Red Hat OpenStack Platform 10
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

OpenStack Team

Red Hat OpenStack Platform 10 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

OpenStack Team
rhos-docs@redhat.com

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

Red Hat OpenStack Platform は、Red Hat Enterprise Linux をベースとして、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で参照してください。

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
- **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 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

Image サービスでは、以下のような注目すべき機能を提供しています。

- **イメージの署名および検証**: 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 サービスでのイメージ作成し、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。

- **イメージの変換**: 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。

  注記

  フォーマットの変換は、イメージ (元のコピー) を **インポート** するとトリガーされます。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 --os-image-api-version 2 task-create --type import -
  -input '{"import_from_format": "qcow2", "import_from":
  "http://127.0.0.1:8000/test.qcow2", "image_properties":
  {"disk_format": "qcow2", "container_format": "bare"}}'
  ```

- **イメージのイントロスペクション**: すべてのイメージフォーマットには、イメージ自体の中に埋め込まれたメタデータセットがあります。

  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。このように Image サービス内のメタデータを抽出することにより、管理者は、上書きする必要がなければ、それらのメタデータについて注意を払う必要はありません。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。

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追加の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。

- RHEL 7.2 KVM Guest Image
- RHEL 6.8 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るので、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** パスワードの挿入」の手順に従って **admin** パスワードを設定してください。

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 (**yum 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 (「**yum install -y libguestfs-tools-c**」のコマンドを実行してください)。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 または 6 ISO ファイル (RHEL 7.2 Binary DVD または RHEL 6.8 Binary DVD を参照)、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。
- テキストエディター (**kickstart** ファイルを編集する必要がある場合 - RHEL のみ)

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します。

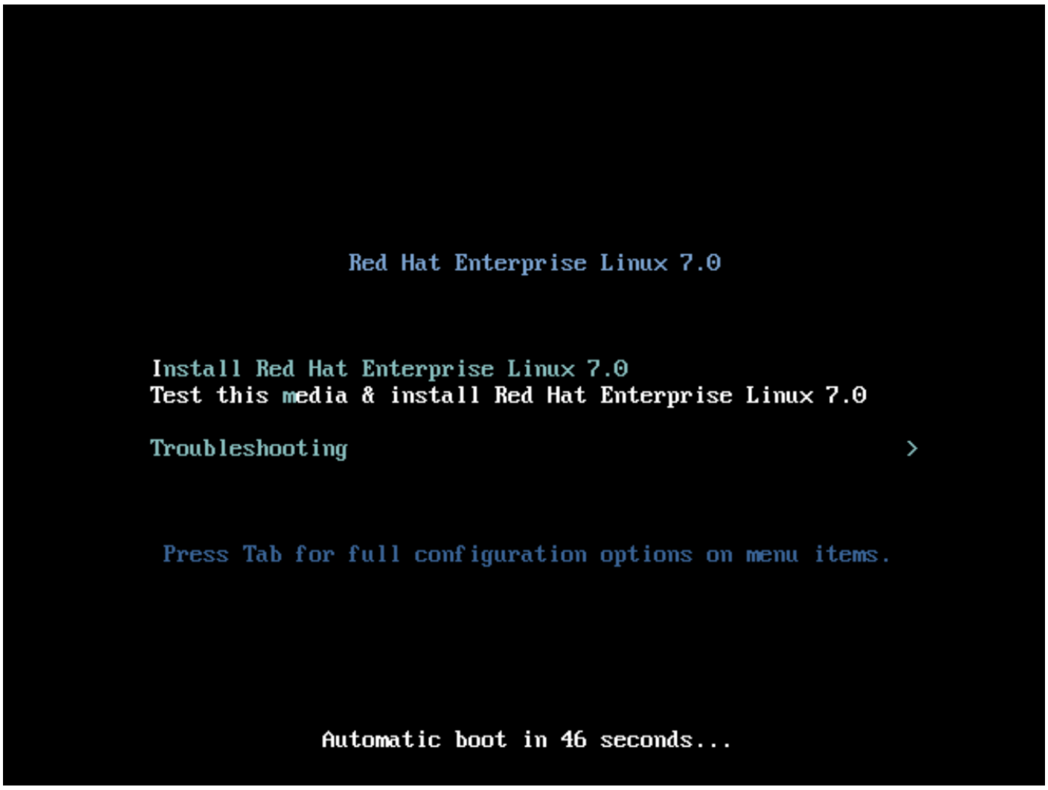
注記

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
```

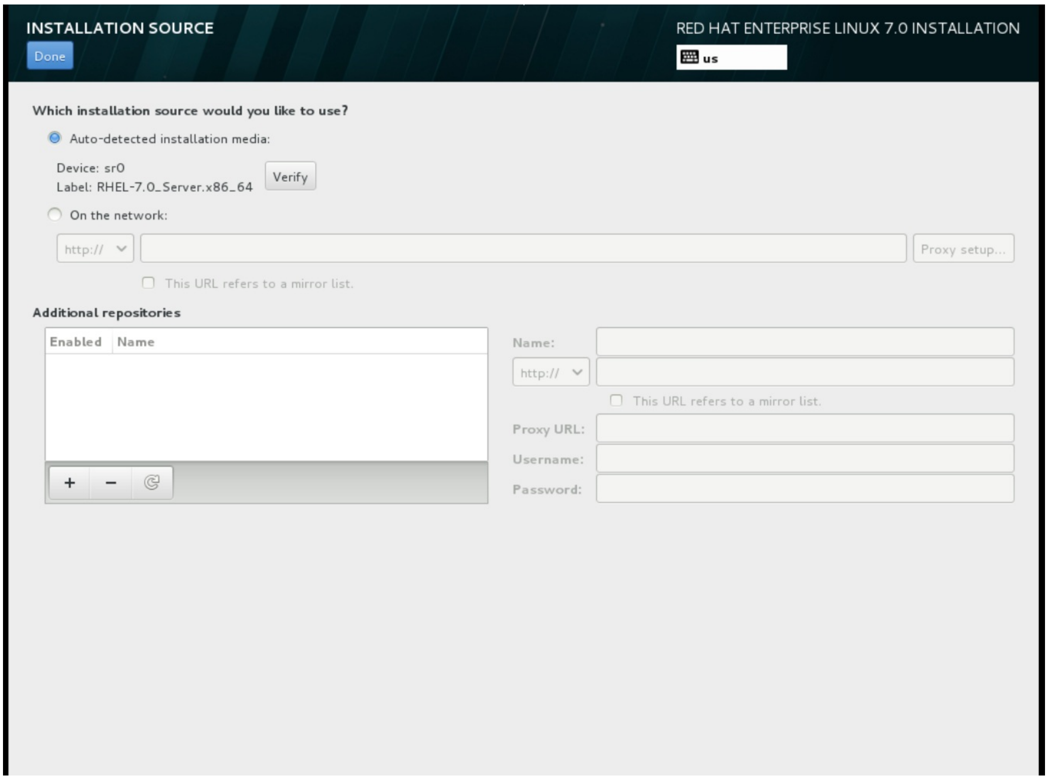
2. 以下の手順に従って、仮想マシンを設定します。

   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。

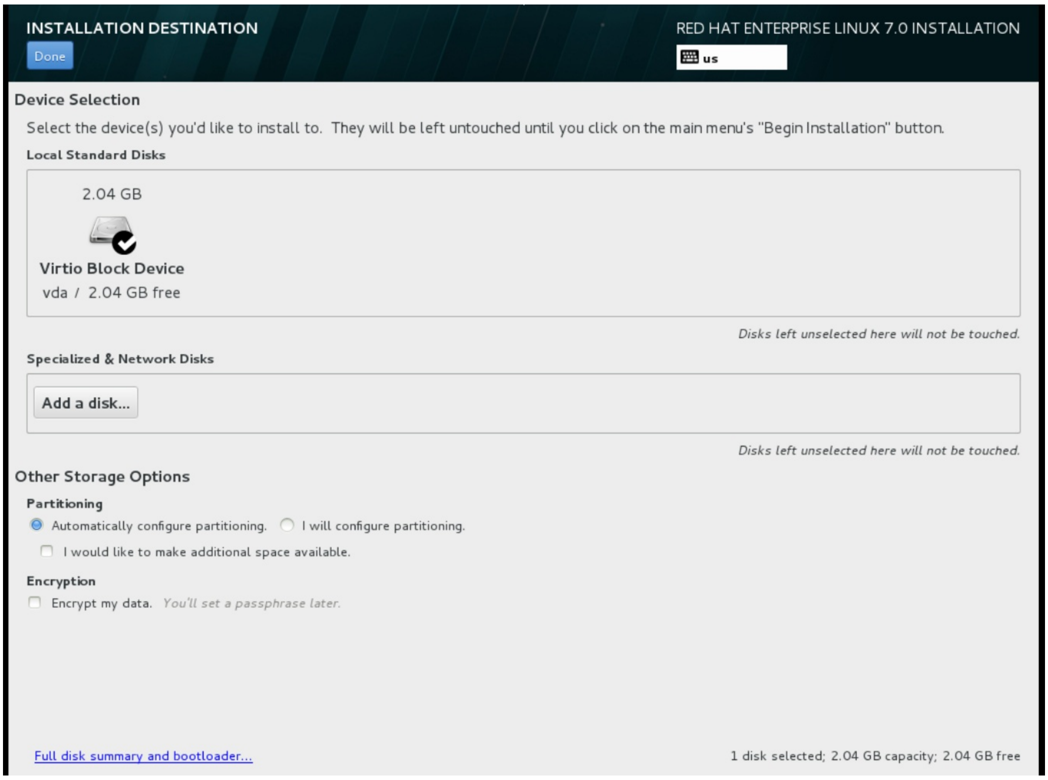
   

   b. 適切な **言語** および **キーボード** オプションを選択します。

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。

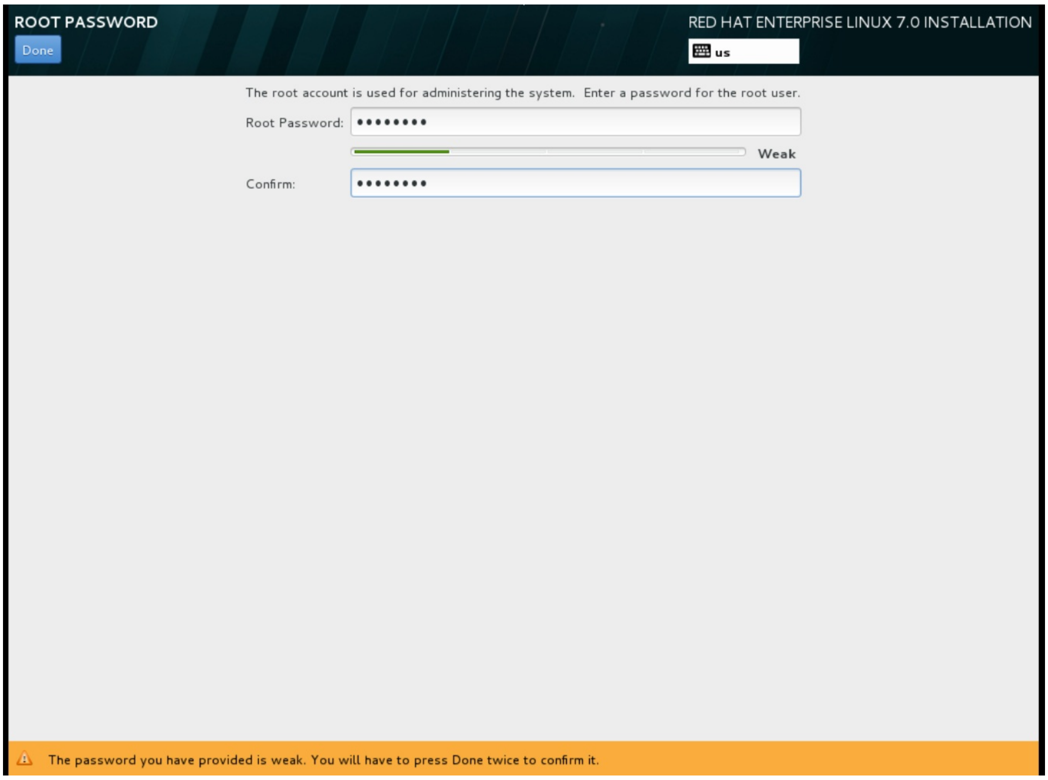


その他のストレージタイプオプションには、**自動構成のパーティション構成** を選択します。

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。



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。詳細は **『手動インストール手順』** の「必要なチャンネルのサブスクライブ」を参照してください。

7. システムを更新します。

```
# yum -y update
```

8. **cloud-init** パッケージをインストールします。

```
# yum 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
# yum 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します。

   > **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。

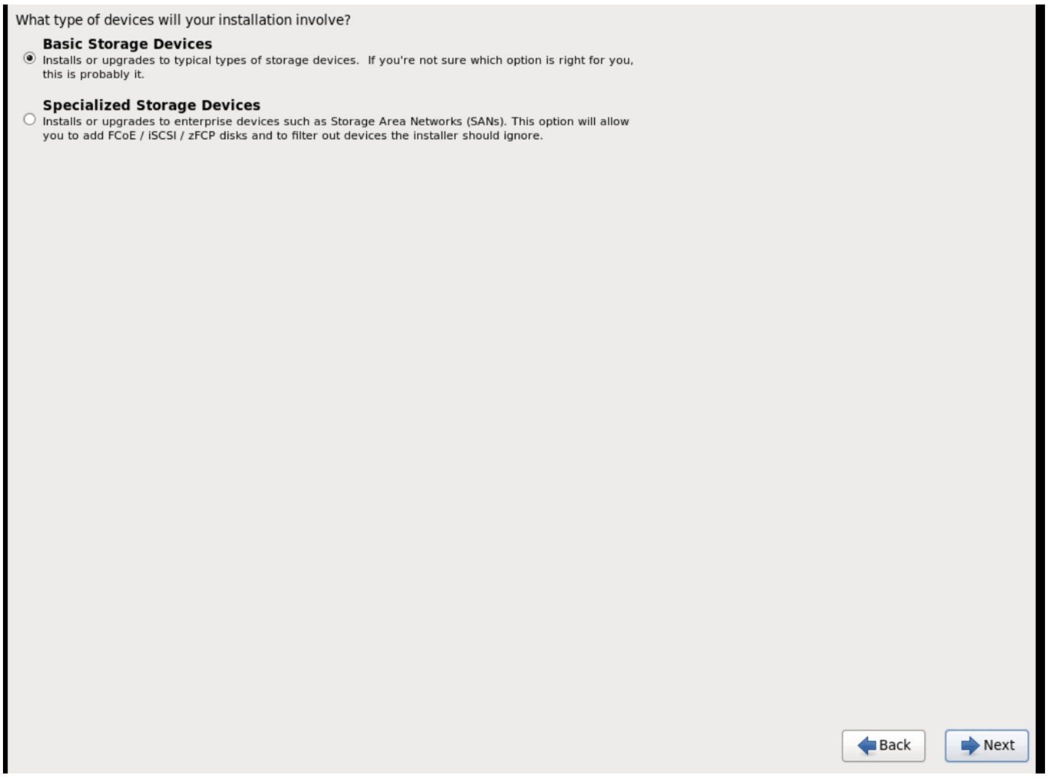
a. インストーラーの初期起動メニューで、**Install or upgrade an existing system** のオプションを選択します。

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。

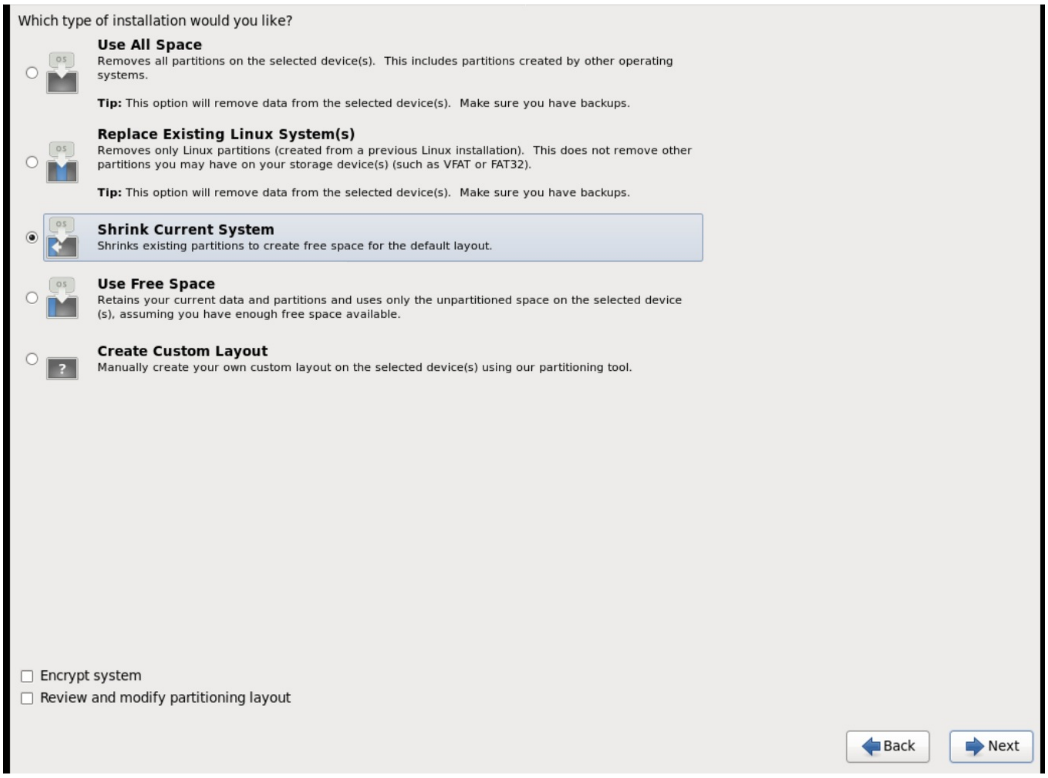
c. インストールに使用するデバイスタイプを尋ねるプロンプトが表示されたら、**基本ストレージデバイス** を選択します。



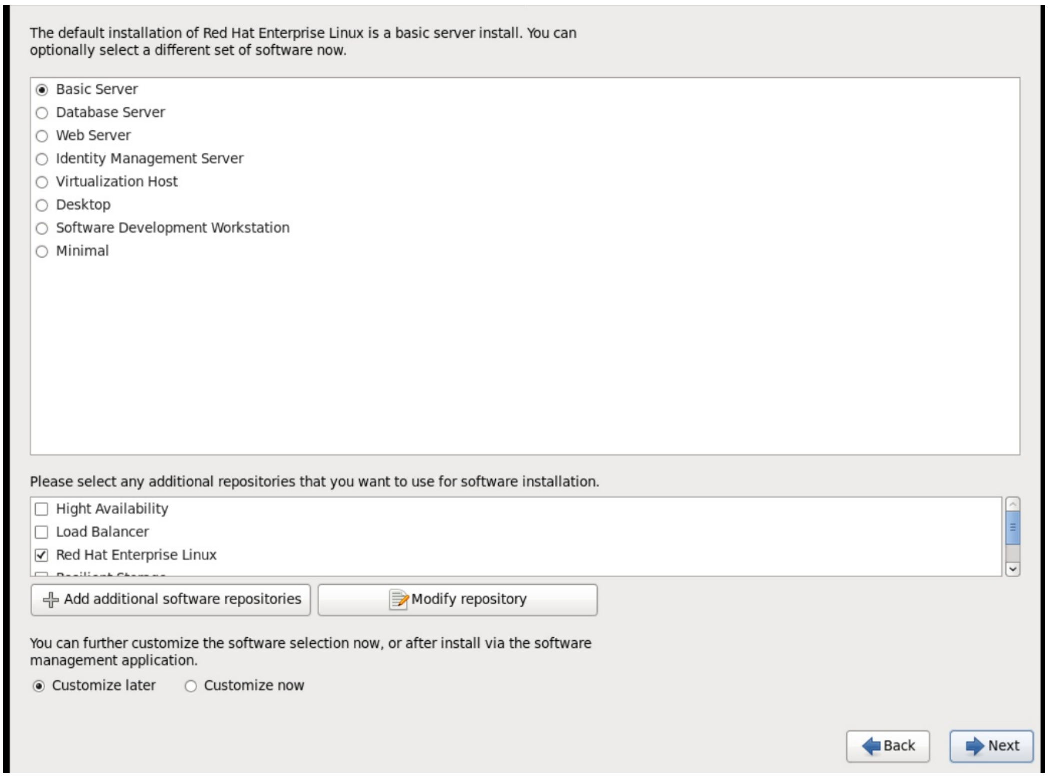
d. デバイスの **ホスト名** を指定します。デフォルトのホスト名は **`localhost.localdomain`** です。

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。詳細は**『手動インストール手順』**の「必要なチャンネルのサブスクライブ」を参照してください。

7. システムを更新します。

```
# yum -y update
```

8. **cloud-init** パッケージをインストールします。

```
# yum 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
# yum 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。**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に関する方法は、**『仮想化の導入および管理ガイド』**の「9.1 章 KVM WINDOWS VIRTIO ドライバーのインストール」を参照してください。

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。

1.2.2. 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)
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。

フィールド	説明
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 **プロパティー** 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。

1.2.4.1. 削除するイメージデータの完全削除

イメージが削除された後でも、イメージに関する情報は、OpenStack Image サービスのデータベースに保持されます。そのため、データベースは時間が経過するにつれ拡大し、使用に時間がかかり、アップグレードが困難になります。

削除したイメージに関する情報を削除するには、**`glance-manage db purge`** コマンドを使用します。デフォルトでは、削除されるデータベースのエントリーは、30 日 (以上) 前に削除したイメージを参照する場合ですが、パラメーターとして、別の期間 (日数) を指定することができます。たとえば、60 日よりも前に削除したイメージに関する情報を削除するには、**`glance-manage`** のコマンドを実行します。

```
$ glance-manage db purge 60
```

第2章 OPENSTACK COMPUTE 用のストレージの設定

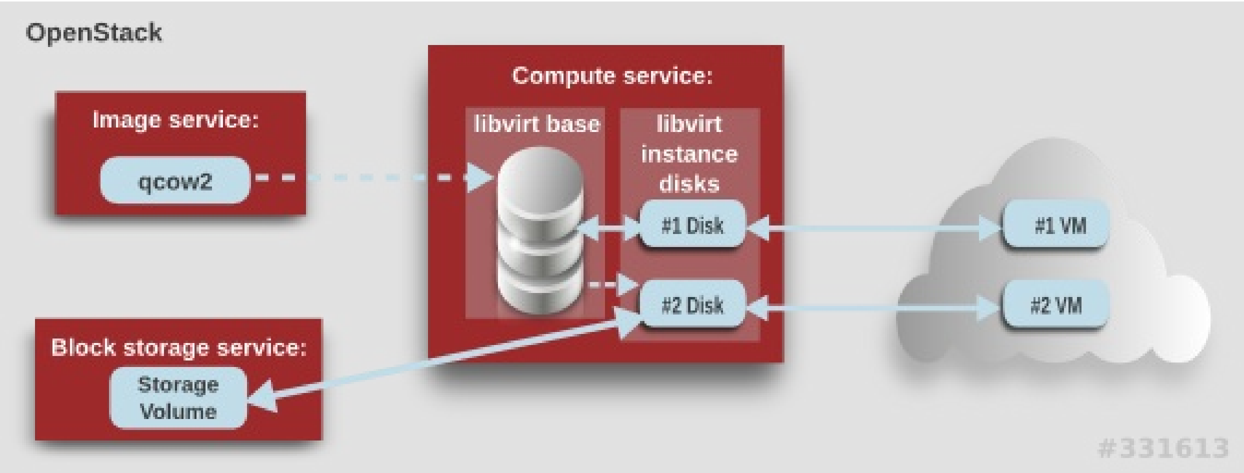
本章では、OpenStack Compute (nova) のイメージのバックエンドストレージのアーキテクチャーについて説明し、基本的な設定オプションを記載します。

2.1. 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、仮想マシンの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の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。

2.2. 設定

libvirt ベースとインスタンスディスクを処理するための Compute の設定により、お使いの環境のパフォーマンスとセキュリティー両方を決定することができます。パラメーターは、**/etc/nova/nova.conf** ファイルで設定します。

表2.1 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。値は以下の通りです。 - **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3章 仮想マシンのインスタンス

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。

3.1. 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。

3.1.1. 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 > 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」の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 (イメージを作成する場合は 『**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、Red Hat OpenStack Platform で『**User and Identity Management**』の「Project Security Management」を参照してください)。

3.1.2. インスタンスの作成

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 (「*」でマークされているフィールドは必須) にインスタンスの設定値を入力し、完了したら **起動** をクリックします。

表3.1 インスタンスのオプション

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。
	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)。

タブ	フィールド	説明
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『**Users and Identity Management Guide**』の「Project Security Management」を参照)。
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が、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)。

3.1.3. 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3.2 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、『**Users and Identity Management Guide**』の「Project Security Management」を参照してください)。

アクション	説明
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。
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。
インスタンスの再作成	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作成します (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。

アクション	説明
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、解放」を参照してください。

3.1.4. 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)。

   SSH 鍵認証についての詳しい情報は、『**Migrating Instances** ガイド』の「Configure SSH Tunneling Between Nodes」の項を参照してください。

2. 元のホストでリサイズを有効にするには、**/etc/nova/nova.conf** ファイルで以下のパラメーターを設定します。

   ```
   [DEFAULT] allow_resize_to_same_host = True
   ```

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。

3.1.5. インスタンスへの接続

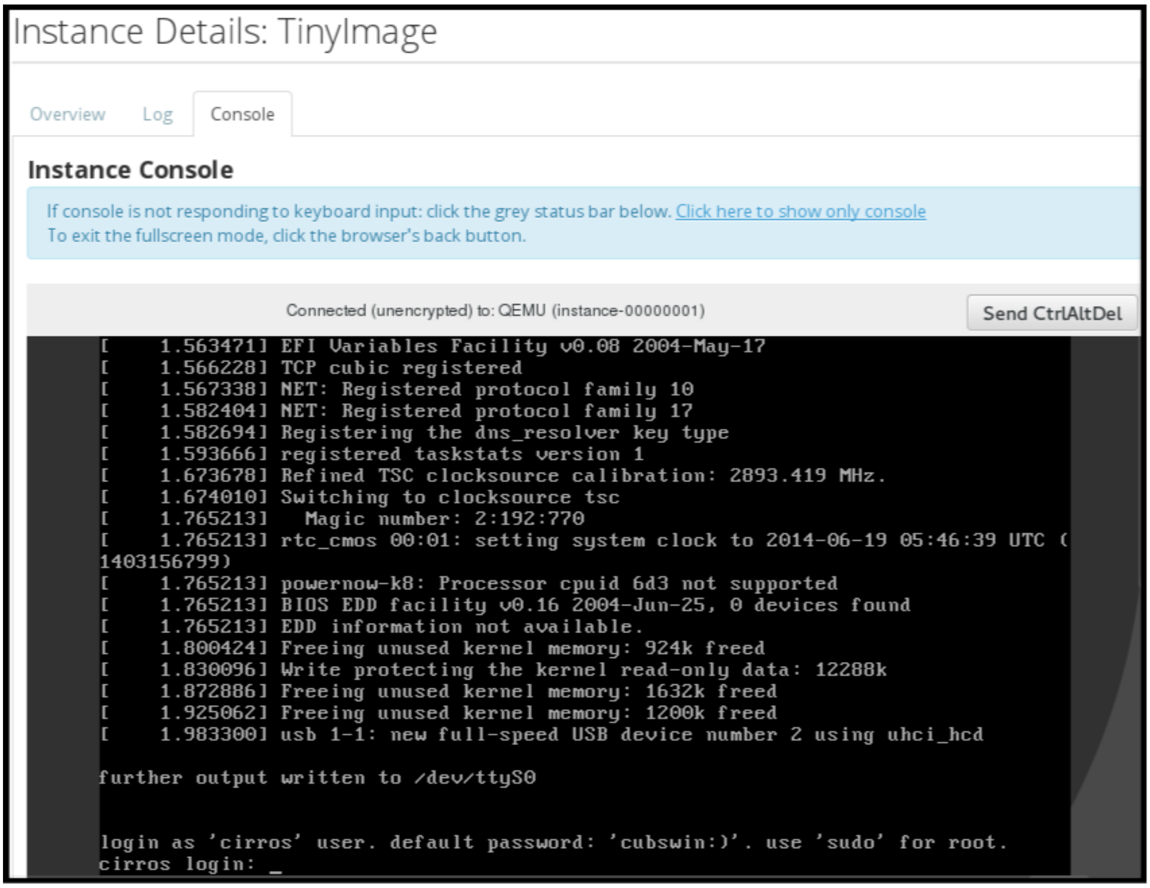
本項では、Dashboard またはコマンドラインインターフェースを使用して、インスタンスのコンソールにアクセスする複数の方法について説明します。また、インスタンスのシリアルポートに直接接続して、ネットワーク接続が失敗しても、デバッグすることが可能です。

3.1.5.1. 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)。

3.1.5.2. 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。

3.1.5.3. シリアルコンソールへの直接接続

Websocket クライアントを使用すると、コンソールのシリアルポートに直接アクセスすることが可能です。シリアル接続は、通常デバッグツールで使用されます (たとえば、ネットワーク設定に問題がある場合でもインスタンスにアクセス可能)。実行中のインスタンスのシリアル URL を取得するには、以下のコマンドを実行します。

```
$ nova get-serial-console INSTANCE_ID
```

注記

「novaconsole」は、Websocket 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/larsks/novaconsole/
# cd novaconsole
```

インスタンスのシリアル URL を使用してクライアントを実行します。

```
# python console-client-poll.py
```

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。

ただし、インストールによっては、管理者があらかじめ「nova-serialproxy」サービスを設定する必要がある場合があります。プロキシーサービスは、OpenStack Compute のシリアルポートへの接続が可能な Websocket プロキシーです。

3.1.5.3.1. nova-serialproxy のインストールと設定

1. **nova-serialproxy** サービスをインストールします。

```
# yum install openstack-nova-serialproxy
```

2. **/etc/nova/nova.conf** の **serial_console** セクションを更新します。

   a. **nova-serialproxy** サービスを有効化します。

   ```
   $ openstack-config --set /etc/nova/nova.conf serial_console
   enabled true
   ```

   b. **nova get-serial-console** コマンドで提供される URL の生成に使用する文字列を指定します。

   ```
   $ openstack-config --set /etc/nova/nova.conf serial_console
   base_url ws://PUBLIC_IP:6083/
   ```

   PUBLIC_IP は、**nova-serialproxy** サービスを実行するホストのパブリック IP アドレスに置き換えます。

   c. インスタンスのシリアルコンソールをリッスンする IP アドレスを指定します (文字列)。

   ```
   $ openstack-config --set /etc/nova/nova.conf serial_console
   listen 0.0.0.0
   ```

   d. プロキシークライアントが接続する必要のあるアドレスを指定します (文字列)。

   ```
   $ openstack-config --set /etc/nova/nova.conf serial_console
   proxyclient_address ws://HOST_IP:6083/
   ```

   HOST_IP は、Compute ホストの IP アドレスに置き換えます。たとえば、**nova-serialproxy** サービスを有効化する設定は、以下のようになります。

   ```
   [serial_console]
   enabled=true
   base_url=ws://192.0.2.0:6083/
   listen=0.0.0.0
   proxyclient_address=192.0.2.3
   ```

3. Compute サービスを再起動します。

   ```
   # openstack-service restart nova
   ```

4. **nova-serialproxy** サービスを起動します。

   ```
   # systemctl enable openstack-nova-serialproxy
   # systemctl start openstack-nova-serialproxy
   ```

5. 実行中のインスタンスを再起動して、正しいソケットをリッスンするようになったことを確認します。

6. シリアル/コンソール間のポート接続のためにファイアウォールを開きます。シリアルポートは、**/etc/nova/nova.conf** で **[serial_console]** port_range を使用して設定します。デフォルトの範囲は、10000:20000 です。以下のコマンドを実行して、iptables を更新します。

```
# iptables -I INPUT 1 -p tcp --dport 10000:20000 -j ACCEPT
```

3.1.6. 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。

3.1.7. 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)。

3.1.8. 複数のインスタンスの一括管理

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。

3.2. 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、**『Users and Identity Management Guide』**の「Project Security Management」を参照してください。

3.2.1. 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。

3.2.1.1. 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
```

3.2.1.2. 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。

3.2.1.3. 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。

3.2.2. セキュリティーグループの作成

3.2.2. 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、**『Users and Identity Management Guide』**の「Project Security Management」を参照してください。

3.2.3. Floating IP アドレスの作成、割り当て、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が、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。詳細は**『ネットワークガイド』**の「Floating IP プールの作成」を参照してください。

3.2.3.1. 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** のフィールドから、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。

3.2.3.2. 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。

3.2.3.3. 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。

3.2.4. 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 (**『Users and Identity Management Guide』** の「Project Security Management」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、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
```

3.2.5. 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. **/etc/nova/nova.conf** ファイルで、**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
   ```

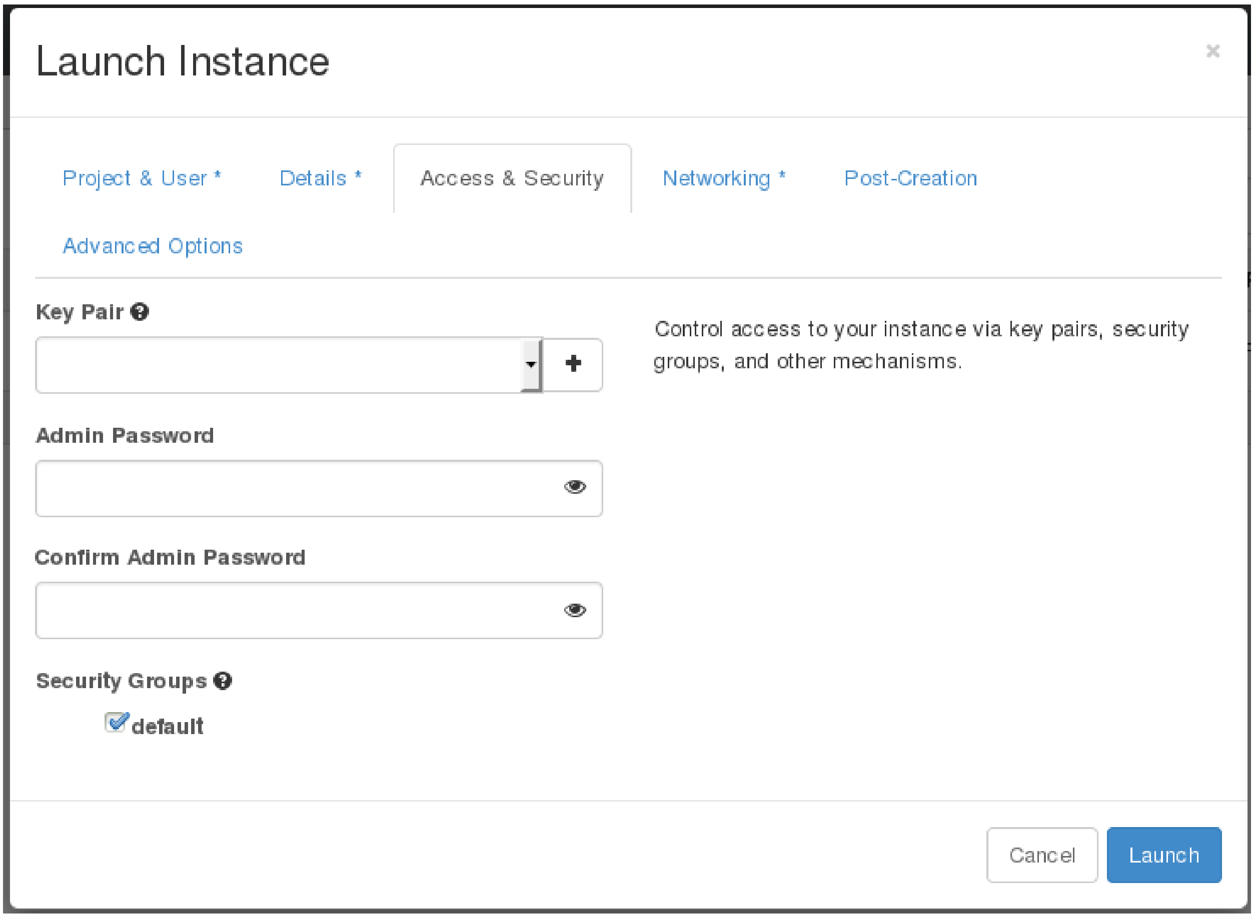
nova boot コマンドを使用して、新規インスタンスを起動する際には、コマンドの出力に **adminPass** パラメーターが表示されます。このパスワードを使用して、インスタンスに **root** ユーザーとしてログインすることができます。

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。

3.3. 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3.3 デフォルトのフレーバー

名前	仮想 CPU	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

名前	仮想 **CPU**	メモリー	ルートディスクのサイズ
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が、特化したフレーバーを作成/管理する必要がある場合もあります。たとえば、以下の設定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。

3.3.1. 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": "",
```

3.3.2. 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3.4 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

タブ	フィールド	説明
	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。
	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。

3.3.3. 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。

3.3.4. 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3.5「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)。

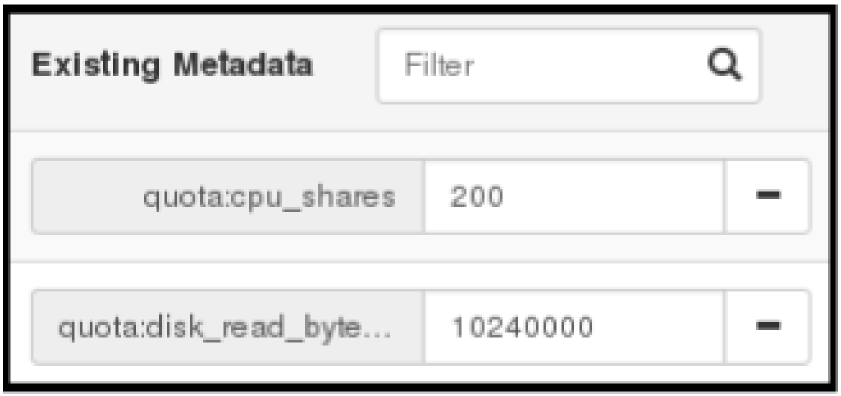
3.3.4.1. メタデータの表示

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。

3.3.4.2. 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3.5「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3.5 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通りです。 » **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**

<table>
<tr><th>キー</th><th>説明</th></tr>
<tr><td>hw:NUMA_def</td><td>
インスタンスの NUMA トポロジーの定義。RAM および vCPU 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通りです。

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_mempolicy: メモリーの割り当てポリシー。以下が有効なポリシーです。

strict: バインディングされる NUMA ノードからインスタンスの RAM が割り当てられるようにするには必須です (numa_nodes が指定されている場合にはデフォルト)。
preferred: カーネルは、代替ノードを使用するようにフォールバックすることが可能です。numa_nodes が「1」に設定されている場合に有用です。

numa_cpus.0: vCPU 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: vCPU 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 GB を NUMA ノード 0 へマッピング
numa_mem.1: メモリー N GB 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 vCPU、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=3
hw:numa_mem.1=1

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 3 GB のメモリーを実行し、別のノードで 2 つの CPU および 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。numa_mempolicy の設定に関わらず、同様のロジックがスケジューラーで適用されます。
</td></tr>
</table>

<table>
<tr><th>キー</th><th>説明</th></tr>
<tr><td><code>hw:watchdog_action</code></td><td>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通りです。
<code>disabled</code>: デバイスは接続されません (デフォルト値)。
<code>pause</code>: インスタンスを一時停止します。
<code>poweroff</code>: インスタンスを強制終了します。
<code>reset</code>: 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: <code>hw:watchdog_action=poweroff</code></td></tr>
<tr><td><code>hw_rng:action</code></td><td>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 <code>hw_rng_model</code> 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通りです。
<code>allowed</code>: <code>True</code> に指定すると、デバイスが有効化され、<code>False</code> に指定すると無効化されます。デフォルトでは無効となっています。
<code>rate_bytes</code>: 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
<code>rate_period</code>: 秒単位で示した読み取り期間 (整数)
例: <code>hw_rng:allowed=True</code></td></tr>
<tr><td><code>hw_video:ram_max_mb</code></td><td>ビデオデバイスの最大許容 RAM (MB 単位)

例: <code>hw:ram_max_mb=64</code></td></tr>
<tr><td><code>quota:option</code></td><td>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通りです。
<code>cpu_period</code>: <code>cpu_quota</code> を強制する時間 (マイクロ秒)。指定した <code>cpu_period</code> 内では、各仮想 CPU は <code>cpu_quota</code> 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。
<code>cpu_quota</code>: 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 が制御されていないことを意味します。<code>cpu_quota</code> および <code>cpu_period</code> 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。
<code>cpu_shares</code>: 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</td></tr>
</table>

キー	説明
	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 - **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) - **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) - **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) - **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) - **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) - **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) - **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。

キー	説明
	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) **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) **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。 **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。

3.4. 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。

3.4.1. 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. **/etc/nova/nova.conf** ファイルを編集します (root または nova ユーザーのパーミッションが必要です)。

2. **scheduler_default_filters** パラメーターに以下の値が含まれていることを確認します。

   - ホストアグリゲートのメタデータ用の **AggregateInstanceExtraSpecsFilter**。た

とえば、以下のように記載します。

```
scheduler_default_filters=AggregateInstanceExtraSpecsFilter,RetryFilter,RamFilter,ComputeFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter,CoreFilter
```

- 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の仕様用の **AvailabilityZoneFilter**。たとえば、以下のように記載します。

```
scheduler_default_filters=AvailabilityZoneFilter,RetryFilter,RamFilter,ComputeFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter,CoreFilter
```

3. 設定ファイルを保存します。

3.4.2. 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。

3.4.3. 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。

3.4.4. 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3.6「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3.6 ホストアグリゲートのメタデータ

キー	説明
`cpu_allocation_ratio`	物理 CPU に対する仮想 CPU の割り当ての比率を設定します。これは、Compute のスケジューラーに設定されている **`AggregateCoreFilter`** フィルターによって異なります。
`disk_allocation_ratio`	物理ディスクに対する仮想ディスクの割り当ての比率を設定します。これは、Compute のスケジューラーに設定されている **`AggregateDiskFilter`** フィルターによって異なります。
`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。
`ram_allocation_ratio`	物理メモリーに対する仮想メモリーの割り当ての比率を設定します。これは、Compute のスケジューラーに設定されている **`AggregateRamFilter`** フィルターによって異なります。

3.4.5. 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。

3.5. ホストとセルのスケジュール

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セルまたはホスト (もしく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。

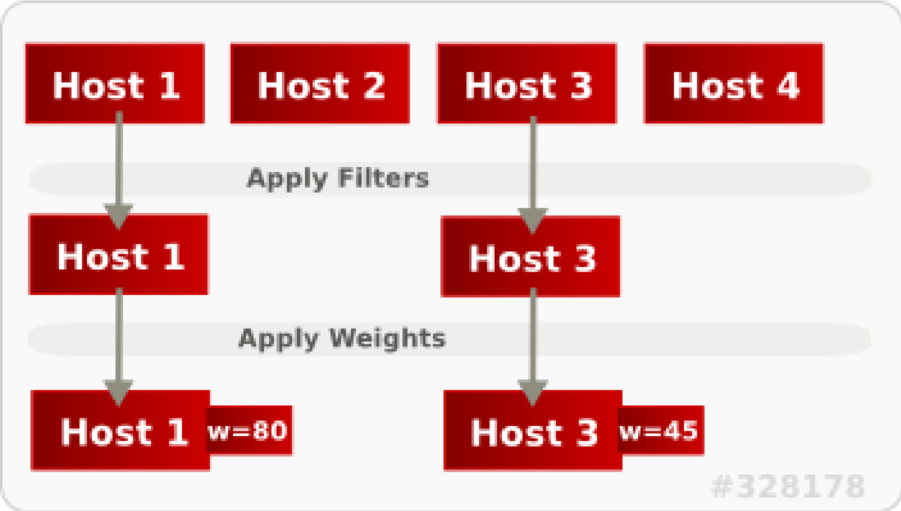
以下のコンポーネントを設定することができます。

- 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 **`/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。ホストとセルの両方にスケジューラーがあります。これらのオプションの一覧は『Configuration Reference』を参照してください。

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。



3.5.1. スケジューリングフィルターの設定

scheduler_default_filters オプションでスケジューラーが使用するフィルターを定義します (**/etc/nova/nova.conf** ファイル。root または nova ユーザーのパーミッションが必要)。

デフォルトでは、以下のフィルターがスケジューラーで実行されるように設定されています。

```
scheduler_default_filters=RetryFilter,AvailabilityZoneFilter,RamFilter,
ComputeFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter,ServerGro
upAntiAffinityFilter,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** コマンドについては『Command-Line Interface Reference』ガイドを参照してください。
- 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)

以下の表には、利用可能な全フィルターをまとめています。

表3.7 スケジューリングフィルター

フィルター	説明
AggregateCoreFilter	ホストアグリゲートのメタデータキー cpu_allocation_ratio を使用して、オーバーコミット比 (物理 CPU に対する仮想 CPU の割り当ての比率) を超過したホストを除外します。これは、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のみに有効です。
	この比率が設定されている場合には、フィルターは /etc/nova/nova.conf ファイルの cpu_allocation_ratio の値を使用します。デフォルト値は **16.0** です (1 物理 CPU に対して 16 仮想 CPU を割り当て可能)。

フィルター	説明
AggregateDiskFilter	ホストアグリゲートのメタデータキー disk_allocation_ratio を使用して、オーバーコミット比 (物理ディスクに対する仮想ディスクの割り当ての比率) を超過したホストを除外します。これは、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のみに有効です。
	この比率が設定されている場合には、フィルターは /etc/nova/nova.conf ファイルの disk_allocation_ratio の値を使用します。デフォルト値は **1.0** です (1 物理ディスク に対して 1 仮想ディスクを割り当て可能)。
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。詳しい情報は、「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。詳しい情報は、「フレーバーのメタデータの更新」を参照してください。
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。
AggregateRamFilter	ホストアグリゲートのメタデータキー ram_allocation_ratio を使用して、オーバーコミット比 (物理メモリーに対する仮想メモリーの割り当ての比率) を超過したホストを除外します。これは、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のみに有効です。
	この比率が設定されている場合には、フィルターは /etc/nova/nova.conf ファイルの ram_allocation_ratio の値を使用します。デフォルト値は **1.5** です (1 物理メモリーに対して 1.5 仮想メモリーを割り当て可能)。
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。
CoreFilter	**/etc/nova/nova.conf** ファイルの cpu_allocation_ratio を使用して、オーバーコミット比 (物理 CPU に対する仮想 CPU の比率) を超過したホストを除外します。デフォルトの値は **16.0** です (1 物理 CPU に対して 16 仮想 CPU を割り当て可能)。
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。
DiskFilter	**/etc/nova/nova.conf** ファイルの disk_allocation_ratio を使用して、オーバーコミット比 (物理ディスクに対する仮想ディスクの比率) を超過したホストを除外します。デフォルトの値は **1.0** です (1 物理ディスクに対して 1 仮想ディスクを割り当て可能)。
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。詳しい情報は、「イメージの作成」を参照してください。
IsolatedHostsFilter	**/etc/nova/nova.conf** ファイルで **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

フィルター	説明
	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]'**
MetricFilter	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。
RamFilter	**/etc/nova/nova.conf** ファイルの ram_allocation_ratio を使用して、オーバーコミット比 (物理メモリーに対する仮想メモリーの比率) を超過したホストを除外します。デフォルトの値は **1.5** です (1 物理メモリーに対して 1 仮想メモリーを割り当て可能)。
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 の値がゼロを超える場合に有効です (デフォルトでは、**scheduler_max_attempts=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)。

フィルター	説明
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**

3.5.2. スケジューリングの重みの設定

セルとホストはいずれも、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またはセルが選択されます。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、ホストの **/etc/nova/nova.conf** ファイルで設定することができます (root または nova ユーザーのパーミッションが必要です)。

3.5.2.1. 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通りです。

- **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3.8 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
all	[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)。
アフィニティー	[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。
アフィニティー	[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。
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。

重み付け関数	オプション	説明
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通りです。 » **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。

3.5.2.2. セルの重みオプションの設定

[cells] scheduler_weight_classes オプション (**/etc/nova/nova.conf** ファイル。**root** または **nova** ユーザーのパーミッションが必要) でスケジューラーが使用するセルの重み付け関数を定義します。

注記

本リリースでは、セルの使用機能は **テクノロジープレビュー** として提供されているため、Red Hat では全面的にはサポートしていません。これは、テスト目的のみでご利用いただく機能で、実稼働環境にデプロイすべきではありません。 テクノロジープレビューについての詳しい情報は Scope of Coverage Details を参照してください。

有効な重み付け関数:

- **nova.cells.weights.all_weighers**: すべてのセルの重み付け関数を使用します (デフォルト)。
- **nova.cells.weights.mute_child**: セルが容量や機能の更新をしばらく送信していなかったかどうかによって重み付けします。
- **nova.cells.weights.ram_by_instance_type**: セルの使用可能な RAM を重み付けします。
- **nova.cells.weights.weight_offset**: セルの重みのオフセットを評価します。

注記

セルの重みのオフセットは、**--woffset `in the `nova-manage cell create** コマンドを使用して指定します。

表3.9 セルの重みのオプション

重み付け関数	オプション	説明
mute_child	[cells] **mute_weight_multiplier**	しばらく更新がなかったホストの乗数 (負の浮動小数点)。デフォルトではこの値は **-10.0** です。
mute_child	[cells] **mute_weight_value**	更新がなかったホストに適用される重みの値 (正の浮動小数点)。デフォルトでは、この値は **1000.0** です。
ram_by_instance_type	[cells] **ram_weight_multiplier**	RAM の重み付け乗数 (浮動小数点)。デフォルトではこの値は **1.0** に設定されており、使用可能な全セル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 が低いセルが優先され、インスタンスがセルにスタッキングされます。

重み付け関数	オプション	説明
weight_offset	[cells] **offset_weight_multiplier**	セルを重み付けする乗数 (浮動小数点)。重みのオフセットを **999999999999999** に設定することにより (重みが最も高いセルが優先)、インスタンスが優先するセル (浮動小数点) を指定できるようになります。デフォルトではこの値は **1.0** です。

3.6. インスタンスの退避

停止中またはシャットダウンされたコンピュートノードから同じ環境内の新規ホストサーバーにインスタンスを移動するには (例: サーバーをスワップアウトする必要がある場合など)、**nova evacuate** を使用してそのサーバーを退避することができます。

- 退避は、インスタンスのディスクが共有ストレージ上にある場合またはインスタンスのディスクが Block Storage ボリュームである場合にのみ有用です。それ以外の場合には、ディスクへのアクセスは不可能なので、新規コンピュートノードからアクセスはできません。
- インスタンスは、サーバーがシャットダウンされている場合にのみ退避させることができます。サーバーがシャットダウンされなていない場合には、**evacuate** コマンドは失敗します。

注記

正常に機能するコンピュートノードがある場合には、以下のような操作が可能です。

- バックアップ目的またはインスタンスを別の環境にコピーするために静的な (実行中でない) インスタンスのコピーを作成: **nova image-create** を使用してスナップショットを作成します (「静的なインスタンスの移行」の記事を参照)。
- 静的 (実行中でない) 状態のインスタンスを同じ環境内の別のホストに移動 (共有ストレージが必要): **nova migrate** を使用して移行します (「静的なインスタンスの移行」の記事を参照)。
- 稼働状態 (実行中) のインスタンスを同じ環境内の別のホストに移動: **nova live-migration** コマンドを使用して移行します (「ライブ (実行中の) インスタンスの移行」の記事を参照)。

3.6.1. 単一のインスタンスの退避

1. 以下のコマンドを実行して、インスタンスを 1 つ退避します。

```
# nova evacuate [--password pass] [--on-shared-storage]
instance_name [target_host]
```

–

各オプションについての説明は以下の通りです。

- **--password**: 退避するインスタンスに設定する管理パスワード (**--on-shared-storage** が指定されている場合には使用できません)。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。
- **--on-shared-storage**: インスタンスのファイルがすべて共有ストレージ上にあることを指定します。
- **instance_name**: 退避するインスタンスの名前
- **target_host**: インスタンスの退避先となるホスト。このホストを指定しなかった場合には、Compute のスケジューラーがホストを 1 台選択します。退避先に指定可能なホストを確認するには、以下のコマンドを実行します。

  ```
  # nova host-list | grep compute
  ```

  以下に例を示します。

  ```
  # nova evacuate myDemoInstance Compute2_OnEL7.myDomain
  ```

3.6.2. 全インスタンスの退避

1. 以下のコマンドを実行して、指定したホストに全インスタンスを退避します。

   ```
   # nova host-evacuate instance_name [--target target_host] [--on-shared-storage] source_host
   ```

   各オプションについての説明は以下の通りです。

   - **--target**: インスタンスの退避先となるホスト。このホストを指定しなかった場合には、Compute のスケジューラーがホストを 1 台選択します。退避先に指定可能なホストを確認するには、以下のコマンドを実行します。

     ```
     # nova host-list | grep compute
     ```

   - **--on-shared-storage**: インスタンスのファイルがすべて共有ストレージ上にあることを指定します。
   - **source_host**: 退避されるホストの名前

     以下に例を示します。

     ```
     # nova host-evacuate --target Compute2_OnEL7.localdomain myDemoHost.localdomain
     ```

3.6.3. 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、以下の手順に従って 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、ノードが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 **/etc/nova/nova.conf** ファイルの **state_path** および **instances_path** の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。

1. **コントローラーホストで以下のステップを実行します。**

   a. 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 (このユーザーは、全コントローラー/ノードにおいて同じユーザーである必要があります)。アクセス権はたとえば以下のように表示されます。

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

   b. 以下の行を **/etc/exports** ファイルに追加します。node1_IP および node2_IP は、2 つのコンピュートノードの IP アドレスに置き換えます。

   ```
   /var/lib/nova/instances (rw,sync,fsid=0,no_root_squash)
   /var/lib/nova/instances (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ステップを実行します。**

   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。

   ```
   :/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ます (**/etc/fstab** にリストされている全デバイス)。

   ```
   # 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
```

3.7. 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3.10 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。

3.7.1. 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。以下の例を示します。

```
$ 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。

3.7.2. 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。詳しい情報は **『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。詳しい情報は「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。

3.7.3. スナップショットの状態へのインスタンスの再構築

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「インスタンスの作成」を参照してください。

3.7.4. 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。

3.8. 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。

3.8.1. レスキューモードのインスタンス用のイメージの準備

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。

3.8.1.1. 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。

3.8.2. 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。

3.8.3. 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、**`--rescue_image_ref`** パラメーターを使用します。

   ```
   # nova rescue --rescue_image_ref 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、**`nova.conf`** ファイルの **`shutdown_timeout`** パラメーターで設定することができます。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。

3.8.4. 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。

3.9. 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。

3.9.1. コンフィグドライブのオプション

nova.conf の **[DEFAULT]** で最初のコンフィグドライブオプションを設定します。

- **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=true**: 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。
- **mkisofs_cmd=genisoimage**: 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。

3.9.2. 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。

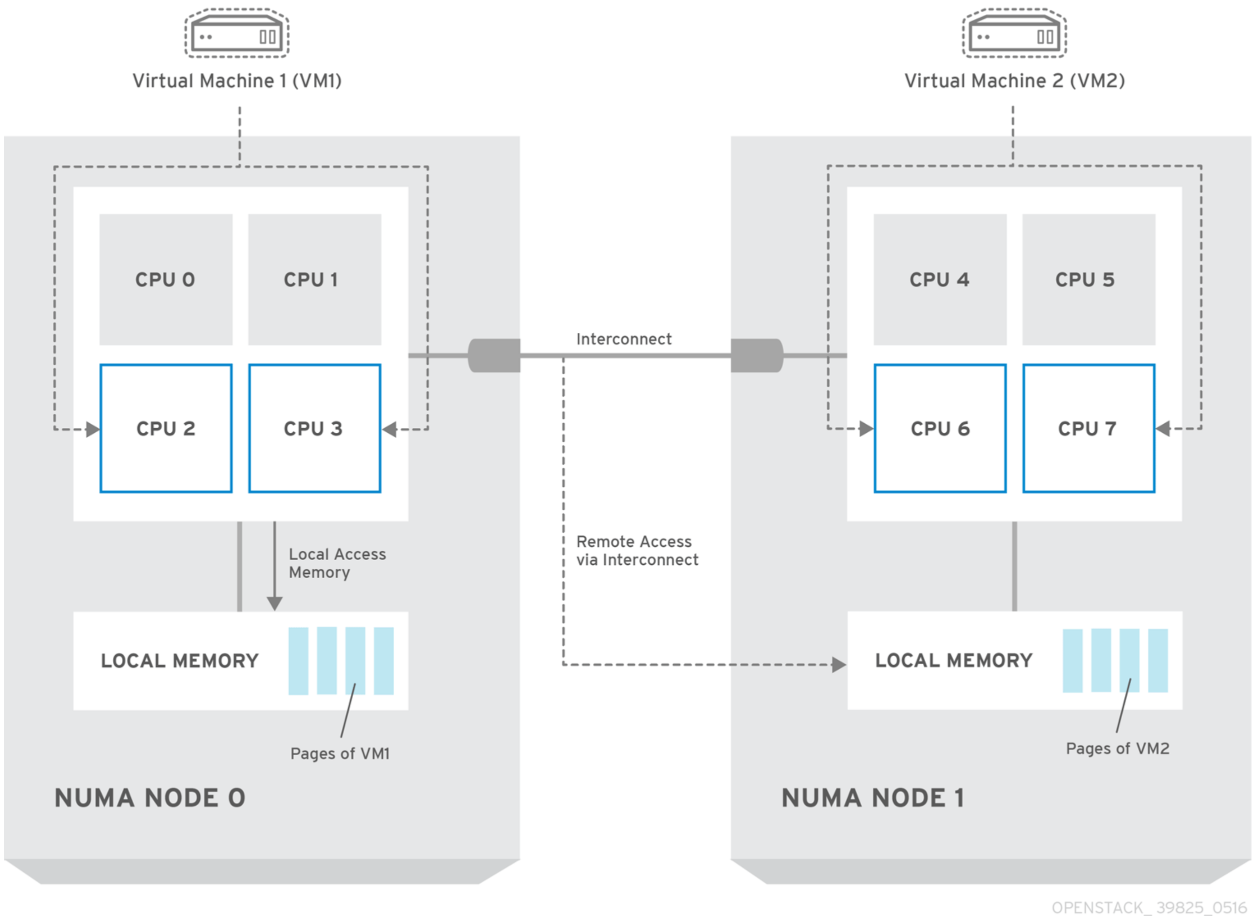
第4章 NUMA ノードを使用する CPU ピニングの設定

本章では、NUMA トポロジーのサポートと、同技術に対応したシステム上における OpenStack 環境の設定について説明します。この構成では、仮想マシンを専用の CPU コアに固定化 (ピニング) することにより、よりスマートなスケジューリングが可能となり、ゲスト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 ?」の記事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 VM 1 に NUMA ノード 1 の CPU コアがある場合 **のみ** アクセスされます。このような場合には、NUMA ノード 1 のメモリーは、VM 1 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 VM 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、『仮想化のチューニングと最適化ガイド』を参照してください。

> **警告**
>
> 現在、CPU ピニングを使用するように設定されているインスタンスは移行できません。この問題に関する詳しい情報は、「Instance migration fails when using cpu-pinning from a numa-cell and flavor-property "hw:cpu_policy=dedicated"」のソリューションを参照してください。

4.1. 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が、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 コアにゲスト仮想マシンインスタンスを処理させるようにする必要があります。たとえば、2 つの NUMA ノード全体に 8 つの CPU コアを均等に分散させる場合には、そのレイアウトは以下の表に示したようになります。

表4.1 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
ゲストのプロセス	コア 2	コア 3	コア 6	コア 7

> **注記**
>
> ホストのプロセス用に確保するコア数は、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決定する必要があります。

コンピュートノードの設定は、以下の手順に従って行います。

1. **/etc/nova/nova.conf** ファイルの **vcpu_pin_set** オプションに、ゲストのプロセス用に確保する CPU コアの一覧を設定します。上記の例を使用する場合、設定は以下のようになります。

   ```
   vcpu_pin_set=2,3,6,7
   ```

   vcpu_pin_set オプションを設定すると、以下のような**cpuset** 属性が libvirt の XML 設定ファイルにも追加されます。

   ```
   <vcpu placement='static' cpuset='2-3,6-7'>1</vcpu>
   ```

これにより、ゲストの仮想 CPU は一覧に設定されている物理 CPU コアに固定され、スケジューラーにはそれらのコアだけが見えるようになります。

2. 同じファイルの **reserved_host_memory_mb** オプションに、ホストのプロセス用に確保するメモリー容量を指定します。512 MB を確保する場合には、設定は以下のようになります。

```
reserved_host_memory_mb=512
```

3. 以下のコマンドを実行して、コンピュートノードで Compute サービスを再起動します。

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

4. システムのブート設定に **isolcpus** の引数を追加して、ホストのプロセスがゲストプロセス用に確保されている CPU コアで実行されないようにします。この引数のパラメーターとして、ゲストプロセスに確保されている CPU コアの一覧を使用します。上記の例のトポロジーを使用する場合には、以下のようなコマンドを実行します。

```
grubby --update-kernel=ALL --args="isolcpus=2,3,6,7"
```

注記

cpuset オプションを **isolcpus** カーネル引数と共に使用することにより、下層のコンピュートノードは、対応する物理 CPU を自らは使用しないようになります。物理 CPU はインスタンス専用となります。

5. ブートレコードを更新して、変更を有効にします。

```
grub2-install /dev/device
```

device は、ブートレーコードが含まれているデバイスの名前に置き換えます。通常は、**sda** です。

6. システムを再起動します。

4.2. スケジューラーの設定

1. OpenStack Compute Scheduler を実行している各システムの **/etc/nova/nova.conf** ファイルを編集します。**scheduler_default_filters** オプションを確認し、コメントアウトされている場合には、コメント解除して、フィルターのリストに **AggregateInstanceExtraSpecFilter** と **NUMATopologyFilter** を追加します。行全体は以下のようになります。

```
scheduler_default_filters=RetryFilter,AvailabilityZoneFilter,RamFilter,
ComputeFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter,CoreFilter,
NUMATopologyFilter,AggregateInstanceExtraSpecsFilter
```

2. openstack-nova-scheduler サービスを再起動します。

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

4.3. アグリゲートとフレーバーの設定

システム上で Compute のコマンドラインインターフェース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。

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 name
   ```

   name は **performance** や **cpu_pinning** などの適切な名前に置き換えます。

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 1 pinned=true
   ```

   このコマンドで、数字の **1** は、前のステップで作成したアグリゲートの ID に置き換えます。

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 name
   ```

   name は、適切な名前 (例: **normal**) に置き換えます。

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 2 pinned=false
   ```

   このコマンドでは、数字の **2** を使用しています。これは、最初のアグリゲートの ID **1** の後で作成されたアグリゲートの ID を指定するためです。

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 name ID RAM disk vCPUs
   ```

   name は適切な名前 (例: **m1.small.performance**、**pinned.small** など)、**ID** は新規フレーバーの識別子 (標準のフレーバーが 5 つある場合には **6**、**nova** が UUID を生成するようにするには **auto** を指定)、**RAM** は指定するメモリー容量 (MB 単位) 、**disk** は指定するディスク容量 (GB 単位) 、**vCPUs** は確保する仮想 CPU 数に置き換えます。

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 vCPU を配置します。

    ```
    nova flavor-key ID set hw:cpu_policy=dedicated
    nova flavor-key ID set hw:cpu_thread_policy=require
    ```

    ID は、前のステップで作成したフレーバーの ID に置き換えます。

    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、ホストに SMT アーキテクチャーがどうしてもない場合には、**hw:cpu_thread_policy** の仕様を使わないか、**require** ではなく **prefer** に設定してください。デフォルトの **prefer** ポリシーでは、利用できる場合はスレッドシブリングが使用されるようにします。

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 ID set aggregate_instance_extra_specs:pinned=true
    ```

    この場合にも、**ID** をフレーバーの ID に置き換えます。

10. 新規アグリゲートにホストを追加します。

    ```
    nova aggregate-add-host ID_1 host_1
    ```

    ID_1 は最初の (「パフォーマンス」/「ピニング」用) アグリゲートの ID に、**host_1** はアグリゲートに追加するホストのホスト名に置き換えます。

    ```
    nova aggregate-add-host ID_2 host_2
    ```

    ID_2 は 2 番目の ID (「通常」の) アグリゲートの ID に、**host_2** はアグリゲートに追加するホストのホスト名に置き換えます。

これで新規フレーバーを使用してインスタンスをブートできるようになりました。

```
nova boot --image image --flavor flavor server_name
```

image は保存した仮想マシンイメージの名前に (**nova image-list** を参照)、**flavor** はフレーバー名に (**m1.small.performance**、**pinned.small**、または使用したその他の名前)、 **server_name** は新規サーバーの名前に置き換えます。

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
```

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。詳しい説明は、「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

対象コンポーネント	キー	説明	サポートされている値
all	アーキテクチャー	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 (例: **x86_64**、**arm**、**ppc64**)。**uname -m** を実行してマシンのアーキテクチャーを確認します。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-bit RISC armv7l-ARM Cortex-A7 MPCore cris-Ethernet, Token Ring, AXis-Code Reduced Instruction Set i686-Intel sixth-generation 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 ppc64-PowerPC 64 ビット ppcemb-PowerPC (Embedded 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

対象コンポーネント	キー	説明	サポートされている値
			» xtensaeb: Tensilica Xtensa 構成可能マイクロプロセッサーコア (Big Endian)
all	hypervisor_type	ハイパーバイザーのタイプ	**`kvm`**、**`vmware`**
all	instance_uuid	スナップショットイメージの場合に、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
all	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

対象コンポーネント	キー	説明	サポートされている値
all	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一般名 (小文字。libosinfo project と同じデータボキャブラリーを使用)。このフィールドで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、非推奨の値を以下にリストします。	- 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。 - 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
all	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」)
all	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
all	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`**
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。

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_mempolicy	NUMA のメモリー割り当てポリシー (フレーバーの定義はオーバーライドしません)	「strict」に設定すると、インスタンスのメモリーが、バインディングされている NUMA ノードから割り当てられます (uma_nodes が指定されている場合にはデフォルト) 。「preferred」に設定すると、カーネルは別のノードを使用してフォールバックすることが可能となります。これは、「hw:numa_nodes」パラメーターが「1」に設定されている場合に有用です。
libvirt API ドライバー	hw_numa_cpus.0	vCPU 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
libvirt API ドライバー	hw_numa_cpus.1	vCPU 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 GB 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 GB 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

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_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**
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、nova.conf ファイルで「rng_dev_path=/dev/hwrng」のオプションを使用します。	**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 とは、より高いスケーラビリティーとパフォーマンスを提供する、高度な SCSI ハードウェア対応の準仮想化 SCSI コントローラーデバイスです。	**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。	thinkvirt.com 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
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**