



# Red Hat OpenStack Platform 13

## Instances and Images Guide

Managing Instances and Images



# Red Hat OpenStack Platform 13 Instances and Images Guide

---

Managing Instances and Images

OpenStack Team  
rhos-docs@redhat.com

## Legal Notice

Copyright © 2022 Red Hat, Inc.

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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js<sup>®</sup> 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<sup>®</sup> 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.

## Abstract

The Instances and Images guide provides procedures for the management of instances, images of a Red Hat OpenStack Platform environment.

# Table of Contents

<b>MAKING OPEN SOURCE MORE INCLUSIVE</b> .....	<b>6</b>
<b>CHAPTER 1. IMAGE SERVICE</b> .....	<b>7</b>
1.1. UNDERSTANDING AND OPTIMIZING THE IMAGE SERVICE	7
1.1.1. Supported Image service (glance) back ends	7
1.1.2. Image signing and verification	8
1.1.3. Image conversion	9
1.1.4. Image introspection	9
1.1.5. Interoperable image import	10
1.2. MANAGING IMAGES	10
1.2.1. Creating an image	10
1.2.1.1. Using a KVM guest image with Red Hat OpenStack Platform	10
1.2.1.2. Creating custom Red Hat Enterprise Linux or Windows images	11
1.2.1.2.1. Creating a Red Hat Enterprise Linux 7 image	12
1.2.1.2.2. Creating a Red Hat Enterprise Linux 6 image	14
1.2.1.2.3. Creating a Windows image	17
1.2.2. Uploading an image	19
1.2.3. Updating an image	20
1.2.4. Importing an image	21
1.2.4.1. Importing from a remote URI	21
1.2.4.2. Importing from a local volume	21
1.2.5. Deleting an image	22
1.2.6. Enabling image conversion	22
1.2.7. Converting an image to RAW format	22
1.2.7.1. Configuring the Image service to accept only RAW and ISO	23
1.2.8. Storing an image in RAW format	23
<b>CHAPTER 2. CONFIGURING THE COMPUTE (NOVA) SERVICE</b> .....	<b>25</b>
2.1. CONFIGURING MEMORY FOR OVERALLOCATION	26
2.2. CALCULATING RESERVED HOST MEMORY ON COMPUTE NODES	26
2.3. CALCULATING SWAP SIZE	27
<b>CHAPTER 3. CONFIGURING OPENSTACK COMPUTE STORAGE</b> .....	<b>28</b>
3.1. ARCHITECTURE OVERVIEW	28
3.2. CONFIGURATION	29
3.3. ENABLING SERVICE TOKENS BETWEEN THE COMPUTE SERVICE AND THE BLOCK STORAGE SERVICE	31
<b>CHAPTER 4. VIRTUAL MACHINE INSTANCES</b> .....	<b>34</b>
4.1. MANAGING INSTANCES	34
4.1.1. Adding components	34
4.1.2. Launching an instance	34
4.1.2.1. Launching instance options	35
4.1.3. Updating an instance	37
4.1.4. Resizing an instance	38
4.1.5. Connecting to an instance	39
4.1.5.1. Accessing an instance console by using the dashboard	39
4.1.5.2. Accessing an instance console by using the CLI	40
4.1.6. Viewing instance usage	40
4.1.7. Deleting an instance	40
4.1.8. Managing multiple instances simultaneously	41
4.2. MANAGING INSTANCE SECURITY	41

4.2.1. Managing key pairs	41
4.2.1.1. Creating a key pair	41
4.2.1.2. Importing a key pair	41
4.2.1.3. Deleting a key pair	42
4.2.2. Creating a security group	42
4.2.3. Creating, assigning, and releasing floating IP addresses	42
4.2.3.1. Allocating a floating IP to the project	42
4.2.3.2. Assigning a floating IP	42
4.2.3.3. Releasing a floating IP	43
4.2.4. Logging in to an instance	43
4.2.5. Injecting an admin password into an instance	44
4.3. MANAGING FLAVORS	45
4.3.1. Updating configuration permissions	46
4.3.2. Creating a flavor	46
4.3.3. Updating general attributes	47
4.3.4. Updating flavor metadata	47
4.3.4.1. Viewing metadata	47
4.3.4.2. Adding metadata	47
4.4. MANAGING HOST AGGREGATES	52
4.4.1. Enabling host aggregate scheduling	53
4.4.2. Viewing availability zones or host aggregates	53
4.4.3. Adding a host aggregate	53
4.4.4. Updating a host aggregate	54
4.4.5. Deleting a host aggregate	54
4.5. SCHEDULING HOSTS	55
4.5.1. Configuring scheduling filters	56
4.5.2. Configuring scheduling weights	59
4.5.3. Reserving NUMA nodes with PCI devices	65
4.6. MANAGING INSTANCE SNAPSHOTS	65
4.6.1. Creating an instance snapshot	66
4.6.2. Managing a snapshot	67
4.6.3. Rebuilding an instance to a state in a snapshot	67
4.6.4. Consistent snapshots	67
4.7. USING RESCUE MODE FOR INSTANCES	68
4.7.1. Preparing an image for a rescue mode instance	68
4.7.1.1. Rescuing an image that uses ext4 file system	68
4.7.2. Adding the rescue image to the OpenStack Image service	69
4.7.3. Launching an instance in rescue mode	69
4.7.4. Unrescuing an instance	70
4.8. CREATING A CUSTOMIZED INSTANCE	70
4.8.1. Customizing an instance by using user data	71
4.8.2. Customizing an instance by using metadata	72
4.8.3. Customizing an instance by using a config drive	72
<b>CHAPTER 5. MIGRATING VIRTUAL MACHINE INSTANCES BETWEEN COMPUTE NODES</b>	<b>74</b>
5.1. MIGRATION TYPES	74
5.2. MIGRATION CONSTRAINTS	76
5.3. PREPARING TO MIGRATE	78
5.4. ADDITIONAL PREPARATION FOR DPDK INSTANCES	78
5.5. COLD MIGRATING AN INSTANCE	80
5.6. LIVE MIGRATING AN INSTANCE	81
5.7. CHECKING MIGRATION STATUS	82
5.8. COMPLETING THE MIGRATION	83

5.9. EVACUATING AN INSTANCE	84
5.9.1. Evacuating one instance	84
5.9.2. Evacuating all instances on a host	85
5.9.3. Configuring shared storage	85
5.10. TROUBLESHOOTING MIGRATION	87
5.10.1. Errors during migration	87
5.10.2. Never-ending live migration	87
5.10.3. Instance performance degrades after migration	88
<b>CHAPTER 6. CONFIGURING PCI PASSTHROUGH</b>	<b>90</b>
6.1. DESIGNATING COMPUTE NODES FOR PCI PASSTHROUGH	90
6.2. CONFIGURING A PCI PASSTHROUGH COMPUTE NODE	92
6.3. PCI PASSTHROUGH DEVICE TYPE FIELD	95
6.4. GUIDELINES FOR CONFIGURING NOVAPCIPASSTHROUGH	95
<b>CHAPTER 7. DATABASE CLEANING</b>	<b>96</b>
7.1. CONFIGURING DATABASE MANAGEMENT	96
7.2. CONFIGURATION OPTIONS FOR OPENSTACK COMPUTE (NOVA) AUTOMATED DATABASE MANAGEMENT	96
<b>CHAPTER 8. CONFIGURING COMPUTE NODES FOR PERFORMANCE</b>	<b>100</b>
8.1. CONFIGURING CPU PINNING WITH NUMA	100
8.1.1. Compute node configuration	101
8.1.2. Configuring emulator threads to run on dedicated physical CPU	102
8.1.3. Scheduler configuration	103
8.1.4. Aggregate and flavor configuration	104
8.2. CONFIGURING HUGE PAGES ON THE COMPUTE NODE	105
8.2.1. Allocating huge pages to instances	107
<b>CHAPTER 9. ADDING METADATA TO INSTANCES</b>	<b>108</b>
9.1. TYPES OF INSTANCE METADATA	108
9.2. ADDING A CONFIG DRIVE TO ALL INSTANCES	108
9.3. ADDING STATIC METADATA TO INSTANCES	110
9.4. ADDING DYNAMIC METADATA TO INSTANCES	110
<b>CHAPTER 10. CONFIGURING REAL-TIME COMPUTE</b>	<b>112</b>
10.1. PREPARING YOUR COMPUTE NODES FOR REAL-TIME	112
10.2. DEPLOYING THE REAL-TIME COMPUTE ROLE	115
10.3. SAMPLE DEPLOYMENT AND TESTING SCENARIO	117
10.4. LAUNCHING AND TUNING REAL-TIME INSTANCES	119
<b>CHAPTER 11. CONFIGURING VIRTUAL GPUS FOR INSTANCES</b>	<b>121</b>
11.1. SUPPORTED CONFIGURATIONS AND LIMITATIONS	121
11.2. CONFIGURING VGPU ON THE COMPUTE NODES	121
11.2.1. Building a custom GPU overcloud image	122
11.2.2. Designating Compute nodes for vGPU	124
11.2.3. Configuring the Compute node for vGPU and deploying the overcloud	126
11.3. CREATING THE VGPU IMAGE AND FLAVOR	128
11.3.1. Creating a custom GPU instance image	128
11.3.2. Creating a vGPU flavor for instances	128
11.3.3. Launching a vGPU instance	129
11.4. ENABLING PCI PASSTHROUGH FOR A GPU DEVICE	130
<b>APPENDIX A. IMAGE CONFIGURATION PARAMETERS</b>	<b>133</b>

APPENDIX B. ENABLING THE LAUNCH INSTANCE WIZARD ..... 144





## MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

# CHAPTER 1. IMAGE SERVICE

You can manage images and storage in Red Hat OpenStack Platform (RHOSP).

A virtual machine image is a file that contains a virtual disk with a bootable operating system installed. Virtual machine images are supported in different formats. The following formats are available in RHOSP:

- **RAW** - Unstructured disk image format.
- **QCOW2** - Disk format supported by QEMU emulator. This format includes QCOW2v3 (sometimes referred to as QCOW3), which requires QEMU 1.1 or higher.
- **ISO** - Sector-by-sector copy of the data on a disk, stored in a binary file.
- **AKI** - Indicates an Amazon Kernel Image.
- **AMI** - Indicates an Amazon Machine Image.
- **ARI** - Indicates an Amazon RAMDisk Image.
- **VDI** - Disk format supported by VirtualBox virtual machine monitor and the QEMU emulator.
- **VHD** - Common disk format used by virtual machine monitors from VMware, VirtualBox, and others.
- **VMDK** - Disk format supported by many common virtual machine monitors.
- **PLOOP** - A disk format supported and used by Virtuozzo to run OS containers.
- **OVA** - Indicates that what is stored in the Image service (glance) is an OVA tar archive file.
- **DOCKER** - Indicates that what is stored in the Image service (glance) is a Docker tar archive of the container file system.

Because **ISO** files contain bootable file systems with an installed operating system, you can use **ISO** files in the same way that you use other virtual machine image files.

To download the official Red Hat Enterprise Linux cloud images, your account must have a valid Red Hat Enterprise Linux subscription:

- [Red Hat Enterprise Linux 8 KVM Guest Image](#)
- [Red Hat Enterprise Linux 7 KVM Guest Image](#)
- [Red Hat Enterprise Linux 6 KVM Guest Image](#)

If you are not logged in to the Customer Portal, a prompt opens where you must enter your Red Hat account credentials.

## 1.1. UNDERSTANDING AND OPTIMIZING THE IMAGE SERVICE

You can use the following Red Hat OpenStack Platform (RHOSP) Image service (glance) features to manage and optimize images and storage in your RHOSP deployment.

### 1.1.1. Supported Image service (glance) back ends

The following Image service (glance) back end scenarios are supported:

- RBD is the default back end when you use Ceph. For more information, see [Configuring Ceph Storage](#) in the *Advanced Opencloud Customization* guide.
- Object Storage (swift). For more information, see [Using an External Object Storage Cluster](#) in the *Advanced Opencloud Customization* guide.
- Block Storage (cinder). For more information, see [Configuring cinder back end for the Image service](#) in the *Advanced Opencloud Customization* guide.

#### Note

The Image service uses the Block Storage type and back end as the default.

- NFS. For more information, see [Configuring NFS Storage](#) in the *Advanced Opencloud Customization* guide.



### IMPORTANT

Although NFS is a supported Image service deployment option, more robust options are available.

NFS is not native to the Image service. When you mount an NFS share on the Image service, the Image service does not manage the operation. The Image service writes data to the file system but is unaware that the back end is an NFS share.

In this type of deployment, the Image service cannot retry a request if the share fails. This means that when a failure occurs on the back end, the store might enter read-only mode, or it might continue to write data to the local file system, in which case you risk data loss. To recover from this situation, you must ensure that the share is mounted and in sync, and then restart the Image service. For these reasons, Red Hat does not recommend NFS as an Image service back end.

However, if you do choose to use NFS as an Image service back end, some of the following best practices can help to mitigate risks:

- Use a production-grade NFS back end.
- Ensure that a Layer 2 connection is established between Controller nodes and the NFS back end.
- Include monitoring and alerts for the mounted share.
- Set underlying FS permissions.
  - Ensure that the user and the group that the glance-api process runs on do not have write permissions on the mount point at the local file system. This means that the process can detect possible mount failure and put the store into read-only mode during a write attempt.
  - The write permissions must be present in the shared file system that you use as a store.

## 1.1.2. Image signing and verification

Image signing and verification protects image integrity and authenticity by enabling deployers to sign images and save the signatures and public key certificates as image properties.

By taking advantage of this feature, you can:

- Sign an image using your private key and upload the image, the signature, and a reference to your public key certificate (the verification metadata). The Image service then verifies that the signature is valid.
- Create an image in the Compute service, have the Compute service sign the image, and upload the image and its verification metadata. The Image service again verifies that the signature is valid.
- Request a signed image in the Compute service. The Image service provides the image and its verification metadata, allowing the Compute service to validate the image before booting it.

For information on image signing and verification, refer to the [Validate Glance Images](#) chapter of the *Manage Secrets with OpenStack Key Manager Guide* .

### 1.1.3. Image conversion

Image conversion converts images by calling the task API while importing an image.

As part of the import workflow, a plugin provides the image conversion. This plugin can be activated or deactivated based on the deployer configuration. Therefore, the deployer needs to specify the preferred format of images for the deployment.

Internally, the Image service receives the bits of the image in a particular format. These bits are stored in a temporary location. The plugin is then triggered to convert the image to the target format and move it to a final destination. When the task is finished, the temporary location is deleted. As a result, the format uploaded initially is not retained by the Image service.

For more information about image conversion, see [Enabling image conversion](#).



#### NOTE

You can trigger the conversion only when you import an image. Conversion does not run when you upload an image. For example:

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

### 1.1.4. Image introspection

Every image format comes with a set of metadata embedded inside the image itself. For example, a stream optimized **vmdk** would contain the following parameters:

```
$ head -20 so-disk.vmdk
# Disk DescriptorFile
```

```

version=1
CID=d5a0bce5
parentCID=ffffff
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"

```

By introspecting this **vmdk**, you can know that the **disk\_type** is **streamOptimized**, and the **adapter\_type** is **buslogic**. These metadata parameters are useful for the consumer of the image. In Compute, the workflow to instantiate a **streamOptimized** disk is different from the one to instantiate a **flat** disk. This new feature allows metadata extraction. You can achieve image introspection by calling the task API while you import the image. An administrator can override metadata settings.

### 1.1.5. Interoperable image import

The OpenStack Image service (glance) provides two methods to import images by using the interoperable image import workflow:

- **web-download** (default) for importing images from a URI
- **glance-direct** for importing from a local file system

## 1.2. MANAGING IMAGES

The OpenStack Image service (glance) provides discovery, registration, and delivery services for disk and server images. It provides the ability to copy or snapshot a server image, and immediately store it. You can use stored images as a template to get new servers up and running quickly and more consistently than installing a server operating system and individually configuring services.

### 1.2.1. Creating an image

Manually create Red Hat OpenStack Platform (RHOSP) compatible images in the QCOW2 format by using Red Hat Enterprise Linux 7 ISO files, Red Hat Enterprise Linux 6 ISO files, or Windows ISO files.

#### 1.2.1.1. Using a KVM guest image with Red Hat OpenStack Platform

You can use a ready RHEL KVM guest QCOW2 image:

- [Red Hat Enterprise Linux 8 KVM Guest Image](#)
- [Red Hat Enterprise Linux 7 KVM Guest Image](#)
- [Red Hat Enterprise Linux 6 KVM Guest Image](#)

These images are configured with **cloud-init** and must take advantage of ec2-compatible metadata services for provisioning SSH keys to function properly.

Ready Windows KVM guest QCOW2 images are not available.



## NOTE

For the KVM guest images:

- The **root** account in the image is disabled, but **sudo** access is granted to a special user named **cloud-user**.
- There is no **root** password set for this image.

The **root** password is locked in **/etc/shadow** by placing **!!** in the second field.

For a RHOSP instance, it is recommended that you generate an ssh keypair from the RHOSP dashboard or command line and use that key combination to perform an SSH public authentication to the instance as root.

When the instance is launched, this public key is injected to it. You can then use the private key you downloaded while you created the keypair to authenticate.

If you do not want to use keypairs, you can use the **admin** password that you can set in the procedure to inject an **admin** password, see [Injecting an admin password into an instance](#).

If you want to create custom Red Hat Enterprise Linux or Windows images, see:

- [Create a Red Hat Enterprise Linux 7 Image](#)
- [Create a Red Hat Enterprise Linux 6 Image](#)
- [Create a Windows Image](#)

### 1.2.1.2. Creating custom Red Hat Enterprise Linux or Windows images

#### Prerequisites

- Linux host machine to create an image. This can be any machine on which you can install and run the Linux packages.
- `libvirt`, `virt-manager` (run command **yum groupinstall -y @virtualization**). This installs all packages necessary to create a guest operating system.
- `Libguestfs` tools (run command **yum install -y libguestfs-tools-c**). This installs a set of tools to access and modify virtual machine images.
- A Red Hat Enterprise Linux 7 or 6 ISO file (see [RHEL 7.2 Binary DVD](#) or [RHEL 6.8 Binary DVD](#)) or a Windows ISO file. If you do not have a Windows ISO file, visit the [Microsoft TechNet Evaluation Center](#) and download an evaluation image.
- A text editor if you want to change the **kickstart** files (RHEL only).



## IMPORTANT

If you install the **libguestfs-tools** package on the undercloud, disable **iscsid.socket** to avoid port conflicts with the **tripleo\_iscsid** service on the undercloud:

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



## NOTE

In the following procedures, you must run all commands with the **[root@host]#** prompt on your host machine.

### 1.2.1.2.1. Creating a Red Hat Enterprise Linux 7 image

Manually create a Red Hat OpenStack Platform (RHOSP) compatible image in the QCOW2 format by using a Red Hat Enterprise Linux 7 ISO file.

#### Procedure

1. Start the installation using **virt-install**:

```
[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
```

This launches an instance and starts the installation process.



## NOTE

If the instance does not launch automatically, run the **virt-viewer** command to view the console:

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

2. Configure the virtual machine as follows:
  - a. At the initial Installer boot menu, choose the **Install Red Hat Enterprise Linux 7.X** option.
  - b. Choose the appropriate **Language** and **Keyboard** options.
  - c. When prompted about which type of devices your installation uses, choose **Auto-detected installation media**.
  - d. When prompted about which type of installation destination, choose **Local Standard Disks**. For other storage options, choose **Automatically configure partitioning**.
  - e. For software selection, choose **Minimal Install**.
  - f. For network and host name, choose **eth0** for network and choose a **hostname** for your device. The default host name is **localhost.localdomain**.



- g. Choose the **root** password. The installation process completes and the **Complete!** screen appears.
3. After the installation is complete, reboot the instance and log in as the root user.
4. Update the **/etc/sysconfig/network-scripts/ifcfg-eth0** file so that it contains only the following values:

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

5. Reboot the machine.
6. Register the machine with the Content Delivery Network.

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

7. Update the system:

```
# yum -y update
```

8. Install the **cloud-init** packages:

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

9. Edit the **/etc/cloud/cloud.cfg** configuration file and under **cloud\_init\_modules** add:

```
- resolv-conf
```

The **resolv-conf** option automatically configures the **resolv.conf** when an instance boots for the first time. This file contains information related to the instance such as **nameservers**, **domain** and other options.

10. Add the following line to **/etc/sysconfig/network** to avoid problems accessing the EC2 metadata service:

```
NOZEROCONF=yes
```

11. To ensure the console messages appear in the **Log** tab on the dashboard and the **nova console-log** output, add the following boot option to the **/etc/default/grub** file:

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

Run the **grub2-mkconfig** command:

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

The output is as follows:

```
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
```

- Un-register the virtual machine so that the resulting image does not contain the same subscription details for every instance cloned based on it:

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

- Power off the instance:

```
# poweroff
```

- Use the **virt-sysprep** command to reset and clean the image so that it can be used to create instances without issues:

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

- Reduce image size by using the **virt-sparsify** command. This command converts any free space within the disk image back to free space within the host:

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

This creates a new **rhel7-cloud.qcow2** file in the location from where the command is run.

The **rhel7-cloud.qcow2** image file is ready to be uploaded to the Image service. For more information about using the dashboard to upload this image to your RHOSP deployment, see [Upload an Image](#).

#### 1.2.1.2.2. Creating a Red Hat Enterprise Linux 6 image

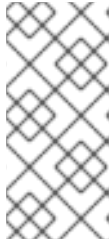
Manually create a Red Hat OpenStack Platform (RHOSP) compatible image in the QCOW2 format by using a Red Hat Enterprise Linux 6 ISO file.

##### Procedure

- Use **virt-install** to start the installation:

```
[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
```

This launches an instance and starts the installation process.

**NOTE**

If the instance does not launch automatically, run the **virt-viewer** command to view the console:

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

2. Configure the virtual machines as follows:
  - a. At the initial Installer boot menu, choose the **Install or upgrade an existing system** option. Follow the installation prompts. Accept the defaults. The installer checks for the disc and lets you decide whether you want to test your installation media before installation. Select **OK** to run the test or **Skip** to proceed without testing.
  - b. Choose the appropriate **Language** and **Keyboard** options.
  - c. When prompted about which type of devices your installation uses, choose **Basic Storage Devices**.
  - d. Choose a **hostname** for your device. The default host name is **localhost.localdomain**.
  - e. Set **timezone** and **root** password.
  - f. Based on the space on the disk, choose the type of installation.
  - g. Choose the **Basic Server** install, which installs an SSH server.
  - h. The installation process completes and **Congratulations, your Red Hat Enterprise Linux installation is complete** screen appears.
3. Reboot the instance and log in as the **root** user.
4. Update the **/etc/sysconfig/network-scripts/ifcfg-eth0** file so it only contains the following values:

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

5. Reboot the machine.
6. Register the machine with the Content Delivery Network:

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

7. Update the system:

```
# yum -y update
```

8. Install the **cloud-init** packages:

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

9. Edit the `/etc/cloud/cloud.cfg` configuration file and under `cloud_init_modules` add:

```
- resolv-conf
```

The `resolv-conf` option automatically configures the `resolv.conf` configuration file when an instance boots for the first time. This file contains information related to the instance such as `nameservers`, `domain`, and other options.

10. To prevent network issues, create the `/etc/udev/rules.d/75-persistent-net-generator.rules` file as follows:

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

This prevents `/etc/udev/rules.d/70-persistent-net.rules` file from being created. If `/etc/udev/rules.d/70-persistent-net.rules` is created, networking might not function correctly when booting from snapshots (the network interface is created as `eth1` rather than `eth0` and IP address is not assigned).

11. Add the following line to `/etc/sysconfig/network` to avoid problems accessing the EC2 metadata service:

```
NOZEROCONF=yes
```

12. To ensure the console messages appear in the **Log** tab on the dashboard and the **nova console-log** output, add the following boot option to the `/etc/grub.conf`:

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

13. Un-register the virtual machine so that the resulting image does not contain the same subscription details for every instance cloned based on it:

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

14. Power off the instance:

```
# poweroff
```

15. Use the `virt-sysprep` command to reset and clean the image so that it can be used to create instances without issues:

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

16. Reduce image size by using the `virt-sparsify` command. This command converts any free space within the disk image back to free space within the host:

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

This creates a new `rhel6-cloud.qcow2` file in the location from where the command is run.

**NOTE**

You must manually resize the partitions of instances based on the image in accordance with the disk space in the flavor that is applied to the instance.

The **rhel6-cloud.qcow2** image file is ready to upload to the Image service. For more information about using the dashboard to upload this image to your RHOSP deployment, see [Upload an Image](#)

**1.2.1.2.3. Creating a Windows image**

Manually create a Red Hat OpenStack Platform (RHOSP) compatible image in the QCOW2 format by using a Windows ISO file.

**Procedure**

1. Use **virt-install** to start the installation:

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

Replace the values of the **virt-install** parameters as follows:

- **<name>** – the name of the Windows guest.
- **<size>** – disk size in GB.
- **<path>** – the path to the Windows installation ISO file.
- **<RAM>** – the requested amount of RAM in MB.

**NOTE**

The **--os-type=windows** parameter ensures that the clock is configured correctly for the Windows guest, and enables its Hyper-V enlightenment features.

**virt-install** saves the guest image as **/var/lib/libvirt/images/<name>.qcow2** by default. If you want to keep the guest image elsewhere, change the parameter of the **--disk** option as follows:

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

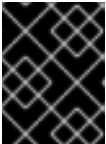
Replace **<filename>** with the name of the file that stores the guest image, and optionally its path, for example, **path=win8.qcow2,size=8** creates an 8 GB file named **win8.qcow2** in the current working directory.

**TIP**

If the guest does not launch automatically, run the **virt-viewer** command to view the console:

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

2. Installation of Windows systems is beyond the scope of this document. For instructions about how to install Windows, see the relevant Microsoft documentation.
3. To allow the newly installed Windows system to use the virtualized hardware, you might need to install **virtio drivers** in it. To do so, first install the **virtio-win** package on the host system. This package contains the virtio ISO image, which you must attach as a CD-ROM drive to the Windows guest. See [Chapter 8. KVM Para-virtualized \(virtio\) Drivers](#) in the *Virtualization Deployment and Administration Guide* for detailed instructions on how to install the **virtio-win** package, add the virtio ISO image to the guest, and install the virtio drivers.
4. To complete the configuration, download and execute [Cloudbase-Init](#) on the Windows system. At the end of the installation of Cloudbase-Init, select the **Run Sysprep** and **Shutdown** check boxes. The **Sysprep** tool makes the guest unique by generating an OS ID, which certain Microsoft services use.

**IMPORTANT**

Red Hat does not provide technical support for Cloudbase-Init. If you encounter an issue, [contact Cloudbase Solutions](#).

When the Windows system shuts down, the **<name>\_qcow2** image file is ready to upload to the Image service. For more information about using the dashboard or the command line to upload this image to your RHOSP deployment, see [Uploading an Image](#).



## NOTE

### libosinfo data

The Compute service has deprecated support for using **libosinfo** data to set default device models. Instead, use the following image metadata properties to configure the optimal virtual hardware for an instance:

- **os\_distro**
- **os\_version**
- **hw\_cdrom\_bus**
- **hw\_disk\_bus**
- **hw\_scsi\_model**
- **hw\_vif\_model**
- **hw\_video\_model**
- **hypervisor\_type**

For more information about these metadata properties, see [Appendix A, Image configuration parameters](#).

## 1.2.2. Uploading an image

### Procedure

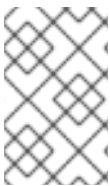
1. In the dashboard, select **Project > Compute > Images**
2. Click **Create Image**.
3. Complete the values, and click **Create Image** when finished.

**Table 1.1. Image options**

Field	Notes
Name	Name for the image. The name must be unique within the project.
Description	Brief description to identify the image.
Image Source	Image source: <b>Image Location</b> or <b>Image File</b> . Based on your selection, the next field is displayed.
Image Location or Image File	<ul style="list-style-type: none"> <li>• Select <b>Image Location</b> option to specify the image location URL.</li> <li>• Select <b>Image File</b> option to upload an image from the local disk.</li> </ul>
Format	Image format (for example, qcow2).

Field	Notes
Architecture	Image architecture. For example, use i686 for a 32-bit architecture or x86_64 for a 64-bit architecture.
Minimum Disk (GB)	Minimum disk size required to boot the image. If this field is not specified, the default value is 0 (no minimum).
Minimum RAM (MB)	Minimum memory size required to boot the image. If this field is not specified, the default value is 0 (no minimum).
Public	If selected, makes the image public to all users with access to the project.
Protected	If selected, ensures only users with specific permissions can delete this image.

When the image has been successfully uploaded, its status is changed to **active**, which indicates that the image is available for use. The Image service can handle even large images that take a long time to upload, longer than the lifetime of the Identity service token which was used to initiate the upload. This is due to the fact that the Image service first creates a trust with the Identity service so that a new token can be obtained and used when the upload is complete and the status of the image is to be updated.



#### NOTE

You can also use the **glance image-create** command with the **--property** option to upload an image. More values are available on the command line. For a complete list of available metadata properties, see [Image Configuration Parameters](#).

### 1.2.3. Updating an image

#### Procedure

1. In the dashboard, select **Project > Compute > Images**
2. Click **Edit Image** from the list.



#### NOTE

The **Edit Image** option is available only when you log in as an **admin** user. When you log in as a **demo** user, you have the option to **Launch an instance** or **Create Volume**.

3. Update the fields and click **Update Image** when finished. You can update the following values - name, description, kernel ID, ramdisk ID, architecture, format, minimum disk, minimum RAM, public, protected.
4. Click the menu and select **Update Metadata** option.
5. Specify metadata by adding items from the left column to the right one. In the left column, there are metadata definitions from the Image Service Metadata Catalog. Select **Other** to add metadata with the key of your choice and click **Save** when finished.



**NOTE**

You can also use the **glance image-update** command with the **--property** option to update an image. More values are available on the command line. For a complete list of available metadata properties, see [Image Configuration Parameters](#).

## 1.2.4. Importing an image

You can import images into the Image service (glance) by using **web-download** to import an image from a URI and **glance-direct** to import an image from a local file system. The **web-download** option is enabled by default.

Import methods are configured by the cloud administrator. Run the **glance import-info** command to list available import options.

### 1.2.4.1. Importing from a remote URI

You can use the **web-download** method to copy an image from a remote URI by using a two-stage process. First, an image record is created and then the image is retrieved from a URI. This method provides a more secure way to import images than the deprecated **copy-from** method used in Image API v1.

#### Procedure

1. Create an image and specify the URI of the image to import.

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

2. You can monitor the availability of the image:

```
$ openstack image show <image_id> command.
```

Replace the ID with the one provided during image creation.

### 1.2.4.2. Importing from a local volume

The **glance-direct** method creates an image record, which generates an image ID. After the image is uploaded to the service from a local volume, it is stored in a staging area and is made active after it passes any configured checks. The **glance-direct** method requires a shared staging area when used in a highly available (HA) configuration.

**NOTE**

Image uploads that use the **glance-direct** method fail in an HA environment if a common staging area is not present. In an HA active-active environment, API calls are distributed to the Image service controllers. The download API call can be sent to a different controller than the API call to upload the image. For more information about configuring the staging area, see [Storage Configuration](#) in the *Advanced OpenStack Customization Guide*.

The **glance-direct** method uses the following calls to import an image:

- **glance image-create**

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

### Procedure

1. You can use the **glance image-create-via-import** command to perform all three of these calls in one command:

```
$ glance image-create-via-import --container-format <format> --disk-format <disk_format> --name <name> --file <path_to_image>
```

After the image moves from the staging area to the back end location, the image is listed. However, it might take some time for the image to become active.

2. You can monitor the availability of the image:

```
$ openstack image show <image_id> command.
```

Replace the ID with the one provided during image creation.

## 1.2.5. Deleting an image

### Procedure

1. In the dashboard, select **Project > Compute > Images**
2. Select the image you want to delete and click **Delete Images**.

## 1.2.6. Enabling image conversion

With the **GlanceImageImportPlugins** parameter enabled, you can upload a QCOW2 image, and the Image service converts it to RAW.

### Procedure

- To enable image conversion, create an environment file that contains the following parameter value and include the new environment file with any other environment files that are relevant to your deployment by using the **-e** option in the **openstack overcloud deploy** command:

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

## 1.2.7. Converting an image to RAW format

Red Hat Ceph can store, but does not support using, QCOW2 images to host virtual machine (VM) disks.

When you upload a QCOW2 image and create a VM from it, the compute node downloads the image, converts the image to RAW, and uploads it back into Ceph, which can then use it. This process affects the time it takes to create VMs, especially during parallel VM creation.

For example, when you create multiple VMs simultaneously, uploading the converted image to the Ceph cluster might impact already running workloads. The upload process can starve those workloads of IOPS and impede storage responsiveness.

To boot VMs in Ceph more efficiently (ephemeral back end or boot from volume), the Image service image format must be RAW.

### Procedure

1. Converting an image to RAW might yield an image that is larger in size than the original QCOW2 image file. Run the following command before the conversion to determine the final RAW image size:

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

2. Convert an image from QCOW2 to RAW format:

```
qemu-img convert -p -f qcow2 -O raw <original_qcow2_image>.qcow2
<new_raw_image>.raw
```

#### 1.2.7.1. Configuring the Image service to accept only RAW and ISO

You can configure the Image service to accept only RAW and ISO image formats.

### Procedure

1. Add an additional environment file that contains the following content in the **openstack overcloud deploy** command with your other environment files:

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

#### 1.2.8. Storing an image in RAW format

### Procedure

- With the **GlanceImageImportPlugins** parameter enabled, run the following command to upload a **QCOW2** image and automatically convert it to **RAW** format.

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

- Replace **<name>** with the name of the image; this is the name that appears in **openstack image list**.

- For **--uri**, replace **<http://server/image.qcow2>** with the location and file name of the QCOW2 image.



#### NOTE

This example command creates the image record and imports it by using the **web-download** method. The **glance-api** downloads the image from the **--uri** location during the import process. If **web-download** is not available, **glanceclient** cannot automatically download the image data. Run the **glance import-info** command to list the available image import methods.

## CHAPTER 2. CONFIGURING THE COMPUTE (NOVA) SERVICE

Use environment files to customize the Compute (nova) service. Puppet generates and stores this configuration in the `/var/lib/config-data/puppet-generated/<nova_container>/etc/nova/nova.conf` file. Use the following configuration methods to customize the Compute service configuration:

- **Heat parameters** - as detailed in the [Compute \(nova\) Parameters](#) section in the *Overcloud Parameters* guide. For example:

```
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 parameters** - as defined in `/etc/puppet/modules/nova/manifests/*`:

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



### NOTE

Only use this method if an equivalent heat parameter does not exist.

- **Manual hieradata overrides** - for customizing parameters when no heat or Puppet parameter exists. For example, the following sets the `disk_allocation_ratio` in the **[DEFAULT]** section on the Compute role:

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



### WARNING

If a heat parameter exists, it must be used instead of the Puppet parameter; if a Puppet parameter exists, but not a heat parameter, then the Puppet parameter must be used instead of the manual override method. The manual override method must only be used if there is no equivalent heat or Puppet parameter.

**TIP**

Follow the guidance in [Identifying Parameters to Modify](#) to determine if a heat or Puppet parameter is available for customizing a particular configuration.

See [Parameters](#) in the *Advanced Overcloud Customization* guide for further details on configuring overcloud services.

**2.1. CONFIGURING MEMORY FOR OVERALLOCATION**

When you use memory overcommit (**NovaRAMAllocationRatio**  $\geq 1.0$ ), you need to deploy your overcloud with enough swap space to support the allocation ratio.

**NOTE**

If your **NovaRAMAllocationRatio** parameter is set to  $< 1$ , follow the RHEL recommendations for swap size. For more information, see [Recommended system swap space](#) in the RHEL *Managing Storage Devices* guide.

**Prerequisites**

- You have calculated the swap size your node requires. For more information, see [Section 2.3, “Calculating swap size”](#).

**Procedure**

- Copy the **/usr/share/openstack-tripleo-heat-templates/environments/enable-swap.yaml** file to your environment file directory:

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

- Configure the swap size by adding the following parameters to your **enable-swap.yaml** file:

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

- To apply this configuration, add the **enable\_swap.yaml** environment file to the stack with your other environment files and deploy the overcloud:

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

**2.2. CALCULATING RESERVED HOST MEMORY ON COMPUTE NODES**

To determine the total amount of RAM to reserve for host processes, you need to allocate enough memory for each of the following:

- The resources that run on the node, for instance, OSD consumes 3 GB of memory.
- The emulator overhead required to visualize instances on a host.

- The hypervisor for each instance.

After you calculate the additional demands on memory, use the following formula to help you determine the amount of memory to reserve for host processes on each node:

$$\text{NovaReservedHostMemory} = \text{total\_RAM} - (\text{vm\_no} * (\text{avg\_instance\_size} + \text{overhead})) + (\text{resource1} * \text{resource\_ram}) + (\text{resource\_n\_} * \text{resource\_ram})$$

- Replace **vm\_no** with the number of instances.
- Replace **avg\_instance\_size** with the average amount of memory each instance can use.
- Replace **overhead** with the hypervisor overhead required for each instance.
- Replace **resource1** with the number of a resource type on the node.
- Replace **resource\_ram** with the amount of RAM each resource of this type requires.

## 2.3. CALCULATING SWAP SIZE

The allocated swap size must be large enough to handle any memory overcommit. You can use the following formulas to calculate the swap size your node requires:

- $\text{overcommit\_ratio} = \text{NovaRAMAllocationRatio} - 1$
- Minimum swap size (MB) =  $(\text{total\_RAM} * \text{overcommit\_ratio}) + \text{RHEL\_min\_swap}$
- Recommended (maximum) swap size (MB) =  $\text{total\_RAM} * (\text{overcommit\_ratio} + \text{percentage\_of\_RAM\_to\_use\_for\_swap})$

The **percentage\_of\_RAM\_to\_use\_for\_swap** variable creates a buffer to account for QEMU overhead and any other resources consumed by the operating system or host services.

For instance, to use 25% of the available RAM for swap, with 64GB total RAM, and **NovaRAMAllocationRatio** set to **1**:

- Recommended (maximum) swap size =  $64000 \text{ MB} * (0 + 0.25) = 16000 \text{ MB}$

For information on how to calculate the **NovaReservedHostMemory** value, see [Section 2.2, “Calculating reserved host memory on Compute nodes”](#).

For information on how to determine the **RHEL\_min\_swap** value, see [Recommended system swap space](#) in the RHEL *Managing Storage Devices* guide.

## CHAPTER 3. CONFIGURING OPENSTACK COMPUTE STORAGE

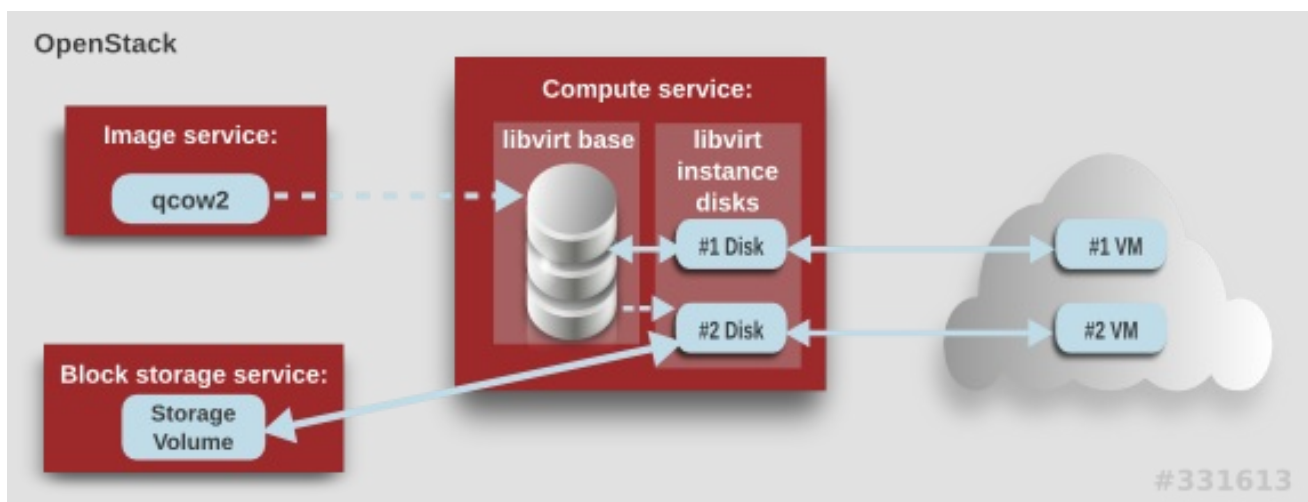
This chapter describes the architecture for the back-end storage of images in OpenStack Compute (nova), and provides basic configuration options.

### 3.1. ARCHITECTURE OVERVIEW

In Red Hat OpenStack Platform, the OpenStack Compute service uses the KVM hypervisor to execute compute workloads. The **libvirt** driver handles all interactions with KVM, and enables the creation of virtual machines.

Two types of **libvirt** storage must be considered for Compute:

- Base image, which is a cached and formatted copy of the Image service image.
- Instance disk, which is created using the **libvirt** base and is the back end for the virtual machine instance. Instance disk data can be stored either in Compute's ephemeral storage (using the **libvirt** base) or in persistent storage (for example, using Block Storage).



The steps that Compute takes to create a virtual machine instance are:

1. Cache the Image service's backing image as the **libvirt** base.
2. Convert the base image to the raw format (if configured).
3. Resize the base image to match the VM's flavor specifications.
4. Use the base image to create the libvirt instance disk.

In the diagram above, the **#1** instance disk uses ephemeral storage; the **#2** disk uses a block-storage volume.

Ephemeral storage is an empty, unformatted, additional disk available to an instance. This storage value is defined by the instance flavor. The value provided by the user must be less than or equal to the ephemeral value defined for the flavor. The default value is **0**, meaning no ephemeral storage is created.

The ephemeral disk appears in the same way as a plugged-in hard drive or thumb drive. It is available as a block device which you can check using the **lsblk** command. You can format it, mount it, and use it however you normally would a block device. There is no way to preserve or reference that disk beyond



the instance it is attached to.

Block storage volume is persistent storage available to an instance regardless of the state of the running instance.

## 3.2. CONFIGURATION

You can configure performance tuning and security for your virtual disks by customizing the Compute (nova) configuration files. Compute is configured in custom environment files and Heat templates using the parameters detailed in the [Compute \(nova\) Parameters](#) section in the *Overcloud Parameters* guide. This configuration is generated and stored in the `/var/lib/config-data/puppet-generated/<nova_container>/etc/nova/nova.conf` file, as detailed in the following table.

Table 3.1. Compute Image Parameters

Section	Parameter	Description	Default
[DEFAULT]	<b>force_raw_images</b>	<p>Whether to convert a <b>non-raw</b> cached base image to be <b>raw</b> (boolean). If a non-raw image is converted to raw, Compute:</p> <ul style="list-style-type: none"> <li>• Disallows backing files (which might be a security issue).</li> <li>• Removes existing compression (to avoid CPU bottlenecks).</li> </ul> <p>Converting the base to raw uses more space for any image that could have been used directly by the hypervisor (for example, a qcow2 image). If you have a system with slower I/O or less available space, you might want to specify <i>false</i>, trading the higher CPU requirements of compression for that of minimized input bandwidth.</p> <p>Raw base images are always used with <b>libvirt_images_type=lvm</b>.</p>	<b>true</b>
[DEFAULT]	<b>use_cow_images</b>	<p>Whether to use CoW (Copy on Write) images for <b>libvirt</b> instance disks (boolean):</p> <ul style="list-style-type: none"> <li>• <b>false</b> - The raw format is used. Without CoW, more space is used for common parts of the disk image</li> <li>• <b>true</b> - The cqow2 format is used. With CoW, depending on the backing store and host caching, there may be better concurrency achieved by having each VM operate on its own copy.</li> </ul>	true

Section	Parameter	Description	Default
[DEFAULT]	<b>preallocate_images</b>	<p>Preallocation mode for <b>libvirt</b> instance disks. Value can be:</p> <ul style="list-style-type: none"> <li>● <b>none</b> - No storage is provisioned at instance start.</li> <li>● <b>space</b> - Storage is fully allocated at instance start (using <b>fallocate</b>), which can help with both space guarantees and I/O performance.</li> </ul> <p>Even when not using CoW instance disks, the copy each VM gets is sparse and so the VM may fail unexpectedly at run time with ENOSPC. By running <b>fallocate(1)</b> on the instance disk images, Compute immediately and efficiently allocates the space for them in the file system (if supported). Run time performance should also be improved because the file system does not have to dynamically allocate blocks at run time (reducing CPU overhead and more importantly file fragmentation).</p>	none
[DEFAULT]	<b>resize_fs_using_block_device</b>	<p>Whether to enable direct resizing of the base image by accessing the image over a block device (boolean). This is only necessary for images with older versions of <b>cloud-init</b> (that cannot resize themselves).</p> <p>Because this parameter enables the direct mounting of images which might otherwise be disabled for security reasons, it is not enabled by default.</p>	<b>false</b>
[DEFAULT]	<b>default_ephemeral_format</b>	<p>The default format that is used for a new ephemeral volume. Value can be: <b>ext2</b>, <b>ext3</b>, or <b>ext4</b>. The <b>ext4</b> format provides much faster initialization times than <b>ext3</b> for new, large disks. You can also override per instance using the <b>guest_format</b> configuration option.</p>	<b>ext4</b>
[DEFAULT]	<b>image_cache_manager_interval</b>	<p>Number of seconds to wait between runs of the image cache manager, which impacts base caching on libvirt compute nodes. This period is used in the auto removal of unused cached images (see <b>remove_unused_base_images</b> and <b>remove_unused_original_minimum_age_seconds</b>).</p>	<b>2400</b>

Section	Parameter	Description	Default
[DEFAULT]	<b>remove_unused_base_images</b>	Whether to enable the automatic removal of unused base images (checked every <b>image_cache_manager_interval</b> seconds). Images are defined as <b>unused</b> if they have not been accessed in <b>remove_unused_original_minimum_age_seconds</b> seconds.	<b>true</b>
[DEFAULT]	<b>remove_unused_original_minimum_age_seconds</b>	How old an unused base image must be before being removed from the <b>libvirt</b> cache (see <b>remove_unused_base_images</b> ).	<b>86400</b>
[libvirt]	<b>images_type</b>	Image type to use for <b>libvirt</b> instance disks (deprecates <b>use_cow_images</b> ). Value can be: <b>raw</b> , <b>qcow2</b> , <b>lvm</b> , <b>rbd</b> , or <b>default</b> . If <b>default</b> is specified, the value used for the <b>use_cow_images</b> parameter is used.	<b>default</b>

### 3.3. ENABLING SERVICE TOKENS BETWEEN THE COMPUTE SERVICE AND THE BLOCK STORAGE SERVICE

As an administrator, if you want to prevent user request token timeouts when attaching or detaching volumes, you must enable service tokens on all overcloud nodes that run the Compute (nova) service or the Block Storage (cinder) service.

#### Procedure

1. Create an environment file to configure the service tokens, such as **service\_tokens.yaml**.
2. Add the following configuration parameters to the service token environment file:

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      service_user/send_service_user_token:
        value: true
      service_user/username:
        value: nova
      service_user/auth_strategy:
        value: keystone
      service_user/auth_type:
        value: password
      service_user/password:
        value: "%{hiera('nova::placement::password')}}"
      service_user/auth_url:
        value: "%{hiera('nova::placement::auth_url')}}"
      service_user/user_domain_name:
        value: "Default"
      service_user/project_name:
```

```

    value: "%{hiera('nova::placement::project_name')}}"
  service_user/project_default_name:
    value: "Default"

```

#### ControllerExtraConfig:

```

nova::config::nova_config:
  keystone_authtoken/service_token_roles_required:
    value: true
  keystone_authtoken/service_token_roles:
    value: admin
  service_user/send_service_user_token:
    value: true
  service_user/username:
    value: nova
  service_user/auth_strategy:
    value: keystone
  service_user/auth_type:
    value: password
  service_user/password:
    value: "%{hiera('nova::keystone::authtoken::password')}}"
  service_user/auth_url:
    value: "%{hiera('nova::keystone::authtoken::auth_url')}}"
  service_user/user_domain_name:
    value: "%{hiera('nova::keystone::authtoken::user_domain_name')}}"
  service_user/project_name:
    value: "%{hiera('nova::keystone::authtoken::project_name')}}"
  service_user/project_domain_name:
    value: "%{hiera('nova::keystone::authtoken::project_domain_name')}}"

```

#### cinder::config::cinder\_config:

```

keystone_authtoken/service_token_roles_required:
  value: true
keystone_authtoken/service_token_roles:
  value: admin
service_user/send_service_user_token:
  value: true
service_user/username:
  value: cinder
service_user/auth_strategy:
  value: keystone
service_user/auth_type:
  value: password
service_user/password:
  value: "%{hiera('cinder::keystone::authtoken::password')}}"
service_user/auth_url:
  value: "%{hiera('cinder::keystone::authtoken::auth_url')}}"
service_user/user_domain_name:
  value: "%{hiera('cinder::keystone::authtoken::user_domain_name')}}"
service_user/project_name:
  value: "%{hiera('cinder::keystone::authtoken::project_name')}}"
service_user/project_domain_name:
  value: "%{hiera('cinder::keystone::authtoken::project_domain_name')}}"

```

#### BlockStorageExtraConfig:

```

cinder::config::cinder_config:
  keystone_authtoken/service_token_roles_required:

```

```

    value: true
  keystone_authtoken/service_token_roles:
    value: admin
  service_user/send_service_user_token:
    value: true
  service_user/username:
    value: cinder
  service_user/auth_strategy:
    value: keystone
  service_user/auth_type:
    value: password
  service_user/password:
    value: "%{hiera('cinder::keystone::authtoken::password')}}"
  service_user/auth_url:
    value: "%{hiera('cinder::keystone::authtoken::auth_url')}}"
  service_user/user_domain_name:
    value: "%{hiera('cinder::keystone::authtoken::user_domain_name')}}"
  service_user/project_name:
    value: "%{hiera('cinder::keystone::authtoken::project_name')}}"
  service_user/project_domain_name:
    value: "%{hiera('cinder::keystone::authtoken::project_domain_name')}}"

```

3. Add the service token environment file to the stack with your other environment files and deploy the overcloud:

```

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

```

## CHAPTER 4. VIRTUAL MACHINE INSTANCES

OpenStack Compute (nova) is the central component that provides virtual machines on demand. Compute interacts with the Identity service (keystone) for authentication, the Image service (glance) for images to launch instances, and the dashboard service for the user and administrative interface.

With Red Hat OpenStack Platform (RHOSP) you can easily manage virtual machine instances in the cloud. The Compute service creates, schedules, and manages instances, and exposes this functionality to other OpenStack components. This chapter discusses these procedures along with procedures to add components like key pairs, security groups, host aggregates and flavors. The term *instance* in OpenStack means a virtual machine instance.

### 4.1. MANAGING INSTANCES

Before you can create an instance, you need to ensure certain other OpenStack components (for example, a network, key pair and an image or a volume as the boot source) are available for the instance.

This section discusses the procedures to add these components, create and manage an instance. Managing an instance refers to updating, and logging in to an instance, viewing how the instances are being used, resizing or deleting them.

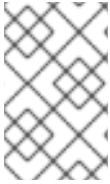
#### 4.1.1. Adding components

Use the following sections to create a network, key pair and upload an image or volume source. Use these components when you create an instance that is not available by default. You must also create a new security group to allow SSH access to the user.

1. In the dashboard, select **Project**.
2. Select **Network > Networks** and ensure there is a private network to which you can attach the new instance (to create a network, see [Creating a Network](#) section in the *Networking Guide*).
3. Select **Compute > Access & Security > Key Pairs** and ensure there is a key pair (to create a key pair, see [Section 4.2.1.1, "Creating a key pair"](#)).
4. Ensure that you have either an image or a volume that can you can use as a boot source:
  - To view boot-source images, select the **Images** tab (to create an image, see [Section 1.2.1, "Creating an image"](#)).
  - To view boot-source volumes, select the **Volumes** tab (to create a volume, see [Create a Volume](#) in the *Storage Guide*).
5. Select **Compute > Access & Security > Security Groups** and ensure you have created a security group rule (to create a security group, see [Project Security Management](#) in the *Users and Identity Management Guide*).

#### 4.1.2. Launching an instance

Launch one or more instances from the dashboard.

**NOTE**

Instances are launched by default using the Launch Instance form. However, you can also enable a Launch Instance wizard that simplifies the steps required. For more information, see [Appendix B, \*Enabling the launch instance wizard\*](#).

1. In the dashboard, select **Project > Compute > Instances**
2. Click **Launch Instance**.
3. Complete the fields (\* indicates a required field), and click **Launch**.

One or more instances are created and launched based on the options provided.

**CAUTION**

It is not possible to launch an instance with a Block Storage (cinder) volume if the root disk size is larger than the HDD of the Compute node. Use one of the following workarounds to allow an instance to be launched with a Block Storage volume:


- Use a flavor with the root disk and ephemeral disk set to **0**.
- Remove **DiskFilter** from the **NovaSchedulerDefaultFilters** configuration.

**4.1.2.1. Launching instance options**

The following table outlines the options available when you use the Launch Instance form to launch a new instance. The same options are also available in the Launch instance wizard.

**Table 4.1. Launch Instance Form options**

Tab	Field	Notes
Project and User	Project	Select the project from the list.
	User	Select the user from the list.
Details	Availability Zone	Zones are logical groupings of cloud resources in which you can place your instance. If you are unsure, use the default zone (for more information, see <a href="#">Section 4.4, “Managing host aggregates”</a> ).
	Instance Name	A name to identify your instance.
	Flavor	The flavor determines what resources to give the instance, for example, memory. For default flavor allocations and information about creating new flavors, see <a href="#">Section 4.3, “Managing flavors”</a> .
	Instance Count	The number of instances to create with these parameters. <b>1</b> is preselected.

Tab	Field	Notes
	Instance Boot Source	<p>Depending on the item selected, new fields are displayed to select the source:</p> <ul style="list-style-type: none"> <li>• Image sources must be compatible with OpenStack (see <a href="#">Section 1.2, "Managing images"</a>).</li> <li>• If a volume or volume source is selected, the source must be formatted by using an image (see <a href="#">Basic Volume Usage and Configuration</a> in the <i>Storage Guide</i>).</li> </ul>
Access and Security	Key Pair	The specified key pair is injected into the instance and is used to remotely access the instance using SSH (if neither a direct login information or a static key pair is provided). Usually one key pair per project is created.
	Security Groups	Security groups contain firewall rules which filter the type and direction of the instance network traffic. For more information about configuring groups, see <a href="#">Project Security Management</a> in the <i>Users and Identity Management Guide</i> .
Networking	Selected Networks	You must select at least one network. Instances are typically assigned to a private network, and then later given a floating IP address to enable external access.
Post-Creation	Customization Script Source	<p>You can provide either a set of commands or a script file, which runs after the instance is booted (for example, to set the instance host name or a user password). If <b>Direct Input</b> is selected, write your commands in the Script Data field; otherwise, specify your script file.</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p><b>NOTE</b></p> <p>Any script that starts with <b>#cloud-config</b> is interpreted as using the cloud-config syntax. For information about the syntax, see <a href="http://cloudinit.readthedocs.org/en/latest/topics/examples.html">http://cloudinit.readthedocs.org/en/latest/topics/examples.html</a>.</p> </div> </div>
Advanced Options	Disk Partition	By default, the instance is built as a single partition and dynamically resized as needed. However, you can choose to manually configure the partitions yourself.



Tab	Field	Notes
	Configuration Drive	If selected, OpenStack writes metadata to a read-only configuration drive that is attached to the instance when it boots (instead of to Compute's metadata service). After the instance has booted, you can mount this drive to view its contents and provide files to the instance.

### 4.1.3. Updating an instance

You can update an instance by selecting **Project > Compute > Instances** and selecting an action for that instance in the **Actions** column. Use actions to manipulate the instance in a number of ways:

**Table 4.2. Update instance options**

Action	Description
Create Snapshot	Snapshots preserve the disk state of a running instance. You can create a snapshot to migrate the instance, as well as to preserve backup copies.
Associate/Disassociate Floating IP	You must associate an instance with a floating IP (external) address before it can communicate with external networks, or be reached by external users. Because there are a limited number of external addresses in your external subnets, it is recommended that you disassociate any unused addresses.
Edit Instance	Update the instance's name and associated security groups.
Edit Security Groups	Add and remove security groups to or from this instance using the list of available security groups (for more information on configuring groups, see <a href="#">Project Security Management</a> in the <i>Users and Identity Management Guide</i> ).
Console	View the instance console in the browser for easy access to the instance.
View Log	View the most recent section of the instance console log. When opened, you can view the full log by clicking <b>View Full Log</b> .
Pause/Resume Instance	Immediately pause the instance (you are not asked for confirmation); the state of the instance is stored in memory (RAM).

Action	Description
Suspend/Resume Instance	Immediately suspend the instance (you are not asked for confirmation); like hibernation, the state of the instance is kept on disk.
Resize Instance	Display the <b>Resize Instance</b> window (see <a href="#">Section 4.1.4, "Resizing an instance"</a> ).
Soft Reboot	Gracefully stop and restart the instance. A soft reboot attempts to gracefully shut down all processes before restarting the instance.
Hard Reboot	Stop and restart the instance. A hard reboot effectively shuts down the <b>power</b> of the instance and then turns it back on.
Shut Off Instance	Gracefully stop the instance.
Rebuild Instance	Use new image and disk-partition options to rebuild the image (shut down, re-image, and re-boot the instance). If encountering operating system issues, this option is easier to try than terminating the instance and starting from the beginning.
Terminate Instance	Permanently destroy the instance (you are asked for confirmation).

You can create and allocate an external IP address, see [Section 4.2.3, "Creating, assigning, and releasing floating IP addresses"](#)

#### 4.1.4. Resizing an instance

To resize an instance (memory or CPU count), you must select a new flavor for the instance that has the right capacity. If you are increasing the size, remember to first ensure that the host has enough space.

1. Ensure communication between hosts by setting up each host with SSH key authentication so that Compute can use SSH to move disks to other hosts. For example, Compute nodes can share the same SSH key.
2. Enable resizing on the original host by setting the **allow\_resize\_to\_same\_host** parameter to **True** for the Controller role.



#### NOTE

The **allow\_resize\_to\_same\_host** parameter does not resize the instance on the same host. Even if the parameter equals **True** on all Compute nodes, the scheduler does not force the instance to resize on the same host. This is the expected behavior.

3. In the dashboard, select **Project > Compute > Instances**
4. Click the instance's **Actions** arrow, and select **Resize Instance**.
5. Select a new flavor in the **New Flavor** field.
6. If you want to manually partition the instance when it launches (results in a faster build time):
  - a. Select **Advanced Options**.
  - b. In the **Disk Partition** field, select **Manual**.
7. Click **Resize**.

### 4.1.5. Connecting to an instance

You can access an instance console by using the dashboard or the command-line interface. You can also directly connect to the serial port of an instance so that you can debug even if the network connection fails.

#### 4.1.5.1. Accessing an instance console by using the dashboard

You can connect to the instance console from the dashboard.

#### Procedure

1. In the dashboard, select **Compute > Instances**
2. Click the instance's **More** button and select **Console**.

The screenshot shows the 'Instance Details: TinyImage' page in the OpenStack dashboard. The 'Console' tab is active, displaying a terminal window. The terminal output shows the following messages:

```

[ 1.563471] EFI Variables Facility v0.08 2004-May-17
[ 1.566228] TCP cubic registered
[ 1.567338] NET: Registered protocol family 10
[ 1.582404] NET: Registered protocol family 17
[ 1.582694] Registering the dns_resolver key type
[ 1.593666] registered taskstats version 1
[ 1.673678] Refined TSC clocksource calibration: 2893.419 MHz.
[ 1.674010] Switching to clocksource tsc
[ 1.765213] Magic number: 2:192:770
[ 1.765213] rtc_cmos 00:01: setting system clock to 2014-06-19 05:46:39 UTC (
1403156799)
[ 1.765213] powernow-k8: Processor cpuid 6d3 not supported
[ 1.765213] BIOS EDD facility v0.16 2004-Jun-25, 0 devices found
[ 1.765213] EDD information not available.
[ 1.800424] Freeing unused kernel memory: 924k freed
[ 1.830096] Write protecting the kernel read-only data: 12288k
[ 1.872886] Freeing unused kernel memory: 1632k freed
[ 1.925062] Freeing unused kernel memory: 1200k freed
[ 1.983300] usb 1-1: new full-speed USB device number 2 using uhci_hcd

further output written to /dev/ttyS0

login as 'cirros' user. default password: 'cubswin:'). use 'sudo' for root.
cirros login: _
  
```

3. Log in using the image's user name and password (for example, a CirrOS image uses *cirros/cubswin:*).

#### 4.1.5.2. Accessing an instance console by using the CLI

You can connect directly to the VNC console for an instance by entering the VNC console URL in a browser.

##### Procedure

1. To display the VNC console URL for an instance, enter the following command:

```
$ openstack console url show <vm_name>
+-----+
| Field | Value          |
+-----+
| type  | novnc          |
| url   | http://172.25.250.50:6080/vnc_auto.html?token=      |
|       | 962dfd71-f047-43d3-89a5-13cb88261eb9                |
+-----+
```

2. To connect directly to the VNC console, enter the displayed URL in a browser.

#### 4.1.6. Viewing instance usage

The following usage statistics are available:

- Per Project  
To view instance usage per project, select **Project > Compute > Overview**. A usage summary is immediately displayed for all project instances.

You can also view statistics for a specific period of time by specifying the date range and clicking **Submit**.

- Per Hypervisor  
If logged in as an administrator, you can also view information for all projects. Click **Admin > System** and select one of the tabs. For example, the **Resource Usage** tab offers a way to view reports for a distinct time period. You might also click **Hypervisors** to view your current vCPU, memory, or disk statistics.

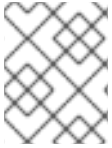


##### NOTE

The **vCPU Usage** value (**x of y**) reflects the number of total vCPUs of all virtual machines (x) and the total number of hypervisor cores (y).

#### 4.1.7. Deleting an instance

1. In the dashboard, select **Project > Compute > Instances** and select your instance.
2. Click **Terminate Instance**.

**NOTE**

Deleting an instance does not delete its attached volumes; you must do this separately (see [Delete a Volume](#) in the *Storage Guide*).

### 4.1.8. Managing multiple instances simultaneously

If you need to start multiple instances at the same time (for example, those that were down for compute or controller maintenance) you can do so easily at **Project > Compute > Instances**

1. Click the check boxes in the first column for the instances that you want to start. If you want to select all of the instances, click the check box in the first row in the table.
2. Click **More Actions** above the table and select **Start Instances**.

Similarly, you can shut off or soft reboot multiple instances by selecting the respective actions.

## 4.2. MANAGING INSTANCE SECURITY

You can manage access to an instance by assigning it the correct security group (set of firewall rules) and key pair (enables SSH user access). Further, you can assign a floating IP address to an instance to enable external network access. The sections below outline how to create and manage key pairs, security groups, floating IP addresses and logging in to an instance using SSH. There is also a procedure for injecting an **admin** password in to an instance.

For information on managing security groups, see [Project Security Management](#) in the *Users and Identity Management Guide*.

### 4.2.1. Managing key pairs

Key pairs provide SSH access to the instances. Each time a key pair is generated, its certificate is downloaded to the local machine and can be distributed to users. Typically, one key pair is created for each project (and used for multiple instances).

You can also import an existing key pair into OpenStack.

#### 4.2.1.1. Creating a key pair

1. In the dashboard, select **Project > Compute > Access & Security**
2. On the **Key Pairs** tab, click **Create Key Pair**.
3. Specify a name in the **Key Pair Name** field, and click **Create Key Pair**.

When the key pair is created, a key pair file is automatically downloaded through the browser. Save this file for later connections from external machines. For command-line SSH connections, you can load this file into SSH by executing:

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

#### 4.2.1.2. Importing a key pair

1. In the dashboard, select **Project > Compute > Access & Security**

2. On the **Key Pairs** tab, click **Import Key Pair**.
3. Specify a name in the **Key Pair Name** field, and copy and paste the contents of your public key into the **Public Key** field.
4. Click **Import Key Pair**.

#### 4.2.1.3. Deleting a key pair

1. In the dashboard, select **Project > Compute > Access & Security**
2. On the **Key Pairs** tab, click the key's **Delete Key Pair** button.

#### 4.2.2. Creating a security group

Security groups are sets of IP filter rules that can be assigned to project instances, and which define networking access to the instance. Security groups are project specific; project members can edit the default rules for their security group and add new rule sets.

1. In the dashboard, select the **Project** tab, and click **Compute > Access & Security**
2. On the **Security Groups** tab, click **+ Create Security Group**.
3. Provide a name and description for the group, and click **Create Security Group**.

For more information on managing project security, see [Project Security Management](#) in the *Users and Identity Management Guide*.

#### 4.2.3. Creating, assigning, and releasing floating IP addresses

By default, an instance is given an internal IP address when it is first created. However, you can enable access through the public network by creating and assigning a floating IP address (external address). You can change an instance's associated IP address regardless of the instance's state.

Projects have a limited range of floating IP address that can be used (by default, the limit is 50), so you should release these addresses for reuse when they are no longer needed. Floating IP addresses can only be allocated from an existing floating IP pool, see [Creating Floating IP Pools](#) in the *Networking Guide*.

##### 4.2.3.1. Allocating a floating IP to the project

1. In the dashboard, select **Project > Compute > Access & Security**
2. On the **Floating IPs** tab, click **Allocate IP to Project**
3. Select a network from which to allocate the IP address in the **Pool** field.
4. Click **Allocate IP**.

##### 4.2.3.2. Assigning a floating IP

1. In the dashboard, select **Project > Compute > Access & Security**
2. On the **Floating IPs** tab, click the address' **Associate** button.

3. Select the address to be assigned in the IP address field.

**NOTE**

If no addresses are available, you can click the **+** button to create a new address.

4. Select the instance to be associated in the **Port** to be **Associated** field. An instance can only be associated with one floating IP address.
5. Click **Associate**.

#### 4.2.3.3. Releasing a floating IP

1. In the dashboard, select **Project > Compute > Access & Security**
2. On the **Floating IPs** tab, click the address' menu arrow (next to the **Associate/Disassociate** button).
3. Select **Release Floating IP**.

#### 4.2.4. Logging in to an instance

##### Prerequisites:

- Ensure that the instance's security group has an SSH rule (see [Project Security Management](#) in the *Users and Identity Management Guide*).
- Ensure the instance has a floating IP address (external address) assigned to it (see [Section 4.2.3, "Creating, assigning, and releasing floating IP addresses"](#)).
- Obtain the instance's key-pair certificate. The certificate is downloaded when the key pair is created; if you did not create the key pair yourself, ask your administrator (see [Section 4.2.1, "Managing key pairs"](#)).

##### To first load the key pair file into SSH, and then use ssh without naming it

1. Change the permissions of the generated key-pair certificate.

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

2. Check whether **ssh-agent** is already running:

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

3. If not already running, start it up with:

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

4. On your local machine, load the key-pair certificate into SSH. For example:

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

5. You can now SSH into the file with the user supplied by the image.

The following example command shows how to SSH into the Red Hat Enterprise Linux guest image with the user **cloud-user**:

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



#### NOTE

You can also use the certificate directly. For example:

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

### 4.2.5. Injecting an admin password into an instance

You can inject an **admin (root)** password into an instance using the following procedure.

1. In the `/etc/openstack-dashboard/local_settings` file, set the **change\_set\_password** parameter value to **True**.

```
can_set_password: True
```

2. Set the **inject\_password** parameter to "True" in your Compute environment file.

```
inject_password=true
```

3. Restart the Compute service.

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

When you use the **nova boot** command to launch a new instance, the output of the command displays an **adminPass** parameter. You can use this password to log into the instance as the **root** user.

The Compute service overwrites the password value in the `/etc/shadow` file for the **root** user. This procedure can also be used to activate the **root** account for the KVM guest images. For more information on how to use KVM guest images, see [Section 1.2.1.1, "Using a KVM guest image with Red Hat OpenStack Platform"](#)

You can also set a custom password from the dashboard. To enable this, run the following command after you have set **can\_set\_password** parameter to **true**.

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

The newly added **admin** password fields are as follows:



## Launch Instance ✕

Project & User \*
Details \*
Access & Security
Networking \*
Post-Creation

[Advanced Options](#)

---

**Key Pair** ⓘ

▼
+

**Admin Password**

👁

**Confirm Admin Password**

👁

**Security Groups** ⓘ

default

Control access to your instance via key pairs, security groups, and other mechanisms.

Cancel
Launch

These fields can be used when you launch or rebuild an instance.

### 4.3. MANAGING FLAVORS

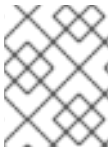
Each created instance is given a flavor (resource template), which determines the instance's size and capacity. Flavors can also specify secondary ephemeral storage, swap disk, metadata to restrict usage, or special project access (none of the default flavors have these additional attributes defined).

**Table 4.3. Default Flavors**

Name	vCPUs	RAM	Root Disk Size
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

The majority of end users will be able to use the default flavors. However, you can create and manage specialized flavors. For example, you can:

- Change default memory and capacity to suit the underlying hardware needs.
- Add metadata to force a specific I/O rate for the instance or to match a host aggregate.

**NOTE**

Behavior set using image properties overrides behavior set using flavors (for more information, see [Section 1.2, “Managing images”](#)).

### 4.3.1. Updating configuration permissions

By default, only administrators can create flavors or view the complete flavor list (select Admin > System > Flavors). To allow all users to configure flavors, specify the following in the `/etc/nova/policy.json` file (nova-api server):

```
"compute_extension:flavormanage": "",
```

### 4.3.2. Creating a flavor

1. As an admin user in the dashboard, select **Admin > System > Flavors**
2. Click **Create Flavor**, and specify the following fields:

**Table 4.4. Flavor Options**

Tab	Field	Description
Flavor Information	Name	Unique name.
	ID	Unique ID. The default value, <b>auto</b> , generates a UUID4 value, but you can also manually specify an integer or UUID4 value.
	VCPUs	Number of virtual CPUs.
	RAM (MB)	Memory (in megabytes).
	Root Disk (GB)	Ephemeral disk size (in gigabytes); to use the native image size, specify <b>0</b> . This disk is not used if <b>Instance Boot Source=Boot from Volume</b> .
	Ephemeral Disk (GB)	Secondary ephemeral disk size (in gigabytes) available to an instance. This disk is destroyed when an instance is deleted.  The default value is <b>0</b> , which implies that no ephemeral disk is created.

Tab	Field	Description
	Swap Disk (MB)	Swap disk size (in megabytes).
Flavor Access	Selected Projects	Projects which can use the flavor. If no projects are selected, all projects have access ( <b>Public=Yes</b> ).

3. Click Create Flavor.

### 4.3.3. Updating general attributes

1. As an admin user in the dashboard, select **Admin > System > Flavors**
2. Click the flavor's **Edit Flavor** button.
3. Update the values, and click **Save**.

### 4.3.4. Updating flavor metadata

In addition to editing general attributes, you can add metadata to a flavor (**extra\_specs**), which can help fine-tune instance usage. For example, you might want to set the maximum-allowed bandwidth or disk writes.

- Pre-defined keys determine hardware support or quotas. Pre-defined keys are limited by the hypervisor you are using (for libvirt, see [Table 4.5, "Libvirt Metadata"](#)).
- Both pre-defined and user-defined keys can determine instance scheduling. For example, you might specify **SpecialComp=True**; any instance with this flavor can then only run in a host aggregate with the same key-value combination in its metadata (see [Section 4.4, "Managing host aggregates"](#)).

#### 4.3.4.1. Viewing metadata

1. As an admin user in the dashboard, select **Admin > System > Flavors**
2. Click the flavor's **Metadata** link (**Yes** or **No**). All current values are listed on the right-hand side under **Existing Metadata**.

#### 4.3.4.2. Adding metadata

You specify a flavor's metadata using a **key/value** pair.

1. As an admin user in the dashboard, select **Admin > System > Flavors**
2. Click the flavor's **Metadata** link (**Yes** or **No**). All current values are listed on the right-hand side under **Existing Metadata**.
3. Under **Available Metadata**, click on the **Other** field, and specify the key you want to add (see [Table 4.5, "Libvirt Metadata"](#)).
4. Click the + button; you can now view the new key under **Existing Metadata**.


5. Fill in the key's value in its right-hand field.

The screenshot shows a web interface titled "Existing Metadata". At the top right, there is a search box labeled "Filter" with a magnifying glass icon. Below this, there are two rows of metadata entries. Each row consists of a key-value pair in a light gray box, followed by a minus sign button in a separate box. The first row shows the key "quota:cpu\_shares" with the value "200". The second row shows the key "quota:disk\_read\_byte..." with the value "10240000".

6. When finished with adding key-value pairs, click **Save**.

Table 4.5. Libvirt Metadata

Key	Description
<b>hw:action</b>	<p>Action that configures support limits per instance. Valid actions are:</p> <ul style="list-style-type: none"> <li>● <b>cpu_max_sockets</b> - Maximum supported CPU sockets.</li> <li>● <b>cpu_max_cores</b> - Maximum supported CPU cores.</li> <li>● <b>cpu_max_threads</b> - Maximum supported CPU threads.</li> <li>● <b>cpu_sockets</b> - Preferred number of CPU sockets.</li> <li>● <b>cpu_cores</b> - Preferred number of CPU cores.</li> <li>● <b>cpu_threads</b> - Preferred number of CPU threads.</li> <li>● <b>serial_port_count</b> - Maximum serial ports per instance.</li> </ul> <p>Example: <b>hw:cpu_max_sockets=2</b></p>

Key	Description
<b>hw:NUMA_def</b>	<p>Definition of NUMA topology for the instance. For flavors whose RAM and vCPU allocations are larger than the size of NUMA nodes in the compute hosts, defining NUMA topology enables hosts to better utilize NUMA and improve performance of the guest OS. NUMA definitions defined through the flavor override image definitions. Valid definitions are:</p> <ul style="list-style-type: none"> <li>● <b>numa_nodes</b> - Number of NUMA nodes to expose to the instance. Specify <i>1</i> to ensure image NUMA settings are overridden.</li> <li>● <b>numa_cpus.0</b> - Mapping of vCPUs N-M to NUMA node 0 (comma-separated list).</li> <li>● <b>numa_cpus.1</b> - Mapping of vCPUs N-M to NUMA node 1 (comma-separated list).</li> <li>● <b>numa_mem.0</b> - Mapping N MB of RAM to NUMA node 0.</li> <li>● <b>numa_mem.1</b> - Mapping N MB of RAM to NUMA node 1.</li> <li>● <b>numa_cpu.N</b> and <b>numa_mem.N</b> are only valid if <b>numa_nodes</b> is set. Additionally, they are only required if the instance's NUMA nodes have an asymmetrical allocation of CPUs and RAM (important for some NFV workloads).</li> </ul> <p> <b>NOTE</b></p> <p>If the values of <b>numa_cpu</b> or <b>numa_mem.N</b> specify more than that available, an exception is raised.</p> <p>Example when the instance has 8 vCPUs and 4GB RAM:</p> <ul style="list-style-type: none"> <li>● <b>hw:numa_nodes=2</b></li> <li>● <b>hw:numa_cpus.0=0,1,2,3,4,5</b></li> <li>● <b>hw:numa_cpus.1=6,7</b></li> <li>● <b>hw:numa_mem.0=3072</b></li> <li>● <b>hw:numa_mem.1=1024</b></li> </ul> <p>The scheduler looks for a host with 2 NUMA nodes with the ability to run 6 CPUs + 3072 MB, or 3 GB, of RAM on one node, and 2 CPUs + 1024 MB, or 1 GB, of RAM on another node. If a host has a single NUMA node with capability to run 8 CPUs and 4 GB of RAM, it will not be considered a valid match.</p>

Key	Description
<b>hw:watchdog_action</b>	<p>An instance watchdog device can be used to trigger an action if the instance somehow fails (or hangs). Valid actions are:</p> <ul style="list-style-type: none"> <li>● <b>disabled</b> - The device is not attached (default value).</li> <li>● <b>pause</b> - Pause the instance.</li> <li>● <b>poweroff</b> - Forcefully shut down the instance.</li> <li>● <b>reset</b> - Forcefully reset the instance.</li> <li>● <b>none</b> - Enable the watchdog, but do nothing if the instance fails.</li> </ul> <p>Example: <b>hw:watchdog_action=poweroff</b></p>
<b>hw:pci_numa_affinity_policy</b>	<p>You can use this parameter to specify the NUMA affinity policy for PCI passthrough devices and SR-IOV interfaces. Set to one of the following valid values:</p> <ul style="list-style-type: none"> <li>● <b>required</b>: The Compute service only creates an instance that requests a PCI device when at least one of the NUMA nodes of the instance has affinity with the PCI device. This option provides the best performance.</li> <li>● <b>preferred</b>: The Compute service attempts a best effort selection of PCI devices based on NUMA affinity. If this is not possible, then the Compute service schedules the instance on a NUMA node that has no affinity with the PCI device.</li> <li>● <b>legacy</b>: (Default) The Compute service creates instances that request a PCI device when either: <ul style="list-style-type: none"> <li>○ The PCI device has affinity with at least one of the NUMA nodes; or</li> <li>○ The PCI devices do not provide information on their NUMA affinities.</li> </ul> </li> </ul> <p>Example: <b>hw:pci_numa_affinity_policy=required</b></p>
<b>hw_rng:action</b>	<p>A random-number generator device can be added to an instance using its image properties (see <b>hw_rng_model</b> in the "Command-Line Interface Reference" in Red Hat OpenStack Platform documentation).</p> <p>If the device has been added, valid actions are:</p> <ul style="list-style-type: none"> <li>● <b>allowed</b> - If <b>True</b>, the device is enabled; if <b>False</b>, disabled. By default, the device is disabled.</li> <li>● <b>rate_bytes</b> - Maximum number of bytes the instance's kernel can read from the host to fill its entropy pool every <b>rate_period</b> (integer).</li> <li>● <b>rate_period</b> - Duration of the read period in seconds (integer).</li> </ul> <p>Example: <b>hw_rng:allowed=True</b>.</p>

Key	Description
<b>hw_video:ram_max_mb</b>	<p>Maximum permitted RAM to be allowed for video devices (in MB).</p> <p>Example: <b>hw:ram_max_mb=64</b></p>
<b>quota:option</b>	<p>Enforcing limit for the instance. Valid options are:</p> <ul style="list-style-type: none"> <li>● <b>cpu_period</b> - Time period for enforcing <b>cpu_quota</b> (in microseconds). Within the specified <b>cpu_period</b>, each vCPU cannot consume more than <b>cpu_quota</b> of runtime. The value must be in range [1000, 1000000]; 0 means <i>no value</i>.</li> <li>● <b>cpu_quota - Maximum allowed bandwidth (in microseconds) for the vCPU in each `cpu_period</b>. The value must be in range [1000, 18446744073709551]. 0 means <i>no value</i>; a negative value means that the vCPU is not controlled. <b>cpu_quota</b> and <b>cpu_period</b> can be used to ensure that all vCPUs run at the same speed.</li> <li>● <b>cpu_shares</b> - Share of CPU time for the domain. The value only has meaning when weighted against other machine values in the same domain. That is, an instance with a flavor with 200 will get twice as much machine time as an instance with 100.</li> <li>● <b>disk_read_bytes_sec</b> - Maximum disk reads in bytes per second.</li> <li>● <b>disk_read_iops_sec</b> - Maximum read I/O operations per second.</li> <li>● <b>disk_write_bytes_sec</b> - Maximum disk writes in bytes per second.</li> <li>● <b>disk_write_iops_sec</b> - Maximum write I/O operations per second.</li> <li>● <b>disk_total_bytes_sec</b> - Maximum total throughput limit in bytes per second.</li> <li>● <b>disk_total_iops_sec</b> - Maximum total I/O operations per second.</li> <li>● <b>vif_inbound_average</b> - Desired average of incoming traffic.</li> <li>● <b>vif_inbound_burst</b> - Maximum amount of traffic that can be received at <b>vif_inbound_peak</b> speed.</li> <li>● <b>vif_inbound_peak</b> - Maximum rate at which incoming traffic can be received.</li> <li>● <b>vif_outbound_average</b> - Desired average of outgoing traffic.</li> <li>● <b>vif_outbound_burst</b> - Maximum amount of traffic that can be sent at <b>vif_outbound_peak</b> speed.</li> <li>● <b>vif_outbound_peak</b> - Maximum rate at which outgoing traffic can be sent.</li> </ul> <p>Example: <b>quota:vif_inbound_average=10240</b></p> <p>In addition, the VMware driver supports the following quota options, which control upper and lower limits for CPUs, RAM, disks, and networks, as well as <i>shares</i>, which can be used to control relative allocation of available resources among tenants:</p>

Key	<ul style="list-style-type: none"> <li>● <b>cpu_limit</b> - Maximum CPU frequency available to a virtual machine (MHz).</li> </ul> Description
	<ul style="list-style-type: none"> <li>● <b>cpu_reservation</b> - Guaranteed minimum amount of CPU resources available to a virtual machine (in MHz).</li> <li>● <b>cpu_shares_level</b> - CPU allocation level (shares) in the case of contention. Possible values are <b>high</b>, <b>normal</b>, <b>low</b>, and <b>custom</b>.</li> <li>● <b>cpu_shares_share</b> - The number of allocated CPU shares. Applicable when <b>cpu_shares_level</b> is set to <b>custom</b>.</li> <li>● <b>memory_limit</b> - Maximum amount of RAM available to a virtual machine (in MB).</li> <li>● <b>memory_reservation</b> - Guaranteed minimum amount of RAM available to a virtual machine (in MB).</li> <li>● <b>memory_shares_level</b> - RAM allocation level (shares) in the case of contention. Possible values are <b>high</b>, <b>normal</b>, <b>low</b>, and <b>custom</b>.</li> <li>● <b>memory_shares_share</b> - The number of allocated RAM shares. Applicable when <b>memory_shares_level</b> is set to <b>custom</b>.</li> <li>● <b>disk_io_limit</b> - Maximum I/O utilization by a virtual machine (in I/O operations per second).</li> <li>● <b>disk_io_reservation</b> - Guaranteed minimum amount of disk resources available to a virtual machine (in I/O operations per second).</li> <li>● <b>disk_io_shares_level</b> - I/O allocation level (shares) in the case of contention. Possible values are <b>high</b>, <b>normal</b>, <b>low</b>, and <b>custom</b>.</li> <li>● <b>disk_io_shares_share</b> - The number of allocated I/O shares. Applicable when <b>disk_io_shares_level</b> is set to <b>custom</b>.</li> <li>● <b>vif_limit</b> - Maximum network bandwidth available to a virtual network adapter (in Mbps).</li> <li>● <b>vif_reservation</b> - Guaranteed minimum network bandwidth available to a virtual network adapter (in Mbps).</li> <li>● <b>vif_shares_level</b> - Network bandwidth allocation level (shares) in the case of contention. Possible values are <b>high</b>, <b>normal</b>, <b>low</b>, and <b>custom</b>.</li> <li>● <b>vif_shares_share</b> - The number of allocated network bandwidth shares. Applicable when <b>vif_shares_level</b> is set to <b>custom</b>.</li> </ul>

## 4.4. MANAGING HOST AGGREGATES

A single Compute deployment can be partitioned into logical groups for performance or administrative purposes. OpenStack uses the following terms:

- *Host aggregates* - A host aggregate creates logical units in a OpenStack deployment by grouping together hosts. Aggregates are assigned Compute hosts and associated metadata; a host can be in more than one host aggregate. Only administrators can see or create host aggregates.  
An aggregate's metadata is commonly used to provide information for use with the Compute scheduler (for example, limiting specific flavors or images to a subset of hosts). Metadata specified in a host aggregate will limit the use of that host to any instance that has the same



metadata specified in its flavor.

Administrators can use host aggregates to handle load balancing, enforce physical isolation (or redundancy), group servers with common attributes, or separate out classes of hardware. When you create an aggregate, a zone name must be specified, and it is this name which is presented to the end user.

- *Availability zones* - An availability zone is the end-user view of a host aggregate. An end user cannot view which hosts make up the zone, nor see the zone's metadata; the user can only see the zone's name.  
End users can be directed to use specific zones which have been configured with certain capabilities or within certain areas.

#### 4.4.1. Enabling host aggregate scheduling

By default, host-aggregate metadata is not used to filter instance usage. You must update the Compute scheduler's configuration to enable metadata usage:

1. Open your Compute environment file.
2. Add the following values to the **NovaSchedulerDefaultFilters** parameter, if they are not already present:
  - **AggregateInstanceExtraSpecsFilter** for host aggregate metadata.



#### NOTE

Scoped specifications must be used for setting flavor **extra\_specs** when specifying both **AggregateInstanceExtraSpecsFilter** and **ComputeCapabilitiesFilter** filters as values of the same **NovaSchedulerDefaultFilters** parameter, otherwise the **ComputeCapabilitiesFilter** will fail to select a suitable host. See [Table 4.7, "Scheduling Filters"](#) for further details.

- **AvailabilityZoneFilter** for availability zone host specification when launching an instance.
3. Save the configuration file.
  4. Deploy the overcloud.

#### 4.4.2. Viewing availability zones or host aggregates

As an admin user in the dashboard, select **Admin > System > Host Aggregates**. All currently defined aggregates are listed in the **Host Aggregates** section; all zones are in the **Availability Zones** section.

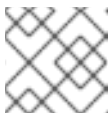
#### 4.4.3. Adding a host aggregate

1. As an admin user in the dashboard, select **Admin > System > Host Aggregates**. All currently defined aggregates are listed in the **Host Aggregates** section.
2. Click **Create Host Aggregate**.
3. Add a name for the aggregate in the **Name** field, and a name by which the end user should see it in the **Availability Zone** field.

4. Click **Manage Hosts within Aggregate**
5. Select a host for use by clicking its + icon.
6. Click **Create Host Aggregate**.

#### 4.4.4. Updating a host aggregate

1. As an admin user in the dashboard, select **Admin > System > Host Aggregates** All currently defined aggregates are listed in the **Host Aggregates** section.
2. To update the instance's **Name** or **Availability zone**:
  - Click the aggregate's **Edit Host Aggregate** button.
  - Update the **Name** or **Availability Zone** field, and click **Save**.
3. To update the instance's **Assigned hosts**:
  - Click the aggregate's arrow icon under **Actions**.
  - Click **Manage Hosts**.
  - Change a host's assignment by clicking its + or - icon.
  - When finished, click **Save**.
4. To update the instance's **Metadata**:
  - Click the aggregate's arrow icon under **Actions**.
  - Click the **Update Metadata** button. All current values are listed on the right-hand side under **Existing Metadata**.
  - Under **Available Metadata**, click on the **Other** field, and specify the key you want to add. Use predefined keys (see [Table 4.6, "Host Aggregate Metadata"](#)) or add your own (which will only be valid if exactly the same key is set in an instance's flavor).
  - Click the + button; you can now view the new key under **Existing Metadata**.



#### NOTE

Remove a key by clicking its - icon.

- Click **Save**.

Table 4.6. Host Aggregate Metadata

Key	Description
<b>filter_tenant_id</b>	If specified, the aggregate only hosts this tenant (project). Depends on the <b>AggregateMultiTenancyIsolation</b> filter being set for the Compute scheduler.

#### 4.4.5. Deleting a host aggregate

1. As an admin user in the dashboard, select **Admin > System > Host Aggregates**. All currently defined aggregates are listed in the **Host Aggregates** section.
2. Remove all assigned hosts from the aggregate:
  - a. Click the aggregate's arrow icon under **Actions**.
  - b. Click **Manage Hosts**.
  - c. Remove all hosts by clicking their - icon.
  - d. When finished, click **Save**.
3. Click the aggregate's arrow icon under **Actions**.
4. Click **Delete Host Aggregate** in this and the next dialog screen.

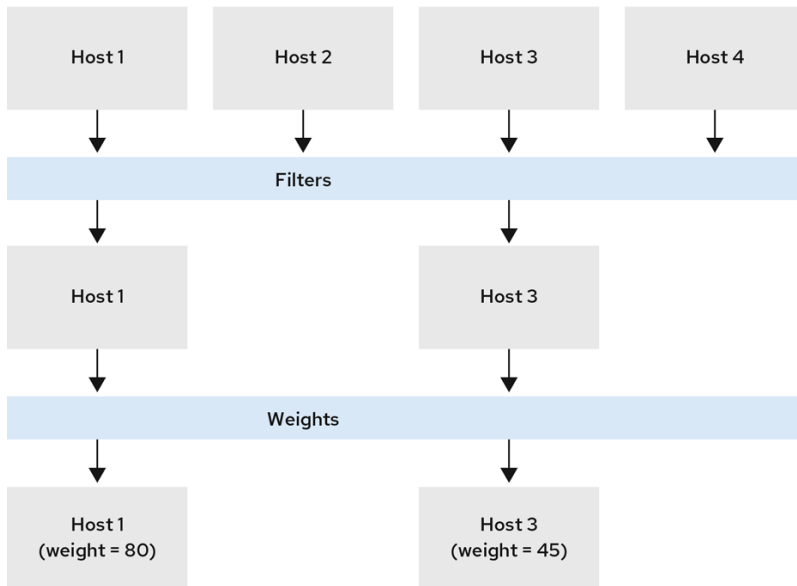
## 4.5. SCHEDULING HOSTS

The Compute scheduling service determines on which host (or host aggregate), an instance will be placed. As an administrator, you can influence where the scheduler will place an instance. For example, you might want to limit scheduling to hosts in a certain group or with the right RAM.

You can configure the following components:

- **Filters** - Determine the initial set of hosts on which an instance might be placed (see [Section 4.5.1, "Configuring scheduling filters"](#)).
- **Weights** - When filtering is complete, the resulting set of hosts are prioritized using the weighting system. The highest weight has the highest priority (see [Section 4.5.2, "Configuring scheduling weights"](#)).
- **Scheduler service** - There are a number of configuration options in the `/var/lib/config-data/puppet-generated/<nova_container>/etc/nova/nova.conf` file (on the scheduler host), which determine how the scheduler executes its tasks, and handles weights and filters.

In the following diagram, both host 1 and 3 are eligible after filtering. Host 1 has the highest weight and therefore has the highest priority for scheduling.



81\_OpenStack\_0520

### 4.5.1. Configuring scheduling filters

You define the filters you want the scheduler to use by adding or removing filters from the **NovaSchedulerDefaultFilters** parameter in your Compute environment file.

The default configuration runs the following filters in the scheduler:

- RetryFilter
- AvailabilityZoneFilter
- ComputeFilter
- ComputeCapabilitiesFilter
- ImagePropertiesFilter
- ServerGroupAntiAffinityFilter
- ServerGroupAffinityFilter

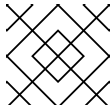
Some filters use information in parameters passed to the instance in:

- The **nova boot** command.
- The instance's flavor (see [Section 4.3.4, "Updating flavor metadata"](#))
- The instance's image (see [Appendix A, Image configuration parameters](#)).

The following table lists all the available filters.

**Table 4.7. Scheduling Filters**

Filter	Description
--------	-------------

Filter	Description
<b>AggregateImagePropertiesIsolation</b>	Only passes hosts in host aggregates whose metadata matches the instance's image metadata; only valid if a host aggregate is specified for the instance. For more information, see <a href="#">Section 1.2.1, "Creating an image"</a> .
<b>AggregateInstanceExtraSpecsFilter</b>	Metadata in the host aggregate must match the host's flavor metadata. For more information, see <a href="#">Section 4.3.4, "Updating flavor metadata"</a> .
	<p>This filter can only be specified in the same <b>NovaSchedulerDefaultFilters</b> parameter as <b>ComputeCapabilitiesFilter</b> when you scope your flavor <b>extra_specs</b> keys by prefixing them with the correct namespace:</p> <ul style="list-style-type: none"> <li>• <b>ComputeCapabilitiesFilter</b> namespace = "<b>capabilities:</b>"</li> <li>• <b>AggregateInstanceExtraSpecsFilter</b> namespace = "<b>aggregate_instance_extra_specs:</b>"</li> </ul>
<b>AggregateMultiTenancyIsolation</b>	<p>A host with the specified <b>filter_tenant_id</b> can only contain instances from that tenant (project).</p> <div style="display: flex; align-items: center;">  <div style="margin-left: 20px;"> <p><b>NOTE</b></p> <p>The tenant can still place instances on other hosts.</p> </div> </div>
<b>AllHostsFilter</b>	Passes all available hosts (however, does not disable other filters).
<b>AvailabilityZoneFilter</b>	Filters using the instance's specified availability zone.
<b>ComputeCapabilitiesFilter</b>	Ensures Compute metadata is read correctly. Anything before the <b>:</b> is read as a namespace. For example, <b>quota:cpu_period</b> uses <b>quota</b> as the namespace and <b>cpu_period</b> as the key.
<b>ComputeFilter</b>	Passes only hosts that are operational and enabled.
<b>DifferentHostFilter</b>	Enables an instance to build on a host that is different from one or more specified hosts. Specify <b>different</b> hosts using the <b>nova boot</b> option <b>--different_host</b> option.
<b>ImagePropertiesFilter</b>	Only passes hosts that match the instance's image properties. For more information, see <a href="#">Section 1.2.1, "Creating an image"</a> .
<b>IsolatedHostsFilter</b>	Passes only isolated hosts running isolated images that are specified using <b>isolated_hosts</b> and <b>isolated_images</b> (comma-separated values).

Filter	Description
<b>JsonFilter</b>	<p>Recognises and uses an instance's custom JSON filters:</p> <ul style="list-style-type: none"> <li>Valid operators are: =, &lt;, &gt;, in, ∈, ≥, not, or, and</li> <li>Recognised variables are: \$free_ram_mb, \$free_disk_mb, \$total_usable_ram_mb, \$vcpus_total, \$vcpus_used</li> </ul>
	<p>The filter is specified as a query hint in the <b>nova boot</b> command. For example:</p> <pre>--hint query='[&gt;=, '\$free_disk_mb', 200 * 1024]'</pre>
<b>MetricsFilter</b>	<p>Use this filter to limit scheduling to Compute nodes that report the metrics configured by using <b>metrics/weight_setting</b>.</p> <p>To use this filter, add the following configuration to your Compute environment file:</p> <pre>parameter_defaults:   ComputeExtraConfig:     nova::config::nova_config:       DEFAULT/compute_monitors:         value: 'cpu.virt_driver'</pre> <p>By default, the Compute scheduling service updates the metrics every 60 seconds. To ensure the metrics are up-to-date, you can increase the frequency at which the metrics data is refreshed using the <b>update_resources_interval</b> configuration option. For example, use the following configuration to refresh the metrics data every 2 seconds:</p> <pre>parameter_defaults:   ComputeExtraConfig:     nova::config::nova_config:       DEFAULT/update_resources_interval:         value: '2'</pre>
<b>NUMATopologyFilter</b>	<p>Filters out hosts based on its NUMA topology. If the instance has no topology defined, any host can be used. The filter tries to match the exact NUMA topology of the instance to those of the host (it does not attempt to pack the instance onto the host). The filter also looks at the standard over-subscription limits for each NUMA node, and provides limits to the compute host accordingly.</p>
<b>RetryFilter</b>	<p>Filters out hosts that have failed a scheduling attempt; valid if <b>scheduler_max_attempts</b> is greater than zero (defaults to "3").</p>
<b>SameHostFilter</b>	<p>Passes one or more specified hosts; specify hosts for the instance using the <b>--hint same_host</b> option for <b>nova boot</b>.</p>

Filter	Description
<b>ServerGroupAffinityFilter</b>	<p>Only passes hosts for a specific server group:</p> <ul style="list-style-type: none"> <li>• Give the server group the affinity policy (<b>nova server-group-create --policy affinity groupName</b>).</li> <li>• Build the instance with that group (<b>nova boot</b> option <b>--hint group=UUID</b>).</li> </ul>
<b>ServerGroupAntiAffinityFilter</b>	<p>Only passes hosts in a server group that do not already host an instance:</p> <ul style="list-style-type: none"> <li>• Give the server group the anti-affinity policy (<b>nova server-group-create --policy anti-affinity groupName</b>).</li> <li>• Build the instance with that group (<b>nova boot</b> option <b>--hint group=UUID</b>).</li> </ul>
<b>SimpleCIDRAffinityFilter</b>	<p>Only passes hosts on the specified IP subnet range specified by the instance's cidr and <b>build_new_host_ip</b> hints. Example:</p> <p><b>--hint build_near_host_ip=192.0.2.0 --hint cidr=/24</b></p>

#### 4.5.2. Configuring scheduling weights

Hosts can be weighted for scheduling; the host with the largest weight (after filtering) is selected. All weighers are given a multiplier that is applied after normalising the node's weight. A node's weight is calculated as:

$$w1\_multiplier * norm(w1) + w2\_multiplier * norm(w2) + \dots$$

You can configure weight options in the Compute node's configuration file.

**Table 4.8. Configuration options for Scheduling service weights**

Configuration option	Description
----------------------	-------------

Configuration option	Description
<b>filter_scheduler/weight_classes</b>	<p>Use this parameter to configure which of the following attributes to use for calculating the weight of each host:</p> <ul style="list-style-type: none"> <li>● <b>nova.scheduler.weights.ram.RAMWeigher</b> - Weighs the available RAM on the Compute node.</li> <li>● <b>nova.scheduler.weights.cpu.CPUWeigher</b> - Weighs the available CPUs on the Compute node.</li> <li>● <b>nova.scheduler.weights.disk.DiskWeigher</b> - Weighs the available disks on the Compute node.</li> <li>● <b>nova.scheduler.weights.metrics.MetricsWeigher</b> - Weighs the metrics of the Compute node.</li> <li>● <b>nova.scheduler.weights.affinity.ServerGroupSoftAffinityWeigher</b> - Weighs the proximity of the Compute node to other nodes in the given instance group.</li> <li>● <b>nova.scheduler.weights.affinity.ServerGroupSoftAntiAffinityWeigher</b> - Weighs the proximity of the Compute node to other nodes in the given instance group.</li> <li>● <b>nova.scheduler.weights.compute.BuildFailureWeigher</b> - Weighs Compute nodes by the number of recent failed boot attempts.</li> <li>● <b>nova.scheduler.weights.io_ops.IoOpsWeigher</b> - Weighs Compute nodes by their workload.</li> <li>● <b>nova.scheduler.weights.pci.PCIWeigher</b> - Weighs Compute nodes by their PCI availability.</li> <li>● <b>nova.scheduler.weights.cross_cell.CrossCellWeigher</b> - Weighs Compute nodes based on which cell they are in, giving preference to Compute nodes in the source cell when moving an instance.</li> <li>● <b>nova.scheduler.weights.all_weighers</b> - (Default) Uses all the above weighers.</li> </ul> <p>Type: String</p>



Configuration option	Description
<b>filter_scheduler/ram_weight_multiplier</b>	<p>Use this parameter to specify the multiplier to use to weigh hosts based on the available RAM.</p> <p>Set to a positive value to prefer hosts with more available RAM, which spreads instances across many hosts.</p> <p>Set to a negative value to prefer hosts with less available RAM, which fills up (stacks) hosts as much as possible before scheduling to a less-used host.</p> <p>The absolute value, whether positive or negative, controls how strong the RAM weigher is relative to other weighers.</p> <p>By default, the scheduler spreads instances across all hosts evenly (<b>ram_weight_multiplier=1.0</b>).</p> <p>Type: Floating point</p>
<b>filter_scheduler/disk_weight_multiplier</b>	<p>Use this parameter to specify the multiplier to use to weigh hosts based on the available disk space.</p> <p>Set to a positive value to prefer hosts with more available disk space, which spreads instances across many hosts.</p> <p>Set to a negative value to prefer hosts with less available disk space, which fills up (stacks) hosts as much as possible before scheduling to a less-used host.</p> <p>The absolute value, whether positive or negative, controls how strong the disk weigher is relative to other weighers.</p> <p>By default, the scheduler spreads instances across all hosts evenly (<b>disk_weight_multiplier=1.0</b>).</p> <p>Type: Floating point</p>
<b>filter_scheduler/cpu_weight_multiplier</b>	<p>Use this parameter to specify the multiplier to use to weigh hosts based on the available vCPUs.</p> <p>Set to a positive value to prefer hosts with more available vCPUs, which spreads instances across many hosts.</p> <p>Set to a negative value to prefer hosts with less available vCPUs, which fills up (stacks) hosts as much as possible before scheduling to a less-used host.</p> <p>The absolute value, whether positive or negative, controls how strong the vCPU weigher is relative to other weighers.</p> <p>By default, the scheduler spreads instances across all hosts evenly (<b>cpu_weight_multiplier=1.0</b>).</p> <p>Type: Floating point</p>

Configuration option	Description
<b>filter_scheduler/io_ops_weight_multiplier</b>	<p>Use this parameter to specify the multiplier to use to weigh hosts based on the host workload.</p> <p>Set to a negative value to prefer hosts with lighter workloads, which distributes the workload across more hosts.</p> <p>Set to a positive value to prefer hosts with heavier workloads, which schedules instances onto hosts that are already busy.</p> <p>The absolute value, whether positive or negative, controls how strong the I/O operations weigher is relative to other weighers.</p> <p>By default, the scheduler distributes the workload across more hosts (<b>io_ops_weight_multiplier=-1.0</b>).</p> <p>Type: Floating point</p>
<b>filter_scheduler/build_failure_weight_multiplier</b>	<p>Use this parameter to specify the multiplier to use to weigh hosts based on recent build failures.</p> <p>Set to a positive value to increase the significance of build failures recently reported by the host. Hosts with recent build failures are then less likely to be chosen.</p> <p>Set to <b>0</b> to disable weighing compute hosts by the number of recent failures.</p> <p>Default: 1000000.0</p> <p>Type: Floating point</p>
<b>filter_scheduler/cross_cell_move_weight_multiplier</b>	<p>Use this parameter to specify the multiplier to use to weigh hosts during a cross-cell move. This option determines how much weight is placed on a host which is within the same source cell when moving an instance. By default, the scheduler prefers hosts within the same source cell when migrating an instance.</p> <p>Set to a positive value to prefer hosts within the same cell the instance is currently running. Set to a negative value to prefer hosts located in a different cell from that where the instance is currently running.</p> <p>Default: 1000000.0</p> <p>Type: Floating point</p>

Configuration option	Description
<b>filter_scheduler/pci_weight_multiplier</b>	<p>Use this parameter to specify the multiplier to use to weigh hosts based on the number of PCI devices on the host and the number of PCI devices requested by an instance. If an instance requests PCI devices, then the more PCI devices a Compute node has the higher the weight allocated to the Compute node.</p> <p>For example, if there are three hosts available, one with a single PCI device, one with multiple PCI devices and one without any PCI devices, then the Compute scheduler prioritizes these hosts based on the demands of the instance. The first host should be preferred if the instance requests one PCI device, the second host if the instance requires multiple PCI devices and the third host if the instance does not request a PCI device.</p> <p>Configure this option to prevent non-PCI instances from occupying resources on hosts with PCI devices.</p> <p>Default: 1.0</p> <p>Type: Positive floating point</p>
<b>filter_scheduler/host_subset_size</b>	<p>Use this parameter to specify the size of the subset of filtered hosts from which to select the host. Must be set to at least 1. A value of 1 selects the first host returned by the weighing functions. Any value less than 1 is ignored and 1 is used instead.</p> <p>Set to a value greater than 1 to prevent multiple scheduler processes handling similar requests selecting the same host, creating a potential race condition. By selecting a host randomly from the N hosts that best fit the request, the chance of a conflict is reduced. However, the higher you set this value, the less optimal the chosen host may be for a given request.</p> <p>Default: 1</p> <p>Type: Integer</p>
<b>filter_scheduler/soft_affinity_weight_multiplier</b>	<p>Use this parameter to specify the multiplier to use to weigh hosts for group soft-affinity.</p> <p>Default: 1.0</p> <p>Type: Positive floating point</p>

Configuration option	Description
<b>filter_scheduler/soft_anti_affinity_weight_multiplier</b>	<p>Use this parameter to specify the multiplier to use to weigh hosts for group soft-anti-affinity.</p> <p>Default: 1.0</p> <p>Type: Positive floating point</p>
<b>metrics/weight_multiplier</b>	<p>Use this parameter to specify the multiplier to use for weighting metrics. By default, <b>weight_multiplier=1.0</b>, which spreads instances across possible hosts.</p> <p>Set to a number greater than 1.0 to increase the effect of the metric on the overall weight.</p> <p>Set to a number between 0.0 and 1.0 to reduce the effect of the metric on the overall weight.</p> <p>Set to 0.0 to ignore the metric value and return the value of the 'weight_of_unavailable' option.</p> <p>Set to a negative number to prioritize the host with lower metrics, and stack instances in hosts.</p> <p>Default: 1.0</p> <p>Type: Floating point</p>
<b>metrics/weight_setting</b>	<p>Use this parameter to specify the metrics to use for weighting, and the ratio to use to calculate the weight of each metric. Valid metric names:</p> <ul style="list-style-type: none"> <li>● <b>cpu.frequency</b> - CPU frequency</li> <li>● <b>cpu.user.time</b> - CPU user mode time</li> <li>● <b>cpu.kernel.time</b> - CPU kernel time</li> <li>● <b>cpu.idle.time</b> - CPU idle time</li> <li>● <b>cpu.iowait.time</b> - CPU I/O wait time</li> <li>● <b>cpu.user.percent</b> - CPU user mode percentage</li> <li>● <b>cpu.kernel.percent</b> - CPU kernel percentage</li> <li>● <b>cpu.idle.percent</b> - CPU idle percentage</li> <li>● <b>cpu.iowait.percent</b> - CPU I/O wait percentage</li> <li>● <b>cpu.percent</b> - Generic CPU utilization</li> </ul> <p>Example: <b>weight_setting=cpu.user.time=1.0</b></p> <p>Type: Comma-separated list of <b>metric=ratio</b> pairs.</p>

Configuration option	Description
<b>metrics/required</b>	<p>Use this parameter to specify how to handle configured <b>metrics/weight_setting</b> metrics that are unavailable:</p> <ul style="list-style-type: none"> <li>• <b>True</b> - Metrics are required. If the metric is unavailable, an exception is raised. To avoid the exception, use the <b>MetricsFilter</b> filter in <b>NovaSchedulerDefaultFilters</b>.</li> <li>• <b>False</b> - The unavailable metric is treated as a negative factor in the weighing process. Set the returned value by using the <b>weight_of_unavailable</b> configuration option.</li> </ul> <p>Type: Boolean</p>
<b>metrics/weight_of_unavailable</b>	<p>Use this parameter to specify the weight to use if any <b>metrics/weight_setting</b> metric is unavailable, and <b>metrics/required=False</b>.</p> <p>Default: -10000.0</p> <p>Type: Floating point</p>

### 4.5.3. Reserving NUMA nodes with PCI devices

Compute uses the filter scheduler to prioritize hosts with PCI devices for instances requesting PCI. The hosts are weighted using the **PCIWeigher** option, based on the number of PCI devices available on the host and the number of PCI devices requested by an instance. If an instance requests PCI devices, then the hosts with more PCI devices are allocated a higher weight than the others. If an instance is not requesting PCI devices, then prioritization does not take place.

This feature is especially useful in the following cases:

- As an operator, if you want to reserve nodes with PCI devices (typically expensive and with limited resources) for guest instances that request them.
- As a user launching instances, you want to ensure that PCI devices are available when required.



#### NOTE

For this value to be considered, one of the following values must be added to the **NovaSchedulerDefaultFilters** parameter in your Compute environment file: **PciPassthroughFilter** or **NUMATopologyFilter**.

The **pci\_weight\_multiplier** configuration option must be a positive value.

## 4.6. MANAGING INSTANCE SNAPSHOTS

You can use an instance snapshot to create a new image from an instance. This is very convenient for upgrading base images or for taking a published image and customizing it for local use.

The difference between an image that you upload directly to the Image service and an image that you create by snapshot is that an image created by snapshot has additional properties in the Image service database. These properties are in the **image\_properties** table and include the following parameters:

**Table 4.9. Snapshot options**

Name	Value
image_type	snapshot
instance_uuid	<uuid_of_instance_that_was_snapshotted>
base_image_ref	<uuid_of_original_image_of_instance_that_was_snapshotted>
image_location	snapshot

Use snapshots to create new instances based on that snapshot, and potentially restore an instance to that state. You can perform this action while the instance is running.

By default, a snapshot is accessible to the users and projects that were selected while launching an instance that the snapshot is based on.

#### 4.6.1. Creating an instance snapshot

##### NOTE

If you intend to use an instance snapshot as a template to create new instances, you must ensure that the disk state is consistent. Before you create a snapshot, set the snapshot image metadata property **os\_require\_quiesce=yes**:

```
$ openstack image set --property os_require_quiesce=yes <image_id>
```

For this to work, the guest must have the **qemu-guest-agent** package installed, and the image must be created with the metadata property parameter **hw\_qemu\_guest\_agent=yes** set.:

```
$ openstack image create \  
--disk-format raw \  
--container-format bare \  
--file <file_name> \  
--is-public True \  
--property hw_qemu_guest_agent=yes \  
--progress \  
--name <name>
```

If you unconditionally enable the **hw\_qemu\_guest\_agent=yes** parameter, then you are adding another device to the guest. This consumes a PCI slot, and limits the number of other devices you can allocate to the guest. It also causes Windows guests to display a warning message about an unknown hardware device.

For these reasons, setting the **hw\_qemu\_guest\_agent=yes** parameter is optional, and you must use the parameter only for images that require the QEMU guest agent.

1. In the dashboard, select **Project > Compute > Instances**
2. Select the instance from which you want to create a snapshot.
3. In the **Actions** column, click **Create Snapshot**.
4. In the **Create Snapshot** dialog, enter a name for the snapshot and click **Create Snapshot**.  
The **Images** category now shows the instance snapshot.

To launch an instance from a snapshot, select the snapshot and click **Launch**.

#### 4.6.2. Managing a snapshot

1. In the dashboard, select **Project > Images**.
2. All snapshots you created, appear under the **Project** option.
3. For every snapshot you create, you can perform the following functions, using the dropdown list:
  - a. Use the **Create Volume** option to create a volume and entering the values for volume name, description, image source, volume type, size and availability zone. For more information, see [Create a Volume](#) in the *Storage Guide*.
  - b. Use the **Edit Image** option to update the snapshot image by updating the values for name, description, Kernel ID, Ramdisk ID, Architecture, Format, Minimum Disk (GB), Minimum RAM (MB), public or private. For more information, see [Section 1.2.3, "Updating an image"](#).
  - c. Use the **Delete Image** option to delete the snapshot.

#### 4.6.3. Rebuilding an instance to a state in a snapshot

In an event that you delete an instance on which a snapshot is based, the snapshot still stores the instance ID. You can check this information by using the `nova image-list` command and use the snapshot to restore the instance.

1. In the dashboard, select **Project > Compute > Images**
2. Select the snapshot from which you want to restore the instance.
3. In the **Actions** column, click **Launch Instance**.
4. In the **Launch Instance** dialog, enter a name and the other details for the instance and click **Launch**.

For more information on launching an instance, see [Section 4.1.2, "Launching an instance"](#).

#### 4.6.4. Consistent snapshots

Previously, file systems had to be quiesced manually (`fsfreeze`) before taking a snapshot of active instances for consistent backups.

The Compute **libvirt** driver automatically requests the QEMU Guest Agent to freeze the file systems (and applications if **fsfreeze-hook** is installed) during an image snapshot. Support for quiescing file systems enables scheduled, automatic snapshots at the block device level.

This feature is valid only if the QEMU Guest Agent is installed (**qemu-ga**) and the image metadata enables the agent (**hw\_qemu\_guest\_agent=yes**).



#### NOTE

Do not use snapshots as a substitute for system backups.

## 4.7. USING RESCUE MODE FOR INSTANCES

Compute has a method to reboot a virtual machine in rescue mode. Rescue mode provides a mechanism for access when the virtual machine image renders the instance inaccessible. A rescue virtual machine allows a user to fix their virtual machine by accessing the instance with a new root password. This feature is useful if the file system of an instance is corrupted. By default, rescue mode starts an instance from the initial image attaching the current boot disk as a secondary one.

### 4.7.1. Preparing an image for a rescue mode instance

Due to the fact that both the boot disk and the disk for rescue mode have same UUID, sometimes the virtual machine can be booted from the boot disk instead of the disk for rescue mode.

To avoid this issue, you should create a new image as rescue image based on the procedure in [Section 1.2.1, "Creating an image"](#):



#### NOTE

The **rescue** image is stored in **glance** and configured in the **nova.conf** as a default, or you can select when you do the rescue.

#### 4.7.1.1. Rescuing an image that uses ext4 file system

When the base image uses **ext4** file system, you can create a rescue image from it by using the following procedure:

1. Change the **UUID** to a random value by using the **tune2fs** command:

```
# tune2fs -U random /dev/<device_node>
```

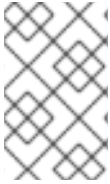
Replace **<device\_node>** with the root device node, for example, **sda** or **vda**.

2. Verify the details of the file system, including the new UUID:

```
# tune2fs -l
```

3. Update the **/etc/fstab** to use the new UUID. You might need to repeat this for any additional partitions that you have that are mounted in the **fstab** by UUID.
4. Update the **/boot/grub2/grub.conf** file and update the UUID parameter with the new UUID of the root disk.
5. Shut down and use this image as your rescue image. This causes the rescue image to have a new random UUID that does not conflict with the instance that you are rescuing.



**NOTE**

The XFS file system cannot change the UUID of the root device on the running virtual machine. Reboot the virtual machine until the virtual machine is launched from the disk for rescue mode.

### 4.7.2. Adding the rescue image to the OpenStack Image service

When you have completed modifying the UUID of your image, use the following commands to add the generated rescue image to the OpenStack Image service:

1. Add the rescue image to the Image service:

```
# openstack image create --name <image_name> --disk-format qcow2 \
  --container-format bare --is-public True --file <image_path>
```

Replace **<image\_name>** with the name of the image and **<image\_path>** with the location of the image.

2. Use the **image list** command to obtain the **<image\_id>** required to launch an instance in the rescue mode.

```
# openstack image list
```

You can also upload an image by using the OpenStack Dashboard, see [Section 1.2.2, “Uploading an image”](#).

### 4.7.3. Launching an instance in rescue mode

1. Because you need to rescue an instance with a specific image, rather than the default one, use the **--image** parameter:

```
# openstack server rescue --image <image> <instance>
```

- Replace **<image>** with the name or ID of the image you want to use.
- Replace **<instance>** with the name or ID of the instance that you want to rescue.

**NOTE**

For more information on rescuing an instance, see [https://access.redhat.com/documentation/en-us/red\\_hat\\_openstack\\_platform/16.1/html/instances\\_and\\_images\\_guide/assembly-managing-an-instance\\_instances#rescuing-an-instance\\_instances](https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/16.1/html/instances_and_images_guide/assembly-managing-an-instance_instances#rescuing-an-instance_instances)

By default, the instance has 60 seconds to shut down. You can override the timeout value on a per image basis by using the image metadata setting **os\_shutdown\_timeout** to specify the time that different types of operating systems require to shut down cleanly.

2. Reboot the virtual machine.
3. Confirm the status of the virtual machine is *RESCUE* on the controller node by using **nova list** command or by using dashboard.

4. Log in to the new virtual machine dashboard by using the password for rescue mode.

You can now make the necessary changes to your instance to fix any issues.

#### 4.7.4. Unrescuing an instance

You can **unrescue** the fixed instance to restart it from the boot disk.

1. Execute the following commands on the controller node.

```
# nova unrescue <virtual_machine_id>
```

Here **<virtual\_machine\_id>** is ID of a virtual machine that you want to unrescue.

The status of your instance returns to **ACTIVE** after the unrescue operation has completed successfully.  
:leveloffset: +3

## 4.8. CREATING A CUSTOMIZED INSTANCE

Cloud users can specify additional data to use when they launch an instance, such as a shell script that the instance runs on boot. The cloud user can use the following methods to pass data to instances:

### User data

Use to include instructions in the instance launch command for **cloud-init** to execute.

### Instance metadata

A list of key-value pairs that you can specify when you create or update an instance.

You can access the additional data passed to the instance by using a config drive or the metadata service.

### Config drive

You can attach a config drive to an instance when it boots. The config drive is presented to the instance as a read-only drive. The instance can mount this drive and read files from it. You can use the config drive as a source for **cloud-init** information. Config drives are useful when combined with **cloud-init** for server bootstrapping, and when you want to pass large files to your instances. For example, you can configure **cloud-init** to automatically mount the config drive and run the setup scripts during the initial instance boot. Config drives are created with the volume label of **config-2**, and attached to the instance when it boots. The contents of any additional files passed to the config drive are added to the **user\_data** file in the **openstack/{version}/** directory of the config drive. **cloud-init** retrieves the user data from this file.

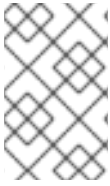
### Metadata service

Uses a REST API to retrieve data specific to an instance. Instances access this service at **169.254.169.254** or at **fe80::a9fe:a9fe**.

**cloud-init** can use both a config drive and the metadata service to consume the additional data for customizing an instance. The **cloud-init** package supports several data input formats. Shell scripts and the **cloud-config** format are the most common input formats:

- Shell scripts: The data declaration begins with **#!** or **Content-Type: text/x-shellscript**. Shell scripts are invoked last in the boot process.

- **cloud-config** format: The data declaration begins with **#cloud-config** or **Content-Type: text/cloud-config**. **cloud-config** files must be valid YAML to be parsed and executed by **cloud-init**.



## NOTE

**cloud-init** has a maximum user data size of 16384 bytes for data passed to an instance. You cannot change the size limit, therefore use gzip compression when you need to exceed the size limit.

### 4.8.1. Customizing an instance by using user data

You can use user data to include instructions in the instance launch command. **cloud-init** executes these commands to customize the instance as the last step in the boot process.

#### Procedure

1. Create a file with instructions for **cloud-init**. For example, create a bash script that installs and enables a web server on the instance:

```
$ vim /home/scripts/install_httpd
#!/bin/bash

yum -y install httpd python-psycopg2
systemctl enable httpd --now
```

2. Launch an instance with the **--user-data** option to pass the bash script:

```
$ openstack server create \
--image rhel8 \
--flavor default \
--nic net-id=web-server-network \
--security-group default \
--key-name web-server-keypair \
--user-data /home/scripts/install_httpd \
--wait web-server-instance
```

3. When the instance state is active, attach a floating IP address:

```
$ openstack floating ip create web-server-network
$ openstack server add floating ip web-server-instance 172.25.250.123
```

4. Log in to the instance with SSH:

```
$ ssh -i ~/.ssh/web-server-keypair cloud-user@172.25.250.123
```

5. Check that the customization was successfully performed. For example, to check that the web server has been installed and enabled, enter the following command:

```
$ curl http://localhost | grep Test
<title>Test Page for the Apache HTTP Server on Red Hat Enterprise Linux</title>
<h1>Red Hat Enterprise Linux <strong>Test Page</strong></h1>
```

- Review the `/var/log/cloud-init.log` file for relevant messages, such as whether or not the **cloud-init** executed:

```
$ sudo less /var/log/cloud-init.log
...output omitted...
...util.py[DEBUG]: Cloud-init v. 0.7.9 finished at Sat, 23 Jun 2018 02:26:02 +0000.
Datasource DataSourceOpenStack [net,ver=2]. Up 21.25 seconds
```

### 4.8.2. Customizing an instance by using metadata

You can use instance metadata to specify the properties of an instance in the instance launch command.

#### Procedure

- Launch an instance with the **--property <key=value>** option. For example, to mark the instance as a webserver, set the following property:

```
$ openstack server create \
--image rhel8 \
--flavor default \
--property role=webservers \
--wait web-server-instance
```

- Optional: Add an additional property to the instance after it is created, for example:

```
$ openstack server set \
--property region=emea \
--wait web-server-instance
```

### 4.8.3. Customizing an instance by using a config drive

You can create a config drive for an instance that is attached during the instance boot process. You can pass content to the config drive that the config drive makes available to the instance.

#### Procedure

- Enable the config drive, and specify a file that contains content that you want to make available in the config drive. For example, the following command creates a new instance named **config-drive-instance** and attaches a config drive that contains the contents of the file **my-user-data.txt**:

```
(overcloud)$ openstack server create --flavor m1.tiny \
--config-drive true \
--user-data ./my-user-data.txt \
--image cirros config-drive-instance
```

This command creates the config drive with the volume label of **config-2**, which is attached to the instance when it boots, and adds the contents of **my-user-data.txt** to the **user\_data** file in the **openstack/{version}/** directory of the config drive.

- Log in to the instance.

### 3. Mount the config drive:

- If the instance OS uses **udev**:

```
# mkdir -p /mnt/config  
# mount /dev/disk/by-label/config-2 /mnt/config
```

- If the instance OS does not use **udev**, you need to first identify the block device that corresponds to the config drive:

```
# blkid -t LABEL="config-2" -odevice  
/dev/vdb  
# mkdir -p /mnt/config  
# mount /dev/vdb /mnt/config
```

## CHAPTER 5. MIGRATING VIRTUAL MACHINE INSTANCES BETWEEN COMPUTE NODES

You sometimes need to migrate instances from one Compute node to another Compute node in the overcloud, to perform maintenance, rebalance the workload, or replace a failed or failing node.

### Compute node maintenance

If you need to temporarily take a Compute node out of service, for instance, to perform hardware maintenance or repair, kernel upgrades and software updates, you can migrate instances running on the Compute node to another Compute node.

### Failing Compute node

If a Compute node is about to fail and you need to service it or replace it, you can migrate instances from the failing Compute node to a healthy Compute node.

### Failed Compute nodes

If a Compute node has already failed, you can evacuate the instances. You can rebuild instances from the original image on another Compute node, using the same name, UUID, network addresses, and any other allocated resources the instance had before the Compute node failed.

### Workload rebalancing

You can migrate one or more instances to another Compute node to rebalance the workload. For example, you can consolidate instances on a Compute node to conserve power, migrate instances to a Compute node that is physically closer to other networked resources to reduce latency, or distribute instances across Compute nodes to avoid hot spots and increase resiliency.

Director configures all Compute nodes to provide secure migration. All Compute nodes also require a shared SSH key to provide the users of each host with access to other Compute nodes during the migration process. Director creates this key using the **OS::TripleO::Services::NovaCompute** composable service. This composable service is one of the main services included on all Compute roles by default. For more information, see [Composable Services and Custom Roles](#) in the *Advanced Overcloud Customization* guide.



### NOTE

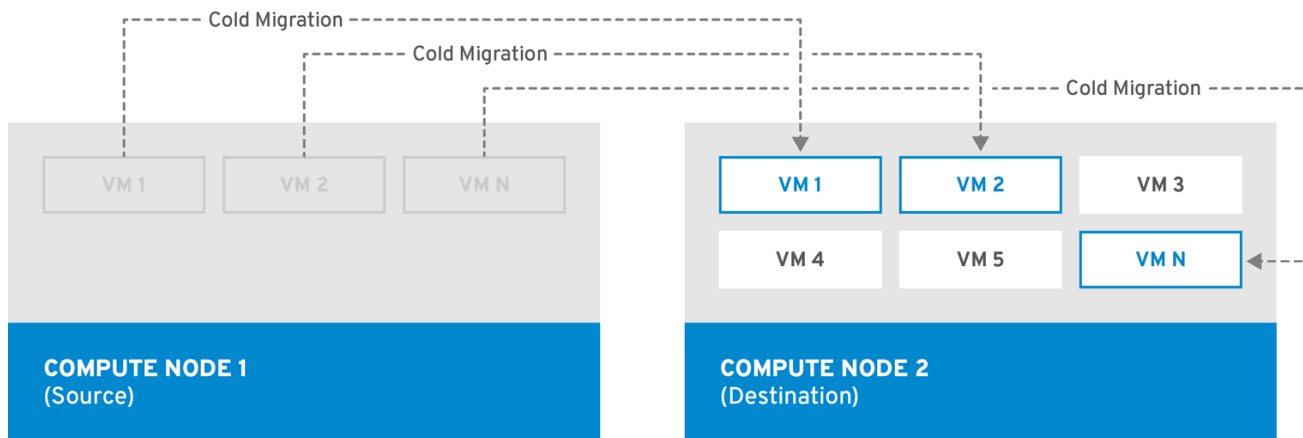
If you have a functioning Compute node, and you want to make a copy of an instance for backup purposes, or to copy the instance to a different environment, follow the procedure in [Importing virtual machines into the overcloud](#) in the *Director Installation and Usage* guide.

## 5.1. MIGRATION TYPES

Red Hat OpenStack Platform (RHOSP) supports the following types of migration.

### Cold migration

Cold migration, or non-live migration, involves shutting down a running instance before migrating it from the source Compute node to the destination Compute node.



OPENSTACK\_11\_0419

Cold migration involves some downtime for the instance. The migrated instance maintains access to the same volumes and IP addresses.

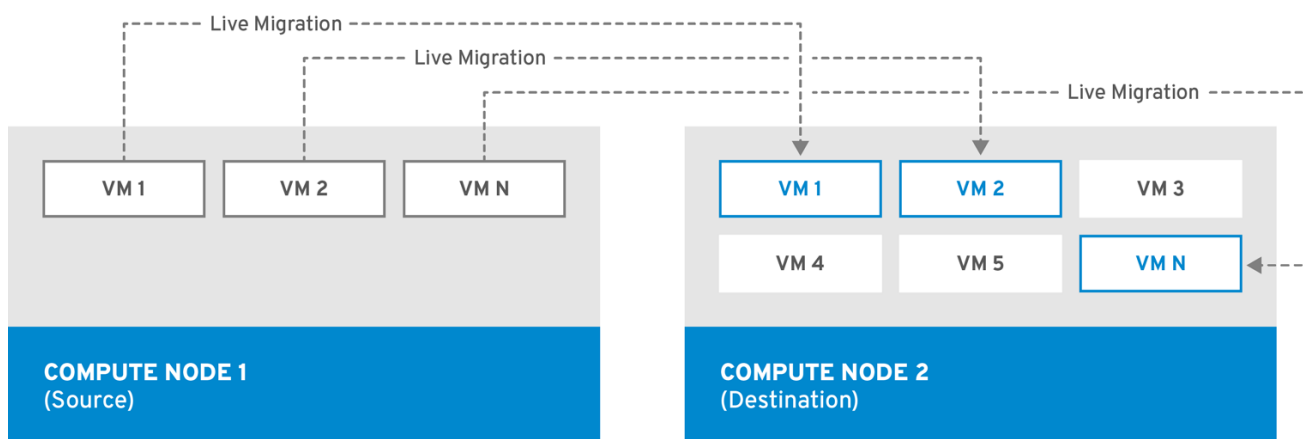


### NOTE

Cold migration requires that both the source and destination Compute nodes are running.

### Live migration

Live migration involves moving the instance from the source Compute node to the destination Compute node without shutting it down, and while maintaining state consistency.



OPENSTACK\_11\_0419

Live migrating an instance involves little or no perceptible downtime. In some cases, instances cannot use live migration. For more information, see [Migration Constraints](#).



### NOTE

Live migration requires that both the source and destination Compute nodes are running.

### Evacuation

If you need to migrate instances because the source Compute node has already failed, you can evacuate the instances.

## 5.2. MIGRATION CONSTRAINTS

In some cases, migrating instances involves additional constraints. Migration constraints typically arise with block migration, configuration disks, or when one or more instances access physical hardware on the Compute node.

### CPU constraints

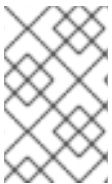
The source and destination Compute nodes must have the same CPU architecture. For example, Red Hat does not support migrating an instance from an **x86\_64** CPU to a **ppc64le** CPU. In some cases, the CPU of the source and destination Compute node must match exactly, such as instances that use CPU host passthrough. In all cases, the CPU features of the destination node must be a superset of the CPU features on the source node. Using CPU pinning introduces additional constraints. For more information, see [Live migration constraints](#).

### Memory constraints

The destination Compute node must have sufficient available RAM. Memory oversubscription can cause migration to fail. Additionally, instances that use a NUMA topology must have sufficient available RAM on the same NUMA node on the destination Compute node.

### Block migration constraints

Migrating instances that use disks that are stored locally on a Compute node takes significantly longer than migrating volume-backed instances that use shared storage, such as Red Hat Ceph Storage. This latency arises because OpenStack Compute (nova) migrates local disks block-by-block between the Compute nodes over the control plane network by default. By contrast, volume-backed instances that use shared storage, such as Red Hat Ceph Storage, do not have to migrate the volumes, because each Compute node already has access to the shared storage.



#### NOTE

Network congestion in the control plane network caused by migrating local disks or instances that consume large amounts of RAM might impact the performance of other systems that use the control plane network, such as RabbitMQ.

### Read-only drive migration constraints

Migrating a drive is supported only if the drive has both read and write capabilities. For example, OpenStack Compute (nova) cannot migrate a CD-ROM drive or a read-only config drive. However, OpenStack Compute (nova) can migrate a drive with both read and write capabilities, including a config drive with a drive format such as **vfat**.

### Live migration constraints

#### Migration between RHEL minor versions

In general, you can live migrate an instance between RHEL minor versions when the instance machine type version on the source Compute node is equal to or less than that of the destination Compute node. For example, you cannot live migrate an instance with a RHEL-7.6 machine type running on a RHEL-7.6 Compute node to a RHEL-7.5 Compute node, because the RHEL-7.5 Compute node does not know of the RHEL-7.6 machine type.

However, if you do not set a machine type explicitly, the instance receives the default machine type and live migration can succeed across supported RHEL minor versions. For example, you can live migrate an instance that has the default machine type, RHEL-7.6, from a RHEL-7.8 Compute node to a RHEL-7.7 Compute node.



## No new operations during migration

To achieve state consistency between the copies of the instance on the source and destination nodes, RHOSP must prevent new operations during live migration. Otherwise, live migration might take a long time or potentially never end if writes to memory occur faster than live migration can replicate the state of the memory.

## NUMA, CPU pinning, huge pages and DPDK

OpenStack Compute can live migrate an instance that uses NUMA, CPU pinning or DPDK when the environment meets the following conditions:

- The destination Compute node must have sufficient capacity on the same NUMA node that the instance uses on the source Compute node. For example, if an instance uses **NUMA 0** on **overcloud-compute-0**, to live migrate the instance to **overcloud-compute-1**, you must ensure that **overcloud-compute-1** has sufficient capacity on **NUMA 0** to support the instance.
- **NovaEnableNUMALiveMigration** is set to "True" in the Compute configuration. This parameter is enabled by default only when the Compute host is configured for an OVS-DPDK deployment.
- The **NovaSchedulerDefaultFilters** parameter in the Compute configuration must include the values **AggregateInstanceExtraSpecsFilter** and **NUMATopologyFilter**.
- **CPU Pinning:** When a flavor uses CPU pinning, the flavor implicitly introduces a NUMA topology to the instance and maps its CPUs and memory to specific host CPUs and memory. The difference between a simple NUMA topology and CPU pinning is that NUMA uses a range of CPU cores, whereas CPU pinning uses specific CPU cores. For more information, see [Configuring CPU pinning with NUMA](#). To live migrate instances that use CPU pinning, the destination host must be empty and must have equivalent hardware.
- **Data Plane Development Kit (DPDK):** When an instance uses DPDK, such as an instance running Open vSwitch with **dpdk-netdev**, the instance also uses huge pages. Huge pages impose a NUMA topology such that OpenStack Compute (nova) pins the instance to a NUMA node. When you migrate instances that use DPDK, the destination Compute node must have an identical hardware specification and configuration as the source Compute node. Additionally, there must not be any instances running on the destination Compute node to ensure that it preserves the NUMA topology of the source Compute node.

## Constraints that preclude live migration

Live migration is not possible when the instance is configured for the following features:

- **Single-root Input/Output Virtualization (SR-IOV):** You can assign SR-IOV Virtual Functions (VFs) to instances. However, this prevents live migration. Unlike a regular network device, an SR-IOV VF network device does not have a permanent unique MAC address. The VF network device receives a new MAC address each time the Compute node reboots, or when the scheduler migrates the instance to a new Compute node. Consequently, OpenStack Compute cannot live migrate instances that use SR-IOV. You must cold migrate instances that use SR-IOV.
- **PCI passthrough:** QEMU/KVM hypervisors support attaching PCI devices on the Compute node to an instance. Use PCI passthrough to give an instance exclusive access to PCI devices, which appear and behave as if they are physically attached to the operating system of the instance. However, because PCI passthrough involves physical addresses, OpenStack Compute does not support live migration of instances using PCI passthrough.

## 5.3. PREPARING TO MIGRATE

Before you migrate one or more instances, you need to determine the Compute node names and the IDs of the instances to migrate.

### Procedure

1. Identify the source Compute node host name and the destination Compute node host name:

```
(undercloud)$ source ~/overcloudrc
(overcloud)$ openstack compute service list
```

2. List the instances on the source Compute node and locate the ID of the instance or instances that you want to migrate:

```
(overcloud)$ openstack server list --host <source> --all-projects
```

Replace **<source>** with the name or ID of the source Compute node.

3. Optional: To shut down the source Compute node for maintenance, disable the source Compute node from the undercloud to ensure that the scheduler does not attempt to assign new instances to the source Compute node during maintenance:

```
(overcloud)$ source ~/stackrc
(undercloud)$ openstack compute service set <source> nova-compute --disable
```

Replace **<source>** with the name or ID of the source Compute node.

If you are not migrating NUMA, CPU-pinned or DPDK instances, you are now ready to perform the migration. Follow the required procedure detailed in [Cold migrating an instance](#) or [Live migrating an instance](#).

If you are migrating NUMA, CPU-pinned or DPDK instances, you need to prepare the destination node. Complete the procedure detailed in [Section 5.4, "Additional preparation for DPDK instances"](#).

## 5.4. ADDITIONAL PREPARATION FOR DPDK INSTANCES

If you are migrating NUMA, CPU-pinned or DPDK instances, you need to prepare the destination node.

### Procedure

1. If the destination Compute node for NUMA, CPU-pinned or DPDK instances is not disabled, disable it to prevent the scheduler from assigning instances to the node:

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

Replace **<dest>** with the name or ID of the destination Compute node.

2. Ensure that the destination Compute node has no instances, except for instances that you previously migrated from the source Compute node when you migrated multiple DPDK or NUMA instances:

```
(overcloud)$ openstack server list --host <dest> --all-projects
```

Replace **<dest>** with the name or ID of the destination Compute node.

3. Ensure that the destination Compute node has sufficient resources to run the NUMA, CPU-pinned or DPDK instance:

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

Replace **<dest>** with the name or ID of the destination Compute node.

4. To discover NUMA information about the source or destination Compute nodes, run the following commands:

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

Use **ssh** to connect to **overcloud-compute-n** where **overcloud-compute-n** is the source or destination Compute node.

5. If you do not know if an instance uses NUMA, check the flavor of the instance:

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

- Replace **<vm>** with the name or ID of the instance.
  - Replace **<flavor>** with the name or ID of the flavor.
    - If the **properties** field includes **hw:mem\_page\_size** with a value other than **any**, such as **2MB**, **2048** or **1GB**, the instance has a NUMA topology.
    - If the **properties** field includes **aggregate\_instance\_extra\_specs:pinned='true'**, the instance uses CPU pinning.
    - If the **properties** field includes **hw:numa\_nodes**, the OpenStack Compute (nova) service restricts the instance to a specific NUMA node.
6. For each instance that uses NUMA, you can retrieve information about the NUMA topology from the underlying Compute node so that you can verify that the NUMA topology on the destination Compute node reflects the NUMA topology of the source Compute node after migration is complete. You can use the following commands to perform this check:

- To view details about NUMA and CPU pinning, run the following command:

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

Replace **<vm>** with the name of the instance.

- To view details about which NUMA node the instance is using, run the following command:

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

Replace **<vm>** with the name of the instance.

## 5.5. COLD MIGRATING AN INSTANCE

Cold migrating an instance involves stopping the instance and moving it to another Compute node. Cold migration facilitates migration scenarios that live migrating cannot facilitate, such as migrating instances that use PCI passthrough or Single-Root Input/Output Virtualization (SR-IOV). The scheduler automatically selects the destination Compute node. For more information, see [Migration Constraints](#).



### NOTE

During cold migrations, the Compute service rebuilds the migrated instances from scratch, and adjusts the machine type to the machine type of the destination Compute node. Therefore, if you cold migrate an instance with a RHEL-7.5 machine type running on a RHEL-7.5 Compute node, to a RHEL-7.6 Compute node, the migrated instance on the destination Compute node will have a RHEL-7.6 machine type.

### Procedure

1. To cold migrate an instance, run the following command to power off and move the instance:

```
(overcloud)$ openstack server migrate <vm> --wait
```

- Replace **<vm>** with the name or ID of the instance to migrate.
  - Specify the **--block-migration** flag if migrating a locally stored volume.
2. Wait for migration to complete. See [Checking migration status](#) to check the status of the migration.
  3. Check the status of the instance:

```
(overcloud)$ openstack server list --all-projects
```

A status of "VERIFY\_RESIZE" indicates you need to confirm or revert the migration:

- If the migration worked as expected, confirm it:

```
(overcloud)$ openstack server resize --confirm <vm>`
```

Replace **<vm>** with the name or ID of the instance to migrate. A status of "ACTIVE" indicates that the instance is ready to use.

- If the migration did not work as expected, revert it:

```
(overcloud)$ openstack server resize --revert <vm>`
```

Replace **<vm>** with the name or ID of the instance.

4. Restart the instance:

```
(overcloud)$ openstack server start <vm>
```

Replace **<vm>** with the name or ID of the instance.

When you finish migrating the instances, proceed to [Completing the migration](#).

## 5.6. LIVE MIGRATING AN INSTANCE

Live migration moves an instance from a source Compute node to a destination Compute node with a minimal amount of downtime. Live migration might not be appropriate for all instances. For more information, see [Section 5.2, "Migration constraints"](#).



### NOTE

Live migrations preserve the instance machine type on the destination Compute node. Therefore, if you live migrate an instance with a RHEL-7.5 machine type running on a RHEL-7.5 Compute node, to a RHEL-7.6 Compute node, the migrated instance on the destination Compute node retains the RHEL-7.5 machine type. To change the machine type, you must set the image metadata property **hw\_machine\_type**, or set the **NovaHWMachineType** parameter on each Compute node.

### Procedure

- To live migrate an instance, specify the instance and the destination Compute node:

```
(overcloud)$ openstack server migrate <vm> --live <dest> --wait
```

- Replace **<vm>** with the name or ID of the instance.
- Replace **<dest>** with the name or ID of the destination Compute node.



### NOTE

The **openstack server migrate** command covers migrating instances with shared storage, which is the default. Specify the **--block-migration** flag to migrate a locally stored volume:

```
(overcloud)$ openstack server migrate <vm> --live <dest> --wait --block-migration
```

- Confirm that the instance is migrating:

```
$ openstack server show <vm>
```

```
+-----+
| Field      | Value                |
+-----+
| ...        | ...                  |
| status     | MIGRATING           |
| ...        | ...                  |
+-----+
```

3. Wait for migration to complete. See [Checking migration status](#) to check the status of the migration.
4. Check the status of the instance to confirm if the migration was successful:

```
(overcloud)$ openstack server list --host <dest> --all-projects
```

Replace **<dest>** with the name or ID of the destination Compute node.

5. Optional: For instances that use NUMA, CPU-pinning, or DPDK, retrieve information about the NUMA topology from a Compute node to compare it with the NUMA topology that you retrieved during the preparing to migrate procedure. Comparing the NUMA topologies of the source and destination Compute nodes ensures that the source and destination Compute nodes use the same NUMA topology.

- To view details about NUMA and CPU pinning, run the following command:

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

- Replace **overcloud-compute-n** with the host name of the Compute node.
- Replace **<vm>** with the name of the instance.

- To view details about which NUMA node the instance is using, run the following command:

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

- Replace **overcloud-compute-n** with the host name of the Compute node.
- Replace **<vm>** with the name or ID of the instance.

When you finish migrating the instances, proceed to [Completing the migration](#).

## 5.7. CHECKING MIGRATION STATUS

Migration involves several state transitions before migration is complete. During a healthy migration, the migration state typically transitions as follows:

1. **Queued:** The Compute service has accepted the request to migrate an instance, and migration is pending.
2. **Preparing:** The Compute service is preparing to migrate the instance.
3. **Running:** The Compute service is migrating the instance.
4. **Post-migrating:** The Compute service has built the instance on the destination Compute node and is releasing resources on the source Compute node.
5. **Completed:** The Compute service has completed migrating the instance and finished releasing resources on the source Compute node.

### Procedure

1. Retrieve the list of migration IDs for the instance:

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

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

Replace **<vm>** with the name or ID of the instance.

2. Show the status of the migration:

```
$ nova server-migration-show <vm> <migration-id>
```

- Replace **<vm>** with the name or ID of the instance.
- Replace **<migration-id>** with the ID of the migration.

Running the **nova server-migration-show** command returns the following example output:

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

### TIP

The OpenStack Compute service measures progress of the migration by the number of remaining memory bytes to copy. If this number does not decrease over time, the migration might be unable to complete, and the Compute service might abort it.

Sometimes instance migration can take a long time or encounter errors. For more information, see [Section 5.10, "Troubleshooting migration"](#).

## 5.8. COMPLETING THE MIGRATION

After you migrate one or more instances, you need to re-enable the source Compute nodes from the undercloud to ensure that the scheduler can assign new instances to the source Compute node. For

migrated instances that use DPDK, you must also re-enable the destination Compute node from the undercloud.

## Procedure

1. Re-enable the source Compute node:

```
(overcloud)$ source ~/stackrc
(undercloud)$ openstack compute service set <source> nova-compute --enable
```

Replace **<source>** with the host name of the source Compute node.

2. Optional: For instances that use DPDK, re-enable the destination Compute node from the undercloud:

```
(undercloud)$ openstack compute service set <dest> nova-compute --enable
```

Replace **<dest>** with the host name of the destination Compute node.

## 5.9. EVACUATING AN INSTANCE

If you want to move an instance from a dead or shut-down Compute node to a new host in the same environment, you can evacuate it. The evacuate process rebuilds the instance on another Compute node. If the instance uses shared storage, the instance root disk is not rebuilt during the evacuate process, as the disk remains accessible by the destination Compute node. If the instance does not use shared storage, then the instance root disk is also rebuilt on the destination Compute node.



### NOTE

- You can only perform an evacuation when the Compute node is fenced, and the API reports that the state of the Compute node is "down" or "forced-down". If the Compute node is not reported as "down" or "forced-down", the **evacuate** command fails.
- To perform an evacuation, you must be a cloud administrator.
- During evacuations, the Compute service rebuilds the evacuated instances from scratch, and adjusts the machine type to the machine type of the destination Compute node. Therefore, if you evacuate an instance with a RHEL-7.5 machine type running on a RHEL-7.5 Compute node, to a RHEL-7.6 Compute node, the migrated instance on the destination Compute node will have a RHEL-7.6 machine type.

### 5.9.1. Evacuating one instance

You can evacuate instances one at a time.

#### Procedure

1. Log onto the failed Compute node as an administrator.
2. Disable the Compute node:



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

- Replace **<host>** with the name of the Compute node to evacuate the instance from.
- Replace **<service>** with the name of the service to disable, for example **nova-compute**.

3. To evacuate an instance, run the following command:

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

- Replace **<pass>** with the admin password to set for the evacuated instance. If a password is not specified, a random password is generated and output when the evacuation is complete.
- Replace **<vm>** with the name or ID of the instance to evacuate.
- Replace **[dest]** with the name of the Compute node to evacuate the instance to. If you do not specify the destination Compute node, the Compute scheduler selects one for you. You can find possible Compute nodes by using the following command:

```
(overcloud)[stack@director ~]$ openstack hypervisor list
```

## 5.9.2. Evacuating all instances on a host

You can evacuate all instances on a specified Compute node.

### Procedure

1. Log onto the failed Compute node as an administrator.
2. Disable the Compute node:

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

- Replace **<host>** with the name of the Compute node to evacuate the instances from.
- Replace **<service>** with the name of the service to disable, for example **nova-compute**.

3. Evacuate all instances on a specified Compute node:

```
(overcloud)[stack@director ~]$ nova host-evacuate [--target_host <dest>] [--force] <host>
```

- Replace **<dest>** with the name of the destination Compute node to evacuate the instances to. If you do not specify the destination, the Compute scheduler selects one for you. You can find possible Compute nodes by using the following command:

```
(overcloud)[stack@director ~]$ openstack hypervisor list
```

- Replace **<host>** with the name of the Compute node to evacuate the instances from.

## 5.9.3. Configuring shared storage

If you are using shared storage, export the instance directory for the Compute service to the two nodes, and ensure that the nodes have access. The directory path is set in the **state\_path** and **instances\_path** parameters in your Compute environment file. This procedure uses the default value, which is **/var/lib/nova/instances**. Only users with root access can set up shared storage. The Compute service user in the following procedure must be the same across Controller and Compute nodes.

## Procedure

1. Perform the following steps on the Controller node:

- a. Ensure that the **/var/lib/nova/instances** directory has read-write access by the Compute service user, as shown in the following example:

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

- b. Add the following lines to the **/etc/exports** file:

```
/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)
```

Replace **node1\_IP** and **node2\_IP** for the IP addresses of the two Compute nodes, for example:

```
/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. Export the **/var/lib/nova/instances** directory to the Compute nodes:

```
# exportfs -avr
```

- d. Restart the NFS server:

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

2. Perform the following steps on each Compute node:

- a. Ensure that the **/var/lib/nova/instances** directory exists locally.

- b. Add the following line to the **/etc/fstab** file:

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

- c. Mount the controller's instance directory to mount all the devices listed in **/etc/fstab**:

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

- d. Ensure that QEMU can access the directory's images:

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

- e. Ensure that the node can see the instances directory with:

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



## NOTE

You can also run the following to view all mounted devices:

```
# df -k
```

## 5.10. TROUBLESHOOTING MIGRATION

The following issues can arise during instance migration:

- The migration process encounters errors.
- The migration process never ends.
- Performance of the instance degrades after migration.

### 5.10.1. Errors during migration

The following issues can send the migration operation into an **error** state:

- Running a cluster with different versions of Red Hat OpenStack Platform (RHOSP).
- Specifying an instance ID that cannot be found.
- The instance you are trying to migrate is in an **error** state.
- The Compute service is shutting down.
- A race condition occurs.
- Live migration enters a **failed** state.

When live migration enters a **failed** state, it is typically followed by an **error** state. The following common issues can cause a **failed** state:

- A destination Compute host is not available.
- A scheduler exception occurs.
- The rebuild process fails due to insufficient computing resources.
- A server group check fails.
- The instance on the source Compute node gets deleted before migration to the destination Compute node is complete.

### 5.10.2. Never-ending live migration

Live migration can fail to complete, which leaves migration in a perpetual **running** state. A common reason for a live migration that never completes is that client requests to the instance running on the source Compute node create changes that occur faster than the Compute service can replicate them to the destination Compute node.

Use one of the following methods to address this situation:

- Abort the live migration.
- Force the live migration to complete.

### Aborting live migration

If the instance state changes faster than the migration procedure can copy it to the destination node, and you do not want to temporarily suspend the instance operations, you can abort the live migration.

#### Procedure

1. Retrieve the list of migrations for the instance:

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

Replace **<vm>** with the name or ID of the instance.

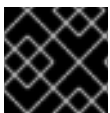
2. Abort the live migration:

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

- Replace **<vm>** with the name or ID of the instance.
- Replace **<migration-id>** with the ID of the migration.

### Forcing live migration to complete

If the instance state changes faster than the migration procedure can copy it to the destination node, and you want to temporarily suspend the instance operations to force migration to complete, you can force the live migration procedure to complete.



#### IMPORTANT

Forcing live migration to complete might lead to perceptible downtime.

#### Procedure

1. Retrieve the list of migrations for the instance:

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

Replace **<vm>** with the name or ID of the instance.

2. Force the live migration to complete:

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

- Replace **<vm>** with the name or ID of the instance.
- Replace **<migration-id>** with the ID of the migration.

### 5.10.3. Instance performance degrades after migration

For instances that use a NUMA topology, the source and destination Compute nodes must have the same NUMA topology and configuration. The NUMA topology of the destination Compute node must have sufficient resources available. If the NUMA configuration between the source and destination Compute nodes is not the same, it is possible that live migration succeeds while the instance performance degrades. For example, if the source Compute node maps NIC 1 to NUMA node 0, but the destination Compute node maps NIC 1 to NUMA node 5, after migration the instance might route network traffic from a first CPU across the bus to a second CPU with NUMA node 5 to route traffic to NIC 1. This can result in expected behavior, but degraded performance. Similarly, if NUMA node 0 on the source Compute node has sufficient available CPU and RAM, but NUMA node 0 on the destination Compute node already has instances using some of the resources, the instance might run correctly but suffer performance degradation. For more information, see [Section 5.2, "Migration constraints"](#).

## CHAPTER 6. CONFIGURING PCI PASSTHROUGH

You can use PCI passthrough to attach a physical PCI device, such as a graphics card or a network device, to an instance. If you use PCI passthrough for a device, the instance reserves exclusive access to the device for performing tasks, and the device is not available to the host.



### IMPORTANT

#### Using PCI passthrough with routed provider networks

The Compute service does not support single networks that span multiple provider networks. When a network contains multiple physical networks, the Compute service only uses the first physical network. Therefore, if you are using routed provider networks you must use the same **physical\_network** name across all the Compute nodes.

If you use routed provider networks with VLAN or flat networks, you must use the same **physical\_network** name for all segments. You then create multiple segments for the network and map the segments to the appropriate subnets.

To enable your cloud users to create instances with PCI devices attached, you must complete the following:

1. Designate Compute nodes for PCI passthrough.
2. Configure the Compute nodes for PCI passthrough that have the required PCI devices.
3. Deploy the overcloud.
4. Create a flavor for launching instances with PCI devices attached.

### Prerequisites

- The Compute nodes have the required PCI devices.

## 6.1. DESIGNATING COMPUTE NODES FOR PCI PASSTHROUGH

To designate Compute nodes for instances with physical PCI devices attached, you must:

- create a new role file to configure the PCI passthrough role
- configure a new overcloud flavor for PCI passthrough to use to tag the Compute nodes for PCI passthrough

### Procedure

1. Generate a new roles data file named **roles\_data\_pci\_passthrough.yaml** that includes the **Controller**, **Compute**, and **ComputePCI** roles:

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

2. Open **roles\_data\_pci\_passthrough.yaml** and edit or add the following parameters and sections:

Section/Parameter	Current value	New value
Role comment	<b>Role: Compute</b>	<b>Role: ComputePCI</b>
Role name	<b>name: Compute</b>	<b>name: ComputePCI</b>
<b>description</b>	<b>Basic Compute Node role</b>	<b>PCI Passthrough Compute Node role</b>
<b>HostnameFormatDefault</b>	<b>%stackname%-novacompute-%index%</b>	<b>%stackname%-novacomputepci-%index%</b>
<b>deprecated_nic_config_name</b>	<b>compute.yaml</b>	<b>compute-pci-passthrough.yaml</b>

- Register the PCI passthrough Compute nodes for the overcloud by adding them to your node definition template, **node.json** or **node.yaml**. For more information, see [Registering nodes for the overcloud](#) in the *Director Installation and Usage* guide.
- Inspect the node hardware:

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

For more information, see [Inspecting the hardware of nodes](#) in the *Director Installation and Usage* guide.

- Create the **compute-pci-passthrough** bare metal flavor to use to tag nodes that you want to designate for PCI passthrough:

```
(undercloud)$ openstack flavor create --id auto \
--ram <ram_size_mb> --disk <disk_size_gb> \
--vcpus <no_vcpus> compute-pci-passthrough
```

- Replace **<ram\_size\_mb>** with the RAM of the bare metal node, in MB.
- Replace **<disk\_size\_gb>** with the size of the disk on the bare metal node, in GB.
- Replace **<no\_vcpus>** with the number of CPUs on the bare metal node.



#### NOTE

These properties are not used for scheduling instances. However, the Compute scheduler does use the disk size to determine the root partition size.

- Tag each bare metal node that you want to designate for PCI passthrough with a custom PCI passthrough resource class:

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

Replace **<node>** with the ID of the bare metal node.

- Associate the **compute-pci-passthrough** flavor with the custom PCI passthrough resource class:

```
(undercloud)$ openstack flavor set \
  --property resources:CUSTOM_BAREMETAL_PCI_PASSTHROUGH=1 \
  compute-pci-passthrough
```

To determine the name of a custom resource class that corresponds to a resource class of a Bare Metal service node, convert the resource class to uppercase, replace all punctuation with an underscore, and prefix with **CUSTOM\_**.



#### NOTE

A flavor can request only one instance of a bare metal resource class.

- Set the following flavor properties to prevent the Compute scheduler from using the bare metal flavor properties to schedule instances:

```
(undercloud)$ openstack flavor set \
  --property resources:VCPU=0 --property resources:MEMORY_MB=0 \
  --property resources:DISK_GB=0 compute-pci-passthrough
```

- Add the following parameters to the **node-info.yaml** file to specify the number of PCI passthrough Compute nodes, and the flavor to use for the PCI passthrough designated Compute nodes:

```
parameter_defaults:
  OvercloudComputePCIFlavor: compute-pci-passthrough
  ComputePCICount: 3
```

- To verify that the role was created, enter the following command:

```
(undercloud)$ openstack overcloud profiles list
```

## 6.2. CONFIGURING A PCI PASSTHROUGH COMPUTE NODE

To enable your cloud users to create instances with PCI devices attached, you must configure both the Compute nodes that have the PCI devices and the Controller nodes.

### Procedure

- Create an environment file to configure the Controller node on the overcloud for PCI passthrough, for example, **pci\_passthrough\_controller.yaml**.
- Add **PciPassthroughFilter** to the **NovaSchedulerDefaultFilters** parameter in **pci\_passthrough\_controller.yaml**:

```
parameter_defaults:
  NovaSchedulerDefaultFilters:
  ['AvailabilityZoneFilter','ComputeFilter','ComputeCapabilitiesFilter','ImagePropertiesFilter','ServerGroupAntiAffinityFilter','ServerGroupAffinityFilter','PciPassthroughFilter','NUMATopologyFilter']
```



- To specify the PCI alias for the devices on the Controller node, add the following configuration to **pci\_passthrough\_controller.yaml**:

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

For more information about configuring the **device\_type** field, see [PCI passthrough device type field](#).



#### NOTE

If the **nova-api** service is running in a role different from the **Controller** role, replace **ControllerExtraConfig** with the user role in the format **<Role>ExtraConfig**.

- Optional: To set a default NUMA affinity policy for PCI passthrough devices, add **numa\_policy** to the **nova::pci::aliases** configuration from step 3:

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

- To configure the Compute node on the overcloud for PCI passthrough, create an environment file, for example, **pci\_passthrough\_compute.yaml**.
- To specify the available PCIs for the devices on the Compute node, use the **vendor\_id** and **product\_id** options to add all matching PCI devices to the pool of PCI devices available for passthrough to instances. For example, to add all Intel® Ethernet Controller X710 devices to the pool of PCI devices available for passthrough to instances, add the following configuration to **pci\_passthrough\_compute.yaml**:

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

For more information about how to configure **NovaPCIPassthrough**, see [Guidelines for configuring NovaPCIPassthrough](#).

- You must create a copy of the PCI alias on the Compute node for instance migration and resize operations. To specify the PCI alias for the devices on the PCI passthrough Compute node, add the following to **pci\_passthrough\_compute.yaml**:

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



#### NOTE

The Compute node aliases must be identical to the aliases on the Controller node. Therefore, if you added **numa\_affinity** to **nova::pci::aliases** in **pci\_passthrough\_controller.yaml**, then you must also add it to **nova::pci::aliases** in **pci\_passthrough\_compute.yaml**.

- To enable IOMMU in the server BIOS of the Compute nodes to support PCI passthrough, add the **KernelArgs** parameter to **pci\_passthrough\_compute.yaml**. For example, use the following **KernelArgs** settings to enable an Intel IOMMU:

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

To enable an AMD IOMMU, set **KernelArgs** to **"amd\_iommu=on iommu=pt"**.

- Add your custom environment files to the stack with your other environment files and deploy the overcloud:

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

- Create and configure the flavors that your cloud users can use to request the PCI devices. The following example requests two devices, each with a vendor ID of **8086** and a product ID of **1572**, using the alias defined in step 7:

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

## Verification

- Create an instance with a PCI passthrough device:

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

2. Log in to the instance as a cloud user. For more information, see [Log in to an Instance](#) .
3. To verify that the PCI device is accessible from the instance, enter the following command from the instance:

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

### 6.3. PCI PASSTHROUGH DEVICE TYPE FIELD

The Compute service categorizes PCI devices into one of three types, depending on the capabilities the devices report. The following lists the valid values that you can set the **device\_type** field to:

#### type-PF

The device supports SR-IOV and is the parent or root device. Specify this device type to passthrough a device that supports SR-IOV in its entirety.

#### type-VF

The device is a child device of a device that supports SR-IOV.

#### type-PCI

The device does not support SR-IOV. This is the default device type if the **device\_type** field is not set.



#### NOTE

You must configure the Compute and Controller nodes with the same **device\_type**.

### 6.4. GUIDELINES FOR CONFIGURING NOVAPCIPASSTHROUGH

- Do not use the **devname** parameter when configuring PCI passthrough, as the device name of a NIC can change. Instead, use **vendor\_id** and **product\_id** because they are more stable, or use the **address** of the NIC.
- To use the **product\_id** parameter to pass through a Physical Function (PF), you must also specify the **address** of the PF. However, you can use just the **address** parameter to specify PFs, because the address is unique on each host.
- To pass through all the Virtual Functions (VFs) you must specify only the **product\_id** and **vendor\_id**. You must also specify the **address** if you are using SRIOV for NIC partitioning and you are running OVS on a VF.
- To pass through only the VFs for a PF but not the PF itself, you can use the **address** parameter to specify the PCI address of the PF and **product\_id** to specify the product ID of the VF.

## CHAPTER 7. DATABASE CLEANING

The Compute service includes an administrative tool, **nova-manage**, that you can use to perform deployment, upgrade, clean-up, and maintenance-related tasks, such as applying database schemas, performing online data migrations during an upgrade, and managing and cleaning up the database.

Director automates the following database management tasks on the overcloud by using cron:

- Archives deleted instance records by moving the deleted rows from the production tables to shadow tables.
- Purges deleted rows from the shadow tables after archiving is complete.

### 7.1. CONFIGURING DATABASE MANAGEMENT

The cron jobs use default settings to perform database management tasks. By default, the database archiving cron jobs run daily at 00:01, and the database purging cron jobs run daily at 05:00, both with a jitter between 0 and 3600 seconds. You can modify these settings as required by using heat parameters.

#### Procedure

1. Open your Compute environment file.
2. Add the heat parameter that controls the cron job that you want to add or modify. For example, to purge the shadow tables immediately after they are archived, set the following parameter to "True":

```
parameter_defaults:
...
NovaCronArchiveDeleteRowsPurge: True
```

For a complete list of the heat parameters to manage database cron jobs, see [Configuration options for Openstack Compute \(nova\) automated database management](#).

3. Save the updates to your Compute environment file.
4. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

### 7.2. CONFIGURATION OPTIONS FOR OPENSTACK COMPUTE (NOVA) AUTOMATED DATABASE MANAGEMENT

Use the following heat parameters to enable and modify the automated cron jobs that manage the database.

**Table 7.1. OpenStack Compute (nova) cron parameters**

Parameter	Description
<b>NovaCronArchiveDeleteAllCells</b>	<p>Set this parameter to "True" to archive deleted instance records from all cells.</p> <p>Default value: <b>True</b></p>
<b>NovaCronArchiveDeleteRowsAge</b>	<p>Use this parameter to archive deleted instance records based on their age in days.</p> <p>Set to <b>0</b> to archive data older than today in shadow tables.</p> <p>Default value: <b>90</b></p>
<b>NovaCronArchiveDeleteRowsDestination</b>	<p>Use this parameter to configure the file for logging deleted instance records.</p> <p>Default value: <b>/var/log/nova/nova-rowsflush.log</b></p>
<b>NovaCronArchiveDeleteRowsHour</b>	<p>Use this parameter to configure the hour at which to run the cron command to move deleted instance records to another table.</p> <p>Default value: <b>0</b></p>
<b>NovaCronArchiveDeleteRowsMaxDelay</b>	<p>Use this parameter to configure the maximum delay, in seconds, before moving deleted instance records to another table.</p> <p>Default value: <b>3600</b></p>
<b>NovaCronArchiveDeleteRowsMaxRows</b>	<p>Use this parameter to configure the maximum number of deleted instance records that can be moved to another table.</p> <p>Default value: <b>1000</b></p>
<b>NovaCronArchiveDeleteRowsMinute</b>	<p>Use this parameter to configure the minute past the hour at which to run the cron command to move deleted instance records to another table.</p> <p>Default value: <b>1</b></p>
<b>NovaCronArchiveDeleteRowsMonthday</b>	<p>Use this parameter to configure on which day of the month to run the cron command to move deleted instance records to another table.</p> <p>Default value: <b>*</b> (every day)</p>

Parameter	Description
<b>NovaCronArchiveDeleteRowsMonth</b>	<p>Use this parameter to configure in which month to run the cron command to move deleted instance records to another table.</p> <p>Default value: * (every month)</p>
<b>NovaCronArchiveDeleteRowsPurge</b>	<p>Set this parameter to "True" to purge shadow tables immediately after scheduled archiving.</p> <p>Default value: <b>False</b></p>
<b>NovaCronArchiveDeleteRowsUntilComplete</b>	<p>Set this parameter to "True" to continue to move deleted instance records to another table until all records are moved.</p> <p>Default value: <b>True</b></p>
<b>NovaCronArchiveDeleteRowsUser</b>	<p>Use this parameter to configure the user that owns the crontab that archives deleted instance records and that has access to the log file the crontab uses.</p> <p>Default value: <b>nova</b></p>
<b>NovaCronArchiveDeleteRowsWeekday</b>	<p>Use this parameter to configure on which day of the week to run the cron command to move deleted instance records to another table.</p> <p>Default value: * (every day)</p>
<b>NovaCronPurgeShadowTablesAge</b>	<p>Use this parameter to purge shadow tables based on their age in days.</p> <p>Set to <b>0</b> to purge shadow tables older than today.</p> <p>Default value: <b>14</b></p>
<b>NovaCronPurgeShadowTablesAllCells</b>	<p>Set this parameter to "True" to purge shadow tables from all cells.</p> <p>Default value: <b>True</b></p>
<b>NovaCronPurgeShadowTablesDestination</b>	<p>Use this parameter to configure the file for logging purged shadow tables.</p> <p>Default value: <b>/var/log/nova/nova-rowspurge.log</b></p>

Parameter	Description
<b>NovaCronPurgeShadowTablesHour</b>	<p>Use this parameter to configure the hour at which to run the cron command to purge shadow tables.</p> <p>Default value: <b>5</b></p>
<b>NovaCronPurgeShadowTablesMaxDelay</b>	<p>Use this parameter to configure the maximum delay, in seconds, before purging shadow tables.</p> <p>Default value: <b>3600</b></p>
<b>NovaCronPurgeShadowTablesMinute</b>	<p>Use this parameter to configure the minute past the hour at which to run the cron command to purge shadow tables.</p> <p>Default value: <b>0</b></p>
<b>NovaCronPurgeShadowTablesMonth</b>	<p>Use this parameter to configure in which month to run the cron command to purge the shadow tables.</p> <p>Default value: <b>*</b> (every month)</p>
<b>NovaCronPurgeShadowTablesMonthday</b>	<p>Use this parameter to configure on which day of the month to run the cron command to purge the shadow tables.</p> <p>Default value: <b>*</b> (every day)</p>
<b>NovaCronPurgeShadowTablesUser</b>	<p>Use this parameter to configure the user that owns the crontab that purges the shadow tables and that has access to the log file the crontab uses.</p> <p>Default value: <b>nova</b></p>
<b>NovaCronPurgeShadowTablesVerbose</b>	<p>Use this parameter to enable verbose logging in the log file for purged shadow tables.</p> <p>Default value: <b>False</b></p>
<b>NovaCronPurgeShadowTablesWeekday</b>	<p>Use this parameter to configure on which day of the week to run the cron command to purge the shadow tables.</p> <p>Default value: <b>*</b> (every day)</p>

## CHAPTER 8. CONFIGURING COMPUTE NODES FOR PERFORMANCE

You can configure the scheduling and placement of instances for optimal performance by creating customized flavors to target specialized workloads, including Network Functions Virtualization (NFV), and High Performance Computing (HPC).

Use the following features to tune your instances for optimal performance:

- CPU pinning: Pin virtual CPUs to physical CPUs.
- Emulator threads: Pin emulator threads associated with the instance to physical CPUs.
- Huge pages: Tune instance memory allocation policies both for normal memory (4k pages) and huge pages (2 MB or 1 GB pages).



### NOTE

Configuring any of these features creates an implicit NUMA topology on the instance if there is no NUMA topology already present.

For more information about NFV and hyper-converged infrastructure (HCI) deployments, see [Deploying an overcloud with HCI and DPDK](#) in the *Network Functions Virtualization Planning and Configuration Guide*.

### 8.1. CONFIGURING CPU PINNING WITH NUMA

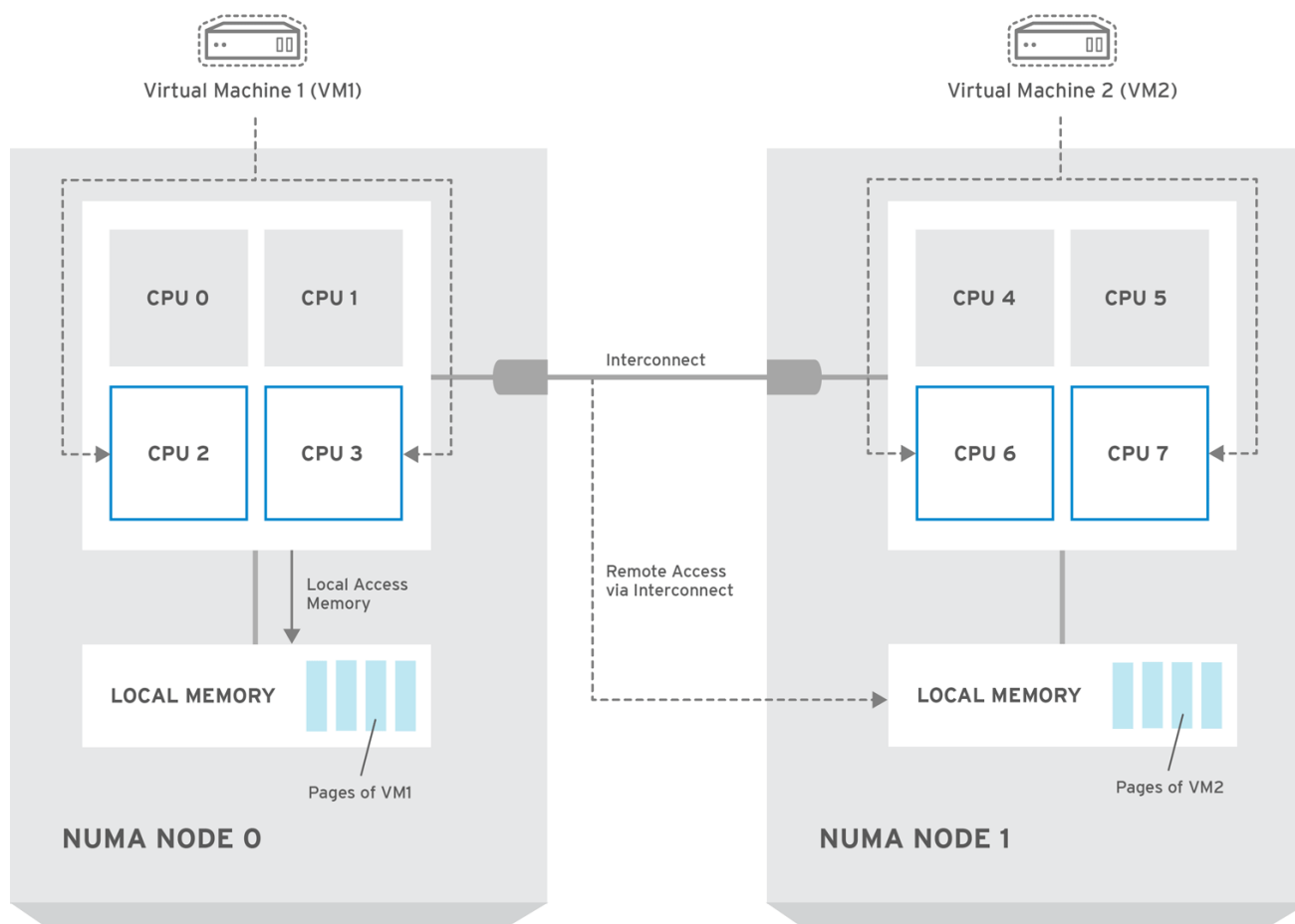
This chapter describes how to use NUMA topology awareness to configure an OpenStack environment on systems with a NUMA architecture. The procedures detailed in this chapter show you how to pin virtual machines (VMs) to dedicated CPU cores, which improves scheduling and VM performance.

#### TIP

Background information about NUMA is available in the following article: [What is NUMA and how does it work on Linux?](#)

The following diagram provides an example of a two-node NUMA system and the way the CPU cores and memory pages are made available:





OPENSTACK\_39825\_0516

**NOTE**

Remote memory available via Interconnect is accessed **only** if VM1 from NUMA node 0 has a CPU core in NUMA node 1. In this case, the memory of NUMA node 1 will act as local for the third CPU core of VM1 (for example, if VM1 is allocated with CPU 4 in the diagram above), but at the same time, it will act as remote memory for the other CPU cores of the same VM.

For more details on NUMA tuning with libvirt, see the [Virtualization Tuning and Optimization Guide](#).

### 8.1.1. Compute node configuration

The exact configuration depends on the NUMA topology of your host system. However, you must reserve some CPU cores across all the NUMA nodes for host processes and let the rest of the CPU cores handle your virtual machines (VMs). The following example illustrates the layout of eight CPU cores evenly spread across two NUMA nodes.

**Table 8.1. Example of NUMA Topology**

	Node 0		Node 1	
Host processes	Core 0	Core 1	Core 4	Core 5
Instances	Core 2	Core 3	Core 6	Core 7

**NOTE**

Determine the number of cores to reserve for host processes by observing the performance of the host under typical workloads.

**Procedure**

1. Reserve CPU cores for the VMs by setting the **NovaVcpuPinSet** configuration in the Compute environment file:

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

2. Set the **NovaReservedHostMemory** option in the same file to the amount of RAM to reserve for host processes. For example, if you want to reserve 512 MB, use:

```
NovaReservedHostMemory: 512
```

3. To ensure that host processes do not run on the CPU cores reserved for VMs, set the parameter **IsolCpusList** in the Compute environment file to the CPU cores you have reserved for VMs. Specify the value of the **IsolCpusList** parameter using a list of CPU indices, or ranges separated by a whitespace. For example:

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

**NOTE**

The **IsolCpusList** parameter and **isolcpus** parameter are different parameters for separate purposes:

- **IsolCpusList**: Use this heat parameter to set **isolated\_cores** in **tuned.conf** using the **cpu-partitioning** profile.
- **isolcpus**: This is a Kernel boot parameter that you set with the **KernelArgs** heat parameter.

Do not use the **IsolCpusList** parameter and the **isolcpus** parameter interchangeably.

**TIP**

To set **IsolCpusList** in non-NFV roles, you must configure **KernelArgs** and **IsolCpusList**, and include the **/usr/share/openstack-tripleo-heat-templates/environments/host-config-and-reboot.yaml** environment file in the overcloud deployment. Contact Red Hat Support if you plan to deploy with **config-download**, and configure **IsolCpusList** for non-NFV roles.

4. To apply this configuration, deploy the overcloud:

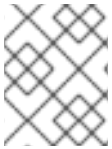
```
(undercloud) $ openstack overcloud deploy --templates \  
-e /home/stack/templates/<compute_environment_file>.yaml
```

## 8.1.2. Configuring emulator threads to run on dedicated physical CPU

The Compute scheduler determines the CPU resource utilization and places instances based on the

number of virtual CPUs (vCPUs) in the flavor. There are a number of hypervisor operations that are performed on the host, on behalf of the guest instance, for example, with QEMU, there are threads used for the QEMU main event loop, asynchronous I/O operations and so on and these operations need to be accounted and scheduled separately.

The **libvirt** driver implements a generic placement policy for KVM which allows QEMU emulator threads to float across the same physical CPUs (pCPUs) that the vCPUs are running on. This leads to the emulator threads using time borrowed from the vCPUs operations. When you need a guest to have dedicated vCPU allocation, it is necessary to allocate one or more pCPUs for emulator threads. It is therefore necessary to describe to the scheduler any other CPU usage that might be associated with a guest and account for that during placement.

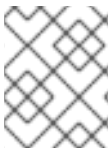


#### NOTE

In an NFV deployment, to avoid packet loss, you have to make sure that the vCPUs are never preempted.

Before you enable the emulator threads placement policy on a flavor, check that the following heat parameters are defined as follows:

- **NovaComputeCpuSharedSet:** Set this parameter to a list of CPUs defined to run emulator threads.
- **NovaSchedulerDefaultFilters:** Include **NUMATopologyFilter** in the list of defined filters.



#### NOTE

You can define or change heat parameter values on an active cluster, and then redeploy for those changes to take effect.

To isolate emulator threads, you must use a flavor configured as follows:

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

### 8.1.3. Scheduler configuration

#### Procedure

1. Open your Compute environment file.
2. Add the following values to the **NovaSchedulerDefaultFilters** parameter, if they are not already present:
  - **NUMATopologyFilter**
  - **AggregateInstanceExtraSpecsFilter**
3. Save the configuration file.
4. Deploy the overcloud.

## 8.1.4. Aggregate and flavor configuration

Configure host aggregates to deploy instances that use CPU pinning on different hosts from instances that do not, to avoid unpinned instances using the resourcing requirements of pinned instances.

### CAUTION

Do not deploy instances with NUMA topology on the same hosts as instances that do not have NUMA topology.

Prepare your OpenStack environment for running virtual machine instances pinned to specific resources by completing the following steps on a system with the Compute CLI.

### Procedure

1. Load the **admin** credentials:

```
source ~/keystonerc_admin
```

2. Create an aggregate for the hosts that will receive pinning requests:

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

3. Enable the pinning by editing the metadata for the aggregate:

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

4. Create an aggregate for other hosts:

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

5. Edit the metadata for this aggregate accordingly:

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

6. Change your existing flavors' specifications to this one:

```
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. Create a flavor for the hosts that will receive pinning requests:

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

Where:

- **<flavor-ID>** - Set to **auto** if you want **nova** to generate a UUID.
- **<RAM>** - Specify the required RAM in MB.
- **<disk-size>** - Specify the required disk size in GB.
- **<vCPUs>** - The number of virtual CPUs that you want to reserve.

- Set the **hw:cpu\_policy** specification of this flavor to **dedicated** so as to require dedicated resources, which enables CPU pinning, and also the **hw:cpu\_thread\_policy** specification to **require**, which places each vCPU on thread siblings:

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



#### NOTE

If the host does not have an SMT architecture or enough CPU cores with free thread siblings, scheduling will fail. If such behavior is undesired, or if your hosts simply do not have an SMT architecture, do not use the **hw:cpu\_thread\_policy** specification, or set it to **prefer** instead of **require**. The (default) **prefer** policy ensures that thread siblings are used when available.

- Set the **aggregate\_instance\_extra\_specs:pinned** specification to "true" to ensure that instances based on this flavor have this specification in their aggregate metadata:

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

- Add some hosts to the new aggregates:

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

- Boot an instance using the new flavor:

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

- To verify that the new server has been placed correctly, run the following command and check for **OS-EXT-SRV-ATTR:hypervisor\_hostname** in the output:

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

## 8.2. CONFIGURING HUGE PAGES ON THE COMPUTE NODE

Configure the Compute node to enable instances to request huge pages.

### Procedure

- Configure the amount of huge page memory to reserve on each NUMA node for processes that are not instances:

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

Where:

Attribute	Description
-----------	-------------

size	The size of the allocated huge page. Valid values: * 2048 (for 2MB) * 1GB
count	The number of huge pages used by OVS per NUMA node. For example, for 4096 of socket memory used by Open vSwitch, set this to 2.

- (Optional) To allow instances to allocate 1GB huge pages, configure the CPU feature flags, **cpu\_model\_extra\_flags**, to include "pdpe1gb":

```
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'
```



## NOTE

- CPU feature flags do not need to be configured to allow instances to only request 2 MB huge pages.
- You can only allocate 1G huge pages to an instance if the host supports 1G huge page allocation.
- You only need to set **cpu\_model\_extra\_flags** to **pdpe1gb** when **cpu\_mode** is set to **host-model** or **custom**.
- If the host supports **pdpe1gb**, and **host-passthrough** is used as the **cpu\_mode**, then you do not need to set **pdpe1gb** as a **cpu\_model\_extra\_flags**. The **pdpe1gb** flag is only included in Opteron\_G4 and Opteron\_G5 CPU models, it is not included in any of the Intel CPU models supported by QEMU.
- To mitigate for CPU hardware issues, such as Microarchitectural Data Sampling (MDS), you might need to configure other CPU flags. For more information, see [RHOS Mitigation for MDS \("Microarchitectural Data Sampling"\) Security Flaws](#).

- To avoid loss of performance after applying Meltdown protection, configure the CPU feature flags, **cpu\_model\_extra\_flags**, to include "+pcid":

```
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'
```

## TIP

For more information, see [Reducing the performance impact of Meltdown CVE fixes for OpenStack guests with "PCID" CPU feature flag](#).

4. Add **NUMATopologyFilter** to the **NovaSchedulerDefaultFilters** parameter in each Compute environment file, if not already present.
5. Apply this huge page configuration by adding the environment file(s) to your deployment command and deploying the overcloud:

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

### 8.2.1. Allocating huge pages to instances

Create a flavor with the **hw:mem\_page\_size** extra specification key to specify that the instance should use huge pages.

#### Prerequisites

- The Compute node is configured for huge pages. For more information, see [Configuring huge pages on the Compute node](#).

#### Procedure

1. Create a flavor for instances that require huge pages:

```
$ openstack flavor create --ram <size-mb> --disk <size-gb> --vcpus <no_reserved_vcpus>
huge_pages
```

2. Set the flavor for huge pages:

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

Valid values for **hw:mem\_page\_size**:

- **large** - Selects the largest page size supported on the host, which may be 2 MB or 1 GB on x86\_64 systems.
  - **small** - (Default) Selects the smallest page size supported on the host. On x86\_64 systems this is 4 kB (normal pages).
  - **any** - Selects the largest available huge page size, as determined by the libvirt driver.
  - **<pagesize>**: (string) Set an explicit page size if the workload has specific requirements. Use an integer value for the page size in KB, or any standard suffix. For example: 4KB, 2MB, 2048, 1GB.
3. Create an instance using the new flavor:

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

#### Validation

The scheduler identifies a host with enough free huge pages of the required size to back the memory of the instance. If the scheduler is unable to find a host and NUMA node with enough pages, then the request will fail with a NoValidHost error.

## CHAPTER 9. ADDING METADATA TO INSTANCES

The Compute (nova) service uses metadata to pass configuration information to instances on launch. The instance can access the metadata by using a config drive or the metadata service.

### Config drive

Config drives are special drives that you can attach to an instance when it boots. The config drive is presented to the instance as a read-only drive. The instance can mount this drive and read files from it to get information that is normally available through the metadata service.

### Metadata service

The Compute service provides the metadata service as a REST API, which can be used to retrieve data specific to an instance. Instances access this service at **169.254.169.254** or at **fe80::a9fe:a9fe**.

## 9.1. TYPES OF INSTANCE METADATA

Cloud users, cloud administrators, and the Compute service can pass metadata to instances:

### Cloud user provided data

Cloud users can specify additional data to use when they launch an instance, such as a shell script that the instance runs on boot. The cloud user can pass data to instances by using the user data feature, and by passing key-value pairs as required properties when creating or updating an instance.

### Cloud administrator provided data

The RHOSP administrator uses the `vendordata` feature to pass data to instances. The Compute service provides the `vendordata` modules **StaticJSON** and **DynamicJSON** to allow administrators to pass metadata to instances:

- **StaticJSON**: (Default) Use for metadata that is the same for all instances.
- **DynamicJSON**: Use for metadata that is different for each instance. This module makes a request to an external REST service to determine what metadata to add to an instance.

Vendordata configuration is located in one of the following read-only files on the instance:

- `/openstack/{version}/vendor_data.json`
- `/openstack/{version}/vendor_data2.json`

### Compute service provided data

The Compute service uses its internal implementation of the metadata service to pass information to the instance, such as the requested hostname for the instance, and the availability zone the instance is in. This happens by default and requires no configuration by the cloud user or administrator.

## 9.2. ADDING A CONFIG DRIVE TO ALL INSTANCES

As an administrator, you can configure the Compute service to always create a config drive for instances, and populate the config drive with metadata that is specific to your deployment. For example, you might use a config drive for the following reasons:

- To pass a networking configuration when your deployment does not use DHCP to assign IP addresses to instances. You can pass the IP address configuration for the instance through the config drive, which the instance can mount and access before you configure the network settings for the instance.



- To pass data to an instance that is not known to the user starting the instance, for example, a cryptographic token to be used to register the instance with Active Directory post boot.
- To create a local cached disk read to manage the load of instance requests, which reduces the impact of instances accessing the metadata servers regularly to check in and build facts.

Any instance operating system that is capable of mounting an ISO 9660 or VFAT file system can use the config drive.

## Procedure

1. Open your Compute environment file.
2. To always attach a config drive when launching an instance, set the following parameter to **True**:

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::force_config_drive: 'true'
```

3. Optional: To change the format of the config drive from the default value of **iso9660** to **vfat**, add the **config\_drive\_format** parameter to your configuration:

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::force_config_drive: 'true'
    nova::compute::config_drive_format: vfat
```

4. Save the updates to your Compute environment file.
5. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

## Verification

1. Create an instance:

```
(overcloud)$ openstack server create --flavor m1.tiny \
--image cirros test-config-drive-instance
```

2. Log in to the instance.
3. Mount the config drive:

- If the instance OS uses **udev**:

```
# mkdir -p /mnt/config
# mount /dev/disk/by-label/config-2 /mnt/config
```

- If the instance OS does not use **udev**, you need to first identify the block device that corresponds to the config drive:

```
# blkid -t LABEL="config-2" -o device
/dev/vdb
# mkdir -p /mnt/config
# mount /dev/vdb /mnt/config
```

4. Inspect the files in the mounted config drive directory, **mnt/config/openstack/{version}/**, for your metadata.

## 9.3. ADDING STATIC METADATA TO INSTANCES

You can make static metadata available to all instances in your deployment.

### Procedure

1. Create the JSON file for the metadata.
2. Open your Compute environment file.
3. Add the path to the JSON file to your environment file:

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      ...
    api/vendordata_jsonfile_path:
      value: <path_to_the_JSON_file>
```

4. Save the updates to your Compute environment file.
5. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

## 9.4. ADDING DYNAMIC METADATA TO INSTANCES

You can configure your deployment to create instance-specific metadata, and make the metadata available to that instance through a JSON file.

### TIP

You can use dynamic metadata on the undercloud to integrate director with a Red Hat Identity Management (IdM) server. An IdM server can be used as a certificate authority and manage the overcloud certificates when SSL/TLS is enabled on the overcloud. For more information, see [Add the undercloud to IdM](#).

### Procedure

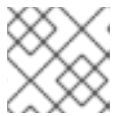
1. Open your Compute environment file.
2. Add **DynamicJSON** to the vendordata provider module:

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
    ...
  api/vendordata_providers:
    value: StaticJSON,DynamicJSON
```

- Specify the REST services to contact to generate the metadata. You can specify as many target REST services as required, for example:

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
    ...
  api/vendordata_providers:
    value: StaticJSON,DynamicJSON
  api/vendordata_dynamic_targets:
    value: target1@http://127.0.0.1:125
  api/vendordata_dynamic_targets:
    value: target2@http://127.0.0.1:126
```

The Compute service generates the JSON file, **vendordata2.json**, to contain the metadata retrieved from the configured target services, and stores it in the config drive directory.



#### NOTE

Do not use the same name for a target service more than once.

- Save the updates to your Compute environment file.
- Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

## CHAPTER 10. CONFIGURING REAL-TIME COMPUTE

In some use-cases, you might need instances on your Compute nodes to adhere to low-latency policies and perform real-time processing. Real-time Compute nodes include a real-time capable kernel, specific virtualization modules, and optimized deployment parameters, to facilitate real-time processing requirements and minimize latency.

The process to enable Real-time Compute includes:

- configuring the BIOS settings of the Compute nodes
- building a real-time image with real-time kernel and Real-Time KVM (RT-KVM) kernel module
- assigning the **ComputeRealTime** role to the Compute nodes

For a use-case example of Real-time Compute deployment for NFV workloads, see the [Example: Configuring OVS-DPDK with ODL and VXLAN tunnelling](#) section in the *Network Functions Virtualization Planning and Configuration Guide*.

### 10.1. PREPARING YOUR COMPUTE NODES FOR REAL-TIME



#### NOTE

Real-time Compute nodes are supported only with Red Hat Enterprise Linux version 7.5 or later.

Before you can deploy Real-time Compute in your overcloud, you must enable Red Hat Enterprise Linux Real-Time KVM (RT-KVM), configure your BIOS to support real-time, and build the real-time image.

#### Prerequisites

- You must use Red Hat certified servers for your RT-KVM Compute nodes. See [Red Hat Enterprise Linux for Real Time 7 certified servers](#) for details.
- You must enable the **rhel-7-server-nfv-rpms** repository for RT-KVM to build the real-time image.



#### NOTE

You need a separate subscription to *Red Hat OpenStack Platform for Real Time* before you can access this repository. For details on managing repositories and subscriptions for your undercloud, see the [Registering and updating your undercloud](#) section in the *Director Installation and Usage* guide.

To check which packages will be installed from the repository, run the following command:

```
$ yum repo-pkgs rhel-7-server-nfv-rpms list
Loaded plugins: product-id, search-disabled-repos, subscription-manager
Available Packages
kernel-rt.x86_64                               3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
kernel-rt-debug.x86_64                         3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
```

```

kernel-rt-debug-devel.x86_64                3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
kernel-rt-debug-kvm.x86_64                  3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
kernel-rt-devel.x86_64                      3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
kernel-rt-doc.noarch                        3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
kernel-rt-kvm.x86_64                        3.10.0-693.21.1.rt56.639.el7
rhel-7-server-nfv-rpms
[ output omitted... ]

```

## Building the real-time image

To build the overcloud image for Real-time Compute nodes:

1. Install the **libguestfs-tools** package on the undercloud to get the **virt-customize** tool:

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



### IMPORTANT

If you install the **libguestfs-tools** package on the undercloud, disable **iscsid.socket** to avoid port conflicts with the **tripleo\_iscsid** service on the undercloud:

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

2. Extract the images:

```
(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. Copy the default image:

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

4. Register the image and configure the required subscriptions:

```
(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
```

Replace the **username** and **password** values with your Red Hat customer account details. For general information about building a Real-time overcloud image, see the [Modifying the Red Hat](#)

[Enterprise Linux OpenStack Platform Overcloud Image with virt-customize](#) knowledgebase article.

- Find the SKU of the *Red Hat OpenStack Platform for Real Time* subscription. The SKU might be located on a system that is already registered to the Red Hat Subscription Manager with the same account and credentials. For example:

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

- Attach the *Red Hat OpenStack Platform for Real Time* subscription to the image:

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

- Create a script to configure **rt** on the image:

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

set -eux

subscription-manager repos --enable=rhel-7-server-rpms --enable=rhel-7-server-openstack-13-rpms --enable=rhel-7-server-nfv-rpms
yum -v -y --setopt=protected_packages= erase kernel.$(uname -m)
yum -v -y install kernel-rt kernel-rt-kvm tuned-profiles-nfv-host

# END OF SCRIPT
```

- Run the script to configure the real-time image:

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

- Re-label SELinux:

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

- Extract **vmlinuz** and **initrd**:

```
(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-3.10.0-862.rt56.804.el7.x86_64 ./overcloud-realtime-compute.vmlinuz
(undercloud) [stack@undercloud-0 ~]$ cp image/boot/initramfs-3.10.0-862.rt56.804.el7.x86_64.img ./overcloud-realtime-compute.initrd
(undercloud) [stack@undercloud-0 ~]$ guestunmount image
```



## NOTE

The software version in the **vmlinuz** and **initramfs** filenames vary with the kernel version.

11. Upload the image:

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

You now have a real-time image you can use with the **ComputeRealTime** composable role on select Compute nodes.

## Modifying BIOS settings on Real-time Compute nodes

To reduce latency on your Real-time Compute nodes, you must modify the BIOS settings in the Compute nodes. You should disable all options for the following components in your Compute node BIOS settings:

- Power Management
- Hyper-Threading
- CPU sleep states
- Logical processors

See [Setting BIOS parameters](#) for descriptions of these settings and the impact of disabling them. See your hardware manufacturer documentation for complete details on how to change BIOS settings.

## 10.2. DEPLOYING THE REAL-TIME COMPUTE ROLE

Red Hat OpenStack Platform Director provides the template for the **ComputeRealTime** role, which you can then use to deploy Real-time Compute nodes. However, you must perform additional steps to designate Compute nodes for real-time.

1. Based on the `/usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-example.yaml` file, create a `compute-real-time.yaml` environment file that sets the parameters for the **ComputeRealTime** role.

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

The file must include values for the following parameters:

- **IsolCpusList** and **NovaVcpuPinSet**. List of isolated CPU cores and virtual CPU pins to reserve for real-time workloads. This value depends on the CPU hardware of your Real-time Compute nodes.
  - **KernelArgs**. Arguments to pass to the kernel of the Real-time Compute nodes. For example, you can use `default_hugepagesz=1G hugepagesz=1G hugepages=<number_of_1G_pages_to_reserve> hugepagesz=2M hugepages=<number_of_2M_pages>` to define the memory requirements of guests that have huge pages with multiple sizes. In this example, the default size is 1GB but you can also reserve 2M huge pages.
2. Add the **ComputeRealTime** role to your roles data file and regenerate the file. For example:

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

This command generates a **ComputeRealTime** role with contents similar to the following example, and also sets the **ImageDefault** option to **overcloud-realtime-compute**.

```
#####
# Role: ComputeRealTime                                     #
#####

- 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
    - Tenant
    - Storage
  HostnameFormatDefault: '%stackname%-computerealtime-%index%'
  disable_upgrade_deployment: True
  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::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::Ipssec
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::LoginDefs
    - 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::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::Vpp
- OS::TripleO::Services::OVNController
- OS::TripleO::Services::OVNMetadataAgent
- OS::TripleO::Services::Ptp

```

For general information about custom roles and about the *roles-data.yaml*, see the [Roles](#) section.

3. Create the **compute-realtime** flavor to tag nodes that you want to designate for real-time workloads. For example:

```

$ 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. Tag each node that you want to designate for real-time workloads with the **compute-realtime** profile.

```

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

```

5. Map the **ComputeRealTime** role to the **compute-realtime** flavor by creating an environment file with the following content:

```

parameter_defaults:
  OvercloudComputeRealTimeFlavor: compute-realtime

```

6. Run the **openstack overcloud deploy** command with the **-e** option and specify all the environment files that you created, as well as the new roles file. For example:

```

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

```



#### NOTE

If you want to run additional real-time instances on the same Compute node, you can change the priority of the instances in the **realtime\_schedule\_priority** parameter in the *nova.conf* file.

## 10.3. SAMPLE DEPLOYMENT AND TESTING SCENARIO

The following example procedure uses a simple single-node deployment to test that the environment variables and other supporting configuration is set up correctly. Actual performance results might vary, depending on the number of nodes and guests that you deploy in your cloud.

1. Create the **compute-real-time.yaml** file with the following parameters:

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

2. Create a new **rt\_roles\_data.yaml** file with the **ComputeRealTime** role.

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

3. Deploy the overcloud, adding both your new real-time roles data file and your real-time environment file to the stack along with your other environment files:

```
(undercloud) $ openstack overcloud deploy --templates \
-r /home/stack/rt_roles_data.yaml
-e [your environment files]
-e /home/stack/templates/compute-real-time.yaml
```

This command deploys one Controller node and one Real-time Compute node.

4. Log into the Real-time Compute node and check the following parameters. Replace **<...>** with the values of the relevant parameters from the **compute-real-time.yaml**.

```
[root@overcloud-computerealttime-0 ~]# uname -a
Linux overcloud-computerealttime-0 3.10.0-693.11.1.rt56.632.el7.x86_64 #1 SMP PREEMPT
RT Wed Dec 13 13:37:53 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
[root@overcloud-computerealttime-0 ~]# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-3.10.0-693.11.1.rt56.632.el7.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-computerealttime-0 ~]# tuned-adm active
Current active profile: realtime-virtual-host
[root@overcloud-computerealttime-0 ~]# grep ^isolated_cores /etc/tuned/realtime-virtual-
variables.conf
isolated_cores=<IsolatedCpusList>
[root@overcloud-computerealttime-0 ~]# cat /usr/lib/tuned/realtime-virtual-
host/lapic_timer_adv_ns
X (X != 0)
[root@overcloud-computerealttime-0 ~]# cat
/sys/module/kvm/parameters/lapic_timer_advance_ns
X (X != 0)
[root@overcloud-computerealttime-0 ~]# cat
/sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
X (X != 0)
[root@overcloud-computerealttime-0 ~]# grep ^vcpu_pin_set /var/lib/config-data/puppet-
generated/nova_libvirt/etc/nova/nova.conf
vcpu_pin_set=<NovaVcpuPinSet>
```

## 10.4. LAUNCHING AND TUNING REAL-TIME INSTANCES

After you deploy and configure Real-time Compute nodes, you can launch real-time instances on those nodes. You can further configure these real-time instances with CPU pinning, NUMA topologies, and huge pages.

### Configuring a real-time policy for instances

A real-time policy prioritizes real-time instances and minimizes latency during peak workload times. To set this policy, add the following parameters to the **compute-realtime** flavor.

```
$ openstack flavor set compute-realtime \
  --property hw:cpu_realtime=yes
  --property hw:cpu_realtime_mask=^0
```

### Launching a real-time instance

1. Make sure that the **compute-realtime** flavor exists on the overcloud, as described in the *Deploying the Real-time Compute Role* section.
2. Launch the real-time instance.

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

3. If you have administrator access to the Compute host, you can optionally verify that the instance uses the assigned emulator threads.

```
# 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>
```

### Pinning CPUs and setting emulator thread policy

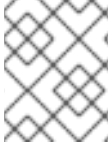
To ensure that there are enough CPUs on each Real-time Compute node for real-time workloads, you need to pin at least one virtual CPU (vCPU) for an instance to a physical CPU (pCPUs) on the host. The emulator threads for that vCPU then remain dedicated to that pCPU.

1. Set the **emulator\_thread\_policy** parameter to **isolate**. For example:

```
# openstack flavor set --property hw:emulator_threads_policy=isolate
```

1. Configure your flavor to use a dedicated CPU policy. To do so, set the **hw:cpu\_policy** parameter to **dedicated** on the flavor. For example:

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

**NOTE**

Make sure that your resources quota has enough pCPUs for the Real-time Compute nodes to consume.

For general information about CPU pinning, see the [CPU Pinning](#) chapter.

**Optimizing your network configuration**

Depending on the needs of your deployment, you might need to set parameters in the ***network-environment.yaml*** file to tune your network for certain real-time workloads.

To review an example configuration optimized for OVS-DPDK, see the [Configuring OVS-DPDK with RT-KVM](#) section of the *Network Functions Virtualization Planning and Configuration Guide* .

**Configuring huge pages**

It is recommended to set the default huge pages size to 1GB. Otherwise, TLB flushes might create jitter in the vCPU execution.

To set the huge pages size for the **compute-realtime** flavor, run the following command:

```
openstack flavor set compute-realtime --property hw:mem_page_size=large
```

For general information about using huge pages, see the [Running DPDK applications](#) web page.

## CHAPTER 11. CONFIGURING VIRTUAL GPUS FOR INSTANCES

To support GPU-based rendering on your instances, you can define and manage virtual GPU (vGPU) resources according to your available physical GPU devices and your hypervisor type. You can use this configuration to divide the rendering workloads between all your physical GPU devices more effectively, and to have more control over scheduling your vGPU-enabled instances.

To enable vGPU in OpenStack Compute, create flavors that your cloud users can use to create Red Hat Enterprise Linux (RHEL) instances with vGPU devices. Each instance can then support GPU workloads with virtual GPU devices that correspond to the physical GPU devices.

The OpenStack Compute service tracks the number of vGPU devices that are available for each GPU profile you define on each host. The Compute service schedules instances to these hosts based on the flavor, attaches the devices, and monitors usage on an ongoing basis. When an instance is deleted, the Compute service adds the vGPU devices back to the available pool.

### 11.1. SUPPORTED CONFIGURATIONS AND LIMITATIONS

#### Supported GPU cards

For a list of supported NVIDIA GPU cards, see [Virtual GPU Software Supported Products](#) on the NVIDIA website.

#### Limitations when using vGPU devices

- You can enable only one vGPU type on each Compute node.
- Each instance can use only one vGPU resource.
- Live migration of vGPU between hosts is not supported.
- Suspend operations on a vGPU-enabled instance is not supported due to a libvirt limitation. Instead, you can snapshot or shelve the instance.
- Resize and cold migration operations on an instance with a vGPU flavor does not automatically re-allocate the vGPU resources to the instance. After you resize or migrate the instance, you must rebuild it manually to re-allocate the vGPU resources.
- By default, vGPU types on Compute hosts are not exposed to API users. To grant access, add the hosts to a host aggregate. For more information, see [Section 4.4, “Managing host aggregates”](#).
- If you use NVIDIA accelerator hardware, you must comply with the NVIDIA licensing requirements. For example, NVIDIA vGPU GRID requires a licensing server. For more information about the NVIDIA licensing requirements, see [NVIDIA License Server Release Notes](#) on the NVIDIA website.

### 11.2. CONFIGURING VGPU ON THE COMPUTE NODES

To enable your cloud users to create instances that use a virtual GPU (vGPU), you must configure the Compute nodes that have the physical GPUs:

1. Build a custom GPU-enabled overcloud image.
2. Prepare the GPU role, profile, and flavor for designating Compute nodes for vGPU.

3. Configure the Compute node for vGPU.
4. Deploy the overcloud.

**NOTE**

To use an NVIDIA GRID vGPU, you must comply with the NVIDIA GRID licensing requirements and you must have the URL of your self-hosted license server. For more information, see the [NVIDIA License Server Release Notes](#) web page.

### 11.2.1. Building a custom GPU overcloud image

Perform the following steps on the director node to install the NVIDIA GRID host driver on an overcloud Compute image and upload the image to the OpenStack Image service (glance).

**Procedure**

1. Copy the overcloud image and add the **gpu** suffix to the copied image.

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

2. Install an ISO image generator tool from YUM.

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

3. Download the NVIDIA GRID host driver RPM package that corresponds to your GPU device from the NVIDIA website. To determine which driver you need, see the [NVIDIA Driver Downloads Portal](#).

**NOTE**

You must be a registered NVIDIA customer to download the drivers from the portal.

4. Create an ISO image from the driver RPM package and save the image in the **nvidia-host** directory.

```
$ genisoimage -o nvidia-host.iso -R -J -V NVIDIA nvidia-host/
l: -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. Create a driver installation script that also disables the nouveau driver and generates a new initramfs. The following example script, **install\_nvidia.sh**, disables the nouveau driver, generates a new initramfs, and installs the NVIDIA GRID host driver on the overcloud image:

```

#!/bin/bash

cat <<EOF >/etc/modprobe.d/disable-nouveau.conf
blacklist nouveau
options nouveau modeset=0
EOF
echo 'omit_drivers+=" nouveau "' > /etc/dracut.conf.d/disable-nouveau.conf
dracut -f

# NVIDIA GRID package
mkdir /tmp/mount
mount LABEL=NVIDIA /tmp/mount
rpm -ivh /tmp/mount/<host_driver>.rpm

```

- Replace **<host\_driver>** with the host driver downloaded in step 3.
6. Customize the overcloud image by attaching the ISO image that you generated in step 4, and running the driver installation script that you created in step 5:

```

$ 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. Relabel the customized image with SELinux:

```

$ 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. Prepare the custom image files for upload to the OpenStack Image Service:

```

$ 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. From the undercloud, upload the custom image to the OpenStack Image Service:

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

## 11.2.2. Designating Compute nodes for vGPU

To designate Compute nodes for vGPU workloads, you must create a new role file to configure the vGPU role, and configure a new flavor to use to tag the GPU-enabled Compute nodes.

### Procedure

1. To create the new **ComputeGpu** role file, copy the file `/usr/share/openstack-tripleo-heat-templates/roles/Compute.yaml` to `/usr/share/openstack-tripleo-heat-templates/roles/ComputeGpu.yaml` and edit or add the following file sections:

Table 11.1. ComputeGpu role file edits

Section/Parameter	Current value	New value
Role comment	<b>Role: Compute</b>	<b>Role: ComputeGpu</b>
Role name	<b>name: Compute</b>	<b>name: ComputeGpu</b>
<b>description</b>	<b>Basic Compute Node role</b>	<b>GPU Compute Node role</b>
<b>ImageDefault</b>	n/a	<b>overcloud-full-gpu</b>
<b>HostnameFormatDefault</b>	<b>-compute-</b>	<b>-computegpu-</b>
<b>deprecated_nic_config_name</b>	<b>compute.yaml</b>	<b>compute-gpu.yaml</b>

The following example shows the **ComputeGpu** role details:

```
#####
# Role: ComputeGpu                                     #
#####
- name: ComputeGpu
  description: |
    GPU Compute Node role
  CountDefault: 1
  ImageDefault: overcloud-full-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: 'NovalImage'
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::Ipssec
  - 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

```

2. Generate a new roles data file named **roles\_data\_gpu.yaml** that includes the **Controller**, **Compute**, and **ComputeGpu** roles:

```
(undercloud) [stack@director templates]$ openstack overcloud roles \
generate -o /home/stack/templates/roles_data_gpu.yaml \
ComputeGpu Compute Controller
```

3. Register the node for the overcloud. For more information, see [Registering nodes for the overcloud](#) in the *Director Installation and Usage* guide.
4. Inspect the node hardware. For more information, see [Inspecting the hardware of nodes](#) in the *Director Installation and Usage* guide.
5. Create the **compute-vgpu-nvidia** flavor to use to tag nodes that you want to designate for vGPU workloads:

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

6. Tag each node that you want to designate for GPU workloads with the **compute-vgpu-nvidia** profile.

```
(undercloud) [stack@director templates]$ openstack baremetal node set --property
capabilities='profile:compute-vgpu-nvidia,boot_option:local' <node>
```

Replace **<node>** with the ID of the baremetal node.

7. To verify the role is created, enter the following command:

```
(undercloud) [stack@director templates]$ openstack overcloud profiles list
```

### 11.2.3. Configuring the Compute node for vGPU and deploying the overcloud

You need to retrieve and assign the vGPU type that corresponds to the physical GPU device in your environment, and prepare the environment files to configure the Compute node for vGPU.

#### Procedure

1. Install Red Hat Enterprise Linux and the NVIDIA GRID driver on a temporary Compute node and launch the node. For more information about installing the NVIDIA GRID driver, see [Section 11.2.1, "Building a custom GPU overcloud image"](#).

- On the Compute node, locate the vGPU type of the physical GPU device that you want to enable. For `libvirt`, virtual GPUs are mediated devices, or **mdev** type devices. To discover the supported **mdev** devices, enter the following command:

```
[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
```

- Add the **compute-gpu.yaml** file to the **network-environment.yaml** file:

```
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
```

- Add the following parameters to the **node-info.yaml** file to specify the number of GPU-enabled Compute nodes, and the flavor to use for the vGPU-designated Compute nodes:

```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeFlavor: compute
  OvercloudComputeGpuFlavor: compute-vgpu-nvidia
  ControllerCount: 1
  ComputeCount: 0
  ComputeGpuCount: 3 #set to the no of GPU nodes you have
```

- Create a **gpu.yaml** file to specify the vGPU type of your GPU device:

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



#### NOTE

Each physical GPU supports only one virtual GPU type. If you specify multiple vGPU types in this property, only the first type is used.

- Deploy the overcloud, adding your new role and environment files to the stack along with your other environment files:

```
(undercloud) $ openstack overcloud deploy --templates \
-r /home/stack/templates/roles_data_gpu.yaml
-e /home/stack/templates/node-info.yaml
```

```
-e /home/stack/templates/network-environment.yaml
-e [your environment files]
-e /home/stack/templates/gpu.yaml
```

## 11.3. CREATING THE VGPU IMAGE AND FLAVOR

To enable your cloud users to create instances that use a virtual GPU (vGPU), you can define a custom vGPU-enabled image, and you can create a vGPU flavor.

### 11.3.1. Creating a custom GPU instance image

After you deploy the overcloud with GPU-enabled Compute nodes, you can create a custom vGPU-enabled instance image with the NVIDIA GRID guest driver and license file.

#### Procedure

1. Create an instance with the hardware and software profile that your vGPU instances require:

```
(overcloud) [stack@director ~]$ openstack server create --flavor <flavor> --image <image>
temp_vgpu_instance
```

- Replace **<flavor>** with the name or ID of the flavor that has the hardware profile that your vGPU instances require. For information on default flavors, see [Manage flavors](#).
  - Replace **<image>** with the name or ID of the image that has the software profile that your vGPU instances require. For information on downloading RHEL cloud images, see [Image service](#).
2. Log in to the instance as a cloud-user. For more information, see [Log in to an Instance](#).
  3. Create the **gridd.conf** NVIDIA GRID license file on the instance, following the NVIDIA guidance: [Licensing an NVIDIA vGPU on Linux by Using a Configuration File](#).
  4. Install the GPU driver on the instance. For more information about installing an NVIDIA driver, see [Installing the NVIDIA vGPU Software Graphics Driver on Linux](#).



#### NOTE

Use the **hw\_video\_model** image property to define the GPU driver type. You can choose **none** if you want to disable the emulated GPUs for your vGPU instances. For more information about supported drivers, see [Appendix A, Image configuration parameters](#).

5. Create an image snapshot of the instance:

```
(overcloud) [stack@director ~]$ openstack server image create --name vgpu_image
temp_vgpu_instance
```

6. Optional: Delete the instance.

### 11.3.2. Creating a vGPU flavor for instances

After you deploy the overcloud with GPU-enabled Compute nodes, you can create a custom flavor that your cloud users can use to launch instances for GPU workloads.

### Procedure

1. Create an NVIDIA GPU flavor. For example:

```
(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. Assign a vGPU resource to the flavor that you created. You can assign only one vGPU for each instance.

```
(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

### 11.3.3. Launching a vGPU instance

You can create a GPU-enabled instance for GPU workloads.

### Procedure

1. Create an instance using a GPU flavor and image. For example:

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

2. Log in to the instance as a cloud-user. For more information, see [Log in to an Instance](#) .
3. To verify that the GPU is accessible from the instance, run the following command from the instance:

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

## 11.4. ENABLING PCI PASSTHROUGH FOR A GPU DEVICE

You can use PCI passthrough to attach a physical PCI device, such as a graphics card, to an instance. If you use PCI passthrough for a device, the instance reserves exclusive access to the device for performing tasks, and the device is not available to the host.

### Prerequisites

- The **pciutils** package is installed on the physical servers that have the PCI cards.
- The GPU driver is available to install on the GPU instances. For more information, see [Section 11.2.1, “Building a custom GPU overcloud image”](#) .

### Procedure

1. To determine the vendor ID and product ID for each passthrough device type, run the following command on the physical server that has the PCI cards:

```
# lspci -nn | grep -i <gpu_name>
```

For example, to determine the vendor and product ID for an NVIDIA GPU, run the following command:

```
# lspci -nn | grep -i nvidia
3b:00.0 3D controller [0302]: NVIDIA Corporation TU104GL [Tesla T4] [10de:1eb8] (rev a1)
d8:00.0 3D controller [0302]: NVIDIA Corporation TU104GL [Tesla T4] [10de:1db4] (rev a1)
```

2. To configure the Controller node on the overcloud for PCI passthrough, create an environment file, for example, **pci\_passthru\_controller.yaml**.
3. Add **PciPassthroughFilter** to the **NovaSchedulerDefaultFilters** parameter in **pci\_passthru\_controller.yaml**:

```
parameter_defaults:
  NovaSchedulerDefaultFilters:
  ['RetryFilter','AvailabilityZoneFilter','ComputeFilter','ComputeCapabilitiesFilter','ImageProperties
Filter','ServerGroupAntiAffinityFilter','ServerGroupAffinityFilter','PciPassthroughFilter','NUMATo
pologyFilter']
```

- To specify the PCI alias for the devices on the Controller node, add the following to **pci\_passthru\_controller.yaml**:

```
ControllerExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
```



#### NOTE

If the **nova-api** service is running in a role other than the Controller, then replace **ControllerExtraConfig** with the user role, in the format **<Role>ExtraConfig**.

- To configure the Compute node on the overcloud for PCI passthrough, create an environment file, for example, **pci\_passthru\_compute.yaml**.
- To specify the available PCIs for the devices on the Compute node, add the following to **pci\_passthru\_compute.yaml**:

```
parameter_defaults:
  NovaPCIPassthrough:
    - vendor_id: "10de"
      product_id: "1eb8"
```

- To enable IOMMU in the server BIOS of the Compute nodes to support PCI passthrough, add the **KernelArgs** parameter to **pci\_passthru\_compute.yaml**:

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

- Deploy the overcloud, adding your custom environment files to the stack along with your other environment files:

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

- Configure a flavor to request the PCI devices. The following example requests two devices, each with a vendor ID of **10de** and a product ID of **13f2**:

```
# openstack flavor set m1.large --property "pci_passthrough:alias"="t4:2"
```

#### Verification

- Create an instance with a PCI passthrough device:

```
# openstack server create --flavor m1.large --image rhelgpu --wait test-pci
```

2. Log in to the instance as a cloud user.
3. Install the GPU driver on the instance. For example, run the following script to install an NVIDIA driver:

```
$ sh NVIDIA-Linux-x86_64-430.24-grid.run
```

4. To verify that the GPU is accessible from the instance, enter the following command from the instance:

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

5. To check the NVIDIA System Management Interface status, run the following command from the instance:

```
$ nvidia-smi
```

Example output:

```
-----
| NVIDIA-SMI 440.33.01   Driver Version: 440.33.01   CUDA Version: 10.2   |
|-----+-----+
| GPU Name      Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|=====+=====+
| 0  Tesla T4           Off | 00000000:01:00:0 Off |             0      |
| N/A   43C    P0   20W / 70W |  0MiB / 15109MiB |  0%      Default |
|-----+-----+

-----
| Processes:                                     GPU Memory |
|  GPU   PID  Type  Process name                               Usage      |
|-----+-----+
| No running processes found                       |
|-----+-----+
```



## APPENDIX A. IMAGE CONFIGURATION PARAMETERS

You can use the following keys with the **property** option for both the **glance image-update** and **glance image-create** commands. For example:

```
$ glance image-update <image_uuid> --property architecture=x86_64
```



### NOTE

If you set an image property that conflicts with the same property on the flavor then either the flavor property is used, or an error is raised. There are some exceptions to this behavior for legacy compatibility.

Table A.1. Property keys

Specific to	Key	Description	Supported values
All	<b>architecture</b>	The CPU architecture that must be supported by the hypervisor. For example, <b>x86_64</b> , <b>arm</b> , or <b>ppc64</b> . Run <b>uname -m</b> to get the architecture of a machine.	<ul style="list-style-type: none"> <li>● <b>alpha</b> - DEC 64-bit RISC</li> <li>● <b>armv7l</b> - ARM Cortex-A7 MPCore</li> <li>● <b>cris</b> - Ethernet, Token Ring, AXis-Code Reduced Instruction Set</li> <li>● <b>i686</b> - Intel sixth-generation x86 (P6 micro architecture)</li> <li>● <b>ia64</b> - Itanium</li> <li>● <b>lm32</b> - Lattice Micro32</li> <li>● <b>m68k</b> - Motorola 68000</li> <li>● <b>microblaze</b> - Xilinx 32-bit FPGA (Big Endian)</li> <li>● <b>microblazeel</b> - Xilinx 32-bit FPGA (Little Endian)</li> <li>● <b>mips</b> - MIPS 32-bit RISC (Big Endian)</li> <li>● <b>mipsel</b> - MIPS 32-bit RISC (Little Endian)</li> <li>● <b>mips64</b> - MIPS 64-bit RISC (Big Endian)</li> <li>● <b>mips64el</b> - MIPS 64-bit RISC (Little Endian)</li> <li>● <b>openrisc</b> - OpenCores RISC</li> <li>● <b>parisc</b> - HP Precision Architecture RISC</li> <li>● <b>parisc64</b> - HP Precision Architecture 64-bit RISC</li> <li>● <b>ppc</b> - PowerPC 32-bit</li> </ul>

Specific to	Key	Description	Supported values
			<ul style="list-style-type: none"> <li>● <b>ppc64</b> - PowerPC 64-bit</li> <li>● <b>ppcemb</b> - PowerPC (Embedded 32-bit)</li> <li>● <b>s390</b> - IBM Enterprise Systems Architecture/390</li> <li>● <b>s390x</b> - S/390 64-bit</li> <li>● <b>sh4</b> - SuperH SH-4 (Little Endian)</li> <li>● <b>sh4eb</b> - SuperH SH-4 (Big Endian)</li> <li>● <b>sparc</b> - Scalable Processor Architecture, 32-bit</li> <li>● <b>sparc64</b> - Scalable Processor Architecture, 64-bit</li> <li>● <b>unicore32</b> - Microprocessor Research and Development Center RISC Unicores2</li> <li>● <b>x86_64</b> - 64-bit extension of IA-32</li> <li>● <b>xtensa</b> - Tensilica Xtensa configurable microprocessor core</li> <li>● <b>xtensaeb</b> - Tensilica Xtensa configurable microprocessor core (Big Endian)</li> </ul>
All	<b>hypervisor_type</b>	The hypervisor type.	<b>kvm, vmware</b>
All	<b>instance_uuid</b>	For snapshot images, this is the UUID of the server used to create this image.	Valid server UUID
All	<b>kernel_id</b>	The ID of an image stored in the Image Service that must be used as the kernel when booting an AMI-style image.	Valid image ID

Specific to	Key	Description	Supported values
All	<b>os_distro</b>	The common name of the operating system distribution in lowercase.	<ul style="list-style-type: none"> <li>● <b>arch</b> - Arch Linux. Do not use <b>archlinux</b> or <b>org.archlinux</b>.</li> <li>● <b>centos</b> - Community Enterprise Operating System. Do not use <b>org.centos</b> or <b>CentOS</b>.</li> <li>● <b>debian</b> - Debian. Do not use <b>Debian</b> or <b>org.debian</b>.</li> <li>● <b>fedora</b> - Fedora. Do not use <b>Fedora</b>, <b>org.fedora</b>, or <b>org.fedoraproject</b>.</li> <li>● <b>freebsd</b> - FreeBSD. Do not use <b>org.freebsd</b>, <b>freeBSD</b>, or <b>FreeBSD</b>.</li> <li>● <b>gentoo</b> - Gentoo Linux. Do not use <b>Gentoo</b> or <b>org.gentoo</b>.</li> <li>● <b>mandrake</b> - Mandrakelinux (MandrakeSoft) distribution. Do not use <b>mandrakelinux</b> or <b>MandrakeLinux</b>.</li> <li>● <b>mandriva</b> - Mandriva Linux. Do not use <b>mandrivalinux</b>.</li> <li>● <b>mes</b> - Mandriva Enterprise Server. Do not use <b>mandrivaent</b> or <b>mandrivaES</b>.</li> <li>● <b>msdos</b> - Microsoft Disc Operating System. Do not use <b>ms-dos</b>.</li> <li>● <b>netbsd</b> - NetBSD. Do not use <b>NetBSD</b> or <b>org.netbsd</b>.</li> <li>● <b>netware</b> - Novell NetWare. Do not use <b>novell</b> or <b>NetWare</b>.</li> <li>● <b>openbsd</b> - OpenBSD. Do not use <b>OpenBSD</b> or <b>org.openbsd</b>.</li> <li>● <b>opensolaris</b> - OpenSolaris. Do not use <b>OpenSolaris</b> or <b>org.opensolaris</b>.</li> <li>● <b>opensuse</b> - openSUSE. Do not use <b>suse</b>, <b>SuSE</b>, or <b>org.opensuse</b>.</li> <li>● <b>rhel</b> - Red Hat Enterprise Linux. Do not use <b>redhat</b>, <b>RedHat</b>, or <b>com.redhat</b>.</li> <li>● <b>sled</b> - SUSE Linux Enterprise Desktop. Do not use <b>com.suse</b>.</li> <li>● <b>ubuntu</b> - Ubuntu. Do not use <b>Ubuntu</b>, <b>com.ubuntu</b>, <b>org.ubuntu</b>, or <b>canonical</b>.</li> </ul>

Specific to	Key	Description	<ul style="list-style-type: none"> <li><b>windows</b> - Microsoft Windows. Do not supported values <b>microsoft.server</b>.</li> </ul>
All	<b>os_version</b>	The operating system version as specified by the distributor.	Version number (for example, "11.10")
All	<b>ramdisk_id</b>	The ID of image stored in the Image Service that should be used as the ramdisk when booting an AMI-style image.	Valid image ID
All	<b>vm_mode</b>	The virtual machine mode. This represents the host/guest ABI (application binary interface) used for the virtual machine.	<b>hvm</b> -Fully virtualized. This is the mode used by QEMU and KVM.
libvirt API driver	<b>hw_disk_bus</b>	Specifies the type of disk controller to attach disk devices to.	<b>scsi, virtio, ide, or usb</b> . Note that if using <b>iscsi</b> , the <b>hw_scsi_model</b> needs to be set to <b>virtio-scsi</b> .
libvirt API driver	<b>hw_cdrom_buses</b>	Specifies the type of disk controller to attach CD-ROM devices to.	<b>scsi, virtio, ide, or usb</b> . If you specify <b>iscsi</b> , you must set the <b>hw_scsi_model</b> parameter to <b>virtio-scsi</b> .
libvirt API driver	<b>hw_numa_nodes</b>	Number of NUMA nodes to expose to the instance (does not override flavor definition).	Integer. For a detailed example of NUMA-topology definition, see the <a href="#">hw:NUMA_def</a> key in <a href="#">Add Metadata</a> .
libvirt API driver	<b>hw_numa_cpus.0</b>	Mapping of vCPUs N-M to NUMA node 0 (does not override flavor definition).	Comma-separated list of integers.

Specific to	Key	Description	Supported values
libvirt API driver	<b>hw_numa_cpus.1</b>	Mapping of vCPUs N-M to NUMA node 1 (does not override flavor definition).	Comma-separated list of integers.
libvirt API driver	<b>hw_numa_mem.0</b>	Mapping N MB of RAM to NUMA node 0 (does not override flavor definition).	Integer
libvirt API driver	<b>hw_numa_mem.1</b>	Mapping N MB of RAM to NUMA node 1 (does not override flavor definition).	Integer
libvirt API driver	<b>hw_qemu_guest_agent</b>	Guest agent support. If set to <b>yes</b> , and if <b>qemu-ga</b> is also installed, file systems can be quiesced (frozen) and snapshots created automatically.	<b>yes / no</b>

Specific to	Key	Description	Supported values
libvirt API driver	<b>hw_rng_mode</b>	<p>Adds a random-number generator device to the image's instances. The cloud administrator can enable and control device behavior by configuring the instance's flavor. By default:</p> <ul style="list-style-type: none"> <li>• The generator device is disabled.</li> <li>• <code>/dev/random</code> is used as the default entropy source. To specify a physical HW RNG device, set <b>rng_dev_path</b> to <code>"/dev/hwrng"</code> in your Compute environment file.</li> </ul>	<b>virtio</b> , or other supported device.

Specific to	Key	Description	Supported values
libvirt API driver	<b>hw_scsi_model</b>	Enables the use of VirtIO SCSI (virtio-scsi) to provide block device access for compute instances; by default, instances use VirtIO Block (virtio-blk). VirtIO SCSI is a para-virtualized SCSI controller device that provides improved scalability and performance, and supports advanced SCSI hardware.	<b>virtio-scsi</b>

Specific to	Key	Description	Supported values
libvirt API driver	<b>hw_video_model</b>	The video device driver to use in virtual machine instances.	<p>List of supported drivers, in order of precedence:</p> <ul style="list-style-type: none"> <li>● <b>virtio</b>. (Recommended) Virtual GPU with the Gallium GPU specification that uses the VIRGL renderer to render OpenGL. This GPU model is supported in all architectures, and can leverage hardware acceleration if the host has a dedicated GPU. For more information, see <a href="https://virgil3d.github.io/">https://virgil3d.github.io/</a>.</li> <li>● <b>qxl</b>. High-performance driver for Spice or noVNC environments.</li> <li>● <b>cirrus</b>. Legacy driver, use if the <b>QXL</b> driver is not available.</li> <li>● <b>vga</b>. Use this driver for IBM Power environments.</li> <li>● <b>gop</b>. Not supported for QEMU/KVM environments.</li> <li>● <b>xen</b>. Not supported for KVM environments.</li> <li>● <b>vmvga</b>. Legacy driver, do not use.</li> <li>● <b>none</b>. Use this value to disable emulated graphics or video in virtual GPU (vGPU) instances where the driver is configured separately. For more information, see <a href="#">Chapter 11, Configuring virtual GPUs for instances</a>.</li> </ul>
libvirt API driver	<b>hw_video_ram</b>	Maximum RAM for the video image. Used only if a <b>hw_video:ram_max_mb</b> value has been set in the flavor's <b>extra_specs</b> and that value is higher than the value set in <b>hw_video_ram</b> .	Integer in MB (for example, 64)



Specific to	Key	Description	Supported values
libvirt API driver	<b>hw_watchdog_action</b>	Enables a virtual hardware watchdog device that carries out the specified action if the server hangs. The watchdog uses the i6300esb device (emulating a PCI Intel 6300ESB). If <b>hw_watchdog_action</b> is not specified, the watchdog is disabled.	<ul style="list-style-type: none"> <li>● disabled-The device is not attached. Allows the user to disable the watchdog for the image, even if it has been enabled using the image's flavor. The default value for this parameter is disabled.</li> <li>● reset-Forcefully reset the guest.</li> <li>● poweroff-Forcefully power off the guest.</li> <li>● pause-Pause the guest.</li> <li>● none-Only enable the watchdog; do nothing if the server hangs.</li> </ul>
libvirt API driver	<b>os_command_line</b>	The kernel command line to be used by the libvirt driver, instead of the default. For Linux Containers (LXC), the value is used as arguments for initialization. This key is valid only for Amazon kernel, ramdisk, or machine images (aki, ari, or ami).	
libvirt API driver and VMware API driver	<b>hw_vif_model</b>	Specifies the model of virtual network interface device to use.	<p>The valid options depend on the configured hypervisor.</p> <ul style="list-style-type: none"> <li>● KVM and QEMU: e1000, ne2k_pci, pcnet, rtl8139, and virtio.</li> <li>● VMware: e1000, e1000e, VirtualE1000, VirtualE1000e, VirtualPCNet32, VirtualSriovEthernetCard, and VirtualVmxnet.</li> <li>● Xen: e1000, netfront, ne2k_pci, pcnet, and rtl8139.</li> </ul>

Specific to	Key	Description	Supported values
VMware API driver	<b>vmware_adaptertype</b>	The virtual SCSI or IDE controller used by the hypervisor.	<b>lsiLogic, busLogic, or ide</b>
VMware API driver	<b>vmware_ostype</b>	A VMware GuestID which describes the operating system installed in the image. This value is passed to the hypervisor when creating a virtual machine. If not specified, the key defaults to <b>otherGuest</b> .	For more information, see <a href="#">Images with VMware vSphere</a> .
VMware API driver	<b>vmware_image_version</b>	Currently unused.	<b>1</b>
XenAPI driver	<b>auto_disk_config</b>	If true, the root partition on the disk is automatically resized before the instance boots. This value is only taken into account by the Compute service when using a Xen-based hypervisor with the XenAPI driver. The Compute service will only attempt to resize if there is a single partition on the image, and only if the partition is in <b>ext3</b> or <b>ext4</b> format.	<b>true / false</b>

Specific to	Key	Description	Supported values
libvirt API driver and XenAPI driver	<b>os_type</b>	The operating system installed on the image. The XenAPI driver contains logic that takes different actions depending on the value of the <b>os_type</b> parameter of the image. For example, for <b>os_type=windows</b> images, it creates a FAT32-based swap partition instead of a Linux swap partition, and it limits the injected host name to less than 16 characters.	<b>linux</b> or <b>windows</b>

## APPENDIX B. ENABLING THE LAUNCH INSTANCE WIZARD

There are two methods that you can use to launch instances from the dashboard:

- The Launch Instance form
- The Launch Instance wizard

The Launch Instance form is enabled by default, but you can enable the Launch Instance wizard at any time. You can also enable both the Launch Instance form and the Launch Instance wizard at the same time. The Launch Instance wizard simplifies the steps required to create instances.

1. Edit `/etc/openstack-dashboard/local_settings` file, and add the following values:

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

2. Restart the `httpd` service:

```
# systemctl restart httpd
```

The preferences for the Launch Instance form and Launch Instance wizard are updated.

If you enabled only one of these options, the **Launch Instance** button in the dashboard opens that option by default. If you enabled both options, two **Launch Instance** buttons are displayed in the dashboard, with the button on the left opening the Launch Instance wizard and the button on the right opening the Launch Instance form.