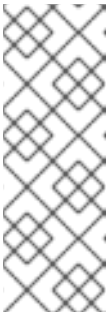
ライブマイグレーションでは、ダウンタイムを最小限に抑えて、インスタンスを移行元コンピュートノードから移行先コンピュートノードに移動します。ライブマイグレーションがすべてのインスタンスに適しているとは限りません。詳細は、「移行の制約」を参照してください。

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 <vm> --live <dest> --wait
```

- **<vm> 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 <vm> --live <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。

```
(overload) $ openstack server show <vm>

+----------------------+--------------------------------------+
| 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、「移行ステータスの確認」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) NUMA、CPU ピニング、または DPDK を使用するインスタンスについて、コンピュートノードから NUMA トポロジーに関する情報を取得して、移行の準備手順の実施中に取得した NUMA トポロジーと比較します。移行元と移行先コンピュートノードの NUMA トポロジーを比較して、移行元と移行先コンピュートノードが同じ NUMA トポロジーを使用するようにします。

- NUMA および CPU ピニングに関する詳細を確認するには、以下のコマンドを実行します。

```
$ ssh root@overcloud-compute-n
# virsh vcpuinfo <vm>
```

  - **overcloud-compute-n** をコンピュートノードのホスト名に置き換えてください。
  - **<vm>** をインスタンスの名前に置き換えてください。

- インスタンスがどの NUMA ノードを使用しているかの詳細を確認するには、以下のコマンドを実行します。

```
$ ssh root@overcloud-compute-n
# virsh numatune <vm>
```

  - **overcloud-compute-n** をコンピュートノードのホスト名に置き換えてください。
  - **<vm>** をインスタンスの名前または ID に置き換えてください。

6. 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) $ source ~/stackrc
(undercloud) $ 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。

7. （オプション）DPDK を使用する移行先インスタンス用に移行先コンピュートノードを無効にした場合には、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) $ source ~/stackrc
(undercloud) $ openstack compute service set <dest> nova-compute --enable
```

<dest> を移行先コンピュートノードのホスト名に置き換えてください。

5.7. 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 の一覧を取得します。

```
$ nova server-migration-list <vm>

+----+-------------+-----------  (...)
| Id | Source Node | Dest Node | (...)
+----+-------------+-----------+ (...)
| 2  | -           | -         | (...)
+----+-------------+-----------+ (...)
```

<vm> 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。

```
$  <vm> <migration-id>
```

- **<vm>** 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。
  nova server-migration-show コマンドを実行すると、以下の例に示すような出力が返されます。

```
+------------------------+--------------------------------------+
| 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

OpenStack 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、「移行に関するトラブルシューティング」を参照してください。

5.8. 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。**

5.8.1. 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. 障害の発生したコンピュートノードに管理者としてログインします。
2. コンピュートノードを無効にします。

```
(overcloud) [stack@director ~]$ openstack compute service set \
<host> <service> --disable
```

- **<host>** 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。
- **<service>** を無効にするサービスの名前 (例:**nova-compute**) に置き換えてください。

3. インスタンスを退避させるには、以下のコマンドを入力します。

```
(overcloud) [stack@director ~]$ nova evacuate [--password <pass>] <vm> [<dest>]
```

- **<pass>** を退避するインスタンスに設定する admin パスワードに置き換えてください。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。
- **<vm>** 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。
- **<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てください。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 の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
```

5.8.2. 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. 障害の発生したコンピュートノードに管理者としてログインします。
2. コンピュートノードを無効にします。

```
(overcloud) [stack@director ~]$ openstack compute service set \
<host> <service> --disable
```

- **<host>** 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。
- **<service>** を無効にするサービスの名前 (例:**nova-compute**) に置き換えてください。

3. 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>] [--force] <host>
```

- **<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てください。退避先を指定しなかった場合には、Compute の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
```

- **<host> 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。**

5.8.3. 共有ストレージの設定

共有ストレージを使用する場合には、Compute サービスのインスタンスディレクトリーを 2 つのノードにエクスポートし、ノードにアクセスできることを確認します。ディレクトリーのパスは、Compute 環境ファイルの state_path および instances_path パラメーターで設定されます。この手順では、デフォルト値の /var/lib/nova/instances を使用しています。共有ストレージを設定することができるのは、root アクセスのあるユーザーのみです。以下の手順の Compute サービスユーザーは、すべてのコントローラーノードおよびコンピュートノードについて同じでなければなりません。

手順

1. **コントローラーノードで以下の手順を実施します。**
   a. **以下の例に示すように、Compute サービスのユーザーに /var/lib/nova/instances ディレクトリーの読み取り/書き込み権限があることを確認します。**

```
drwxr-xr-x.  9 nova nova 4096 Nov  5 20:37 instances
```

   b. **/etc/exports ファイルに以下の行を追加します。**

```
/var/lib/nova/instances node1_IP(rw,sync,fsid=0,no_root_squash)
/var/lib/nova/instances node2_IP(rw,sync,fsid=0,no_root_squash)
```

   node1_IP および node2_IP を 2 つのコンピュートノードの IP アドレスに置き換えてください。以下に例を示します。

```
/var/lib/nova/instances 192.168.24.9(rw,sync,fsid=0,no_root_squash)
/var/lib/nova/instances 192.168.24.21(rw,sync,fsid=0,no_root_squash)
```

   c. **/var/lib/nova/instances ディレクトリーをコンピュートノードにエクスポートします。**

```
# exportfs -avr
```

   d. **NFS サーバーを再起動します。**

```
# systemctl restart nfs-server
```

2. **各コンピュートノードで以下の手順を実施します。**
   a. **/var/lib/nova/instances ディレクトリーがローカルに存在することを確認します。**
   b. **/etc/fstab ファイルに以下の行を追加します。**

```
NFS_SHARE_PATH:/var/lib/nova/instances /var/lib/nova/instances nfs4 defaults 0 0
```

   c. **コントローラーのインスタンスディレクトリーをマウントし、/etc/fstab に記載されているすべてのデバイスをマウントします。**

```
# mount -a -v
```

   d. **QEMU がディレクトリーのイメージにアクセスできることを確認します。**

```
# ls -ld /var/lib/nova/instances
drwxr-xr-x. 9 nova nova 4096 Nov  5 20:37 /var/lib/nova/instances
```

e. ノードでインスタンスディレクトリーを表示できることを確認します。

```
drwxr-xr-x. 9 nova nova 4096 Nov  5 20:37 /var/lib/nova/instances
```

注記

以下のコマンドを実行して、全マウント済みデバイスを確認することもできます。

```
# df -k
```

5.9. 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。

5.9.1. 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。

5.9.2. 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一覧を取得します。

```
$ nova server-migration-list <vm>
```

   <vm> をインスタンスの名前または ID に置き換えてください。

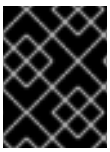
2. ライブマイグレーションを中止します。

```
$ nova live-migration-abort <vm> <migration-id>
```

   - **<vm>** 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一覧を取得します。

```
$ nova server-migration-list <vm>
```

   <vm> 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。

```
$ nova live-migration-force-complete <vm> <migration-id>
```

- **<vm>** 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。

5.9.3. 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。

第6章 インスタンス用のコンフィグドライブの設定

config-drive パラメーターを使用して、読み取り専用のドライブをインスタンスに公開することができます。ファイルを選択してこのドライブに追加すると、インスタンスにアクセスできるようになります。コンフィグドライブは、起動時にインスタンスにアタッチされて、パーティションとしてインスタンスに公開されます。コンフィグドライブは、cloud-init (サーバーのブートストラップ用) と組み合わせる場合や、インスタンスに大容量のファイルを渡す場合に有用です。

6.1. コンフィグドライブのオプション

Compute 環境ファイルを使用して、以下のコンフィグドライブパラメーターを設定します。

- **config_drive_format**: ドライブの形式を設定して、**iso9660** と **vfat** のオプションを確定します。デフォルトでは **iso9660** を使用します。
- **force_config_drive**: このオプションでは、全インスタンスにコンフィグドライブを強制的に公開します。「true」に設定します。
- **mkisofs_cmd**: ISO ファイルの作成に使用するコマンドを指定します。genisoimage しかサポートされないので、この値は変更しないでください。

6.2. コンフィグドライブの使用

インスタンスは、ブート時にコンフィグドライブをアタッチします。これは、**--config-drive** オプションで有効になります。たとえば、このコマンドは test-instance01 という名前の新しいインスタンスを作成して、/root/user-data.txt という名前のファイルが含まれるドライブをアタッチします。

```
# nova boot --flavor m1.tiny --config-drive true --file /root/user-data.txt=/root/user-data.txt --image
cirros test-instance01
```

インスタンスが起動すると、そのインスタンスにログインして、/root/user-data.txt という名前のファイルを確認できます。

注記

このコンフィグドライブを cloud-init の情報源として使用できます。インスタンスの初回起動中には、cloud-init は自動的にコンフィグドライブをマウントして、設定スクリプトを実行することができます。

第7章 パフォーマンス改善のためのコンピュートノードの設定

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、最大の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。
- ヒュージページ: 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。

注記

これらの機能のいずれかを設定すると、インスタンス上に NUMA トポロジーが存在しない場合、暗黙的な NUMA トポロジーが作成されます。

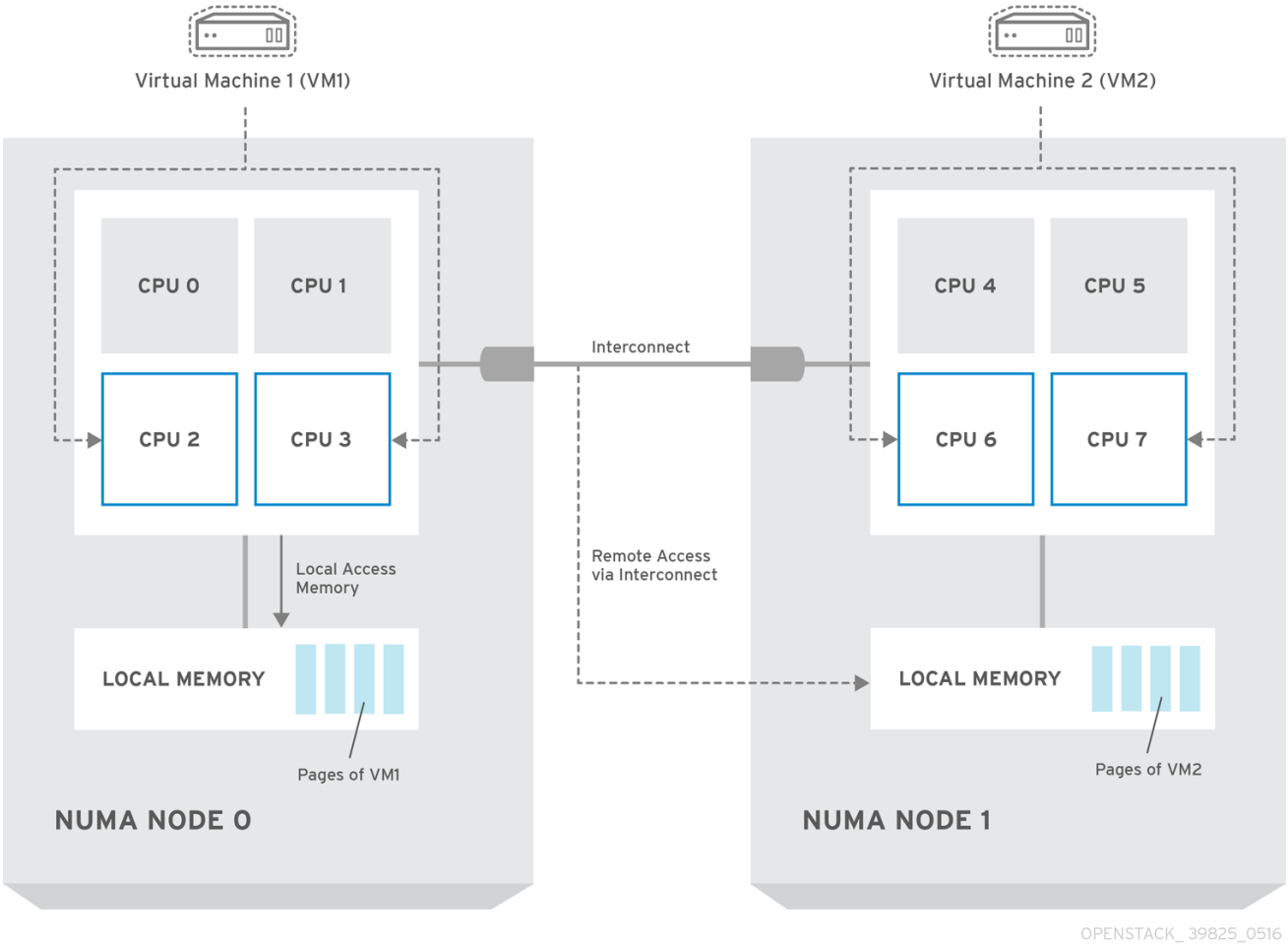
7.1. NUMA ノードを使用する CPU ピニングの設定

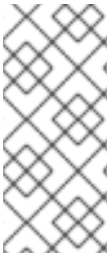
本章では、NUMA トポロジーのサポートを使用して、NUMA アーキテクチャーを持つシステム上で OpenStack 環境を設定する方法について説明します。本章で説明する手順で、仮想マシン (VM) を専用の CPU コアに固定する方法を説明します。これにより、スケジューリング機能および仮想マシンのパフォーマンスが向上します。

ヒント

NUMA についての予備知識は、「What is NUMA and how does it work on Linux ?」のアーティクルに記載されています。

以下の図では、2 つのノードからなる NUMA システムの例と、CPU コアとメモリーページを利用可能にする方法の例が提供されています。





注記

Interconnect 経由で利用可能なリモートのメモリーには、NUMA ノード 0 からの VM1 に NUMA ノード 1 の CPU コアがある場合 のみ アクセスすることができます。このような場合には、NUMA ノード 1 のメモリーは、VM1 の 3 番目の CPU コアのローカルとして機能しますが (例: 上記の図では、VM1 は CPU4 が割り当てられています)、同じ仮想マシンの別の CPU コアに対してはリモートメモリーとして機能します。

libvirt での NUMA のチューニングについての詳しい情報は、『仮想化の設定および管理』を参照してください。

7.1.1. コンピュートノードの設定

具体的な設定は、お使いのホストシステムの NUMA トポロジーによって異なります。ただし、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、それ以外の CPU コアに仮想マシン (VM) を処理させる必要があります。以下の例で、2 つの NUMA ノード間で 8 つの CPU コアを均等に分散させる場合のレイアウトを説明します。

表7.1 NUMA トポロジーの例

	ノード 0		ノード 1	
ホストのプロセス	コア 0	コア 1	コア 4	コア 5
仮想マシン	コア 2	コア 3	コア 6	コア 7

注記

標準的な作業負荷がかかった状態におけるホストのパフォーマンスを観察した上で、ホストのプロセス用に確保するコア数を決定します。

手順

1. Compute 環境ファイルの **NovaVcpuPinSet** 設定を定義して、仮想マシン用に CPU コアを確保します。

```
NovaVcpuPinSet: 2,3,6,7
```

2. 同じファイルの **NovaReservedHostMemory** オプションを、ホストのプロセス用に確保するメモリー容量に設定します。たとえば、512 MB を確保する場合、設定は以下のようになります。

```
NovaReservedHostMemory: 512
```

3. 仮想マシン用に確保した 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、Compute 環境ファイルの **IsolCpusList** パラメーターに、仮想マシン用に確保した CPU コアを設定します。CPU インデックスの一覧またはスペース区切りの範囲を使用して、**IsolCpusList** パラメーターの値を指定します。以下に例を示します。

```
IsolCpusList: 2 3 6 7
```

注記

IsolCpusList パラメーターにより、下層のコンピュートノードは対応する物理 CPU を自らは使用できなくなります。物理 CPU は仮想マシン専用です。

4. この設定を適用するには、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/<compute_environment_file>.yaml
```

7.1.2. スケジューラーの設定

手順

1. Compute 環境ファイルを開きます。
2. **NovaSchedulerDefaultFilters** 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。
   - **NUMATopologyFilter**
   - **AggregateInstanceExtraSpecsFilter**
3. 設定ファイルを保存します。
4. オーバークラウドをデプロイします。

7.1.3. アグリゲートとフレーバーの設定

ホストアグリゲートを設定して、CPU ピニングを使用するインスタンスと使用しないインスタンスを、それぞれ別のホストにデプロイします。これにより、ピニングされていないインスタンスがピニングされたインスタンスのリソース要件を使用するのを防ぐことができます。

注意

NUMA トポリジーを持つインスタンスを、NUMA トポリジーを持たないインスタンスと同じホストにデプロイしないでください。

システム上で Compute CLI を使用して以下の手順を実行して、OpenStack 環境で、特定のリソースにピニングされた仮想マシンインスタンスを実行するための準備を行います。

手順

1. admin の認証情報を読み込みます。

```
source ~/keystonerc_admin
```

2. ピニング要求を受信するホスト用にアグリゲートを作成します。

```
nova aggregate-create <aggregate-name-pinned>
```

3. アグリゲートのメタデータを編集して、ピニングを有効化します。

```
nova aggregate-set-metadata <aggregate-pinned-UUID> pinned=true
```

4. その他のホスト用のアグリゲートを作成します。

```
nova aggregate-create <aggregate-name-unpinned>
```

5. このアグリゲートのメタデータを編集します。

```
nova aggregate-set-metadata <aggregate-unpinned-UUID> pinned=false
```

6. 既存のフレーバーのスペックを以下のように変更します。

```
for i in $(nova flavor-list | cut -f 2 -d ' ' | grep -o '[0-9]*'); do nova flavor-key $i set
"aggregate_instance_extra_specs:pinned"="false"; done
```

7. ピニング要求を受信するホスト用にフレーバーを作成します。

```
nova flavor-create <flavor-name-pinned> <flavor-ID> <RAM> <disk-size> <vCPUs>
```

詳細は以下のようになります。

- **<flavor-ID>: nova** で UUID を生成する場合は、**auto** に設定します。
- **<RAM>:** 必要なメモリーを MB 単位で指定します。
- **<disk-size>:** 必要なディスクサイズを GB 単位で指定します。
- **<vCPUs>:** 確保する仮想 CPU の数

8. このフレーバーの **hw:cpu_policy** のスペックは、**dedicated** に指定して、CPU ピニングを有効化するための専用のリソースを必要とするように設定し、**hw:cpu_thread_policy** の仕様を **require** に指定して、スレッドシブリングに各仮想 CPU を配置します。

```
nova flavor-key <flavor-name-pinned> set hw:cpu_policy=dedicated
nova flavor-key <flavor-name-pinned> set hw:cpu_thread_policy=require
```

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に空きのある 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。このような動作が望ましくない場合や、単にホストが SMT アーキテクチャーを備えていない場合には、hw:cpu_thread_policy の仕様を使わないようにするか、require の代わりに prefer に設定してください。デフォルトの prefer ポリシーでは、スレッドシブリングが利用可能な場合には、必ず使用されます。

9. **aggregate_instance_extra_specs:pinned** のスペックは「true」に指定して、このフレーバーをベースとするインスタンスが、アグリゲートのメタデータ内のこのスペックを使用するように設定します。

```
nova flavor-key <flavor-name-pinned> set aggregate_instance_extra_specs:pinned=true
```

10. **新規アグリゲートにホストを追加します。**

```
nova aggregate-add-host <aggregate-pinned-UUID> <host_name>
nova aggregate-add-host <aggregate-unpinned-UUID> <host_name>
```

11. **新規フレーバーを使用してインスタンスをブートします。**

```
nova boot --image <image-name> --flavor <flavor-name-pinned> <server-name>
```

12. **新規サーバーが正しく配置されたことを確認するには、以下のコマンドを実行して、その出力で OS-EXT-SRV-ATTR:hypervisor_hostname の箇所をチェックします。**

```
nova show <server-name>
```

7.2. コンピュートノードでのヒュージページの設定

インスタンスがヒュージページを要求できるようにコンピュートノードを設定します。

手順

1. **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:
  NovaReservedHugePages: ["node:0,size:2048,count:64","node:1,size:1GB,count:1"]
```

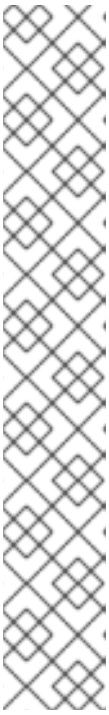
各属性についての説明は以下のとおりです。

属性	説明

size	割り当てるヒュージページのサイズ。有効な値は 2048 (2 MB 用) および 1 GB です。
count	NUMA ノードごとに OVS が使用するヒュージページの数。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。

2. **(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 cpu_model_extra_flags を設定して「pdpe1gb」を指定します。**

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::libvirt::libvirt_cpu_mode: 'custom'
    nova::compute::libvirt::libvirt_cpu_model: 'Haswell-noTSX'
    nova::compute::libvirt::libvirt_cpu_model_extra_flags: '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。**
- **cpu_mode が host-model または custom に設定されている場合に限り、cpu_model_extra_flags 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、cpu_mode に host-passthrough が使用される場合、cpu_model_extra_flags 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。**

3. **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 cpu_model_extra_flags を設定して「+pcid」を指定します。**

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::libvirt::libvirt_cpu_mode: 'custom'
    nova::compute::libvirt::libvirt_cpu_model: 'Haswell-noTSX'
    nova::compute::libvirt::libvirt_cpu_model_extra_flags: 'vmx, pdpe1gb, +pcid'
```

ヒント

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

4. 各 Compute 環境ファイルの **NovaSchedulerDefaultFilters** 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。

5. デプロイコマンドに環境ファイルを追加してオーバークラウドをデプロイし、このヒュージページの設定を適用します。

```
(undercloud) $ openstack overcloud deploy --templates \
  -e [your environment files]
  -e /home/stack/templates/<compute_environment_file>.yaml
```

7.2.1. インスタンスへのヒュージページの割り当て

インスタンスがヒュージページを使用することを指定するには、**hw:mem_page_size** 追加仕様キーを使用して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。詳しくは、「コンピュートノードでのヒュージページの設定」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. ヒュージページのフレーバーを設定します。

```
$ openstack flavor set huge_pages --property hw:mem_page_size=1GB
```

有効な **hw:mem_page_size** の値は以下のとおりです。

- **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 1GB 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**: 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。整数値のページサイズを、kB またはその他の標準単位で指定します。たとえば、4kB、2MB、2048、1GB と設定します。

3. 新しいフレーバーを使用してインスタンスを作成します。

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

検証

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。

第8章 ゲストインスタンス用の仮想 GPU の設定

ゲストインスタンスで 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。この設定により、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 対応ゲストインスタンスのスケジューリング、チューニング、およびモニタリングの制御性を向上させることができます。

OpenStack Compute で仮想 GPU を有効にするには、仮想 GPU デバイスを持つ Red Hat Enterprise Linux ゲストを要求するのに使用できるフレーバーを作成し、それらのフレーバーをコンピュートインスタンスに割り当てます。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。

OpenStack Compute サービスは各ホストで利用可能な仮想 GPU デバイスの数とサイズを追跡し、フレーバーに基づいてゲストをこれらのホストにスケジュールし、デバイスをアタッチし、使用状況を継続的に監視します。ゲストが利用できなくなると、OpenStack Compute は仮想 GPU デバイスを利用可能なプールに戻します。

8.1. サポートされる構成および制限

本項では、現在サポートされている仮想 GPU (vGPU) グラフィックカードの一覧を示すと共に、OpenStack Compute で仮想 GPU デバイスを設定する際の考慮事項および制限について説明します。

サポートされる GPU カード

サポートされる NVIDIA GPU カードの一覧については、NVIDIA の Web サイトで「Virtual GPU Software Supported Products」を参照してください。

制限および考慮事項

- 各コンピュートホストで使用できる仮想 GPU の種別は 1 つだけです。
- 各コンピュートインスタンスで使用できる仮想 GPU のリソースは 1 つだけです。
- ホスト間での仮想 GPU のライブマイグレーションはサポートされていません。
- libvirt の制限により、仮想 GPU 対応ゲスト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。
- 仮想 GPU フレーバーが設定されたインスタンスでサイズ変更およびコールドマイグレーション操作を行っても、仮想 GPU のリソースはインスタンスに自動的には再割り当てされません。インスタンスのサイズ変更または移行後に、インスタンスを手動で再ビルドし、仮想 GPU のリソースを再割り当てする必要があります。
- 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。アクセスを許可するには、ホストをホストアグリゲートに追加します。ホストアグリゲートに関する一般的な情報は、を参照してください。 「ホストアグリゲートの管理」
- 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、「Virtual GPU License Server Release Notes」の Web ページを参照してください。

8.2. NVIDIA GRID VGPU のデプロイ

本項では、コンピュートノードホストおよびゲストインスタンスに、NVIDIA 仮想 GPU (vGPU) デバイスをデプロイする方法について説明します。このエンドツーエンドプロセスには、以下の手順が含まれます。

1. GPU 対応カスタムオーバークラウドイメージのビルド
2. GPU ロール、プロファイル、およびフレーバーの準備
3. オーバークラウドの設定およびデプロイ
4. 仮想 GPU 対応カスタムゲストイメージのビルド
5. インスタンス用の仮想 GPU フレーバーの準備
6. 仮想 GPU 対応インスタンスの起動および設定

前提条件

オーバークラウドに NVIDIA GRID vGPU をデプロイする前に、環境が以下の要件を満たしていることを確認してください。

- デプロイメントは、「サポートされる構成および制限」 の説明に従って、仮想 GPU デバイスの要件を満たしている必要があります。
- アンダークラウドがデプロイされ、デフォルトのオーバークラウドイメージが Glance にアップロードされていること。
- 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 が用意されていること。NVIDIA ライセンス要件およびセルフホストサーバーのインストールに関する詳細は、「Virtual GPU License Server Release Notes」の Web ページを参照してください。

8.2.1. カスタム GPU オーバークラウドイメージのビルド

オーバークラウドの Compute イメージに NVIDIA GRID ホストドライバーをインストールし、イメージを Glance にアップロードするには、アンダークラウドで以下の手順を実施します。

1. オーバークラウドイメージをコピーし、コピーしたイメージに **gpu** の接尾辞を追加します。

```
$ cp overcloud-full.qcow2 overcloud-full-gpu.qcow2
```

2. YUM から ISO イメージ生成器ツールをインストールします。

```
$ sudo yum install genisoimage -y
```

3. 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ます。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。

注記

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。

4. ドライバー RPM パッケージから ISO イメージを作成し、イメージを nvidia-guest ディレクトリーに保存します。この ISO イメージを使用して、以下の手順でコンピュートノードにドライバーをインストールします。

```
$ genisoimage -o nvidia-guest.iso -R -J -V NVIDIA nvidia-guest/
I: -input-charset not specified, using utf-8 (detected in locale settings)
  9.06% done, estimate finish Wed Oct 31 11:24:46 2018
 18.08% done, estimate finish Wed Oct 31 11:24:46 2018
 27.14% done, estimate finish Wed Oct 31 11:24:46 2018
 36.17% done, estimate finish Wed Oct 31 11:24:46 2018
 45.22% done, estimate finish Wed Oct 31 11:24:46 2018
 54.25% done, estimate finish Wed Oct 31 11:24:46 2018
 63.31% done, estimate finish Wed Oct 31 11:24:46 2018
 72.34% done, estimate finish Wed Oct 31 11:24:46 2018
 81.39% done, estimate finish Wed Oct 31 11:24:46 2018
 90.42% done, estimate finish Wed Oct 31 11:24:46 2018
 99.48% done, estimate finish Wed Oct 31 11:24:46 2018
Total translation table size: 0
Total rockridge attributes bytes: 358
Total directory bytes: 0
Path table size(bytes): 10
Max brk space used 0
55297 extents written (108 MB)
```

5. コンピュートノード用のドライバーインストールスクリプトを作成します。このスクリプトにより、スクリプトを実行する各コンピュートノードに NVIDIA GRID ホストドライバーがインストールされます。この例では、スクリプトは install_nvidia.sh という名前です。

```
#/bin/bash

# NVIDIA GRID package
mkdir /tmp/mount
mount LABEL=NVIDIA /tmp/mount
rpm -ivh /tmp/mount/NVIDIA-vGPU-rhel-8.0-430.27.x86_64.rpm
```

6. 生成した ISO イメージをアタッチし、作成したドライバーインストールスクリプトを実行して、オーバークラウドイメージをカスタマイズします。以下に例を示します。

```
$ virt-customize --attach nvidia-packages.iso -a overcloud-full-gpu.qcow2  -v --run
install_nvidia.sh
[   0.0] Examining the guest ...
libguestfs: launch: program=virt-customize
libguestfs: launch: version=1.36.10rhel=8,release=6.el8_5.2,libvirt
libguestfs: launch: backend registered: unix
libguestfs: launch: backend registered: uml
libguestfs: launch: backend registered: libvirt
```

7. SELinux でカスタマイズしたイメージのラベルを付け直します。

```
$ virt-customize -a overcloud-full-gpu.qcow2 --selinux-relabel
[   0.0] Examining the guest ...
[   2.2] Setting a random seed
[   2.2] SELinux relabelling
[  27.4] Finishing off
```

8. **Glance アップロード用にカスタムイメージファイルを準備します。以下に例を示します。**

```
$ mkdir /var/image/x86_64/image
$ guestmount -a overcloud-full-gpu.qcow2 -i --ro image
$ cp image/boot/vmlinuz-3.10.0-862.14.4.el8.x86_64 ./overcloud-full-gpu.vmlinuz
$ cp image/boot/initramfs-3.10.0-862.14.4.el8.x86_64.img ./overcloud-full-gpu.initrd
```

9. **アンダークラウドから、Glance にカスタムイメージをアップロードします。**

```
(undercloud) $ openstack overcloud image upload --update-existing --os-image-name
overcloud-full-gpu.qcow2
```

8.2.2. 仮想 GPU ロール、プロファイル、およびフレーバーの設定

カスタム GPU オーバークラウドイメージをビルドしたら、GPU 対応オーバークラウドのデプロイメント用にコンピュートノードを準備します。本項では、GPU 対応コンピュートノード用にロール、プロファイル、およびフレーバーを設定する方法について説明します。

1. **ファイル /home/stack/templates/roles/Compute.yaml を /home/stack/templates/roles/ComputeGPU.yaml にコピーし、ファイルの以下のセクションを編集して、新たな ComputeGPU ロールファイルを作成します。**

表8.1 ComputeGPU ロールファイルの編集

セクション	現在の値	新しい値
ロールのコメント	Role: Compute	Role: ComputeGpu
name (ロール名)	Compute	ComputeGpu
説明	Basic Compute Node role	GPU Compute Node role
CountDefault	1	0
ImageDefault	overcloud-full	overcloud-gpu
HostnameFormatDefault	-compute-	-computegpu-
deprecated_nic_config_name	compute.yaml	compute-gpu.yaml

2. **Controller、Compute、および ComputeGpu ロールが含まれる gpu_roles_data.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) [stack@director templates]$ openstack overcloud roles generate -o
/home/stack/templates/gpu_roles_data.yaml Controller Compute ComputeGpu
```

ComputeGpu ロールの詳細の例を以下に示します。

```
###############################################################################
# Role: ComputeGpu                                                            #
###############################################################################
```

```
- name: ComputeGpu
  description: |
    GPU Compute Node role
  CountDefault: 1
  ImageDefault: overcloud-gpu
  networks:
    - InternalApi
    - Tenant
    - Storage
  HostnameFormatDefault: '%stackname%-computegpu-%index%'
  RoleParametersDefault:
    TunedProfileName: "virtual-host"
  # Deprecated & backward-compatible values (FIXME: Make parameters consistent)
  # Set uses_deprecated_params to True if any deprecated params are used.
  uses_deprecated_params: True
  deprecated_param_image: 'NovaImage'
  deprecated_param_extraconfig: 'NovaComputeExtraConfig'
  deprecated_param_metadata: 'NovaComputeServerMetadata'
  deprecated_param_scheduler_hints: 'NovaComputeSchedulerHints'
  deprecated_param_ips: 'NovaComputeIPs'
  deprecated_server_resource_name: 'NovaCompute'
  deprecated_nic_config_name: 'compute-gpu.yaml'
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::ComputeCeilometerAgent
    - OS::TripleO::Services::ComputeNeutronCorePlugin
    - OS::TripleO::Services::ComputeNeutronL3Agent
    - OS::TripleO::Services::ComputeNeutronMetadataAgent
    - OS::TripleO::Services::ComputeNeutronOvsAgent
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MetricsQdr
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronBgpVpnBagpipe
    - OS::TripleO::Services::NeutronLinuxbridgeAgent
    - OS::TripleO::Services::NeutronVppAgent
    - OS::TripleO::Services::NovaCompute
    - OS::TripleO::Services::NovaLibvirt
    - OS::TripleO::Services::NovaLibvirtGuests
    - OS::TripleO::Services::NovaMigrationTarget
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::OpenDaylightOvs
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::Securetty
```

```
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::SkydiveAgent
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    - OS::TripleO::Services::Vpp
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::OVNMetadataAgent
    - OS::TripleO::Services::Ptp
```

3. **仮想 GPU 負荷用に指定するノードをタグ付けするための、compute-vgpu-nvidia フレーバーを作成します。**

```
(undercloud) [stack@director templates]$ openstack flavor create --id auto --ram 6144 --disk
40 --vcpus 4 compute-vgpu-nvidia
+----------------------------+--------------------------------------+
| Field                      | Value                                |
+----------------------------+--------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                |
| OS-FLV-EXT-DATA:ephemeral  | 0                                    |
| disk                       | 40                                   |
| id                         | 9cb47954-be00-47c6-a57f-44db35be3e69 |
| name                       | compute-vgpu-nvidia                  |
| os-flavor-access:is_public | True                                 |
| properties                 |                                      |
| ram                        | 6144                                 |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 4                                    |
+----------------------------+--------------------------------------+
```

4. **GPU 負荷用に指定するそれぞれのノードを、compute-vgpu-nvidia プロファイルでタグ付けします。**

```
(undercloud) [stack@director templates]$ openstack baremetal node set --property
capabilities='profile:compute-vgpu-nvidia,boot_option:local' 9d07a673-b6bf-4a20-a538-
3b05e8fa2c13
```

5. **オーバークラウドを登録し、ノードで標準のハードウェアイントロスペクションを実行します。**

8.2.3. 設定ファイルの準備とオーバークラウドのデプロイ

仮想 GPU 用のオーバークラウドを準備したら、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、設定テンプレートを準備します。

NVIDIA デバイスの仮想 GPU 種別の設定

物理 GPU デバイスに対応する仮想 GPU の種別を判断するには、別のマシンから利用可能なデバイス種別を確認する必要があります。任意の一時的な Red Hat Enterprise Linux の未使用コンピュートノードから、この手順を実施することができます。その後、ノードを削除します。これらのステップを実施するために、オーバークラウドをデプロイする必要はありません。

1. Red Hat Enterprise Linux と NVIDIA GRID ドライバーを 1 つの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。NVIDIA GRID ドライバーのインストールについての詳細は、「カスタム GPU オーバークラウドイメージのビルド」を参照してください。

2. コンピュートノードで、有効にする物理 GPU デバイスに対応する仮想 GPU の種別を確認します。Libvirt の場合、仮想 GPU は仲介デバイスまたは **mdev** 種別のデバイスとして表示されます。サポートされている **mdev** デバイスを検出するには、以下のコマンドを実行します。

```
[root@overcloud-computegpu-0 ~]# ls
/sys/class/mdev_bus/0000\:06\:00.0/mdev_supported_types/
nvidia-11  nvidia-12  nvidia-13  nvidia-14  nvidia-15  nvidia-16  nvidia-17  nvidia-18  nvidia-19
nvidia-20  nvidia-21  nvidia-210  nvidia-22

[root@overcloud-computegpu-0 ~]# cat
/sys/class/mdev_bus/0000\:06\:00.0/mdev_supported_types/nvidia-18/description
num_heads=4, frl_config=60, framebuffer=2048M, max_resolution=4096x2160,
max_instance=4
```

設定テンプレートの準備

1. **compute-gpu.yaml** ファイルを **network-environment.yaml** ファイルに追加します。以下に例を示します。

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::ComputeGpu::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute-gpu.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
  #OS::TripleO::AllNodes::Validation: OS::Heat::None
```

2. **OvercloudComputeGpuFlavor** フレーバーを **node-info.yaml** ファイルに追加します。以下に例を示します。

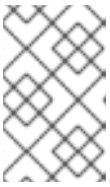
```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeFlavor: compute
  OvercloudComputeGpuFlavor: compute-vgpu-nvidia
  ControllerCount: 1
  ComputeCount: 0
  ComputeGpuCount: 1
  NtpServer: `NTP_SERVER_URL`
  NeutronNetworkType: vxlan,vlan
  NeutronTunnelTypes: vxlan
```

NTP_SERVER_URL 変数を実際の NTP サーバーのアドレスに置き換えます。

3. GPU デバイス用に取得した仮想 GPU の種別で **gpu.yaml** ファイルを作成します。以下に例を示します。

```
parameter_defaults:
  ComputeGpuExtraConfig:
    nova::compute::vgpu::enabled_vgpu_types:
      - nvidia-18
```

-



注記

サポートされる仮想 GPU の種別は、物理 GPU 1つにつき 1つだけです。このプロパティーで複数の仮想 GPU 種別を指定した場合は、最初の種別だけが使用されます。

オーバークラウドのデプロイ

カスタム GPU イメージおよび準備した設定テンプレートを指定して、**overcloud deploy** コマンドを実行します。

```
$ openstack overcloud deploy -r /home/stack/templates/nvidia/gpu_roles_data.yaml  -e
/home/stack/templates/nvidia/gpu.yaml
```

8.2.4. カスタム GPU ゲストイメージのビルド

GPU 対応コンピュートノードと共にオーバークラウドをデプロイしたら、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ビルドします。

NVIDIA GRID ライセンスファイルの作成

オーバークラウドホストで、NVIDIA GRID のライセンス情報が含まれる gridd.conf ファイルを作成します。前のステップでインストールしたセルフホスト NVIDIA GRID ライセンスサーバーからのライセンスサーバー情報を使用します。以下に例を示します。

```
# /etc/nvidia/gridd.conf.template - Configuration file for NVIDIA Grid Daemon

# This is a template for the configuration file for NVIDIA Grid Daemon.
# For details on the file format, please refer to the nvidia-gridd(1)
# man page.

# Description: Set License Server Address
# Data type: string
# Format:  "<address>"
ServerAddress=[NVIDIA_LICENSE_SERVER_URL]

# Description: Set License Server port number
# Data type: integer
# Format:  <port>, default is 7070
ServerPort=[PORT_NUMBER]

# Description: Set Backup License Server Address
# Data type: string
# Format:  "<address>"
#BackupServerAddress=

# Description: Set Backup License Server port number
# Data type: integer
# Format:  <port>, default is 7070
#BackupServerPort=

# Description: Set Feature to be enabled
# Data type: integer
```

```
# Possible values:
#    0 => for unlicensed state
#    1 => for GRID vGPU
#    2 => for Quadro Virtual Datacenter Workstation
FeatureType=[TYPE_ID]

# Description: Parameter to enable or disable Grid Licensing tab in nvidia-settings
# Data type: boolean
# Possible values: TRUE or FALSE, default is FALSE
EnableUI=TRUE

# Description: Set license borrow period in minutes
# Data type: integer
# Possible values: 10 to 10080 mins(7 days), default is 1440 mins(1 day)
#LicenseInterval=1440

# Description: Set license linger period in minutes
# Data type: integer
# Possible values: 0 to 10080 mins(7 days), default is 0 mins
#LingerInterval=10
```

ゲストイメージおよび NVIDIA GRID ゲストドライバーの準備

1. NVIDIA の Web サイトから、GPU デバイスに対応する NVIDIA GRID ゲストドライバー RPM パッケージをダウンロードします。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。

注記

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。

2. ドライバー RPM パッケージから ISO イメージを作成します。この ISO イメージを使用して、以下の手順でコンピュートノードにドライバーをインストールします。

```
[root@virtlab607 guest]# genisoimage -o nvidia-guest.iso -R -J -V NVIDIA nvidia-guest/
I: -input-charset not specified, using utf-8 (detected in locale settings)
  9.06% done, estimate finish Wed Oct 31 10:59:50 2018
 18.08% done, estimate finish Wed Oct 31 10:59:50 2018
 27.14% done, estimate finish Wed Oct 31 10:59:50 2018
 36.17% done, estimate finish Wed Oct 31 10:59:50 2018
 45.22% done, estimate finish Wed Oct 31 10:59:50 2018
 54.25% done, estimate finish Wed Oct 31 10:59:50 2018
 63.31% done, estimate finish Wed Oct 31 10:59:50 2018
 72.34% done, estimate finish Wed Oct 31 10:59:50 2018
 81.39% done, estimate finish Wed Oct 31 10:59:50 2018
 90.42% done, estimate finish Wed Oct 31 10:59:50 2018
 99.48% done, estimate finish Wed Oct 31 10:59:50 2018
Total translation table size: 0
Total rockridge attributes bytes: 358
Total directory bytes: 0
Path table size(bytes): 10
Max brk space used 0
55297 extents written (108 MB)
```

3. GPU インスタンス用にカスタマイズするゲストイメージをコピーします。以下に例を示します。

```
[root@virtlab607 guest]# cp rhel-server-8.0-update-4-x86_64-kvm.qcow2 rhel-server-8.0-update-4-x86_64-kvm-gpu.qcow2
```

カスタマイズスクリプトの作成および実行

デフォルトでは、GPU 負荷用に指定する各インスタンスに NVIDIA GRID ドライバーをインストールする必要があります。そのためには、ゲストイメージを変更し、再起動してからゲストドライバーをインストールします。ゲストインスタンスに対するこのプロセスを自動化するスクリプトを作成することができます。

1. nvidia-prepare-guest.sh という名前のスクリプトを作成します。このスクリプトにより、必要なリポジトリーを有効にし、インスタンスを最新のカーネルに更新し、NVIDIA GRID ゲストドライバーをインストールし、インスタンスに gridd.conf ライセンスファイルをアタッチします。

```
#/bin/bash

# Add build tooling
subscription-manager register --username [USERNAME] --password [PASSWORD]
subscription-manager attach --pool=8a85f98c651a88990165399d8eea03e7
subscription-manager repos --disable=*
subscription-manager repos --enable=rhel-8-server-rpms
dnf upgrade -y
dnf install -y gcc make kernel-devel cpp glibc-devel glibc-headers kernel-headers libmpc mpfr elfutils-libelf-devel

# NVIDIA GRID guest script
mkdir /tmp/mount
mount LABEL=NVIDIA /tmp/mount
/bin/sh /tmp/mount/NVIDIA-Linux-x86_64-430.24-grid.run

mkdir -p /etc/nvidia
cp /tmp/mount/gridd.conf /etc/nvidia
```

2. 前のステップでコピーしたゲストイメージで、スクリプトを実行します。以下に例を示します。

```
$ virt-customize --attach nvidia-guest.iso -a rhel-server-8.0-update-4-x86_64-kvm-gpu.qcow2 -v --run nvidia-prepare-guest.sh
```

3. カスタムゲストイメージを Glance にアップロードします。

```
(overcloud) [stack@director ~]$ openstack image create rhelgpu --file /var/images/x86_64/rhel-server-8.0-update-4-x86_64-kvm-gpu.qcow2 --disk-format qcow2 --container-format bare --public
```

8.2.5. インスタンスの仮想 GPU プロファイルの作成

カスタムゲストイメージをビルドしたら、GPU フレーバーを作成し、そのフレーバーに仮想 GPU のリソースを割り当てます。その後このフレーバーでインスタンスを起動すると、各インスタンスで仮想 GPU のリソースが利用可能になります。

注記

各インスタンスに割り当てられる仮想 GPU のリソースは 1 つだけです。

1. **GPU 負荷用に指定する各インスタンスをタグ付けする NVIDIA GPU フレーバーを作成します。以下に例を示します。**

```
(overcloud) [stack@virtlab-director2 ~]$ openstack flavor create --vcpus 6 --ram 8192 --disk
100 m1.small-gpu
+----------------------------+-------------------------------------+
| Field                      | Value                               |
+----------------------------+-------------------------------------+
| OS-FLV-DISABLED:disabled   | False                               |
| OS-FLV-EXT-DATA:ephemeral  | 0                                   |
| disk                       | 100                                 |
| id                         | a27b14dd-c42d-4084-9b6a-225555876f68 |
| name                       | m1.small-gpu                        |
| os-flavor-access:is_public | True                                |
| properties                 |                                     |
| ram                        | 8192                                |
| rxtx_factor                | 1.0                                 |
| swap                       |                                     |
| vcpus                      | 6                                   |
+----------------------------+-------------------------------------+
```

2. **作成したフレーバーに仮想 GPU のリソースを割り当てます。現時点では、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。**

```
(overcloud) [stack@virtlab-director2 ~]$ openstack flavor set m1.small-gpu --property
"resources:VGPU=1"

(overcloud) [stack@virtlab-director2 ~]$ openstack flavor show m1.small-gpu
+----------------------------+-------------------------------------+
| Field                      | Value                               |
+----------------------------+-------------------------------------+
| OS-FLV-DISABLED:disabled   | False                               |
| OS-FLV-EXT-DATA:ephemeral  | 0                                   |
| access_project_ids         | None                                |
| disk                       | 100                                 |
| id                         | a27b14dd-c42d-4084-9b6a-225555876f68 |
| name                       | m1.small-gpu                        |
| os-flavor-access:is_public | True                                |
| properties                 | resources:VGPU='1'                  |
| ram                        | 8192                                |
| rxtx_factor                | 1.0                                 |
| swap                       |                                     |
| vcpus                      | 6                                   |
+----------------------------+-------------------------------------+
```

8.2.6. 仮想 GPU インスタンスの起動およびテスト

ゲストイメージを準備して GPU フレーバーを作成したら、GPU 対応インスタンスを起動し、「カスタム GPU ゲストイメージのビルド」でカスタムイメージにアタッチした ISO から NVIDIA ゲストドライバーをインストールします。

1. 「インスタンスの仮想 GPU プロファイルの作成」で作成した GPU フレーバーで新規インスタンスを起動します。以下に例を示します。

```
(overcloud) [stack@virtlab-director2 ~]$ openstack server create --flavor m1.small-gpu --
image rhelgpu --security-group web --nic net-id=internal0 --key-name lambda instance0
```

2. インスタンスにログインし、NVIDIA GRID ドライバーをインストールします。インストーラーの正確な名前は、ゲストイメージにアタッチしたファイルで確認することができます。以下に例を示します。

```
[root@instance0 tmp]# sh NVIDIA-Linux-x86_64-430.24-grid.run
```

3. NVIDIA GRID デーモンのステータスを確認します。

```
[root@instance0 nvidia]# systemctl status nvidia-gridd.service
● nvidia-gridd.service - NVIDIA Grid Daemon
   Loaded: loaded (/usr/lib/systemd/system/nvidia-gridd.service; enabled; vendor preset:
disabled)
   Active: active (running) since Wed 2018-10-31 20:00:41 EDT; 15s ago
  Process: 18143 ExecStopPost=/bin/rm -rf /var/run/nvidia-gridd (code=exited,
status=0/SUCCESS)
  Process: 18145 ExecStart=/usr/bin/nvidia-gridd (code=exited, status=0/SUCCESS)
 Main PID: 18146 (nvidia-gridd)
   CGroup: /system.slice/nvidia-gridd.service
           └─18146 /usr/bin/nvidia-gridd

Oct 31 20:00:41 instance0 systemd[1]: Stopped NVIDIA Grid Daemon.
Oct 31 20:00:41 instance0 systemd[1]: Starting NVIDIA Grid Daemon...
Oct 31 20:00:41 instance0 systemd[1]: Started NVIDIA Grid Daemon.
Oct 31 20:00:41 instance0 nvidia-gridd[18146]: Started (18146)
Oct 31 20:00:41 instance0 nvidia-gridd[18146]: Ignore Service Provider Licensing.
Oct 31 20:00:41 instance0 nvidia-gridd[18146]: Calling load_byte_array(tra)
Oct 31 20:00:42 instance0 nvidia-gridd[18146]: Acquiring license for GRID vGPU Edition.
Oct 31 20:00:42 instance0 nvidia-gridd[18146]: Calling load_byte_array(tra)
Oct 31 20:00:45 instance0 nvidia-gridd[18146]: License acquired successfully. (Info:
http://dhcp158-15.virt.lab.eng.bos.redhat.com:7070/request; GRID-Virtual-WS,2.0)
```

第9章 REAL-TIME COMPUTE の設定

一部のユースケースでは、低レイテンシーのポリシーを順守しリアルタイム処理を実行するために、コンピュートノードにインスタンスが必要となります。Real-time コンピュートノードには、リアルタイム対応のカーネル、特定の仮想化モジュール、および最適化されたデプロイメントパラメーターが設定され、リアルタイム処理の要求に対応してレイテンシーを最小限に抑えます。

リアルタイムコンピュートを有効にするプロセスは、以下のステップで構成されます。

- コンピュートノードの BIOS 設定の定義
- real-time カーネルおよび Real-Time KVM (RT-KVM) カーネルモジュールを持つ real-time のイメージのビルド
- コンピュートノードへの **ComputeRealTime** ロールの割り当て

NFV 負荷に対して Real-time Compute をデプロイするユースケースの例については、『ネットワーク機能仮想化(NFV)のプランニングおよび設定ガイド』の「例： ODL および VXLAN トンネリングを使用する OVS-DPDK の設定」セクションを参照してください。

9.1. REAL-TIME 用コンピュートノードの準備

注記

Real-time コンピュートノードは、Red Hat Enterprise Linux バージョン 7.5 以降でのみサポートされます。

オーバークラウドに Real-time Compute をデプロイするには、Red Hat Enterprise Linux Real-Time KVM (RT-KVM) を有効にし、real-time をサポートするように BIOS を設定し、real-time のイメージをビルドする必要があります。

前提条件

- RT-KVM コンピュートノードには、Red Hat 認定済みサーバーを使用する必要があります。詳しくは、Red Hat Enterprise Linux for Real Time 7 用認定サーバーを参照してください。
- real-time のイメージをビルドするには、RT-KVM 用の **rhel-8-for-x86_64-nfv-rpms** リポジトリーを有効にする必要があります。

注記

このリポジトリーにアクセスするためには、Red Hat OpenStack Platform for Real Time に対する別のサブスクリプションが必要です。リポジトリーの管理およびアンダークラウド用のサブスクリプションに関する詳細は、『director のインストールと使用方法』の「アンダークラウドの登録と更新」セクションを参照してください。

リポジトリーからインストールされるパッケージを確認するには、以下のコマンドを実行します。

```
$ dnf repo-pkgs rhel-8-for-x86_64-nfv-rpms list
Loaded plugins: product-id, search-disabled-repos, subscription-manager
Available Packages
```

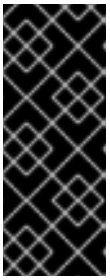
```
kernel-rt.x86_64                4.18.0-80.7.1.rt9.153.el8_0          rhel-8-for-x86_64-nfv-rpms
kernel-rt-debug.x86_64          4.18.0-80.7.1.rt9.153.el8_0          rhel-8-for-x86_64-nfv-
rpms
kernel-rt-debug-devel.x86_64    4.18.0-80.7.1.rt9.153.el8_0          rhel-8-for-x86_64-nfv-
rpms
kernel-rt-debug-kvm.x86_64      4.18.0-80.7.1.rt9.153.el8_0          rhel-8-for-x86_64-nfv-
rpms
kernel-rt-devel.x86_64          4.18.0-80.7.1.rt9.153.el8_0          rhel-8-for-x86_64-nfv-
rpms
kernel-rt-doc.noarch            4.18.0-80.7.1.rt9.153.el8_0          rhel-8-for-x86_64-nfv-rpms
kernel-rt-kvm.x86_64            4.18.0-80.7.1.rt9.153.el8_0          rhel-8-for-x86_64-nfv-
rpms
[ output omitted…]
```

real-time のイメージのビルド

Real-time コンピュートノード用のオーバークラウドイメージをビルドするには、以下のステップを実行します。

1. **アンダークラウドに libguestfs-tools パッケージをインストールして、virt-customize ツールを取得します。**

```
(undercloud) [stack@undercloud-0 ~]$ sudo dnf install libguestfs-tools
```



重要

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

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

2. **イメージを抽出します。**

```
(undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-director-images/overcloud-
full.tar
(undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-director-images/ironic-python-
agent.tar
```

3. **デフォルトのイメージをコピーします。**

```
(undercloud) [stack@undercloud-0 ~]$ cp overcloud-full.qcow2 overcloud-realtime-
compute.qcow2
```

4. **イメージを登録して、必要なサブスクリプションを設定します。**

```
(undercloud) [stack@undercloud-0 ~]$  virt-customize -a overcloud-realtime-compute.qcow2
--run-command 'subscription-manager register --username=[username] --password=
[password]'
[  0.0] Examining the guest ...
[ 10.0] Setting a random seed
```

```
[ 10.0] Running: subscription-manager register --username=[username] --password=
[password]
[ 24.0] Finishing off
```

username および password の値を、ご自分の Red Hat カスタマーアカウント情報に置き換えてください。Real-time オーバークラウドイメージのビルドに関する一般的な情報は、ナレッジベースの記事「Modifying the Red Hat Enterprise Linux OpenStack Platform Overcloud Image with virt-customize」を参照してください。

5. **以下の例に示すように、Red Hat OpenStack Platform for Real Timeサブスクリプションの SKU を探します。SKU は、同じアカウントおよび認証情報を使用してすでに Red Hat サブスクリプションマネージャーに登録済みのシステムに置かれている場合があります。以下に例を示します。**

```
$ sudo subscription-manager list
```

6. **Red Hat OpenStack Platform for Real Timeサブスクリプションをイメージにアタッチします。**

```
(undercloud) [stack@undercloud-0 ~]$  virt-customize -a overcloud-realtime-compute.qcow2
--run-command 'subscription-manager attach --pool [subscription-pool]'
```

7. **イメージ上で rt を設定するためのスクリプトを作成します。**

```
(undercloud) [stack@undercloud-0 ~]$ cat rt.sh
  #!/bin/bash

  set -eux

  subscription-manager repos --enable=[REPO_ID]
  dnf -v -y --setopt=protected_packages= erase kernel.$(uname -m)
  dnf -v -y install kernel-rt kernel-rt-kvm tuned-profiles-nfv-host

  # END OF SCRIPT
```

8. **リアルタイムイメージを設定するスクリプトを実行します。**

```
(undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-realtime-compute.qcow2 -
v --run rt.sh 2>&1 | tee virt-customize.log
```

9. **SELinux の再ラベル付けをします。**

```
(undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-realtime-compute.qcow2 -
-selinux-relabel
```

10. **vmlinuz および initrd を抽出します。以下は例になります。**

```
(undercloud) [stack@undercloud-0 ~]$ mkdir image
(undercloud) [stack@undercloud-0 ~]$ guestmount -a overcloud-realtime-compute.qcow2 -i -
-ro image
(undercloud) [stack@undercloud-0 ~]$ cp image/boot/vmlinuz-4.18.0-
80.7.1.rt9.153.el8_0.x86_64 ./overcloud-realtime-compute.vmlinuz
```

```
(undercloud) [stack@undercloud-0 ~]$ cp image/boot/initramfs-4.18.0-
80.7.1.rt9.153.el8_0.x86_64.img ./overcloud-realtime-compute.initrd
(undercloud) [stack@undercloud-0 ~]$ guestunmount image
```

注記

vmlinuz および initramfs のファイル名に含まれるソフトウェアバージョンは、カーネルバージョンによって異なります。

11. **イメージをアップロードします。**

```
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud image upload --update-existing -
-os-image-name overcloud-realtime-compute.qcow2
```

これで、選択したコンピュートノード上の ComputeRealTime コンポーザブルロールで使用することのできる real-time イメージの準備ができました。

Real-time コンピュートノード上での BIOS 設定の変更

Real-time コンピュートノードのレイテンシーを短縮するには、コンピュートノードの BIOS 設定を変更する必要があります。コンピュートノードの BIOS 設定で、以下のコンポーネントの全オプションを無効にする必要があります。

- **電源管理**
- **ハイパースレッディング**
- **CPU のスリープ状態**
- **論理プロセッサー**

これらの設定に関する説明と、無効化の影響については、「BIOS パラメーターの設定」を参照してください。BIOS 設定の変更方法に関する詳しい情報は、ハードウェアの製造会社のドキュメントを参照してください。

9.2. REAL-TIME COMPUTE ロールのデプロイメント

Red Hat OpenStack Platform director では、ComputeRealTime ロールのテンプレートが利用可能です。これを使用して、Real-time コンピュートノードをデプロイすることができます。ただし、コンピュートノードを real-time 用に指定するためには、追加のステップを実施する必要があります。

1. **/usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-example.yaml ファイルをベースに、ComputeRealTime ロールのパラメーターを設定する compute-real-time.yaml 環境ファイルを作成します。**

```
cp /usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-
example.yaml /home/stack/templates/compute-real-time.yaml
```

ファイルには、以下のパラメーター値を含める必要があります。

- **IsolCpusList および NovaVcpuPinSet: リアルタイム負荷のために確保する分離 CPU コアおよび仮想 CPU のピニングの一覧。この値は、Real-time コンピュートノードの CPU ハードウェアにより異なります。**
- **KernelArgs: Real-time コンピュートノードのカーネルに渡す引数。たとえ**

ば、default_hugepagesz=1G hugepagesz=1G hugepages=<number_of_1G_pages_to_reserve> hugepagesz=2M hugepages=<number_of_2M_pages> を使用して、複数のサイズのヒュージページを持つゲストのメモリー要求を定義することができます。この例では、デフォルトのサイズは 1GB ですが、2 MB のヒュージページを確保することもできます。

2. **ComputeRealTime ロールをロールデータのファイルに追加し、ファイルを再生成します。以下は例になります。**

```
$ openstack overcloud roles generate -o /home/stack/templates/rt_roles_data.yaml Controller
Compute ComputeRealTime
```

このコマンドにより、以下の例のような内容で ComputeRealTime ロールが生成され、ImageDefault オプションに overcloud-realtime-compute が設定されます。

```
- name: ComputeRealTime
  description: |
    Compute role that is optimized for real-time behaviour. When using this role
    it is mandatory that an overcloud-realtime-compute image is available and
    the role specific parameters IsolCpusList and NovaVcpuPinSet are set
    accordingly to the hardware of the real-time compute nodes.
  CountDefault: 1
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Tenant:
      subnet: tenant_subnet
    Storage:
      subnet: storage_subnet
  HostnameFormatDefault: '%stackname%-computerealtime-%index%'
  ImageDefault: overcloud-realtime-compute
  RoleParametersDefault:
    TunedProfileName: "realtime-virtual-host"
    KernelArgs: ""      # these must be set in an environment file or similar
    IsolCpusList: ""    # according to the hardware of real-time nodes
    NovaVcpuPinSet: ""  #
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::BootParams
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::ComputeCeilometerAgent
    - OS::TripleO::Services::ComputeNeutronCorePlugin
    - OS::TripleO::Services::ComputeNeutronL3Agent
    - OS::TripleO::Services::ComputeNeutronMetadataAgent
    - OS::TripleO::Services::ComputeNeutronOvsAgent
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::IpaClient
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Kernel
```

```
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MetricsQdr
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronBgpVpnBagpipe
    - OS::TripleO::Services::NeutronLinuxbridgeAgent
    - OS::TripleO::Services::NeutronVppAgent
    - OS::TripleO::Services::NovaCompute
    - OS::TripleO::Services::NovaLibvirt
    - OS::TripleO::Services::NovaLibvirtGuests
    - OS::TripleO::Services::NovaMigrationTarget
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::OpenDaylightOvs
    - OS::TripleO::Services::Podman
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::SkydiveAgent
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timesync
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Vpp
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::OVNMetadataAgent
```

カスタムロールおよび roles-data.yaml に関する一般的な情報は、「ロール」セクションを参照してください。

3. **リアルタイム負荷用に指定するノードをタグ付けするために、compute-realtime フレーバーを作成します。以下は例になります。**

```
$ source ~/stackrc
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 compute-realtime
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property "capabilities:profile"="compute-realtime"
compute-realtime
```

4. **リアルタイム負荷用に指定するそれぞれのノードを、compute-realtime プロファイルでタグ付けします。**

```
$ openstack baremetal node set --property capabilities='profile:compute-
realtime,boot_option:local' <NODE UUID>
```

5. **以下の内容の環境ファイルを作成して、ComputeRealTime ロールを compute-realtime フレーバーにマッピングします。**

```
parameter_defaults:
  OvercloudComputeRealTimeFlavor: compute-realtime
```

6. **-e オプションを使用して openstack overcloud deploy コマンドを実行し、新しいロールファイルと共に作成したすべての環境ファイルを指定します。以下に例を示します。**

```
$ openstack overcloud deploy -r /home/stack/templates/rt~/my_roles_data.yaml  -e
home/stack/templates/compute-real-time.yaml <FLAVOR_ENV_FILE>
```

9.3. デプロイメントの例およびテストシナリオ

以下の手順の例では、単純な単一ノードのデプロイメントを使用して、環境変数およびその他の補助設定が正しく定義されていることをテストします。実際の実行結果は、クラウドにデプロイするノードおよびゲストの数により異なります。

1. **以下のパラメーターで compute-real-time.yaml ファイルを作成します。**

```
parameter_defaults:
  ComputeRealTimeParameters:
    IsolCpusList: "1"
    NovaVcpuPinSet: "1"
    KernelArgs: "default_hugepagesz=1G hugepagesz=1G hugepages=16"
```

2. **ComputeRealTime ロールを設定して新たな roles_data.yaml ファイルを作成します。**

```
$ openstack overcloud roles generate -o ~/rt_roles_data.yaml Controller ComputeRealTime
```

このコマンドにより、コントローラーノードおよびリアルタイムコンピュートノードが 1 台ずつデプロイされます。

3. **Real-time コンピュートノードにログインし、以下のパラメーターを確認します。<...> は、compute-real-time.yaml からの該当するパラメーターの値に置き換えてください。**

```
[root@overcloud-computerealtime-0 ~]# uname -a
Linux overcloud-computerealtime-0 4.18.0-80.7.1.rt9.153.el8_0.x86_64 #1 SMP PREEMPT
RT Wed Dec 13 13:37:53 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
[root@overcloud-computerealtime-0 ~]# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.18.0-80.7.1.rt9.153.el8_0.x86_64 root=UUID=45ae42d0-
58e7-44fe-b5b1-993fe97b760f ro console=tty0 crashkernel=auto console=ttyS0,115200
default_hugepagesz=1G hugepagesz=1G hugepages=16
[root@overcloud-computerealtime-0 ~]# tuned-adm active
Current active profile: realtime-virtual-host
[root@overcloud-computerealtime-0 ~]# grep ^isolated_cores /etc/tuned/realtime-virtual-host-
variables.conf
isolated_cores=<IsolCpusList>
[root@overcloud-computerealtime-0 ~]# cat /usr/lib/tuned/realtime-virtual-
host/lapic_timer_adv_ns
X (X != 0)
[root@overcloud-computerealtime-0 ~]# cat
/sys/module/kvm/parameters/lapic_timer_advance_ns
X (X != 0)
[root@overcloud-computerealtime-0 ~]# cat
/sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
X (X != 0)
[root@overcloud-computerealtime-0 ~]# grep ^vcpu_pin_set /var/lib/config-data/puppet-
generated/nova_libvirt/etc/nova/nova.conf
vcpu_pin_set=<NovaVcpuPinSet>
```

9.4. REAL-TIME インスタンスの起動およびチューニング

Real-time コンピュートノードをデプロイして設定したら、それらのノードで real-time インスタンスを起動することができます。CPU ピニング、NUMA トポロジーフィルター、およびヒュージページを使用して、これらの real-time インスタンスをさらに設定することができます。

real-time インスタンスの起動

1. 「Real-time Compute ロールのデプロイメント」セクションで説明したように、オーバークラウド上に **compute-realtime** フレーバーが存在する状態にしてください。
2. real-time インスタンスを起動します。

```
# openstack server create  --image <rhel> --flavor r1.small --nic net-id=<dpdk-net> test-rt
```

3. オプションとして、割り当てられたエミュレータースレッドをインスタンスが使用していることを確認します。

```
# virsh dumpxml <instance-id> | grep vcpu -A1
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='3'/>
  <vcpupin vcpu='2' cpuset='5'/>
  <vcpupin vcpu='3' cpuset='7'/>
  <emulatorpin cpuset='0-1'/>
  <vcpusched vcpus='2-3' scheduler='fifo'
  priority='1'/>
</cputune>
```

CPU のピニングおよびエミュレータースレッドポリシーの設定

リアルタイム負荷用に各リアルタイムコンピュートノードの CPU を十分に確保するためには、インスタンス用仮想 CPU (vCPU) の少なくとも 1 つをホストの物理 CPU (pCPU) にピニングする必要があります。その結果、その仮想 CPU のエミュレータースレッドは、ピニングした物理 CPU 専用として維持されます。

専用 CPU のポリシーを使用するようにフレーバーを設定します。そのためには、フレーバーで **hw:cpu_policy** パラメーターを **dedicated** に設定します。以下は例になります。

```
# openstack flavor set --property hw:cpu_policy=dedicated 99
```

注記

リソースクオータに、Real-time コンピュートノードが消費するのに十分な pCPU があることを確認してください。

ネットワーク設定の最適化

デプロイメントのニーズによっては、特定のリアルタイム負荷に合わせてネットワークをチューニングするために、network-environment.yaml ファイルのパラメーターを設定しなければならない場合があります。

OVS-DPDK 用に最適化した設定の例を確認するには、『 ネットワーク機能仮想化(NFV)のプランニングおよび設定ガイド』の「 OVS-DPDK パラメーター の設定 」セクションを参照してください。

ヒュージページの設定

デフォルトのヒュージページサイズを 1GB に設定することを推奨します。このように設定しないと、TLB のフラッシュにより仮想 CPU の実行にジッターが生じます。ヒュージページの使用に関する一般的な情報については、『DPDK Getting Started Guide for Linux』の「Running DPDK applications」を参照してください。

付録A イメージの設定パラメーター

以下のキーは、glance image-update および glance image-create の両コマンドの property オプションに使用することができます。

```
$ glance image-update IMG-UUID --property architecture=x86_64
```

注記

イメージのプロパティーを使用して設定した動作は、フレーバーを使用して設定した動作よりも優先されます。詳細は、「フレーバーの管理」を参照してください。

表A.1 プロパティーのキー

対象コンポーネント	キー	説明	サポートされている値
すべて	architecture	ハイパーバイザーがサポートする必要のある CPU アーキテクチャー。たとえば、**x86_64**、**arm**、**ppc64** 等。マシンのアーキテクチャーを確認するには、**uname -m** を実行します。このためには、libosinfo project で定義されているアーキテクチャーデータボキャブラリーを使用することを強く推奨します。	• alpha: DEC 64 ビット RISC • armv7l: ARM Cortex-A7 MPCore • cris: Ethernet, Token Ring, AXis-Code Reduced Instruction Set • i686: Intel 第 6 世代 x86 (P6 マイクロアーキテクチャー) • ia64: Itanium • lm32: Lattice Micro32 • m68k: Motorola 68000 • microblaze: Xilinx 32 ビット FPGA (Big Endian) • microblazeel: Xilinx 32 ビット FPGA (Little Endian) • mips: MIPS 32 ビット RISC (Big Endian) • mipsel: MIPS 32 ビット RISC (Little Endian) • mips64: MIPS 64 ビット RISC (Big Endian) • mips64el: MIPS 64 ビット RISC (Little Endian) • openrisc: OpenCores RISC • parisc: HP Precision Architecture RISC • parisc64: HP Precision Architecture 64 ビット RISC • ppc: PowerPC 32 ビット

対象コンポーネント	キー	説明	サポートされている値
			● ppc64: PowerPC 64 ビット ● ppcemb: PowerPC (組み込み向け 32 ビット) ● s390: IBM Enterprise Systems Architecture/390 ● s390x: S/390 64 ビット ● sh4: SuperH SH-4 (Little Endian) ● sh4eb: SuperH SH-4 (Big Endian) ● sparc: Scalable Processor Architecture 32 ビット ● sparc64: Scalable Processor Architecture 64 ビット ● unicore32: Microprocessor Research and Development Center RISC Unicore32 ● x86_64: IA-32 の 64 ビット拡張 ● xtensa: Tensilica Xtensa 構成可能マイクロプロセッサーコア ● xtensaeb: Tensilica Xtensa 構成可能マイクロプロセッサーコア (Big Endian)
すべて	hypervisor_type	ハイパーバイザーのタイプ	**kvm、vmware**
すべて	instance_uuid	スナップショットイメージの場合、このイメージを作成するのに使用したサーバーの UUID	有効なサーバーの UUID
すべて	kernel_id	AMI 形式のイメージをブートする際にカーネルとして使用する必要のある Image サービスに保管されているイメージの ID	有効なイメージ ID
すべて	os_distro	オペレーティングシステムのディストリビューションの小文字による一般名 (libosinfo project と同じ	● arch: Arch Linux。archlinux および org.archlinux は使用しないでください。 ● centos: Community Enterprise Operating System。org.centos および CentOS は使用しないでください。

対象コンポーネント	キー	説明	サポートされている値
		データボキャブラリーを使用)。このフィールドには認識済みの値のみを指定します。認識済みの値の検索で役立つように、非推奨の値を示します。	● debian: Debian。Debian および org.debian は使用しないでください。 ● fedora: Fedora。Fedora、org.fedora、org.fedoraproject は使用しないでください。 ● freebsd: FreeBSD。org.freebsd、freeBSD、FreeBSD は使用しないでください。 ● gentoo: Gentoo Linux。Gentoo および org.gentoo は使用しないでください。 ● mandrake: Mandrakelinux (MandrakeSoft) ディストリビューション。mandrakelinux および MandrakeLinux は使用しないでください。 ● mandriva: Mandriva Linux。mandrivalinux は使用しないでください。 ● mes: Mandriva Enterprise Server。mandrivaent および mandrivaES は使用しないでください。 ● msdos: Microsoft Disc Operating System。ms-dos は使用しないでください。 ● netbsd: NetBSD。NetBSD および org.netbsd は使用しないでください。 ● netware: Novell NetWare。novell および NetWare は使用しないでください。 ● openbsd: OpenBSD。OpenBSD および org.openbsd は使用しないでください。 ● opensolaris: OpenSolaris。OpenSolaris および org.opensolaris は使用しないでください。 ● opensuse: openSUSE。suse、SuSE、org.opensuse は使用しないでください。 ● rhel: Red Hat Enterprise Linux。redhat、RedHat、com.redhat は使用しないでください。 ● sled: SUSE Linux Enterprise Desktop。com.suse は使用しないでください。 ● ubuntu: Ubuntu。Ubuntu、com.ubuntu、org.ubuntu、canonical は使用しないでください。 ● windows: Microsoft Windows。com.microsoft.server は使用しないでください。

対象コンポーネント	キー	説明	サポートされている値
すべて	os_version	ディストリビューターによって指定されるオペレーティングシステムのバージョン	バージョン番号 (例:「11.10」)
すべて	ramdisk_id	AMI 形式のイメージをブートする際に ramdisk として使用する必要のある、Image サービスに保管されているイメージの ID	有効なイメージ ID
すべて	vm_mode	仮想マシンのモード。仮想マシンに使用されるホスト/ゲストの ABI (アプリケーションバイナリーインターフェース) を示します。	**hvm**: 完全仮想化。これは QEMU および KVM で使用されるモードです。
libvirt API ドライバー	hw_disk_bus	ディスクデバイスの接続先となるディスクコントローラーのタイプを指定します。	**scsi**、**virtio**、**ide**、**usb** のいずれか。**iscsi** を使用している場合には、**hw_scsi_model** を **virtio-scsi** に設定する必要がある点に注意してください。
libvirt API ドライバー	hw_numa_nodes	インスタンスに公開する NUMA ノードの数 (フレーバーの定義はオーバーライドしません)	整数。NUMA トポロジー定義の詳しい例は、「メタデータの追加」で「hw:NUMA_def key」を参照してください。
libvirt API ドライバー	hw_numa_cpus.0	仮想 CPU N-M から NUMA ノード 0 へのマッピング (フレーバーの定義はオーバーライドしません)	整数のコンマ区切りリスト

対象コンポーネント	キー	説明	サポートされている値
libvirt API ドライバー	hw_numa_cpus.1	仮想 CPU N-M から NUMA ノード 1 へのマッピング (フレーバーの定義はオーバーライドしません)	整数のコンマ区切りリスト
libvirt API ドライバー	hw_numa_mem.0	N MB の RAM から NUMA ノード 0 へのマッピング (フレーバーの定義はオーバーライドしません)	整数
libvirt API ドライバー	hw_numa_mem.1	N MB の RAM から NUMA ノード 1 へのマッピング (フレーバーの定義はオーバーライドしません)	整数
libvirt API ドライバー	hw_qemu_guest_agent	ゲストエージェントのサポート。**yes** に設定し、かつ **qemu-ga** もインストールされている場合には、ファイルシステムが休止 (フリーズ) し、スナップショットが自動的に作成されます。	**yes** / **no**

対象コンポーネント	キー	説明	サポートされている値
libvirt API ドライバー	hw_rng_model	乱数生成器をイメージのインスタンスに追加します。インスタンスのフレーバーを設定することにより、クラウド管理者は、デバイスの動作を有効化して制御することができます。デフォルトでは以下のように設定されます。 ● 乱数生成器は無効化されます。 ● /dev/random がデフォルトのエントロピーソースとして使用されます。物理ハードウェアの乱数生成器を指定するには、Compute 環境ファイルで **rng_dev_path** を「/dev/hwrng」に設定します。	**virtio** またはその他のサポートされているデバイス

対象コンポーネント	キー	説明	サポートされている値
libvirt API ドライバー	hw_scsi_model	VirtIO SCSI (virtio-scsi) の使用を有効にして、コンピュートインスタンスのブロックデバイスアクセスを提供します。デフォルトでは、インスタンスは VirtIO Block (virtio-blk) を使用します。VirtIO SCSI は準仮想化 SCSI コントローラーデバイスで、より高いスケーラビリティーとパフォーマンスを提供し、高度な SCSI ハードウェアに対応します。	**virtio-scsi**
libvirt API ドライバー	hw_video_model	使用されるビデオイメージドライバー	**vga**、**cirrus**、**vmvga**、**xen**、**qxl**
libvirt API ドライバー	hw_video_ram	ビデオイメージの最大 RAM。フレーバーの **extra_specs** で **hw_video:ram_max_mb** の値が設定済みで、かつその値が **hw_video_ram** で設定されている値を上回る場合にのみ使用されます。	整数 (MB 単位。例: 64)

対象コンポーネント	キー	説明	サポートされている値
libvirt API ドライバー	hw_watchdog_action	サーバーがハングした場合に指定したアクションを実行する仮想ハードウェアウォッチドッグデバイスを有効にします。このウォッチドッグは、i6300esb デバイスを使用します (PCI Intel 6300ESB をエミュレート)。**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。	• disabled: デバイスは接続されていません。イメージのフレーバーを使用して有効化されている場合でも、ユーザーがイメージのウォッチドッグを無効にすることができます。このパラメーターのデフォルト値は「disabled」です。 • reset: ゲストを強制的にリセットします。 • poweroff: ゲストの電源を強制的に切断します。 • pause: ゲストを一時停止します。 • none: ウォッチドッグを有効化するのみで、サーバーがハングした場合には何もしません。
libvirt API ドライバー	os_command_line	デフォルトではなく、libvirt ドライバーで使用されるカーネルコマンドライン。Linux Containers (LXC) の場合は、この値が初期化の引数として使用されます。このキーは、Amazon カーネル、ramdisk、またはマシンイメージ (aki、ari、または ami) にのみ有効です。	

対象コンポーネント	キー	説明	サポートされている値
libvirt API ドライバーおよび VMware API ドライバー	hw_vif_model	使用する仮想ネットワークインターフェースデバイスのモデルを指定します。	設定したハイパーバイザーによって有効なオプションは異なります。 • KVM および QEMU: e1000、ne2k_pci、pcnet、rtl8139、virtio • VMware: e1000、e1000e、VirtualE1000、VirtualE1000e、VirtualPCNet32、VirtualSriovEthernetCard、VirtualVmxnet • Xen: e1000、netfront、ne2k_pci、pcnet、rtl8139
VMware API ドライバー	vmware_adaptertype	ハイパーバイザーが使用する仮想 SCSI または IDE コントローラー	**lsiLogic**、**busLogic**、または **ide**
VMware API ドライバー	vmware_ostype	イメージにインストールされているオペレーティングシステムを示す VMware GuestID。この値は、仮想マシンの作成時にハイパーバイザーに渡されます。指定しなかった場合には、このキーの値はデフォルトで **otherGuest** に設定されます。	アップストリームのドキュメント を参照してください。
VMware API ドライバー	vmware_image_version	現在は使用されていません。	**1**

対象コンポーネント	キー	説明	サポートされている値
XenAPI ドライバー	auto_disk_config	true に指定した場合には、ディスク上の root パーティションは、インスタンスがブートする前に自動的にリサイズされます。この値は、Xen ベースのハイパーバイザーを XenAPI ドライバーと共に使用する場合にのみ Compute サービスによって考慮されます。Compute サービスは、イメージに単一のパーティションがあり、かつそのパーティションが **ext3** または**ext4** のフォーマットの場合にのみリサイズを試みます。	**true** / **false**

対象コンポーネント	キー	説明	サポートされている値
libvirt API ドライバーおよび XenAPI ドライバー	os_type	イメージ上にインストールされるオペレーティングシステム。XenAPI ドライバーには、イメージの **os_type** パラメーターの値によって異なるアクションを実行するロジックが組み込まれています。たとえば、**os_type=windows** イメージの場合には、Linux スワップパーティションの代わりに、FAT32 ベースのスワップパーティションを作成し、挿入されるホスト名を 16 文字未満に制限します。	**linux** または **windows**

付録B インスタンスの起動ウィザードの有効化

Dashboard からインスタンスを起動するには、2 つの方法があります。

- インスタンスの起動フォーム
- インスタンスの起動ウィザード

インスタンスの起動フォームはデフォルトで有効化されますが、インスタンスの起動ウィザードを随時有効化することができます。インスタンスの起動フォームとウィザードを同時に有効化することも可能です。インスタンスの起動ウィザードにより、インスタンスの作成に必要なステップが簡素化されます。

1. **/etc/openstack-dashboard/local_settings** ファイルを編集して以下の値を追加します。

```
LAUNCH_INSTANCE_LEGACY_ENABLED = False
LAUNCH_INSTANCE_NG_ENABLED = True
```

2. **httpd** サービスを再起動します。

```
# systemctl restart httpd
```

インスタンスの起動フォームとウィザードの設定が更新されます。

これらのオプションのいずれか 1 つのみを有効にした場合には、Dashboard で インスタンスの起動 ボタンをクリックすると、そのオプションがデフォルトで開きます。両方のオプションを有効化した場合には、Dashboard に インスタンスの起動 ボタンが 2 つ表示され、左のボタンはインスタンスの起動ウィザード、右のボタンはインスタンスの起動フォームを表示します。