Hewlett Packard Enterprise

Helion OpenStack Carrier Grade 4.0

CLOUD ADMINISTRATION

16.10

Acknowledgements

Java® and Oracle® are registered trademarks of Oracle and/or its affiliates

http://www.hpe.com/info/storagewarranty

Helion Carrier Grade 4.0

Cloud Administration, 16.10

Contents

1
Cloud Administration Introduction

Cloud Administration Overview

This guide provides information on topics that an OpenStack administrator would be responsible for, except for the management of the physical nodes and physical networks.

Information for managing the physical nodes and physical networks is provided in the *Helion Carrier Grade 4.0 System Administration Guide*.

The following content is covered in this guide:

- Image Management
- Volume Management
- Flavor Management
- Tenant Network Management
- Router Management

2

Volume and Image Management

Volume and Image Management Overview

Volume Management is primarily the same as upstream OpenStack.

Generally, you should refer to upstream OpenStack documentation for information about volume management. The topics in this section highlight Helion Carrier Grade 4.0 extensions.

Creating Cached RAW Boot Images

You can use cached RAW boot image files for faster volume creation on Ceph-backed systems.

On systems using Ceph storage, boot image files must be converted to RAW format before they can be used to create volumes. You can accelerate volume creation in Helion Carrier Grade 4.0 (and therefore instance launch time) by caching the RAW images as they are created. The cached images are maintained in the Glance image storage space and used for volume creation, eliminating conversion time.

By default, caching is performed in the background to ensure low processing overhead. If multiple requests are made at the same time, they are executed serially. You can use the **--wait** option to override background processing, so that the command does not return until the operation is complete and the raw image is ready for use. If you include a time in seconds (for example, **--wait 10**), the command returns after the time has elapsed and reports the current status of the operation.

This feature is available from the CLI only. To create a cache, include the **--cached-raw** option when creating an image, as shown in the following example:

```
~(keystone_admin)$ glance image-create --name "cirros" --visibility public \
--disk-format=qcow2 --container-format=bare \
--file /home/wrsroot/images/cirros.img --cache-raw
+----------------------+--------------------------------------+
| Property             | Value                                |
+----------------------+--------------------------------------+
| Property 'cache_raw' | True                                 |
| checksum             | 9ffe77b56668bf3fad95c64df4a8ac4f     |
| container_format     | bare                                 |
| created_at           | 2016-01-07T22:06:22.769563           |
| deleted              | False                                |
| deleted_at           | None                                 |
| disk_format          | qcow2                                |
| id                   | a5c6ef41-fe73-4d25-abbc-64380646e705 |
| visibility           | public                               |
| min_disk             | 0                                    |
| min_ram              | 0                                    |
| name                 | cirros                               |
| owner                | f34e518e265a402aa53ce7c29475cab1     |
| protected            | False                                |
| size                 | 636485632                            |
| status               | active                               |
| updated_at           | 2016-01-07T22:06:39.225343           |
| virtual_size         | None                                 |
+----------------------+--------------------------------------+
```

The cache size and status are included in the images list for the CLI.

```
~(keystone_admin)$ glance -v image-list
+----...+--------+...+--------+------------+-----------+
| ID ...| Name   |...| Status | Cache Size | Raw Cache |
+----...+--------+...+--------+------------+-----------+
| a5c...| cirros |...| active | 5636485632 | Cached    |
+----...+--------+...+--------+------------+-----------+
```

In the web administration interface, this list is shown on the Images page accessible from the **System** menu.

NOTE: Until image caching is complete, an image created using **--cached-raw** behaves as any regular uncached image.

Creating Cinder Volumes for Boot Images

You can create Cinder volumes to store VM boot images.

The use of Cinder volumes is recommended for fast instance booting.

NOTE: You can accelerate volume creation by using a cached boot image. For more information, see Creating Cached RAW Boot Images on page 3.

Procedure

1. Log in as the appropriate tenant.
2. On the **Project** menu, open the **Compute** section, and then click **Volumes**.
3. On the Volumes page, click **Create Volume**
4. Complete the Create Volume form.

Cinder Volume Backups

Cinder volume backups are executed selectively, one volume at a time. They can include virtual disk images and any other Cinder volumes available in the system.

Consider the following recommendations when executing Cinder volume backups.

Aim at backing up an idle system

In an idle system, there are no running virtual machines. All VMs are either shut down or terminated. This is the ideal state for ensuring that the current content of all volumes is captured for replication at the next restore operation. Volumes reported as *available* are not associated with any VMs, while volumes reported as *in-use* are associated only with shut-down VMs, and are not being actively modified.

In a non-idle system, at least one VM is running. Volumes reported as *in-use* may be subject to write operations. In this case, you can create snapshots and back them up, but there is a risk of changes after the snapshot that are not captured.

Instructions to back up Cinder volumes in *available* and *in-use* states are provided in the following sections.

Back up one volume at a time

Backing up Cinder volumes can be a time- and space- consuming process, as determined by the size of the volumes themselves. Having two or more backup processes running simultaneously is likely to result in a severe performance hit to the controller and the running instances. This not only slows the normal system operations, but also makes the backup operations themselves take much longer.

Clean up as soon as possible

Backup files for Cinder volumes can fill up the **/opt/backups** partition on the controller very quickly. This is likely to cause further backup operations to fail in the middle of the execution. Therefore, as soon as a Cinder volume is backed up, and its backup files transferred and securely stored elsewhere:

- remove any snapshots you may have created during the backup process
- remove the backup files themselves

Backing Up Available Cinder Volumes

You can back up Cinder volumes in the *available* state.

A Cinder volume in the *available* state is the ideal candidate for a backup operation in that there are no running instances using it.

Procedure

1. Log in as Keystone user **admin** to the active controller.

   Log in as user **wrsroot** to the active controller and source the script **/etc/nova/openrc** to obtain administrative privileges. For more information, see *Helion Carrier Grade 4.0 System Administration: Linux User Accounts*.

2. Identify the volume you want to back up.

   a) List Cinder volumes from all tenants.

   ```
   ~(keystone_admin)$ cinder list --all-tenants--+------+-------------+----------
   |    ID        |   Status   |    Display Name    | Size | Volume Type | Bootable
   +--------------+-----------+-----------------+------+-------------+----------
   | 593111d8-... | in-use    | volume-1         |  1   |     None     |   true
   | 53ad007a-... | available | volume-2         |  10  |     None     |   false
   +--------------+-----------+-----------------+------+-------------+----------
   +---------------+
   | Attached to   |
   +---------------+
   |  b998107b-... |
   |               |
   +---------------+
   ```

   Cinder volumes are listed also on the Volumes page of the web administration interface.

   b) Identify the volume to back up.

   In the list above, **volume-1** is a disk image associated with a running instance. The volume **volume-2**, with UUID **53ad007a-...**, is available and can be safely backed up.

3. Export the target volume to the controller's file system.

   ```
   ~(keystone_admin)$ cinder export 53ad007a-841a-4fd3-a22b-813d4a6a8a85
   +---------------------+--------------------------------------+
   |       Property      |                Value                 |
   +---------------------+--------------------------------------+
   | display_description |                                      |
   |          id         | 53ad007a-841a-4fd3-a22b-813d4a6a8a85 |
   |         size        |                  1                   |
   |        status       |              exporting               |
   |      updated_at     |      2014-09-18T21:36:15.247881      |
   |     volume_type     |                 None                 |
   +---------------------+--------------------------------------+
   ```

   Exporting the volume takes some time, longer times for larger volumes. While it is taking place, the status of the volume is set to *exporting*, which you can verify by issuing the command **cinder show** *UUID*, or **cinder list --all-tenants** again. This new state is automatically reported on the Web administration interface.

   You must wait for the export operation to complete. Use the command **cinder show 53ad007a-...** and look for the **backup_status** field to determine whether the export operation was successful. The state of the volume should be set to *available* again.

When the export operation completes, the backup file is ready, as illustrated in the following example:

```
~(keystone_admin)$ ls -lh /opt/backups
total 306M
drwx------ 2 root root  16K Sep 10 17:42 lost+found
drwxr-xr-x 2 root root 4.0K Sep 18 21:42 temp
-rw-r--r-- 1 root root 305M Sep 18 21:42 volume-53ad007a-841a-4fd3-
a22b-813d4a6a8a85-20140918-213918.tgz
```

4. Transfer the backup file to an external storage resource.

   The following example uses the **scp** command to transfer the backup file to the directory **/backups/titanium** on a server named **backups-server.wrs.com**:

```
~(keystone_admin)$ scp /opt/backups/volume-53ad007a-841a-4fd3-
a22b-813d4a6a8a85-20140918-213918.tgz admin@backups-server.wrs.com:/backups/titanium
```

5. Delete the backup file.

   You may want to free up disk space on the controller to accommodate additional Cinder volume backups.

The volume backup is complete. Repeat this procedure with all other Cinder volumes in the *available* state in the system.

Backing Up In-Use Cinder Volumes

You can back up Cinder volumes in the *in-use* state.

A Cinder volume in the *in-use* state is associated with an existing instance, either as the instance's disk image or as an attached storage resource. The instance can be either running or shut down. If the instance is running, you must take a snapshot of the volume first, and then export it to the controller's file system. An effective and reliable strategy is to take snapshots of all *in-use* volumes, regardless of whether the associated instances are running. If you prefer, you can use the **nova list** command with the option **--all-tenants** to identify which instances are running, and then take snapshots selectively.

```
~[keystone_admin]$ nova list --all-tenants

+--------+--------+-----------+--------+------------+-------------+----------+
| ID     | Name   | Tenant ID | Status | Task State | Power State | Networks
|
                  |
+--------+--------+-----------+--------+------------+-------------+----------+
| 1f6... | t1-vm1 | 5e38b9... | ACTIVE | -          | Running     | tenan... |
| b92... | t1-vm2 | 5e38b9... | ACTIVE | -          | Running     | tenan... |
+--------+--------+-----------+--------+------------+-------------+----------+
```

Procedure

1. Log in as Keystone user **admin** to the active controller.

   Log in as user **wrsroot** to the active controller and source the script **/etc/nova/openrc** to obtain administrative privileges as described in *Linux User Accounts*.

2. Identify the volume you want to back up.

a) List Cinder volumes from all tenants.

```
~(keystone_admin)$ cinder list --all-tenants
+-------------+----------+-------------+... +---------+-------------+
|     ID      |  Status  | Display Name |... | Bootable | Attached to  |
+-------------+----------+-------------+... +---------+-------------+
| 53ad007a-... |  in-use  |  volume-1   |... |  false   | c820df55-... |
| 578da734-... |  in-use  |             |... |  true    | c820df55-... |
| 593111d8-... | available |  volume-2   |... |  true    |              |
+-------------+----------+-------------+... +---------+-------------+
```

Cinder volumes are also listed on the Volumes page of the Web administration interface.

b) Identify the volume to back up.

In the list above, two volumes are listed in the *in-use* state:

- A volume named **volume-1** with ID **53ad007a-...**, attached to the running image **c820df55-...**. The bootable flag is *false*, which indicates that this is not a disk image.
- An unnamed volume with ID **578da734-...**, attached to the running image **c820df55-...**. The bootable flag is *true*, which indicates that this is the instance's disk image. Unnamed volumes are dynamically created when the option **Instance Boot Source** in the Launch Instance window of the Web administration interface is set to *Boot from image (creates a new volume)* or *Boot from volume snapshot (creates a new volume)*.

This example uses the unnamed disk image to illustrate the backup procedure, but you must back up all *in-use* volumes appropriately.

3. Create a volume snapshot.

```
~(keystone_admin)$ cinder snapshot-create --force True --display-name test-vm-
snapshot \
578da734-a416-47e7-97de-137aa10b90f8
+--------------------+--------------------------------------+
|      Property      |                Value                 |
+--------------------+--------------------------------------+
|     created_at     |      2014-09-19T14:18:47.269583      |
| display_description |                 None                 |
|    display_name    |           test-vm-snapshot           |
|         id         | 53b5237c-5cd0-4939-ad6b-cf612e449216 |
|      metadata      |                  {}                  |
|        size        |                  1                   |
|       status       |               creating               |
|     volume_id      | 578da734-a416-47e7-97de-137aa10b90f8 |
+--------------------+--------------------------------------+
```

The status of the new snapshot is *creating*, which means that it is in process. You must wait until it becomes *available* before proceeding.

You can verify the status of all Cinder snapshots using the following command:

```
~(keystone_admin)$ cinder snapshot-list --all-tenants
+--------------+--------------+-----------+------------------+------+
|     ID       | Volume ID    |  Status   |   Display Name   | Size |
+--------------+--------------+-----------+------------------+------+
| 53b5237c-... | 578da734-... | available | test-vm-snapshot |  1   |
+--------------+--------------+-----------+------------------+------+
```

In this example, the state of the snapshot **test-vm-snapshot** is *available* already.

4. Export the snapshot to the controller's file system.

```
~(keystone_admin)$ cinder snapshot-export test-vm-snapshot
+--------------------+--------------------------------------+
|      Property      |                Value                 |
+--------------------+--------------------------------------+
| display_description |                 None                 |
|          id         | 53b5237c-5cd0-4939-ad6b-cf612e449216 |
|        status       |              exporting               |
|      updated_at     |      2014-09-19T14:18:47.393220      |
|     volume_size     |                  1                   |
|     volume_type     |                 None                 |
+---------------------+--------------------------------------+
```

The status of the snapshot is *exporting*, which means that it is not yet ready on the controller's file system. As before, you can use the **cinder snapshot-show** *UUID* command to verify the current status, and wait until the snapshot's state is *available* again.

When the export operation completes, the backup file is ready, as illustrated in the following example:

```
~(keystone_admin)$ ls -lh /opt/backups
total 258M
drwx------ 2 root root  16K Sep 10 17:42 lost+found
drwxr-xr-x 2 root root 4.0K Sep 19 14:30 temp
-rw-r--r-- 1 root root 258M Sep 19 14:30 volume-578da734-
a416-47e7-97de-137aa10b90f8-20140919-142732.tgz
```

Note that the UUID used in the file name is the volume's and not the snapshot's.

5. Transfer the backup file to an external storage resource.

   The following example uses the **scp** command to transfer the backup file to a server named **backups-server.wrs.com** in the directory **/backups/titanium**:

   ```
   ~(keystone_admin)$ scp /opt/backups/volume-578da734-
   a416-47e7-97de-137aa10b90f8-20140919-142732.tgz \
   admin@backups-server.wrs.com:/backups/titanium
   ```

6. Delete the snapshot.

   ```
   ~(keystone_admin)$ cinder snapshot-delete test-vm-snapshot
   ```

   You can confirm that the snapshot is deleted using **cinder snapshot-list --all-tenants**.

   If a standard deletion fails, you can force deletion using the following command:

   ```
   ~(keystone_admin)$ cinder snapshot-force-delete snapshot-id [snapshot-id]
   ```

   Once the snapshot is exported, you should remove it from the Cinder repositories to free up disk space.

7. Delete the backup file.

   You may want to free up disk space on the controller to accommodate additional Cinder volume backups.

The volume backup is complete. Repeat this procedure with all other Cinder volumes in the *in-use* state in the system.

3

VM Resource Usage and Behavior Customization

Options for VM Customization

You can use flavors, flavor extra specifications, and image metadata to allocate host resources for a VM, and to fine-tune the use of host resources.

Administrators can define the basic resources available to VMs by creating and using *flavors*. Each flavor defines the virtual CPU, RAM, and disk resources to be allotted when a VM is launched. For more information, see Virtual Machine Flavors on page 12

To adjust how VMs use host resources such as NUMA nodes, physical interfaces, or selected PCI devices, administrators can add *extra specifications* to the flavors. For more information, see Flavor Extra Specifications on page 14

In many cases, administrators or tenants can use *image metadata* as an alternative to flavor extra specs. This can be useful where the added capability is required for only a few particular instances. For more information, see Image Metadata on page 17

NOTE: If an extra spec method and an image metadata method are both applied to the same instance, and the settings conflict, the method that is given priority depends on the specific capability. In cases where the flavor extra spec overrides the image metadata, an administator can use this to help manage tenant resource usage.

Virtual Machine Flavors

A flavor is a list of attributes applied to a virtual machine when a new instance is created. It specifies resources to be allocated by the system, such as size of RAM, number of cores, storage resources, and so on.

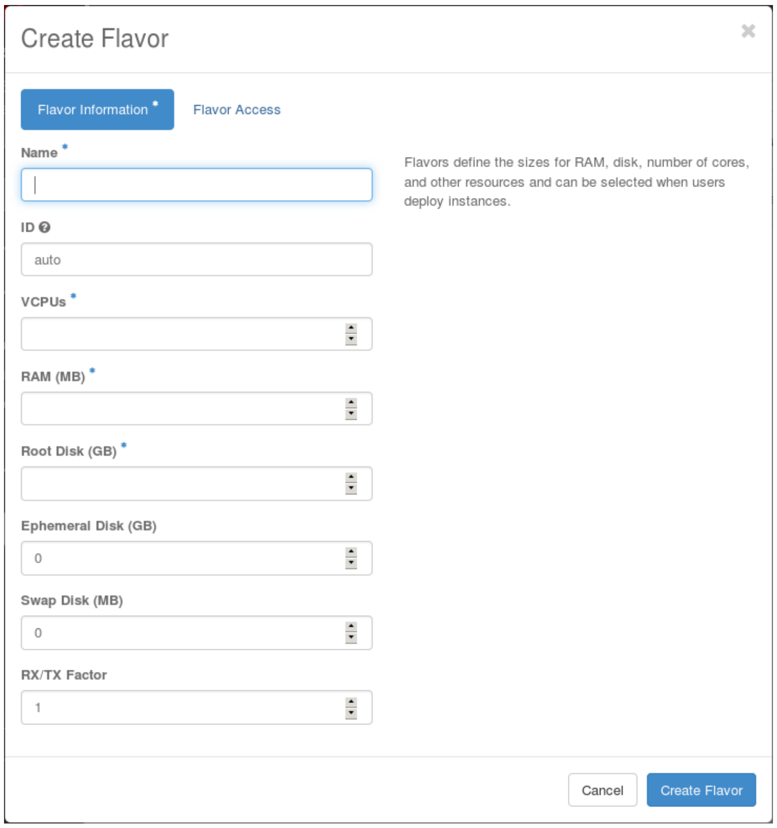
To work with flavors, see Creating and Editing Flavors on page 12.

Creating and Editing Flavors

You can create and edit flavors using the Flavors page of the web administration interface.

You can create and edit flavors using the Flavors page of the web administration interface. To access this page, select **Admin** > **System** > **Flavors**.

- To create a flavor, click the **Create Flavor** button. The Create Flavor window appears, as shown below:



Name *

ID

auto

Flavors define the sizes for RAM, disk, number of cores, and other resources and can be selected when users deploy instances.

VCPUs *

RAM (MB) *

Root Disk (GB) *

Ephemeral Disk (GB)

0

Swap Disk (MB)

0

RX/TX Factor

1

The Create Flavor window has two tabs.

Flavor Information

defines basic information for the flavor

Flavor Access

controls which projects can see and use the flavor

You can set the following basic attributes when you create or edit a flavor:

Name

The name to be associated with the flavor.

ID

The object ID to associate with the flavor. In most situations you should leave the default value of **auto** for the system to auto-generate a unique ID.

VCPUs

Number of virtual CPUs to be allocated to the virtual machine.

RAM MB

Memory, in megabytes, to be allocated to the virtual machine.

Root Disk GB

Specifies the virtual root disk size in gigabytes. This is an ephemeral disk to which the base image is copied. Use the value **0** to set the ephemeral disk size equal to the base image size. This value is ignored when booting from a Cinder volume.

NOTE: For information about storage allocation for the virtual root disk, see Specifying the Storage Type for VM Ephemeral Disks on page 33.

Ephemeral Disk GB

Specifies the size, in gigabytes, of a secondary ephemeral data disk. This is an empty, unformatted disk that exists only for the life time of the instance.

NOTE: For information about storage allocation for the ephemeral disk, see Specifying the Storage Type for VM Ephemeral Disks on page 33.

Swap Disk MB

Ephemeral swap space, in megabytes, to be allocated to the instance.

NOTE: For information about storage allocation for the swap disk, see Specifying the Storage Type for VM Ephemeral Disks on page 33.

TX/RX Factor

(not applicable to Helion Carrier Grade 4.0)

Editing Flavors

To edit the information or project access for an existing flavor, open the Flavors page, and then click **Edit Flavor** in the **Actions** column for the flavor.

To add extra specifications, click the **Flavor Name** to open the Flavor Detail page, and then select the **Extra Specs** tab. For more information, see Flavor Extra Specifications on page 14.

Deleting Flavors

To delete a flavor, first terminate any instances that use the flavor. You can identify the instances to be terminated using the following command:

```
~(keystone_admin)$ nova list --all-tenants --flavor name
```

where *name* is the name of the flavor.

Then, on the Flavors page, open the drop-down menu in the **Actions** column and select **Delete Flavor**.

NOTE: Before you can delete a flavor, you must teminate any instances that use the flavor.

Flavor Extra Specifications

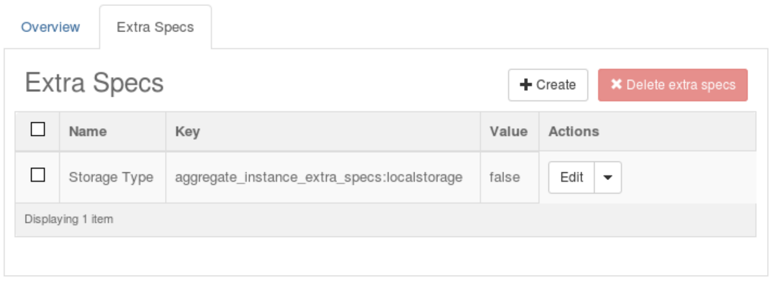
You can edit an existing flavor to include additional attributes using *extra specifications*.

Extra specifications are key-value pairs you can add to an existing flavor to be included when the flavor is used with a new virtual machine. The Helion Carrier Grade 4.0 **Extra Specs** tab for a flavor includes extensions specific to Helion Carrier Grade 4.0.

NOTE:

You can also use the **nova flavor-key** command to add or edit flavor extra specs from the command line. For specific examples and parameters, refer to the documentation for managing a specific resource.

To access the extra specifications settings for a flavor, open the Flavors window and click the flavor name. For help accessing the Flavors window, see Creating and Editing Flavors on page 12.



☐	Name	Key	Value	Actions
☐	Storage Type	aggregate_instance_extra_specs:localstorage	false	Edit ▾

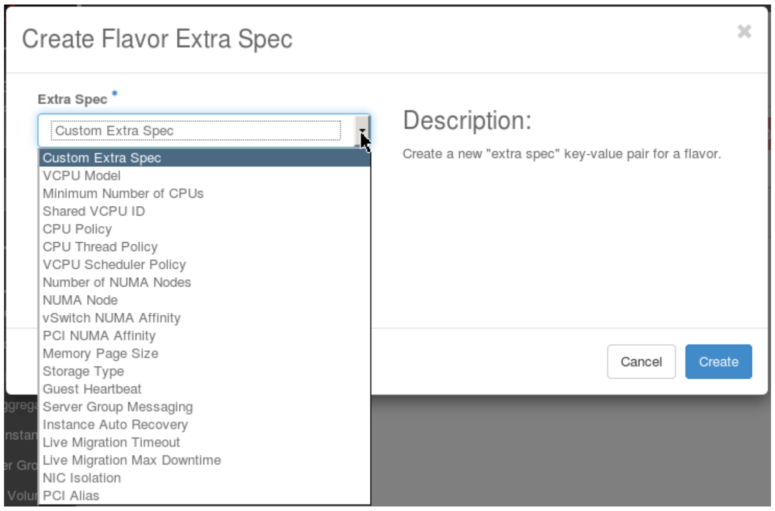
Displaying 1 item

The **Extra Specs** tab lists extra specifications that have been added for the flavor.

NOTE: A **Storage Type** extra spec is added by default for all new flavors. To ensure that instantiated VMs use storage of a known type, do not delete this extra spec. For more information, see Specifying the Storage Type for VM Ephemeral Disks on page 33.

To modify an existing entry, use the **Edit** button. To remove a single entry, open the drop-down menu associated with the extra spec, and select **Delete extra spec** from the menu. You can also remove all extra specs at once using the **Delete extra specs** button.

To add a new extra specification, click **Create**. This opens the Create Flavor Extra Spec window, as illustrated below:


Create Flavor Extra Spec

Extra Spec *

Custom Extra Spec

Custom Extra Spec
VCPU Model
Minimum Number of CPUs
Shared VCPU ID
CPU Policy
CPU Thread Policy
VCPU Scheduler Policy
Number of NUMA Nodes
NUMA Node
vSwitch NUMA Affinity
PCI NUMA Affinity
Memory Page Size
Storage Type
Guest Heartbeat
Server Group Messaging
Instance Auto Recovery
Live Migration Timeout
Live Migration Max Downtime
NIC Isolation
PCI Alias

Description:

Create a new "extra spec" key-value pair for a flavor.

Cancel Create

Use the **Extra Spec** drop-down list to add specifications. The following options are available:

Custom Extra Spec

Available for internal use only.

VCPU Model

Specifies the vCPU model to use with the virtual machine. If this extra spec is not added, then by default the QEMU processor is used. For more information, see Specifying the VCPU Model for a VM on page 20.

Minimum Number of CPUs

Sets the minimum number of virtual CPUs for the flavor. If this extra spec is not added, then by default the minimum is one. For more information, see Setting the CPU Scaling Range on page 24.

Shared VCPU ID

Specifies the ID of a virtual CPU scheduled to run on a shared physical CPU in the compute host. If this extra spec is not added, then by default the VM does not use a shared CPU. For more information, see *Helion Carrier Grade 4.0 System Administration: Designating Shared Physical CPUs on a Compute Host* and Pinning a vCPU to a Shared Physical CPU on page 21.

NOTE: To use this extra specification, you must also set the **CPU Policy** extra specification for the flavor to **Dedicated**. This enables the VM to run a single house-keeping virtual CPU on a shared physical core, while keeping all its high-performance virtual CPUs on dedicated physical cores.

CPU Policy

Specifies the policy for assigning virtual CPUs to available CPUs. If this extra spec is not added, then by default a vCPU on the VM can be assigned to any CPU, subject to the over-commit limit. For more information, see Specifying Host CPU Threading Policies for a VM on page 18.

CPU Thread Policy

Specifies the policy for using sibling threads on a hyperthreaded core when the **CPU Policy** is set to **Dedicated**. If this extra spec is not added, then vCPUs on the VM are assigned without regard to sibling assignments. For more information, see Specifying Host CPU Threading Policies for a VM on page 18.

VCPU Scheduler Policy

Sets the scheduling priority for non-boot virtual CPUs. If this extra spec is not added, then by default *nice* priority 0 is assigned.

NOTE: This extra spec is shown only for flavors with more than one CPU.

Helion Carrier Grade 4.0 also supports the OpenStack **hw:cpu_realtime** extra specification as an alternative scheduling method. This alternative is available from the command line only.

For more information about these options, see Configuring vCPU Scheduling and Priority on page 22.

Number of NUMA Nodes

Sets the number of virtual NUMA Nodes. If this extra spec is not added, then by default the memory and CPU resources for a flavor are assigned to a single virtual NUMA Node. For more information, see Configuring the NUMA Node Allocations for a VM on page 27.

NUMA Node

Specifies the NUMA node to use when launching a virtual machine. If this extra spec is not added, then by default any available NUMA node is used, subject to other NUMA node settings. For more information, see Pinning a Guest NUMA Node to a Host NUMA Node on page 29.

Vswitch NUMA Affinity

Specifies that a NUMA node with a vSwitch core is required or preferred when instantiating the VM. If this extra spec is not added, then by default any available NUMA node is used, subject to other NUMA node settings. For more information, see Affining a VM to a NUMA Node with a vSwitch Core on page 30.

PCI NUMA Affinity

For a VM configured to use an SR-IOV or PCI passthrough device, specifies that a NUMA node with direct access to the physical device is either required or preferred for instantiation. If this extra spec is not added, then by default a NUMA node with direct access is used. For more information, see Specifying Best Effort for PCI NUMA Node Affinity on page 29.

Memory Page Size

Sets the page size for VM memory. If this extra spec is not added, then by default small pages are used. For more information, see Specifying a Page Size for a VM on page 32.

Storage Type

Specifies how storage is instantiated for ephemeral and swap disks, and for the instance boot image if used. This extra spec is added by default and set for LVM-backed local storage. For more information, see Specifying the Storage Type for VM Ephemeral Disks on page 33.

Guest Heartbeat

Enables the Heartbeat API for use by guests on the VM. If this extra spec is not added, then by default this API is disabled. For more information, see Enabling the Guest Heartbeat API for a VM on page 34.

Server Group Messaging

Enables the VM to communicate with other VMs in the same server group using a messaging API. If this extra spec is not added, then by default this API is disabled. For more information, see Enabling Server Group Messaging for a VM on page 35.

Instance Auto Recovery

Disables auto recovery of failed virtual machine instances. If this extra spec is not added, then the system will automatically recover failed virtual machine instances. For more information, see Disabling Instance Auto Recovery on page 36.

Live Migration Timeout

Specifies the number of seconds to wait for a live migration to complete. If this extra spec is not added, then it defaults to 800 seconds. For more information, see *Helion Carrier Grade 4.0 Cloud Operation: Live Migration of Virtual Machines* and Specifying a Live Migration Timeout on page 37.

Live Migration Max Downtime

Specifies the number of milliseconds of downtime to accept during live migrations. If this extra spec is not set, then it defaults to 500 milliseconds. For more information, see Specifying Live Migration Maximum Downtime on page 38.

NIC Isolation

Enables isolating a virtual machine instance's network interface cards from all other interfaces running on the host. If this extra spec is not set, then the instance's interfaces are not isolated. For more information see Isolating an Untrusted Guest on page 39.

PCI Alias

Enables VM access to PCI-passthrough or SR-IOV devices (other than NICs). If this extra spec is not added, then no non-NIC devices are made available to the guest. For more information, see Accessing a PCI Device from a VM on page 39.

Image Metadata

You can use image metadata to adjust access to host resources for a VM when the VM is launched, or when it is running.

The **nova image-meta** command is used to apply metadata to an image.

The **glance image-update** command is used to update image metadata after it has been applied.

For specific examples and parameters, refer to the documentation for managing a specific resource.

Flavor Modification Using Extra Specs or Image Metadata

Specifying Host CPU Threading Policies for a VM

You can control the use of host CPU multithreading for a VM using **CPU Policy** and **CPU Thread Policy** extra specifications or image metadata.

When a virtual machine is launched, each of its virtual CPU threads is scheduled for processing on a physical or logical core. The **CPU Policy**, in conjunction with the **CPU Thread Policy**, determines whether the core can be over-committed (that is, used to execute multiple threads).

NOTE: For CPU scaling, the **CPU Policy** must be set to **Dedicated**. If the **CPU Thread Policy** is also present, the **CPU Thread Policy** must be set to **Isolate** or **Prefer**.

CPU Policy

The following settings are available for the **CPU Policy**:

Dedicated

When the **CPU Policy** is set to **Dedicated**, each virtual CPU thread is scheduled to run as a *pinned* vCPU on a dedicated core on the compute node.

Shared

When the **CPU Policy** is set to **Shared**, each virtual CPU thread is scheduled to run as a *floating* vCPU on any available core on the compute node, subject to an over-commit limit. The OpenStack virtual CPU over-commit ratio is 16:1, which means that up to 16 virtual CPU threads can share a single core.

In contrast to the standard OpenStack implementation, instances with **Dedicated** and **Shared** policies can safely be instantiated on the same host. Instances with **Shared** policies are prevented from using threads on host CPUs assigned to instances with **Dedicated** policies. The availability of host CPUs for shared use is updated as pinned CPUs are assigned or released. This ensures optimal processor time for dedicated applications, as well as efficient use of CPUs on the hosts.

To support this model, one host CPU is reserved for each increment of 16 floating vCPUs. This is reflected in the vCPU usage statistics for the host, as given by the **nova hypervisor-show** command in Helion Carrier Grade 4.0. Each pinned vCPU is reported as using one host CPU, and each floating vCPU is reported as using 1/16 of a host CPU. For example, on a compute node that is hosting five dedicated and four floating vCPUs, the CPU usage (the **vcpus_used** value) is shown as **5.25**, providing an indication of the total CPU utilization for the host.

The default **CPU Policy** setting is **Shared**. You can change this by adding the **CPU Policy** extra spec and setting it to **Dedicated**. When you do so using the web administration interface, the non-default policy **Dedicated** is offered for convenience.

The **Dedicated** setting is recommended for Carrier-Grade requirements to prevent over-commitment of host resources, and to minimize the impact that scheduling events may have on the latency and throughput responses of the guest kernel.

CAUTION: When the **CPU Policy** setting is used, an eligible host NUMA node is required for each virtual NUMA node in the instance. If this requirement cannot be met, the instantiation fails. For more information, see *Helion Carrier Grade 4.0 Cloud Operation: The Virtual Machine Scheduler*.

CPU Thread Policy

To manage how sibling threads are utilitized under the **CPU Policy**, you can add a **CPU Thread Policy** extra specification. The **CPU Thread Policy** controls the use of sibling pairs. If this extra spec is not present, then by default the use of sibling pairs by the VM is not enforced. This is equivalent to the **prefer** setting.

The following settings are available for the **CPU Thread Policy**:

Isolate

When you add the **CPU Thread Policy** extra spec, the **Isolate** policy is offered. This policy enforces the use of only one hyperthreaded sibling in each dedicated core, optimizing processor cycles.

Require

You can change the **CPU Thread Policy** to **Require** to enforce the use of both siblings by the VM, optimizing resource allocations. For instantiation, a host that supports hyperthreading must be available.

Prefer

You can change the **CPU Thread Policy** to **Prefer** to prefer the use of siblings if available. Otherwise the VM can use non-sibling threads.

NOTE:

- The **CPU Thread Policy** is incompatible with the **Shared VCPU ID** extra specification. Do not use both in the same flavor.
- The **Require** setting for the **CPU Thread Policy** effectively causes the VM to require threads in multiples of two. It is therefore not compatible with CPU scaling, which assumes increments of one.

Specifying CPU Policies as Flavor Extra Specs

You can specify dedicated or shared CPU policies for a flavor using extra specifications.

The **CPU Policy** and **CPU Thread Policy** selections in the **Create Flavor Extra Spec** drop-down menu control these policies. To access this menu, see Flavor Extra Specifications on page 14.

Specifying CPU Policies as Flavor Extra Specs Using the CLI

You can specify dedicated or shared CPU policies for a flavor using the CLI.

To add the **CPU Policy** extra spec to a flavor using the CLI, use the following command:

```
~(keystone_admin)$ nova flavor-key flavor_name set hw:cpu_policy=policy
```

where *policy* is either **dedicated** or **shared**. If any other value is supplied, the policy is set to **dedicated**.

To add the **CPU Thread Policy** to a flavor using the CLI, use the following command:

```
~(keystone_admin)$ nova flavor-key flavor_name set hw:cpu_thread_policy=thread_policy
```

where *thread_policy* is either **isolate**, **require**, or **prefer**. Any other value is rejected with an error message.

Specifying CPU Policies as Image Metadata

You can specify dedicated or shared CPU policies for an image by adding metadata.

You can use the following commands:

```
~(keystone_admin)$ nova image-meta image_name set hw_cpu_policy=policy
```

where *policy* is either **dedicated** or **shared**. Other values are ignored.

```
~(keystone_admin)$ nova image-meta image_name set hw_cpu_thread_policy=thread_policy
```

where *thread_policy* is either **isolate**, **require**, or **prefer**. Other values are ignored.

You can update the CPU policies for an image using the following commands:

```
~(keystone_admin)$ glance image-update --property hw_cpu_policy=policy image_name
```

```
~(keystone_admin)$ glance image-update --property hw_cpu_thread_policy=thread_policy
image_name
```

Specifying the VCPU Model for a VM

You can select a particular VCPU model for a VM in order to leverage advanced CPU features such as SSE4.2, AES, or AVX on the compute nodes.

When a virtual machine is launched, a Nova scheduler filter restricts the target compute nodes to those with available cores of the requested model or better. If no such compute node is available, the error *No valid host was found* is reported.

If no **VCPU Model** extra spec is added, then by default the QEMU virtual processor is used.

The following selections are available:

- Intel Core i7 9xx (Nehalem Class Core i7)

  (This is offered by default when the extra spec is added using the web administration interface.)
- Intel Westmere E56xx/L56xx/X56xx (Nehalem-C)
- Intel Xeon E312xx (Sandy Bridge)
- Intel Core Processor (Haswell)
- Intel Core Processor (Broadwell-noTSX)
- Intel Core Processor (Broadwell)

NOTE: For some selections, limitations apply:

- The default QEMU processor is based on the Intel Core Duo CPU, which does not support extended instruction sets such as SSSE3 or SSSE4.2 used for DPDK networking or other advanced features.
- The Haswell model does not currently support transactional synchronization extensions (TSX).

To add this extra spec to a flavor using the web administration interface, use the **VCPU Model** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.

You can also specify the extra spec from the CLI using the following command:

```
~(keystone_admin)$ nova flavor-key flavor_name set hw:cpu_model=cpu_model
```

where *cpu_model* is one of the following:

- **Nehalem**
- **Westmere**
- **SandyBridge**
- **Haswell**
- **Broadwell-noTSX**
- **Broadwell**

NOTE: If any other value is supplied, it is ignored and the default QEMU model is used.

If the **hw:cpu_model** parameter is not supplied with the **nova flavor-key** command, then the default QEMU model is used.

Pinning a vCPU to a Shared Physical CPU

You can pin a vCPU to a shared physical CPU by using a flavor with the required extra specification.

You can set up a shared physical CPU on the host to run low-load or non-real-time tasks for multiple VMs, freeing other cores on the host for dedicated high-load tasks. You can then use an extra spec to pin a specified vCPU to the shared physical CPU.

To add the required extra specification to a flavor using the web administration interface, use the **Shared VCPU ID** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.

To set the Shared VCPU ID from the CLI, use a command of the following form:

```
~(keystone_admin)$ nova flavor-key flavor_name set hw:wrs:shared_vcpu=vcpu_id
```

where **vcpu_id** is an integer that identifies the vcpu. The valid range starts at 0 and must be less than the **VCPUs** value defined for the flavor.

Prerequisites

To use this extra spec, you must define a shared physical CPU on at least one host. For more information, see *Helion Carrier Grade 4.0 System Administration: Designating Shared Physical CPUs on a Compute Host*. To support migration, shared physical CPUs on multiple hosts are recommended.

You must also set the **CPU Policy** extra specification for the flavor to **Dedicated**. This enables the high-load vCPUs for the instance to be pinned.

NOTE: The **CPU Thread Policy** is incompatible with the **Shared VCPU ID** extra specification. Do not use both in the same flavor.

Configuring vCPU Scheduling and Priority

You can assign the Linux scheduler and priority for non-boot virtual CPUs using extra specifications.

Two methods are supported. You cannot use both in the same flavor.

VCPU Scheduler Policy

You can use this method to configure the scheduling priority and time-sharing algorithm for individual vCPUs. This option is available from the web administration interface or the CLI.

VCPU Real Time Policy

You can use this method to assign real-time privileges to one or more vCPUs. This option is available only from the CLI.

Both methods require the **CPU Policy** to be set to **Dedicated**. For more about the **CPU Policy**, see Specifying Host CPU Threading Policies for a VM on page 18.

Both methods lock guest RAM into memory. For improved engineerability, the Helion Carrier Grade 4.0 implementation does not lock lock the entire QEMU memory.

NOTE: These extra specifications apply to non-boot virtual CPUs only. For the boot CPU, the Linux scheduler and priority are fixed to normal time-shared scheduling policy with a *nice* priority of 0.

VCPU Scheduler Policy

This extra spec is shown only for flavors with more than one CPU.

For each additional virtual CPU, the available options are:

Default Policy

Assigns a normal time-shared scheduling policy with *nice* priority of 0.

Real-Time FIFO

Assigns a real-time, first-in-first-out policy, with priority in the range of 1–99, specified in the associated **VCPU Priority** field.

Real-Time Round Robin

Assigns a real-time, round-robin policy, with priority of 1–99, specified in the associated **VCPU Priority** field.

NOTE: The scheduling policy and priority level set by this extra spec become effective when the virtual CPU thread is scheduled to run on dedicated cores. Virtual CPU threads pinned to run on shared-only CPUs always use the default policy of time-shared scheduling with *nice* priority of 0. See Pinning a vCPU to a Shared Physical CPU on page 21 for details.

To add this extra spec to a flavor the flavor must first be configured with two or more VCPUs. Use the web administration interface to select **VCPU Scheduler Policy** in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.

You can specify the scheduler policy and priority for non-boot virtual CPUs from the CLI using the **hw:wrs:vcpu:scheduler** parameter on the **nova flavor-key** command. This parameter accepts a semicolon-separated list of *scheduler:priority:vcpus* values enclosed by quotes, as follows.

scheduler

The scheduler policy. One of *other*, *fifo*, or *rr* to indicate normal time-shared, FIFO, and Round Robin policies respectively.

priority

The real-time scheduler priority. A value between 1 and 99.

vcpus

An optional list of virtual CPUs as a comma-separated list (1,2,3), or a range specification (1-3). Virtual CPU number 0 refers to the boot virtual CPU and therefore cannot be used.

To set real-time schedulers and priorities on virtual CPU 1 (FIFO, 50) and virtual CPU 2 (Round Robin, 80):

```
~(keystone_admin)$ nova flavor-key flavor_name set hw:wrs:vcpu:scheduler="fifo:50:1;rr:
80:2"
```

To set real-time Round Robin schedulers for three virtual CPU (Round Robin, 80):

```
~(keystone_admin)$ nova flavor-key flavor_name set hw:wrs:vcpu:scheduler=rr:80:1-3
```

To set the FIFO scheduler with priority 50 on all virtual CPUs except virtual CPU 0:

```
~(keystone_admin)$ nova flavor-key flavor_name set hw:wrs:vcpu:scheduler=fifo:50
```

To reset all scheduler settings to default values (non real-time scheduler with priority 0):

```
~(keystone_admin)$ nova flavor-key flavor_name unset hw:wrs:vcpu:scheduler
```

VCPU Real Time Policy

Helion Carrier Grade 4.0 supports the OpenStack **hw:cpu_realtime** extra specification for assigning a real-time FIFO policy to selected vCPUs. For the OpenStack documentation, see https://specs.openstack.org/openstack/nova-specs/specs/mitaka/implemented/libvirt-real-time.html.

Unlike the OpenStack implemenation, the Helion Carrier Grade 4.0 implementation locks only the guest RAM, not the entire QEMU process. This ensures that resource allocations can be fully managed by the system at all times.

To enable this policy on a flavor, use the following command:

```
~(keystone_admin)$ nova flavor-key flavor_name set hw:cpu_realtime=yes
```

You must exclude at least one VCPU from the policy to provide for housekeeping processes. To exclude vCPUs 0 and 1 from the policy, use the following command:

```
~(keystone_admin)$ nova flavor-key flavor_name set hw:cpu_realtime_mask=^0-1
```

Setting the CPU Scaling Range

You can define the maximum and minimum CPU resources available to an instance by using a flavor.

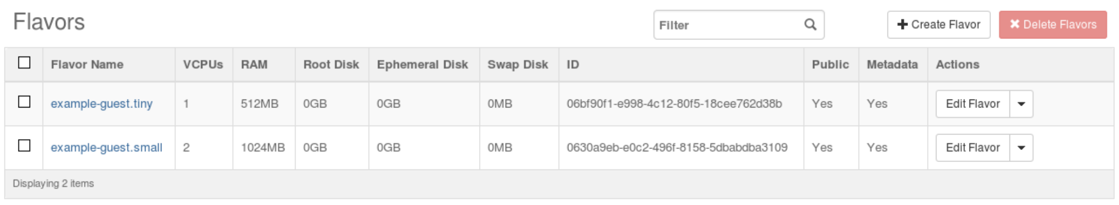
Prerequisites

For CPU scaling to take effect, the **CPU Policy** for the VM must be set to **Dedicated**. For more information, see Specifying Host CPU Threading Policies for a VM on page 18.

Procedure

1. Display the Flavors list.

   Select the **Admin** menu in the web administration interface, and then in the **System Panel** section of the menu, click **Flavors**.



☐	Flavor Name	VCPUs	RAM	Root Disk	Ephemeral Disk	Swap Disk	ID	Public	Metadata	Actions
☐	example-guest.tiny	1	512MB	0GB	0GB	0MB	06bf90f1-e998-4c12-80f5-18cee762d38b	Yes	Yes	Edit Flavor
☐	example-guest.small	2	1024MB	0GB	0GB	0MB	0630a9eb-e0c2-496f-8158-5dbabdba3109	Yes	Yes	Edit Flavor

Displaying 2 items

2. Optional: Select a suitable flavor.

   You can edit it to specify the maximum and minimum vCPUs.

3. Specify the maximum number of vCPUs for the instance.

   In the Flavors list, locate the flavor, and then click **Edit** to open the Edit Flavor dialog box.

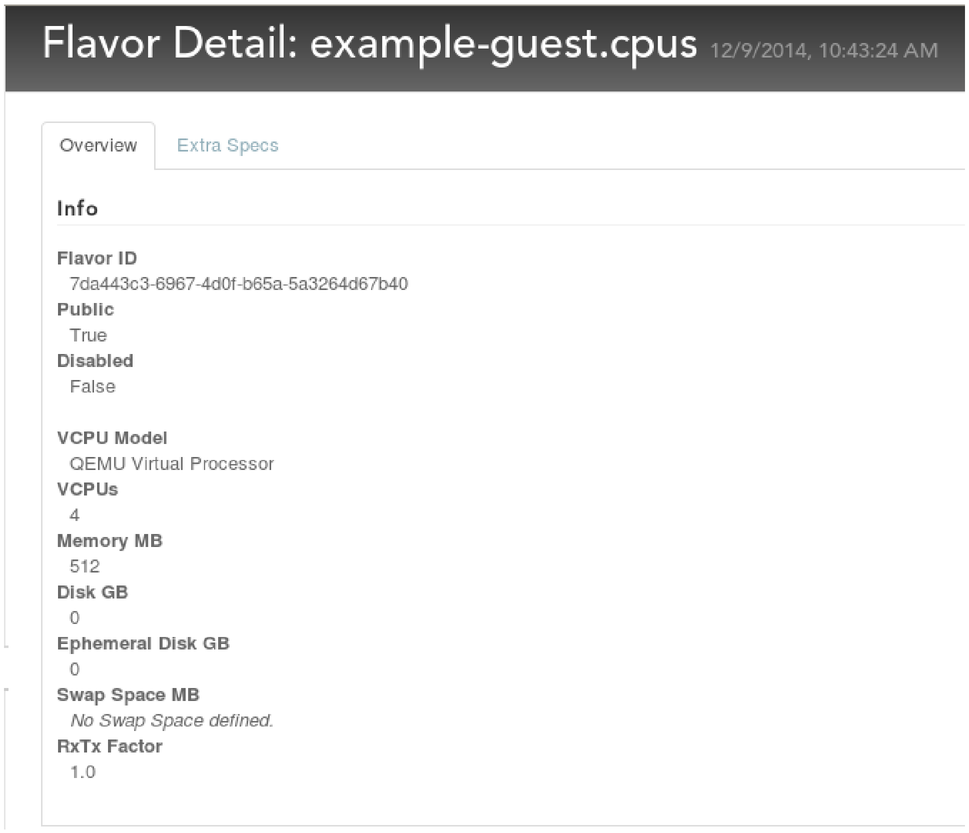
   In the **vCPU** field, enter the maximum number of vCPUs.

   When you are finished editing, click **Save** to return to the Flavors list.

4. Specify the minimum number of vCPUs for the instance.

   You can specify the minimum number of vCPUs by adding an Extra Spec for the flavor.

   a) In the Flavors list, click the **Flavor Name** for the flavor to open the Flavor Details dialog box.



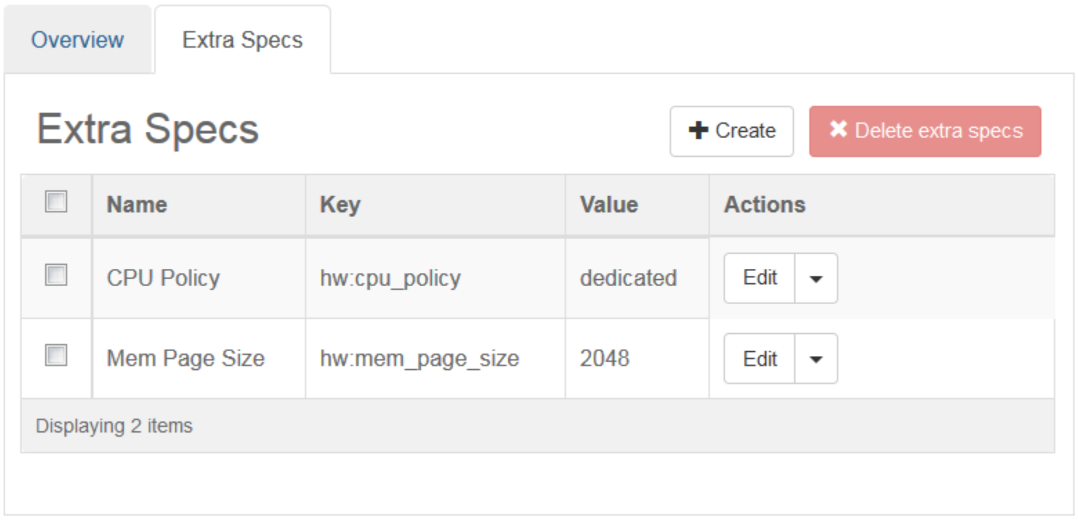
Flavor Detail: example-guest.cpus 12/9/2014, 10:43:24 AM

Info

Flavor ID
7da443c3-6967-4d0f-b65a-5a3264d67b40
Public
True
Disabled
False

VCPU Model
QEMU Virtual Processor
VCPUs
4
Memory MB
512
Disk GB
0
Ephemeral Disk GB
0
Swap Space MB
No Swap Space defined.
RxTx Factor
1.0

b) On the **Extra Specs** tab, click **Create**.

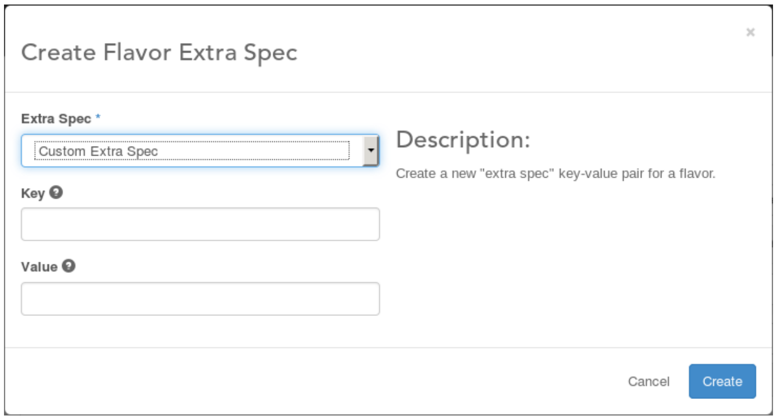


Extra Specs

	Name	Key	Value	Actions
☐	CPU Policy	hw:cpu_policy	dedicated	Edit
☐	Mem Page Size	hw:mem_page_size	2048	Edit

Displaying 2 items

c) In the **Create Flavor Extra Spec** dialog, select **Minimum Number of CPUs** from the **Extra Spec** drop-down menu.


Create Flavor Extra Spec

Extra Spec *

Custom Extra Spec

Key

Value

Description:

Create a new "extra spec" key-value pair for a flavor.

Cancel Create

d) In the **Key** field, enter wrs:min_vcpus.

e) In the **Value** field, enter the minimum allowed number of vCPUs for the flavor.

f) Click **Create**.

Setting the CPU Scaling Range Using the CLI

You can define the maximum and minimum CPU resources available to an instance by using a flavor using the CLI.

- To set the maximum number of VCPUs for an instance, define a flavor with the required number of vCPUs.
- To set the minimum number of vCPUs, edit the flavor to include an Extra Spec. The minimum cannot be less than one.

You can use the web administration interface or the CLI to edit the flavor. The CLI parameter for setting the minimum number of CPUs is as follows:

```
hw:wrs:min_vcpus=min
```

where **min** is the minimum number of CPUs.

For example:

```
~(keystone_admin)$ nova flavor-key flavor_name set hw:wrs:min_vcpus=integer_value
```

For complete information about working with flavors, see Virtual Machine Flavors on page 12.

NUMA Node Affinity Management

By default, the memory and CPU resources defined for a flavor are assigned to a single virtual NUMA node, which is mapped to an available host NUMA node when an instance is launched or migrated.

You can specify the use of multiple NUMA nodes using either flavor extra specifications (which take the general form **hw:specification**), or image properties (which take the general form **hw_specification**).

If you specify the use of multiple NUMA nodes, then by default, the available CPU and memory resources for the flavor are distributed equally among the NUMA nodes. You can customize the distribution of resources. For example, given a flavor with two NUMA nodes, three vCPUs, and 1024 MB of RAM, you can assign one vCPU and 512 MB to virtual NUMA node 0, and two VCPUs and 512 MB to virtual NUMA node 1. The ability to allocate resources is useful when pinning virtual NUMA nodes to host NUMA nodes to optimize VM performance.

Normally the memory required for a virtual NUMA node is pinned to a single host NUMA node to ensure high performance. For applications where high performance is not a concern, you can relax the memory allocation requirements so that the memory for a virtual NUMA node can be drawn from more than one host NUMA node if necessary.

When deploying network guest images operating on the data path, it is advisable to co-locate the virtual machines, the AVS switch, physical ports, and all other networking elements on the same node.

Use of this option should be limited to cases where fine-tuning of the data path on guest applications is important.

⚠ **CAUTION:** When this extra spec is used, an eligible host NUMA node is required for each virtual NUMA node in the instance. If this requirement cannot be met, the instantiation fails. For more information, see *Helion Carrier Grade 4.0 Cloud Operation: The Virtual Machine Scheduler*.

Configuring the NUMA Node Allocations for a VM

You can use flavor extra specs or image properties to allocate virtual memory and vCPU resources to virtual NUMA nodes.

Procedure

1. Declare the number of NUMA nodes to create on the guest.

   The following extra specification sets the number of NUMA nodes:

   ```
   hw:numa_nodes=n
   ```

   where *n* is the number of NUMA nodes to create (1 or more).

   For example, given the flavor **numa.pinned.asym**, use the following command to create two NUMA nodes.

   ```
   ~(keystone_admin)$ nova flavor-key numa.pinned.asym set hw:numa_nodes=2
   ```

   To set an image property instead, you can use the following command:

   ```
   ~(keystone_admin)$ nova image-meta image hw_numa_nodes=2
   ```

2. Optional: For each virtual NUMA node, assign a list of CPUs.

   This step is optional. By default, available CPUs for the flavor are distributed evenly across the available virtual NUMA nodes.

   The following extra specification assigns CPUs to a virtual NUMA node:

   ```
   hw:numa_cpus.vnode_id=cpu_list
   ```

   where

vnode_id

is the number used to identify the virtual NUMA node (0 for the first node, 1 for the second, and so on).

cpu_list

is a comma-separated list of vCPUs to assign to the node. The vCPUs for the flavor are enumerated beginning with 0.

For example, given flavor **numa.pinned.asym**, use the following command to assign vCPU 0 to the first virtual NUMA node, and vCPUs 1 and 2 to the second virtual NUMA node:

```
~(keystone_admin)$ nova flavor-key numa.pinned.asym set hw:numa_cpus.0=0 \
hw:numa_cpus.1=1,2
```

3. Optional: For each virtual NUMA node, assign an amount of memory.

   This step is optional. By default, available memory for the flavor is distributed evenly across the available virtual NUMA nodes.

   The following extra specification assigns memory to a virtual NUMA node:

   ```
   hw:numa_mem.vnode_id=ram_size
   ```

   where

   vnode_id

   is the number used to identify the virtual NUMA node (0 for the first node, 1 for the second, and so on).

   ram_size

   is the amount of RAM in MB.

   For example, given flavor **numa.pinned.asym**, use the following command to assign 512 MB of RAM to each of two virtual NUMA nodes:

   ```
   ~(keystone_admin)$ nova flavor-key numa.pinned.asym set hw:numa_mem.0=512 \
   hw:numa_mem.1=512
   ```

4. Optional: Specify whether memory for the flavor can be drawn from more than one host NUMA node if necessary.

   This step is optional. By default, memory is allocated from the designated host NUMA node only.

   To control whether memory for the flavor can be drawn from more than one host NUMA node, use the following extra specification:

   ```
   hw:numa_mempolicy=policy
   ```

   where **policy** is either **strict** (to use only memory from the designated host NUMA node) or **preferred** (to permit memory from other host NUMA nodes to be used if necessary).

Postrequisites

To pin the virtual NUMA nodes to host NUMA nodes, see Pinning a Guest NUMA Node to a Host NUMA Node on page 29.

Viewing the NUMA Node Configuration for a VM

You can use the CLI to display the NUMA node configuration for a VM.

Use the following command:

```
~(keystone_admin)$ nova show instance
```

where **instance** is the name or UUID of the instance.

Specifying Best Effort for PCI NUMA Node Affinity

You can select whether to require direct NUMA-node access to a PCI interface when instantiating a VM.

By default, when a VM is configured to use an SR-IOV or PCI-passthrough interface, the VM is affined to a NUMA node that has direct hardware access to the physical PCI interface. If none is available, the instantiation fails.

You can set an extra spec so that if the requirement cannot be satisifed, the VM is instantiated on another NUMA node with mediated access to the interface.

To add this extra spec to a flavor using the web administration interface, use the **PCI NUMA Affinity** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.

You can also specify the extra spec from the CLI by setting the following parameter for a flavor:

```
hw:wrs:pci_numa_affinity=pci_affinity
```

where *pci_affinity* is one of the following:

strict

(default) Requires a host NUMA node with direct access to a PCI interface. For a VM configured to use multiple NUMA nodes, at least one host NUMA node must have direct PCI access. If these conditions cannot be satisfied, the instantiation fails.

prefer

Prefers a host NUMA node with direct access to a PCI interface. If none is available, another NUMA node can be used.

For more information about using PCI interfaces, see Configuring PCI Passthrough Ethernet Interfaces on page 41 and Configuring SR-IOV Ethernet Interfaces on page 46.

Pinning a Guest NUMA Node to a Host NUMA Node

You can use flavor extra specs or image properties to pin a guest NUMA node to a host NUMA node.

By default, when instances are launched or migrated, the virtual NUMA nodes defined for the VMs are mapped to available host NUMA nodes. You can optionally designate a specific host NUMA node for a virtual NUMA node, using either a flavor extra specification (which takes the general form **hw:specification**), or an image property (which takes the general form

hw_specification). This enables you to co-locate VM processes with AVS vSwitch processes for high-performance networking.

For information about assigning vSwitch processes to host NUMA nodes, see *Helion Carrier Grade 4.0 System Administration: Processor Tab*.

For a VM with only one virtual NUMA node, you can use an extra specification to specify the host NUMA node. To add this extra spec to a flavor using the web administration interface, use the **NUMA Node** selection in the **Create Flavor Extra Spec** drop-down menu. This provides fields to associate a **Guest NUMA Node** with a **Host NUMA Node**. To access this menu, see Flavor Extra Specifications on page 14.

You can also pin NUMA nodes using the CLI. The following extra specification assigns a specific host NUMA node to a virtual NUMA node:

```
hw:numa_node.vnode_id=pnode_id
```

where

vnode_id

is the number used to identify the virtual NUMA node (0 for the first node, 1 for the second, and so on).

pnode_id

is the number used to identify the host NUMA node (0 for the first node, 1 for the second, and so on).

For example, given flavor **numa.pinned.asym**, use the following command to assign virtual NUMA node 0 to host NUMA node 1:

```
~(keystone_admin)$ nova flavor-key numa.pinned.asym set hw:numa_node.0=1
```

CAUTION: When this extra spec is used, an eligible host NUMA node is required for each virtual NUMA node in the instance. If this requirement cannot be met, the instantiation fails. For more information, see *Helion Carrier Grade 4.0 Cloud Operation: The Virtual Machine Scheduler*.

Affining a VM to a NUMA Node with a vSwitch Core

You can select whether to place a VM in the same NUMA node as a vSwitch (AVS) core.

For optimal networking performance, you can set an extra spec so that a VM is instantiated on a host only if the host can offer a NUMA node with at least one core assigned as a vSwitch. You can optionally specify best-effort placement, so that if a suitable host is unavailable, the VM is still instantiated, but using a NUMA node without a vSwitch core.

To add this extra spec to a flavor using the web administration interface, use the **Vswitch NUMA Affinity** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.

You can also specify the extra spec from the CLI by setting the following parameter for a flavor:

```
hw:wrs:vswitch_numa_affinity=vswitch_affinity
```

where *vswitch_affinity* is one of the following:

strict

Requires a host NUMA node that has a vSwitch core. For a VM configured to use multiple NUMA nodes, each host NUMA node must have a vSwitch core. If these conditions cannot be satisfied, the instantiation fails.

prefer

Prefers a host NUMA node that has a vSwitch core. For a VM configured to use multiple NUMA nodes, prefers host NUMA nodes with a vSwitch core for as many virtual NUMA nodes as possible. If these conditions cannot be satisfied, another NUMA node can be used.

For information about assigning vSwitch cores on a host, see *Helion Carrier Grade 4.0 System Administration: Processor Tab*.

⚠ **CAUTION:** When this extra spec is used, an eligible host NUMA node is required for each virtual NUMA node in the instance. If this requirement cannot be met, the instantiation fails. For more information, see *Helion Carrier Grade 4.0 Cloud Operation: The Virtual Machine Scheduler*.

Ensuring Optimal vSwitch Processing of Physical Ports

To ensure optimal vSwitch processing of physical ports, you can align the vSwitch NUMA node presence with physical-port NUMA node connectivity.

By default, each vSwitch core on the host services all physical ports on the host, including those connected to other NUMA nodes. In this configuration, traffic can cross the QPI bus between the NUMA nodes, presenting a bottleneck that can decrease overall performance.

To address this, Helion Carrier Grade 4.0 supports a split configuration, in which each AVS core services only the physical ports in the same NUMA node. This split configuration is automatically used on a compute host if and only if the following conditions are met:

- Every NUMA node with a vSwitch core has at least one data interface.
- Every NUMA node with a data interface has at least one vSwitch core.

Procedure

1. Identify the processors associated with data-interface ports on the compute host.

   The **system host-port-list** command shows the processor associated with each port.

```
~(keystone_admin)$ system host-port-list compute-0
+-----+--------+----------+-------------+--------+-----------+...
| uuid | name   | type     | pci address  | device | processor |...
+------+--------+----------+-------------+--------+-----------+...
| 4d...| enp0s3 | ethernet | 0000:00:03.0 | 0      | -1        |...
| 15...| enp0s8 | ethernet | 0000:00:08.0 | 0      | -1        |...
| ef...| eth0   | ethernet | 0000:00:09.0 | 0      | -1        |...
| ac...| eth1   | ethernet | 0000:00:0a.0 | 0      | -1        |...
+------+--------+----------+-------------+--------+-----------+...
...
```

2. For each processor identified, ensure that at least one vSwitch core is allocated.

   The system host-cpu-list command shows which cores are currently assigned as vSwitch cores.

```
~(keystone_admin)$ system host-cpu-list compute-0
```

```
+------+----------+-----------+----------+--------+...+-------------------+
| uuid | log_core | processor | phy_core | thread |...| assigned_function |
+------+----------+-----------+----------+--------+...+-------------------+
| 81...| 0        | 0         | 0        | 0      |...| Platform          |
| 05...| 1        | 0         | 1        | 0      |...| Platform          |
| 90...| 2        | 0         | 2        | 0      |...| vSwitch           |
| 18...| 3        | 0         | 3        | 0      |...| vSwitch           |
| 39...| 4        | 0         | 4        | 0      |...| Shared            |
| 52...| 5        | 0         | 8        | 0      |...| VMs               |
| 0a...| 6        | 0         | 9        | 0      |...| VMs               |
| fb...| 7        | 0         | 10       | 0      |...| VMs               |
| 05...| 8        | 0         | 11       | 0      |...| VMs               |
| a0...| 9        | 0         | 12       | 0      |...| VMs               |
| cd...| 10       | 1         | 0        | 0      |...| VMs               |
| c9...| 11       | 1         | 1        | 0      |...| VMs               |
| 10...| 12       | 1         | 2        | 0      |...| VMs               |
| 6c...| 13       | 1         | 3        | 0      |...| VMs               |
| 26...| 14       | 1         | 4        | 0      |...| VMs               |
| af...| 15       | 1         | 8        | 0      |...| VMs               |
| 7c...| 16       | 1         | 9        | 0      |...| VMs               |
| b0...| 17       | 1         | 10       | 0      |...| VMs               |
| 9e...| 18       | 1         | 11       | 0      |...| VMs               |
| 62...| 19       | 1         | 12       | 0      |...| VMs               |
+------+----------+-----------+----------+--------+...+-------------------+
```

In this example, processor 0 has two vSwitch cores. However, processor 1 does not have a vSwitch core. To ensure a split configuration on this host, you must assign at least one core from processor 1 as a vSwitch core.

To assign vSwitch cores, see *Helion Carrier Grade 4.0 System Administration: Processor Tab*.

3. Ensure that no vSwitch cores are assigned for any processors not associated with data-interface ports.

Specifying a Page Size for a VM

You can request a specific memory page size for a VM by using a flavor with the required extra spec, or by defining an image with the required metadata property.

Prerequisites

Host memory on compute nodes reserved for use by VMs is partitioned by default using only 2 MiB huge page blocks. These blocks are used by default as the backing mechanism for VM virtual memory allocation requests. See *Helion Carrier Grade 4.0 System Administration: Host Memory Provisioning* for details on how to modify the memory partitioning scheme for a host.

CAUTION: When this extra spec is used, an eligible host NUMA node is required for each virtual NUMA node in the instance. If this requirement cannot be met, the instantiation fails. For more information, see *Helion Carrier Grade 4.0 Cloud Operation: The Virtual Machine Scheduler*.

Memory requested by a guest is allocated as one or more pages of a specified size. Once allocated, the memory is unavailable for use by other guests until the instance is terminated.

To add this extra spec to a flavor using the web administration interface, use the **Memory Page Size** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.

You can also specify the extra spec from the CLI by setting the following parameter for a flavor:

```
hw:mem_page_size=pagesize
```

where *pagesize* is one of the following:

small

Requests the smallest available size on the compute node, which is always 4KiB of regular memory.

large

Requests the largest available huge page size, 1GiB or 2MiB.

any

Requests any available size, including small pages. Helion Carrier Grade 4.0 uses the largest available size, 1GiB, then 2MiB, and then 4KiB.

2048

Requests a huge page of size 2 MiB. This is the default value when the extra spec is not used.

1048576

requests a huge page of size 1GiB.

If any other value is supplied, it is ignored and the default value of **2048** is used.

It is suggested that NFV applications use explicit huge page sizes to ensure predictable memory access at run time. Use of **large** and **any** sizes may cause VM migration issues when the available page size on the destination node is different.

For example, to set a 1 GiB huge page size on a flavor that has already been created, use the following command:

```
~(keystone)admin)$ nova flavor-key flavor_name set hw:mem_page_size=1048576
```

You can also define an image with the required property by using the **hw_mem_page_size** parameter to include image metadata, as in the following example:

```
~(keystone)admin)$ nova image-meta image set hw_mem_page_size=pagesize
```

NOTE: If you use image metadata to specify a memory page size, the flavor extra specification must be set to **large** or **any**.

Specifying the Storage Type for VM Ephemeral Disks

You can specify the ephemeral storage type for virtual machines (VMs) by using a flavor with the appropriate extra specification.

Each new flavor is automatically assigned a Storage Type extra spec that specifies, as the default, instantiation on compute hosts configured for image-backed local storage (**Local CoW Image Backed**). You can change the extra spec to specify instantiation on compute hosts configured for LVM-backed local storage (**Local LVM Backed**) or Ceph-backed remote storage, if this is available (**Remote Storage Backed**). Ceph-backed remote storage is available only on systems configured with a Ceph storage backend.

The designated storage type is used for ephemeral disk and swap disk space, and for the root disk if the virtual machine is launched using boot-from-image. Local storage is allocated from the Local Volume Group on the host, and does not persist when the instance is terminated. Remote storage is allocated from a Ceph storage pool configured on the storage host resources, and persists until the pool resources are reallocated for other purposes. The choice of storage type affects migration behavior; for more information, see *Helion Carrier Grade 4.0 System Administration: VM Storage Settings for Migration, Resize, or Evacuation*.

If the instance is configured to boot from volume, the root disk is implemented using persistent Cinder-based storage allocated from the controller (for a system using LVM) or from storage hosts (for a system using Ceph).

To specify the type of storage offered by a compute host, see *Helion Carrier Grade 4.0 System Administration: Managing Local Volume Groups*.

⚠ **CAUTION:** Unlike Cinder-based storage, ephemeral storage does not persist if the instance is terminated or the compute node fails.

In addition, for local ephemeral storage, migration and resizing support depends on the storage backing type specified for the instance, as well as the boot source selected at launch.

To change the storage type using the Web administration interface, click **Edit** for the existing **Storage Type** extra specification, and select from the **Storage** drop-down menu. To access the extra specification, see *Helion Carrier Grade 4.0 Cloud Administration: Flavor Extra Specifications on page 14*.

⚠ **CAUTION:** If the **Storage Type** extra spec is not specified, the VM can be instantiated on any host. To ensure that instantiated VMs use storage of a known type, do not delete this extra spec.

You can also specify the extra spec from the CLI by setting the following parameter for a flavor:

```
aggregate_instance_extra_specs:storage=storage_type
```

where *storage_type* is one of the following:

local_lvm

Specifies hosts with LVM-backed local storage for use by the VM.

local_image

Specifies hosts with image-backed local storage for use by the VM.

remote

Specifies hosts with Ceph-backed remote storage for use by the VM.

For example:

```
~(keystone_admin)$ nova flavor-key flavor_name \
set aggregate_instance_extra_specs:storage=local_image
```

The local storage key is added by default on flavor creation and set for **local_image** storage.

Enabling the Guest Heartbeat API for a VM

You can accommodate the use of guest heartbeats on a VM using an extra specification.

Select this option when you expect one or more of the guest applications running on the virtual machine to make use of the Helion Carrier Grade 4.0 Guest Heartbeat API. If no extra spec is added, then by default the API is disabled.

If this option is selected, the controller node starts heartbeat application-level polling cycles on virtual machines launched using the flavor. For more information about the Guest Heartbeat API, refer to the *Helion Carrier Grade 4.0 Software Development Kit*.

A guest application modified to use the Titanium Guest Server Heartbeat API can be more accurately monitored by internal messaging within the virtual machine. For more about application monitoring, see *Helion Carrier Grade 4.0 VNF Integration: Virtual Machines and Carrier-Grade Availability*.

To add this extra spec to a flavor using the web administration interface, use the **Guest Heartbeat** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.

NOTE: The non-default value **True** is offered when the extra spec is added using the web administration interface.

To enable the Guest Heartbeat API using the CLI, use the following command:

```
~(keystone_admin)$ nova flavor-key flavor_name set sw:wrs:guest:heartbeat=value
```

where *value* is either **True** or **False**. If any other value is supplied, it is ignored and the default value **False** is used.

Enabling Server Group Messaging for a VM

You can enable a messaging API for a VM, for use with other VMs in the same server group.

If this extra spec is not added, then by default this API is disabled.

Server Group Messaging is a service that provides simple low-bandwidth datagram messaging and notifications for virtual machines that belong to the same server group. This message channel is available regardless of whether IP networking is functional within the server, and requires no knowledge about the other members in the group.

Guest applications can access this service if the Server Group Messaging API is enabled for the VM. To enable the API, select this option. For more information about the Server Group Messaging API, refer to the *Helion Carrier Grade 4.0 Software Development Kit*.

NOTE: The non-default value **True** is offered when the extra spec is added.

The service provides three types of messaging:

- **Broadcast**—allows the server to send a datagram of up to 3050 bytes to all other servers in the server group.
- **Notification**—provides servers with information about changes to the state of other servers in the server group.
- **Status**—allows the server to query the current state of all servers in the group (including the originating server).

CAUTION: This service is not intended for high-bandwidth or low-latency operations. If reliability is an important consideration, use acknowledgements and retries.

To add this extra spec to a flavor using the web administration interface, use the **Server Group Messaging** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.

To enable the Server Group Messaging API using the CLI, use the following command:

```
~(keystone_admin)$ nova flavor-key flavor_name set sw:wrs:srv_grp_messaging=value
```

where *value* is either **False** or **True**.

Disabling Instance Auto Recovery

You can disable auto recovery of failed virtual machine instances.

If this extra spec is not added, then the system will automatically recover failed virtual machine instances. Set this option to false if, for example, you are writing a Virtual Network Function Manager (VNFM) and the VNFM is to manage the auto-recovery of its VMs. Setting it to false disables this behavior. To re-enable virtual machine instance auto recovery, set the value to true.

Instance auto-recovery can be configured as an extra spec on a flavor, such that any VM instance created with this flavor will have the specified instance auto-recovery. Alternatively, instance auto-recovery can be configured as meta-data on an image, such that any VM instance created based on the image will have the specified instance auto-recovery. Note that Flavor extra-specs will override the image meta-data.

To set the option using the web administration interface, do one of the following:

- As *admin*, use the **Instance Auto Recovery** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.
- As either *admin* or a tenant, select **System** > **Images** > **Edit Image** > for the desired images and uncheck the **Instance Auto Recovery** option.

Disabling Instance Auto Recovery Using the CLI

You can disable auto recovery of failed virtual machine instances using the CLI.

Procedure

To set the option from the command line do one of the following.

- As *admin*, add an extra spec to a flavor:

  ```
  ~(keystone)admin)$ nova flavor-key flavorName set sw:wrs:auto_recovery=value
  ```

  where *value* is **false** to disable instance auto recovery or **true** to enable it.
- As *admin* or a tenant:

  ```
  ~(keystone)admin)$ glance image-update --property sw_wrs_auto_recovery=value
  imageName
  ```

  where *value* is **false** to disable instance auto recovery or **true** to enable it.

Specifying a Live Migration Timeout

Live migrations can take a long time to complete for reasons such as high levels of guest activity, network latency, and so on. You can specify the number of seconds to wait for a live migration to complete.

If no value is set, the default is 800 seconds.

If the value is exceeded, the migration is canceled and a customer log entry is created. The counter starts when the migration begins. The minimum timeout value is 120 seconds and the maximum is 800 seconds. To disable the live migration timeout feature, set this value to 0 using the CLI.

NOTE: NOVA maintains an internal timer which guards a complete lack of VM instance live migration progress for longer than 150 seconds. If this timer expires, the live migration will be canceled; regardless of the value of the live migration maximum timeout.

A live migration timeout can be configured as an extra spec on a flavor, such that any VM instance created with the flavor will inherit the specified timeout value. Alternatively, a live migration timeout can be configured as meta-data on an image, such that any VM instance created based on this Image will have the specified timeout value. If the timeout value is provisioned in both the flavor and the image, the smaller value is used.

To set this option using the web administration interface:

- As *admin*, use the **Live Migration Timeout** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.

or

- As either *admin* or a tenant, select **System** > **Images** > **Edit Image** > **Update Metadata** for the desired images and add an entry with:

```
hw_wrs_live_migration_timeout=seconds
```

Specifying a Live Migration Timeout Using the CLI

You can specify the number of seconds to wait for a live migration to complete using the CLI

Procedure

Set the timer from the command line:

The following examples set the timeout to 400 seconds.

- As *admin*, add an extra spec to a flavor by:

```
~(keystone)admin)$ nova flavor-key flavorName set hw:wrs:live_migration_timeout=400
```

- As *admin* or a tenant, update the metadata of an image by:

```
~(keystone)admin)$ glance image-update --property hw_wrs_live_migration_timeout=400
imageName
```

Specifying Live Migration Maximum Downtime

You can change the maximum amount of downtime to tolerate during a live migration.

If this extra spec is not added, then the default 500 milliseconds is used. The minimum timer value is 100 milliseconds.

NOTE: Downtime during a live migration is the period during which the VM will be suspended in order to journal the final increment of data between source VM instance and destination VM instance, resulting in a temporary service interruption. Adjust this timer to avoid excessive down times that may violate service level agreements. When this threshold cannot be achieved within the *live_migration_timeout*, the migration is aborted. For more information about *live_migration_timeout*, see Specifying a Live Migration Timeout on page 37.

The live migration process will actually attempt to achieve a much smaller downtime than specified by this parameter by starting with a lower value approximately 1/10th of specified value, and increasing the downtime target every 75 seconds for a total of 10 attempts. Setting the Live Migration Maximum Timeout, as described in Specifying a Live Migration Timeout on page 37, lower than 750 seconds results in the actual maximum tolerated live migration downtime being shorter than specified by this parameter.

Live migration maximum downtime can be configured as an extra spec on a flavor, such that any VM Instance created with the flavor will inherit the specified value. Alternatively, it can be configured as meta-data on an image, such that any VM instance created based on this image will have the specified value. If live migration maximum downtime is set in both the flavor and the image, the value from the flavor overrides the value from the image.

To set this option using the web administration interface:

- As *admin*, use the **Live Migration Max Downtime** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.

or

- As either *admin* or a tenant, select **System** > **Images** > **Edit Image** > **Update Metadata** for the desired images and add an entry with:

```
hw_wrs_live_migration_max_downtime=milliseconds
```

Specifying Live Migration Maximum Downtime Using the CLI

You can change the maximum amount of downtime to tolerate during a live migration using the CLI.

Procedure

Set the timer from the command line:

The following examples set the timeout to 350 milliseconds.

- As *admin*, add an extra spec to a flavor:

```
~(keystone)admin)$ nova flavor-key flavorName set
hw:wrs:live_migration_max_downtime=350
```

- As *admin* or a tenant, update the metadata of an image:

```
~(keystone)admin)$ glance image-update --property
hw_wrs_live_migration_max_downtime=350 imageName
```

Isolating an Untrusted Guest

You can isolate the network resources used and accessible by a potentially untrusted virtual machine instance.

This ensures that the untrusted virtual machine cannot exhaust or corrupt shared network resources and impact the operation or performance of other virtual machines hosted on the same compute node. Note that network resource isolation comes at a slight degradation of network performance for this guest.

Set this option to true when you want to isolate the network resources used and accessible by an untrusted guest instance from all other instances running on the host. Set it to false to turn isolation off. If this extra spec is not set, then the instance's NICs use of network resources are not isolated.

To add this extra spec to a flavor using the web administration interface, use the **NIC Isolation** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14. This enables isolation of a virtual machine instance's NICs' use of network resources from all other physical or virtual interfaces on the host. If this extra spec is not set, then the instance's network interfaces are not isolated.

Prerequisites

To use this feature, the compute host must have sufficient 2 MB huge pages available for isolated NIC creation. For example, a guest with 3 NICs would require that 48 MB of 2 MB huge pages be available on its host. Isolated guests will only be launched on hosts that meet this requirement. For more information on using huge pages, see *Helion Carrier Grade 4.0 System Administration: Host Memory Provisioning*.

Procedure

Set this option using the CLI:

```
~(keystone_admin)$ nova flavor-key flavorName set hw:wrs:nic_isolation=true
```

Accessing a PCI Device from a VM

You can specify whether a VM has access to exposed PCI passthrough or PCI SR-IOV devices.

Prerequisites

To use PCI passthrough or SR-IOV devices, you must have Intel-VTx and Intel VT-d features enabled in the BIOS.

For PCI-passthrough or SR-IOV devices other than NICs that are exposed for VM use, you can manage access for individual VMs using **PCI Alias** extra specifications. To review the available advanced devices for a host, see *Helion Carrier Grade 4.0 System Administration: Devices Tab*. To expose a device, see *Helion Carrier Grade 4.0 System Administration: Exposing a Device for Use by VMs*.

NOTE: NIC devices that support PCI passthrough or SR-IOV are always exposed to VMs, and are managed and displayed separately. For more information, see Configuring PCI Passthrough Ethernet Interfaces on page 41 and Configuring SR-IOV Ethernet Interfaces on page 46.

If no **PCI Alias** extra spec is added for a device, then by default the device is not available to the guest. If a **PCI Alias** is added, then the Nova scheduler attempts to schedule the VM on a host containing the device. If no suitable compute node is available, the error **No valid host was found** is reported.

CAUTION: When this extra spec is used, an eligible host NUMA node is required for each virtual NUMA node in the instance. If this requirement cannot be met, the instantiation fails. For more information, see *Helion Carrier Grade 4.0 Cloud Operation: The Virtual Machine Scheduler*.

If a suitable compute node is available, then the scheduler attempts to instantiate the VM in a NUMA node with direct access to the device, subject to the **PCI NUMA Affinity** extra specification.

To add this extra spec to a flavor using the web administration interface, use the **PCI Alias** selection in the **Create Flavor Extra Spec** drop-down menu. To access this menu, see Flavor Extra Specifications on page 14.

You can also specify the extra spec from the CLI using a command of the following form:

```
~(keystone_admin)$ nova flavor-key flavor_name \
set pci_passthrough:alias=device_type[:number_of_devices]
```

where

flavor_name

is the name of the flavor

device_type

is the advanced PCI-passthrough or SR-IOV-capable hardware to expose to the VM

NOTE: The parameter **pci_passthrough:alias** is used to expose both PCI passthrough devices and SR-IOV devices.

The following options are available:

- **qat-vf**

  Exposes an Intel AV-ICE02 VPN Acceleration Card for SR-IOV access. For more information, see SR-IOV Encryption Acceleration on page 49.

  NOTE: Due to driver limitations, PCI passthrough access for the Intel AV-ICE02 VPN Acceleration Card (**qat-pf** option) is not supported.

number_of_devices

is the number of SR-IOV devices to expose to the VM

For example, to make two QuickAssist SR-IOV devices available to a guest:

```
~(keystone_admin)$ nova flavor-key flavor_name set pci_passthrough:alias=qat-vf:2
```

PCI Passthrough Ethernet Interface Devices

For all purposes, a PCI passthrough interface behaves as if it were physically attached to the virtual machine.

Therefore, any potential throughput limitations coming from the virtualized environment, such as the ones introduced by internal copying of data buffers, are eliminated. However, by bypassing the virtualized environment, the use of PCI passthrough Ethernet devices introduces several restrictions that must be taken into consideration. They include:

- no support for LAG, QoS, ACL, or host interface monitoring
- no support for live migration
- no access to the compute node's AVS switch

A passthrough interface bypasses the compute node's AVS switch completely, and is attached instead directly to the provider network's access switch. Therefore, proper routing of traffic to connect the passthrough interface to a particular tenant network depends entirely on the VLAN tagging options configured on both the passthrough interface and the access port on the switch.

The access switch routes incoming traffic based on a VLAN ID, which ultimately determines the tenant network to which the traffic belongs. The VLAN ID is either explicit, as found in incoming tagged packets, or implicit, as defined by the access port's default VLAN ID when the incoming packets are untagged. In both cases the access switch must be configured to process the proper VLAN ID, which therefore has to be known in advance.

⚠ **CAUTION:** On cold migration, a PCI passthrough interface receives a new MAC address, and therefore a new **eth**x interface. The IP address is retained.

To review the current PCI passthrough and SR-IOV interface assignments for a given provider network, see *Helion Carrier Grade 4.0 System Administration: Displaying Provider Network Information*.

In the following example a new virtual machine is launched by user **user1** on tenant **tenant1**, with a passthrough interface connected to the tenant network **net0** identified with VLAN ID 10.

Configuring PCI Passthrough Ethernet Interfaces

A *passthrough* Ethernet interface is a physical PCI Ethernet NIC on a compute node to which a virtual machine is granted direct access. This minimizes packet processing delays but at the same time demands special operational considerations.

Prerequisites

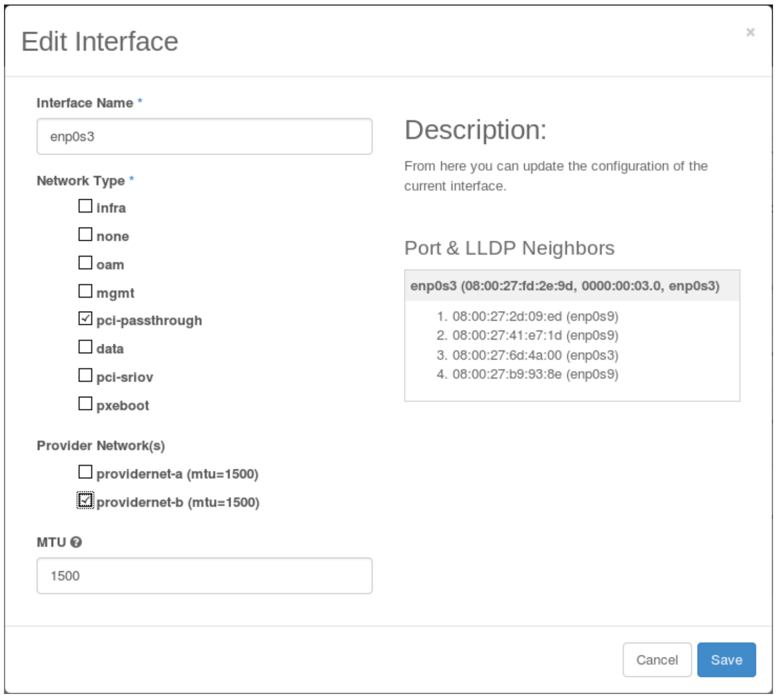
NOTE: To use PCI passthrough or SR-IOV devices, you must have Intel VT-x and Intel VT-d features enabled in the BIOS.

The exercise assumes that the underlying provider network **providernet-b** exists already, and that VLAN ID 10 is a valid segmentation ID assigned to **tenant1**.

Procedure

1. Log in as the **admin** user to the web management interface.
2. Lock the compute node you want to configure.
3. Configure the Ethernet interface to be used as a PCI passthrough interface.

   Select **Admin** > **Platform** > **Host Inventory** from the left-hand pane, click the **Hosts** tab, click the name of the compute node where the PCI interface is available, click the **Interfaces** tab, and finally click the **Edit Interface** button associated with the interface you want to configure. Fill in the window as illustrated below:



Edit Interface

Interface Name *
enp0s3

Network Type *
- [ ] infra
- [ ] none
- [ ] oam
- [ ] mgmt
- [✓] pci-passthrough
- [ ] data
- [ ] pci-sriov
- [ ] pxeboot

Provider Network(s)
- [ ] providernet-a (mtu=1500)
- [✓] providernet-b (mtu=1500)

MTU
1500

Description:
From here you can update the configuration of the current interface.

Port & LLDP Neighbors
enp0s3 (08:00:27:fd:2e:9d, 0000:00:03.0, enp0s3)
1. 08:00:27:2d:09:ed (enp0s9)
2. 08:00:27:41:e7:1d (enp0s9)
3. 08:00:27:6d:4a:00 (enp0s3)
4. 08:00:27:b9:93:8e (enp0s9)

   In this example, Ethernet interface **enp0s3** is assigned the network type *pci-passthrough*, and configured as connected to provider network **providernet-b**. Click the **Save** button to proceed.

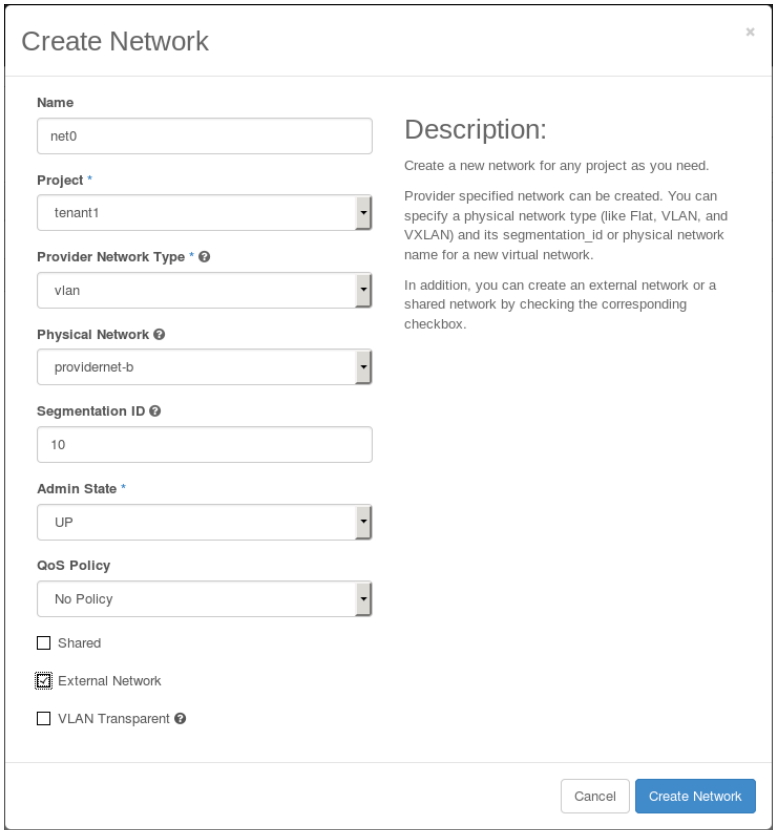
   The interface can also be configured from the CLI as illustrated below:

```
~(keystone_admin)$ system host-if-modify \
-nt pci-passthrough -p group0-data0 compute-0 enp0s3
```

4. Create the **net0** tenant network

   Select **Admin** > **System** > **Networks**, select the Networks tab, and then click **Create Network**. Fill in the Create Network dialog box as illustrated below. You must ensure that:

   - The **tenant1** tenant has access to the tenant network, either assigning it as the owner, as in the illustration (using **Project**), or by enabling the shared flag.
   - The segmentation ID is set to 10.



Create Network

Name: net0

Project *: tenant1

Provider Network Type *: vlan

Physical Network: providernet-b

Segmentation ID: 10

Admin State *: UP

QoS Policy: No Policy

☐ Shared

☑ External Network

☐ VLAN Transparent

Description:

Create a new network for any project as you need.

Provider specified network can be created. You can specify a physical network type (like Flat, VLAN, and VXLAN) and its segmentation_id or physical network name for a new virtual network.

In addition, you can create an external network or a shared network by checking the corresponding checkbox.

Click the **Create Network** button to proceed.

5. Configure the access switch.

   Configure the physical port on the access switch used to connect to Ethernet interface **enp0s3** as an access port with default VLAN ID of 10. Traffic across the connection is therefore untagged, and effectively integrated into the targeted tenant network.

   You can also use a trunk port on the access switch so that it handles tagged packets as well. However, this opens the possibility for guest applications to join other tenant networks using tagged packets with different VLAN IDs, which might compromise the security of the system. See *Helion Carrier Grade 4.0 Planning: L2 Access Switches* for other details regarding the configuration of the access switch.

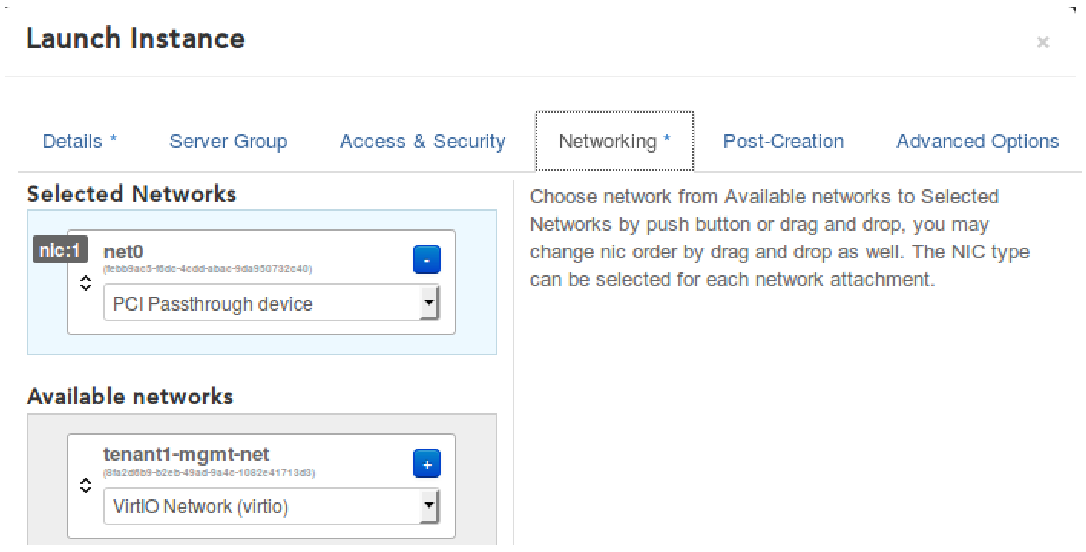
6. Unlock the compute node.

7. Launch the virtual machine.

   a) Log in as user **user1** to the web management interface.

   b) Configure the network interfaces for the new virtual machine.

      Open the Launch Instance dialog by clicking the **Launch Interface** button on the Instances window. Configure the necessary options for the new virtual machine. In particular, click the **Networking** tab and add a PCI passthrough interface on the tenant network **net0**, as illustrated below. Add other network interfaces as needed.



Selected Networks

net0 (febb9ac5-f6dc-4cdd-abac-9da950732c40)
PCI Passthrough device

Available networks

tenant1-mgmt-net (8fa2d6b9-b2eb-49ad-9a4c-1082e41713d3)
VirtIO Network (virtio)

Choose network from Available networks to Selected Networks by push button or drag and drop, you may change nic order by drag and drop as well. The NIC type can be selected for each network attachment.

Click the **Launch** button to proceed.

Passthrough interfaces can be attached from the CLI when booting a new virtual machine, as illustrated below:

```
~(keystone_admin)$ nova boot \
--nic net-id=704e9f3b,vif-model=pci-passthrough \
--flavor small --image cust-guest my-new-vm
```

NOTE: By default, a VM that requires a PCI passthrough interface is instantiated only if a host NUMA node with a directly attached PCI interface is available. You can specify best-effort instantiation by using an extra specification. For more information, see Specifying Best Effort for PCI NUMA Node Affinity on page 29.

The new virtual machine instance is up now. It has a PCI passthrough connection to the **net0** tenant network identified with VLAN ID 10.

Postrequisites

Access switches must be properly configured to ensure that virtual machines using PCI-passthrough or SR-IOV Ethernet interfaces (the latter discussed in Configuring SR-IOV Ethernet Interfaces on page 46) have the expected connectivity. In a common scenario, the virtual machine using these interfaces connects to external end points only, that is, it does not connect to other virtual machines in the same Helion Carrier Grade 4.0 cluster. In this case:

- Traffic between the virtual machine and the access switch can be tagged or untagged.
- The connecting port on the access switch is part of a port-based VLAN.
- If the port is tagged, the allowed VLAN ID range must not overlap with VLAN ID ranges used by the AVS ports.
- The port-based VLAN provides the required connectivity to external switching and routing equipment needed by guest applications to establish connections to the intended end points.

For connectivity to other virtual machines in the Helion Carrier Grade 4.0 cluster the following configuration is also required:

- The VLAN ID used for the tenant network, 10 in this example, and the default port VLAN ID of the access port on the switch are the same. This ensures that incoming traffic from the virtual machine is tagged internally by the switch as belonging to VLAN ID 10, and switched to the appropriate exit ports.
- The target virtual machines are reachable through another port on the compute node, which is managed by the AVS.
- That other port is configured as usual, as a VLAN trunk port, and the tenant network's VLAN ID (10) is included in the tagged range. This ensures that VLAN 10 is common to both the passthrough/SR-IOV interface and the AVS port.

PCI SR-IOV Ethernet Interface Devices

A SR-IOV Ethernet interface is a physical PCI Ethernet NIC that implements hardware-based virtualization mechanisms to expose multiple virtual network interfaces that can be used by one or more virtual machines simultaneously.

The PCI-SIG Single Root I/O Virtualization and Sharing (SR-IOV) specification defines a standardized mechanism to create individual virtual Ethernet devices from a single physical Ethernet interface. For each exposed virtual Ethernet device, formally referred to as a *Virtual Function* (VF), the SR-IOV interface provides separate management memory space, work queues, interrupts resources, and DMA streams, while utilizing common resources behind the host interface. Each VF therefore has direct access to the hardware and can be considered to be an independent Ethernet interface.

When compared with a PCI Passthrough Ethernet interface, a SR-IOV Ethernet interface:

- Provides benefits similar to those of a PCI Passthrough Ethernet interface, including lower latency packet processing.
- Scales up more easily in a virtualized environment by providing multiple VFs that can be attached to multiple virtual machine interfaces.
- Shares the same limitations, including the lack of support for LAG, QoS, ACL, and live migration.
- Has the same requirements regarding the VLAN configuration of the access switches.
- Provides a similar configuration workflow when used on Helion Carrier Grade 4.0.

It is suggested that you read Configuring PCI Passthrough Ethernet Interfaces on page 41 first. Most configuration steps are similar, with the difference that you use **pci-sriov** instead of **pci-passthrough** when defining the network type of an interface.

To review the current PCI passthrough and SR-IOV interface assignments for a given provider network, see *Helion Carrier Grade 4.0 System Administration: Displaying Provider Network Information*.

In the following example a new virtual machine is launched by user **user1** on tenant **tenant1**, with a VF interface connected to the tenant network **tenant1-net1** identified with VLAN ID 20.

Configuring SR-IOV Ethernet Interfaces

You can configure SR-IOV Ethernet interfaces using the web interface.

Prerequisites

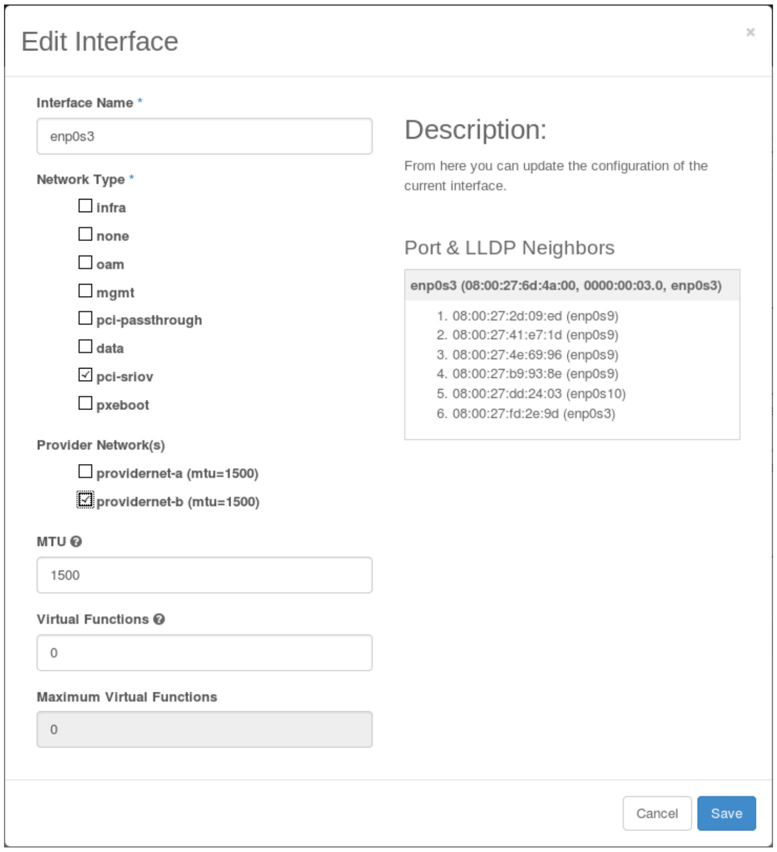
NOTE: To use PCI passthrough or SR-IOV devices, you must have Intel VT-x and Intel VT-d features enabled in the BIOS.

The exercise assumes that the underlying provider network **providernet-b** exists already, and that VLAN ID 20 is a valid segmentation ID assigned to **tenant1**.

Procedure

1. Log in as the **admin** user to the web administration interface.
2. Lock the compute node you want to configure.
3. Configure the Ethernet interface to be used as a SR-IOV interface.

   Select **Admin** > **Platform** > **Host Inventory** from the left-hand pane, click the **Hosts** tab, click the name of the compute node where the PCI interface is available, click the **Interfaces** tab, and finally click the **Edit Interface** button associated with the interface you want to configure. Fill in the window as illustrated below:


Edit Interface

Interface Name *: enp0s3

Network Type *
- ☐ infra
- ☐ none
- ☐ oam
- ☐ mgmt
- ☐ pci-passthrough
- ☐ data
- ☑ pci-sriov
- ☐ pxeboot

Provider Network(s)
- ☐ providernet-a (mtu=1500)
- ☑ providernet-b (mtu=1500)

MTU: 1500

Virtual Functions: 0

Maximum Virtual Functions: 0

Description:

From here you can update the configuration of the current interface.

Port & LLDP Neighbors

enp0s3 (08:00:27:6d:4a:00, 0000:00:03.0, enp0s3)

1. 08:00:27:2d:09:ed (enp0s9)
2. 08:00:27:41:e7:1d (enp0s9)
3. 08:00:27:4e:69:96 (enp0s9)
4. 08:00:27:b9:93:8e (enp0s9)
5. 08:00:27:dd:24:03 (enp0s10)
6. 08:00:27:fd:2e:9d (enp0s3)

In this example, Ethernet interface **enp0s3** is assigned the network type *pci-sriov*, enabled to use 4 VFs, and configured as connected to provider network **providernet-b**. Click the **Save** button to proceed.

The Maximum number of VFs displayed in this form is a read-only item which is auto-discovered by the system by inspecting the specific NIC model. This value is reported as 0 when the selected interface does not support SR-IOV.

The interface can also be configured from the CLI as illustrated below:

```
~(keystone_admin)$ system host-if-modify \
-nt pci-sriov -N 4 -p group0-data0 compute-0 enp0s3
```

NOTE: By default, a VM that requires an SR-IOV interface is instantiated only if a host NUMA node with a directly attached PCI interface is available. You can specify best-effort instantiation by using an extra specification. For more information, see Specifying Best Effort for PCI NUMA Node Affinity on page 29.

4. Create the **tenant1-net1** tenant network.

   You must ensure that:

   - The **tenant1** tenant has access to the tenant network, either assigning it as the owner, or by enabling the tenant network's shared flag.
   - The segmentation ID is set to 20.

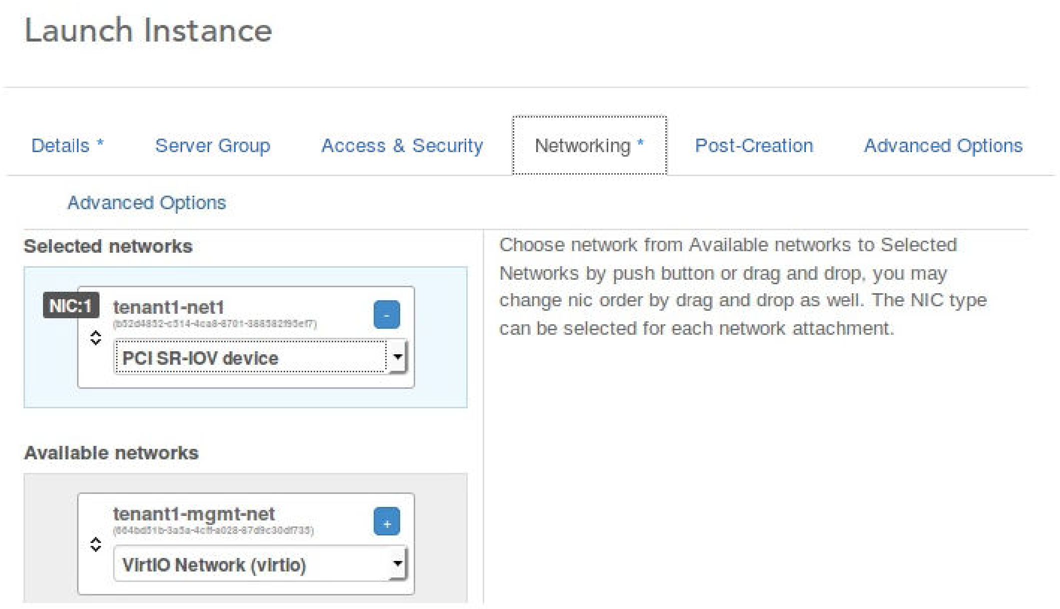
5. Configure the access switch.

   Configure the physical port on the access switch used to connect to Ethernet interface **enp0s3** as an access port with default VLAN ID of 20.

6. Unlock the compute node.

7. Launch the virtual machine.
   a) Log in as user **user1** to the web administration interface.
   b) Configure the network interfaces for the new virtual machine.

      Open the Launch Instance dialog by clicking the **Launch Interface** button on the Instances window. Configure the necessary options for the new virtual machine. In particular, click the **Networking** tab and add a SR-IOV interface on the tenant network **tenant1-net1**, as illustrated below. Add other network interfaces as needed, there are as many SR-IOV ports available as VFs have been enabled.



Selected networks

NIC:1 tenant1-net1 (b52d4852-c314-4ca8-8701-388382f93ef7) PCI SR-IOV device

Available networks

tenant1-mgmt-net (664bd51b-3a3a-4cff-a028-87d9c30df735) VirtIO Network (virtio)

Choose network from Available networks to Selected Networks by push button or drag and drop, you may change nic order by drag and drop as well. The NIC type can be selected for each network attachment.

Click the **Launch** button to proceed.

SR-IOV interfaces can be attached from the CLI when booting a new virtual machine, as illustrated below:

```
~(keystone_admin)$ nova boot \
--nic net-id=704e9f3b,vif-model=pci-sriov --flavor small \
--image cust-guest my-new-vm
```

The new virtual machine instance is up now. It has a SR-IOV VF connection to the **tenant1-net1** tenant network identified with VLAN ID 20.

Postrequisites

Access switches must be properly configured to ensure that virtual machines using PCI-passthrough or SR-IOV Ethernet interfaces (the latter discussed in Configuring SR-IOV Ethernet Interfaces on page 46) have the expected connectivity. In a common scenario, the virtual machine using these interfaces connects to external end points only, that is, it does not connect to other virtual machines in the same Helion Carrier Grade 4.0 cluster. In this case:

- Traffic between the virtual machine and the access switch can be tagged or untagged.
- The connecting port on the access switch is part of a port-based VLAN.
- If the port is tagged, the allowed VLAN ID range must not overlap with VLAN ID ranges used by the AVS ports.
- The port-based VLAN provides the required connectivity to external switching and routing equipment needed by guest applications to establish connections to the intended end points.

For connectivity to other virtual machines in the Helion Carrier Grade 4.0 cluster the following configuration is also required:

- The VLAN ID used for the tenant network, 10 in this example, and the default port VLAN ID of the access port on the switch are the same. This ensures that incoming traffic from the virtual machine is tagged internally by the switch as belonging to VLAN ID 10, and switched to the appropriate exit ports.

- The target virtual machines are reachable through another port on the compute node, which is managed by the AVS.
- That other port is configured as usual, as a VLAN trunk port, and the tenant network's VLAN ID (10) is included in the tagged range. This ensures that VLAN 10 is common to both the passthrough/SR-IOV interface and the AVS port.

SR-IOV Encryption Acceleration

Helion Carrier Grade 4.0 supports PCI SR-IOV access for encryption acceleration.

Helion Carrier Grade 4.0 supports SR-IOV access for the Intel AV-ICE02 VPN Acceleration Card, based on the Intel Coleto Creek 8925/8950 chipset with QuickAssist™ technology. (Due to driver limitations, PCI passthrough access is not currently supported.) If this card is present on an available host, you can provide VMs with access to one or more SR-IOV devices to improve performance for encrypted communications.

⚠ **CAUTION:** Live migration is not supported for instances using SR-IOV devices.

To expose the device to VMs, see *Helion Carrier Grade 4.0 System Administration: Exposing a Device for Use by VMs*. To provide a VM with access to this device, see *Helion Carrier Grade 4.0 Cloud Administration: Accessing a PCI Device from a VM on page 39*.

NOTE: To use PCI passthrough or SR-IOV devices, you must have Intel VT-x and Intel VT-d features enabled in the BIOS.

Enabling Multi-queue Support for virtio NICs

You can enable multi-queue support for enhanced throughput on virtio interfaces by setting a property on the associated image.

When multi-queue support is enabled, guests can scale the network performance as the number of guest vCPUs increases. This is because the vCPUs can transmit and receive packets in parallel as they each have their own queues. Without multi-queue, there is only one receive and transmit queue shared amongst all the vCPUs.

Procedure

Use the **glance image-update** command to update the property for the image.

```
~(keystone_admin)$glance image update \
--property vw_vif_multiqueue_enabled=True image_name
```

where *image_name* is the name of the image.

4 Tenant Network Management

Tenant Networks

Tenant networks are logical networking entities visible to tenant users, and around which working network topologies are built.

Tenant networks need support from the physical layers to work as intended. This means that the access L2 switches, providers' networks, and data interface definitions on the compute nodes, must all be properly configured. In particular, when using provider networks of the VLAN or VXLAN type, getting the proper configuration in place requires additional planning.

For provider networks of the VLAN type, consider the following guidelines:

- All ports on the access L2 switches must be statically configured to support all the VLANs defined on the provider networks they provide access to. The dynamic nature of the cloud might force the set of VLANs in use by a particular L2 switch to change at any moment.
- The set of VLANs used by each compute node is not fixed; it changes over time. The current VLAN set in use is determined by the configuration details of the tenant networks, and the scheduling on the compute nodes of the virtual machines that use them. This information is provided to the Neutron's AVS plugin, which then uses it to configure the AVS as required.

  When a tenant creates a new network, the Neutron segmentation allocation strategy is to look first for an available segmentation identifier owned by the tenant. If none is available, the search continues over the available shared segmentation identifiers. The allocation process returns an error if no segmentation identifiers are available.

The VLAN ID assigned to a tenant network is fixed for as long as the tenant network is defined in the system. If for some reason the VLAN ID has to change, the tenant network must be deleted and recreated again.

- Configuring a tenant network to have access to external networks (not just providing local networking) requires the following elements:
    - A physical router, and the provider network's access L2 switch, must be part of the same Layer-2 network. Because this Layer 2 network uses a unique VLAN ID, this means also that the router's port used in the connection must be statically configured to support the corresponding VLAN ID.
    - The router must be configured to be part of the same IP subnet that the tenant network is intending to use.
    - When configuring the IP subnet, the tenant must use the router's port IP address as its external gateway.
    - The tenant network must have the **external** flag set. Only the **admin** user can set this flag when the tenant network is created.

For provider networks of the VXLAN type, consider the following guidelines:

- Layer 3 routers used to interconnect compute nodes must be multicast-enabled, as required by the VXLAN protocol.
- To minimize flooding of multicast packets, IGMP and MLD snooping is recommended on all Layer 2 switches. The AVS switch supports IGMP V1, V2 and V3, and MLD V1 and V2.
- To support IGMP and MDL snooping, Layer 3 routers must be configured for IGMP and MDL querying.
- To accommodate VXLAN encapsulation, the MTU values for Layer 2 switches and compute node data interfaces must allow for additional headers. For more information, see *Helion Carrier Grade 4.0 Planning: The Ethernet MTU*.
- To participate in a VXLAN network, the data interfaces on the compute nodes must be configured with IP addresses, and with route table entries for the destination subnets or the local gateway. For more information, see *Helion Carrier Grade 4.0 System Administration: Managing Data Interface Static IP Addresses Using the CLI*, and *Helion Carrier Grade 4.0 System Administration: Adding and Maintaining Routes for a VXLAN Network on page 74*.

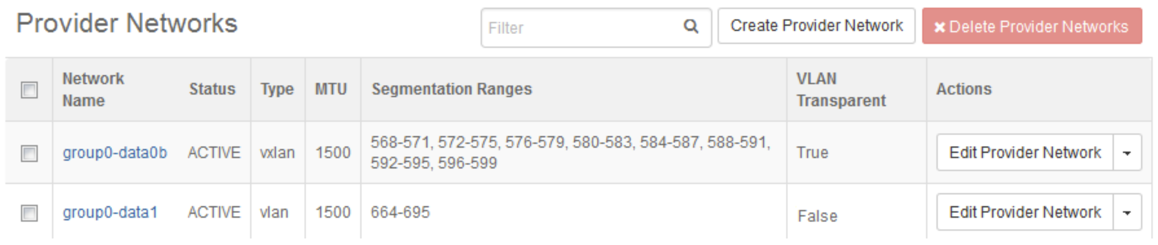
In some circumstances, tenant networks can be configured to use VLAN Transparent mode, in which VLAN tagged packets from the guest are encapsulated within a provider network segment (VLAN) without removing or modifying the guest VLAN tag. For more information, see *Helion Carrier Grade 4.0 Tenant User's Guide: VLAN Transparent on page 53*. Alternately, guest VLAN-tagged traffic can be supported by Helion Carrier Grade 4.0 tenants explicitly defining one or more VLAN-tagged IP subnets on a single tenant network. With this approach, the guest VLAN-tagged IP subnets have access to all of the services of the virtualized network infrastructure, such as DHCP, virtual routing, meta-data server, etc. For more information, see *Helion Carrier Grade 4.0 Introduction: Helion Carrier Grade 4.0 Overview*.

VLAN Transparent

A *vlan transparent* tenant network is one that allows VLAN tagged packets to be encapsulated within a provider network segment without removing or modifying the guest VLAN tag. VLAN Transparent is provided in addition to VLAN-tagged Neutron subnets for guest VLANs.

VLAN Transparent must be supported on a provider network before a VLAN Transparent tenant network can be created on that provider network. The **VLAN Transparent** column in the list of defined provider networks on the page of the **Admin** > **Platform** > **Provider Networks** page indicates availability.

VLAN Priority attributes are propagated to the provider VLAN tag and then restored to the guest VLAN tag.

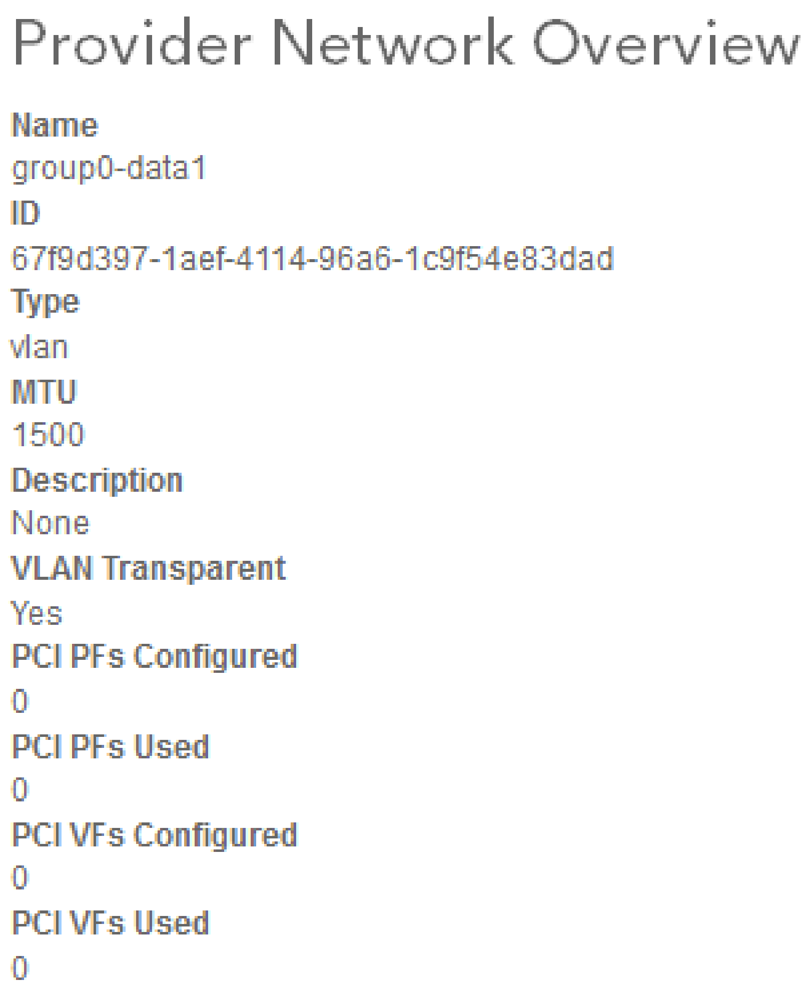


	Network Name	Status	Type	MTU	Segmentation Ranges	VLAN Transparent	Actions
☐	group0-data0b	ACTIVE	vxlan	1500	568-571, 572-575, 576-579, 580-583, 584-587, 588-591, 592-595, 596-599	True	Edit Provider Network
☐	group0-data1	ACTIVE	vlan	1500	664-695	False	Edit Provider Network

This information is also available from the following command.

```
~(keystone_admin)$ neutron providernet-show provider1-vt
+-------------------+--------------------------------------------------------+
| Field             | Value                                                  |
+-------------------+--------------------------------------------------------+
| description       |                                                        |
| id                | 72a09e68-7170-493e-b9c6-d2bcb881f458                   |
| mtu               | 1500                                                   |
| name              | provider1-vt                                           |
| ranges            | {                                                      |
|                   |      "name": "provider1-vt-r1-0",                      |
|                   |      "tenant_id": "c17a85ac5283496da1152ce68b79e606",  |
|                   |      "maximum": 50,                                    |
|                   |      "minimum": 59,                                    |
|                   |      "shared": false,                                  |
|                   |      "id": "8b1fa184-80e0-4555-bd9f-bcf444384189",     |
|                   |      "description": null                               |
|                   | }                                                      |
| status            | DOWN                                                   |
| type              | vlan                                                   |
| vlan_transparent  | True                                                   |
+-------------------+--------------------------------------------------------+
```

The summary information available by selecting a tenant network name form the **Provider Networks** tab of the **System** > **Networks** page indicates if a tenant network will request VLAN Transparent services from a provider network.



Provider Network Overview

Name
group0-data1
ID
67f9d397-1aef-4114-96a6-1c9f54e83dad
Type
vlan
MTU
1500
Description
None
VLAN Transparent
Yes
PCI PFs Configured
0
PCI PFs Used
0
PCI VFs Configured
0
PCI VFs Used
0

This information is also available from the following command.

```
~(keystone_admin)$ neutron net-show tenant1-vt
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | 011cea1e-422f-49d1-9d80-ba864bab361f |
| mtu                       | 1500                                 |
| name                      | tenant1-vt                           |
| provider:network_type     | vlan                                 |
| provider:physical_network | provider1-vt                         |
| provider:segmentation_id  | 50                                   |
| router:external           | False                                |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   | 3bff6cb0-8422-4596-857f-a02d68a051b1 |
|                           | dedcd9e0-d4ee-4306-a6bf-194104baa9b4 |
| tenant_id                 | c17a85ac5283496da1152ce68b79e606     |
| vlan_transparent          | True                                 |
+---------------------------+--------------------------------------+
```

For more information on setting up provider networks, see *Helion Carrier Grade 4.0 System Administration: Configuring Provider Networks* and *Configuring Provider Networks Using the CLI*. For more information on setting up tenant networks, see Creating Tenant Networks on page 57.

Table 1 **VLAN Transparent Compatibility**

	802.1p	802.1q	802.1ad	QinQ
Flat Provider Network	Yes	Yes	Yes	Yes
VLAN Provider Network	Yes	Yes	Yes	Yes
VXLAN Provider Network	Yes	Yes	Yes[1]	Yes
VLAN Tagged Subnets	Yes	Yes	No	No

Guest VLANs

Use guest VLANs to segregate IP traffic from a single virtual Ethernet interface on a virtual machine into dedicated VLANs. Together with the capability to define multiple IP subnets on a single tenant network, guest VLANs facilitate the transitioning of existing guest applications to run on the Helion Carrier Grade 4.0 virtualized network environment.

Guest VLANs are useful when guest applications rely on the capability to configure a single virtual Ethernet interface with multiple, probably overlapping, IP addresses. From the point of view of the guest, this is done by defining VLAN Ethernet interfaces, and associating one or more IP addresses to them. If implementing overlapping IP addresses, typically in support of VPN applications, the guest must use different VLAN IDs to separate traffic from different VPNs.

For example, on a Linux guest, the virtual interfaces eth0.10:1, eth0.10:2, and eth0.20 refer to the same eth0 physical interface with two virtual interfaces on VLAN ID 10, and a single virtual interface on VLAN ID 20. A common scenario in a VLAN application is to allocate distinct IP addresses from the same IP subnet to virtual interfaces on the same VLAN. In this example, eth0.10:1 and eth0.10:2 could be assigned distinct IP addresses from the subnet 192.168.1.0/24, and eth0.20 an address from the subnet 192.168.2.0/24. In the case of a VPN application, overlapping IP addresses are allowed to exist on eth0.20 and either eth0.10:1 or eth0.10:2.

Helion Carrier Grade 4.0 supports these deployment scenarios with the help of guest VLANs which enable the transport of IP subnets traffic over VLAN-tagged Ethernet frames. To support the example above, a tenant user would define the following two IP subnets, both on the same tenant network, using guest VLAN IDs as follows:

- Subnet 192.168.1.0/24 with guest VLAN ID set to 10
- Subnet 192.168.2.0/24 with guest VLAN ID set to 20

The subnet-to-VLAN_ID mapping can be one-to-one, as in this example, or many-to-one. This means that tenant users can use a single VLAN ID of their choice to encapsulate traffic from one or more IP subnets.

Alternately, tenant networks can be configured to use VLAN Transparent mode, in which VLAN tagged guest packets are encapsulated within a provider network segment without removing or modifying the guest VLAN tag. For more information, see *Helion Carrier Grade 4.0 Tenant User's Guide: VLAN Transparent on page 53*.

[1] VLAN priority attributes not propagated to IP header differentiated services field.

Guest VLAN Implementation

Guest VLANs are implemented using available segmentation ranges from suitable provider networks, just as it is done when new tenant networks are created. Therefore all network design considerations regarding the configuration of L2 switches and the Neutron allocation strategy, described in Tenant Networks on page 51, must be taken into consideration.

Additionally, note that the AVS will silently discard incoming VLAN-tagged VM traffic with unknown VLAN IDs, that is, with VLAN IDs not defined as guest VLANs on the particular tenant network.

Managed and Unmanaged Subnets

Use the **System Managed Subnet** and **Enable DHCP** subnet attributes to determine how IP addresses are allocated and offered on an IP subnet.

With the proper configuration in place, DHCP services can be provided by the built-in Neutron DHCP server, by a standalone server available from an external network infrastructure, or by both.

When creating a new IP subnet for a tenant network you can specify the following attributes:

System Managed Subnet

When this attribute is enabled, the subnet is *system managed*. The Neutron service automatically allocates an IP address from the address allocation pools defined for the subnet to any new virtual machine (VM) instance with a virtual Ethernet interface attached to the tenant network. Once allocated, the pair (MAC address, IP address) is registered in the Neutron database as part of the overall registration process for the new virtual machine.

When the system managed subnet attribute is disabled, the subnet is *unmanaged*. No automatic allocation of IP addresses takes place, and the Neutron DHCP service for the subnet is disabled. Allocation of IP addresses for new virtual machines must be done at boot time using the CLI or the API interfaces.

Enable DHCP

When this attribute is enabled, a virtual DHCP server becomes available when the subnet is created. It uses the (MAC address, IP address) pairs registered in the Neutron database to offer IP addresses in response to DHCP discovery requests broadcast on the subnet. DHCP discovery requests from unknown MAC addresses are ignored.

The Neutron DHCP server can only be enabled on system managed subnets. DHCP services for unmanaged subnets, if required, must be provisioned by external, non-Neutron, DHCP servers.

When the DHCP attribute is disabled, all DHCP and DNS services, and all static routes, if any, must be provisioned externally.

Allocation Pools

This a list attribute where each element in the list specifies an IP address range, or address pool, in the subnet address space that can be used for dynamic offering of IP addresses. By default there is a single allocation pool comprised of the entire subnet's IP address space, with the exception of the default gateway's IP address.

An external, non-Neutron, DHCP server can be attached to a system managed subnet to support specific deployment needs as required. For example, it can be configured to offer IP addresses on ranges outside the Neutron allocation pools to service physical devices attached to the tenant network, such as testing equipment and servers.

Allocation pools can only be specified on system managed subnets.

The method used to access the server depends on your deployment scenario.

- If the VM is attached to a tenant network with a virtual router then the metadata server is reachable using the virtual router as the route gateway.
- If the VM instance is attached to multiple tenant networks, each with access to a virtual router, then any of the virtual routers provides access to the metadata server. However, installing multiple default routes on the VM might impact the VM's ability to route packets back to external networks.
- If the VM is attached to a tenant network that does not have a virtual router then the a route to the metadata server can be retrieved from the Neutron DHCP service, provided that this service is enabled. If DHCP is enabled on a tenant network, then the VM can use DHCP option 121, Classless Static Route, to retrieve the metadata server static route information, which uses the DHCP server's address as the gateway.

  Note that when using DHCP option 121, the answer from the DHCP server will also include other applicable static routes. They include:

  - any static routes configured when the IP subnet was created
  - default route to the subnet gateway IP address, if one is configured on the subnet
  - on-link routes for all other subnets on the same network and same guest VLAN

The DHCP service only responds to option 121 is there are no virtual routers on the network.

The following requirements must be satisfied in order for a guest application to access the metadata service:

- There is a route table entry to route traffic destined to the 169.254.169.254 address via a Neutron router, or via a suitable static route to the 169.254.169.254 address.
- The metadata server knows about the virtual machine instance associated with the MAC and IP addresses of the virtual Ethernet interface issuing the metadata service request. This is necessary for the metadata server to be able to validate the request, and to identify the virtual machine's specific data to be returned.

  On system managed subnets, the Neutron service has all address information associated with the virtual machines in its database.

  On unmanaged subnets, you must tell the Neutron service the IP address of the network interface issuing the metadata service requests.

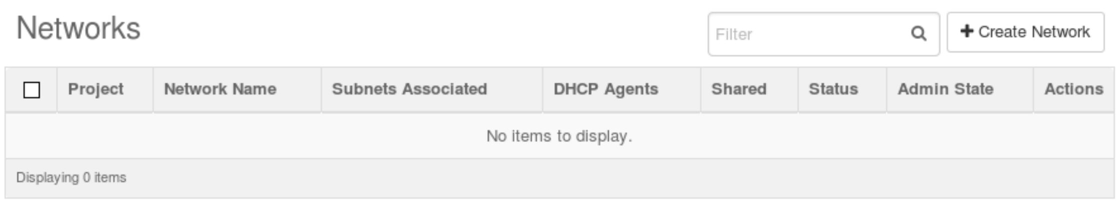
Creating Tenant Networks

You can use the CLI or Web interface to set up tenant networks and their associated IP subnets.

Procedure

1. List the tenant networks currently defined on the system.

   Select **Admin** > **System** > **Networks** to open the Networks page.



Networks

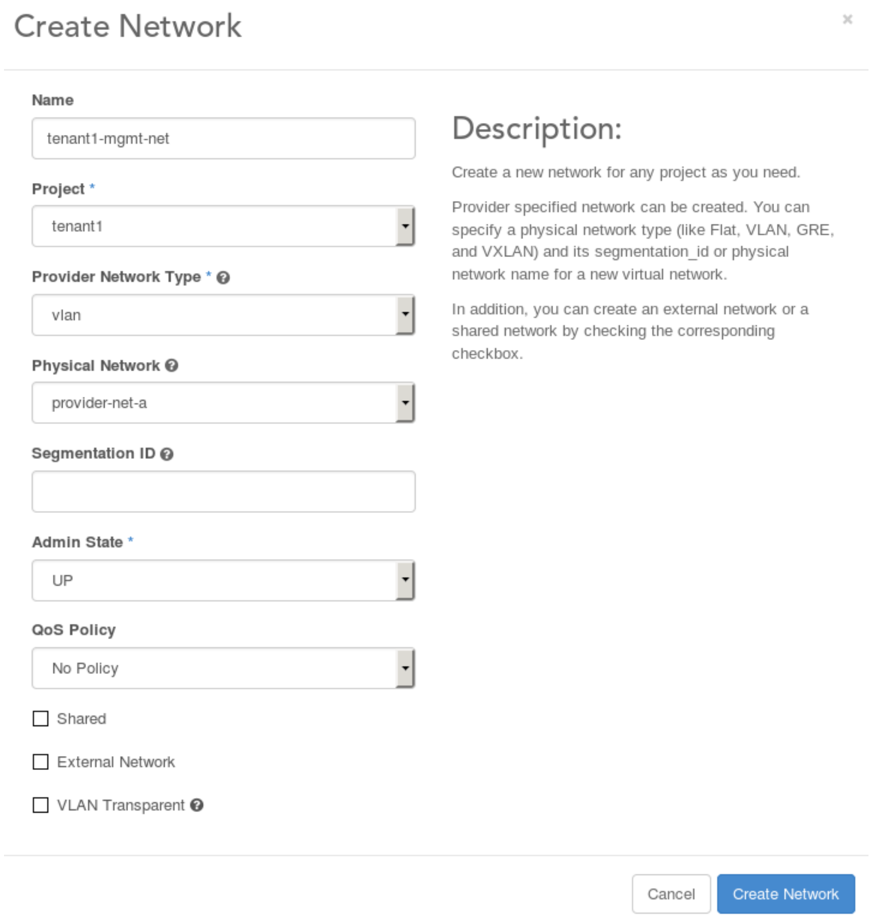
Filter | + Create Network

☐	Project	Network Name	Subnets Associated	DHCP Agents	Shared	Status	Admin State	Actions
No items to display.								

Displaying 0 items

2. Create the tenant network.

   From the **Networks** tab, click **Create Network** and fill in the form as illustrated below:



Create Network

Name: tenant1-mgmt-net

Project *: tenant1

Provider Network Type *: vlan

Physical Network: provider-net-a

Segmentation ID:

Admin State *: UP

QoS Policy: No Policy

☐ Shared

☐ External Network

☐ VLAN Transparent

Description:

Create a new network for any project as you need.

Provider specified network can be created. You can specify a physical network type (like Flat, VLAN, GRE, and VXLAN) and its segmentation_id or physical network name for a new virtual network.

In addition, you can create an external network or a shared network by checking the corresponding checkbox.

Cancel | Create Network

   Click **Create Network** to commit the changes.

3. Create any required subnets.
   a) Select **Networks** in the System Panel section of the **Admin** tab to open the Networks page.
   b) Select the name of the tenant network you just created.
   c) Click **Create Subnet**.
   d) Complete the forms on the Subnet and Subnet Details tabs, then click **Create**.

Creating Tenant Networks Using the CLI

You can use CLI commands to set up tenant networks.

To create a tenant network using the CLI, use the following command:

```
~(keystone_admin)$ neutron net-create --tenant-id UUID \
--provider:physical_network=network \
--provider:network_type=type \
--provider:segmentation_id=segment \
--shared --router:external --admin-state-down \
--vlan-transparent=state \
name
```

where

UUID

is the UUID of the tenant network

network

is the provider network to use

type

is the type of network to create, this can be one of flat, vlan or vxlan

segment

is the provider network segment ID to use. You must be logged in as Admin to use this parameter and it must be used in conjunction with *network* and *type*.

state

is either "True" or "False", indicating if this tenant network will attempt to use VLAN Transparent mode. For more information, see VLAN Transparent on page 53.

name

is the name of the tenant network

To create a tenant subnet using the CLI, use the following command:

```
~(keystone_admin)$ neutron subnet-create --tenant-id UUID \
--name name --gateway gateway --disable-dhcp external-net externalIP
```

where

UUID

is the ID of the tenant network the subnet is being created for

name

is the name of the subnet

gateway

is the IP address of the gateway

address

is the IP address/bitmap of the external network

5
Virtual Router Management

Virtual Routers

Virtual routers provide internal and external network connectivity for tenants.

The user associated with a tenant can add a designated number of virtual routers (Neutron routers) to the tenant. The maximum number is specified in the quotas for the tenant.

The virtual router automatically provides a connection to system services required by the tenant, such as the metadata service that supplies instances with user data. You can configure the virtual routers with interfaces to tenant networks to provide internal and external connectivity.

A virtual router can be implemented as a centralized router, or a distributed virtual router (DVR).

Only the **admin** user can specify a distributed router. For other tenants, this choice is not available, and a centralized router is implemented by default. The **admin** user can change a centralized router to a distributed router on behalf of other tenants (see *Helion Carrier Grade 4.0 Tenant User's Guide: Virtual Router Administration on page 62*).

Centralized

A centralized virtual router is instantiated on a single compute node. All traffic using the router must pass through the compute node.

Distributed

A distributed virtual router is instantiated in a distributed manner across multiple hosts. Distributed virtual routers provide more efficient routing than standard virtual routers for east-west (tenant-to-tenant) or floating IP address traffic. Local traffic is routed within the local host, while external L3 traffic is routed directly between the local host and the gateway router.

To implement the distributed model, a centralized-portion of the router is still deployed on one host. The centralized portion manages north-south (external network) traffic and source network address translation (SNAT) traffic, as well as agents deployed on other hosts. The agents offload the centralized router for east-west (tenant-to-tenant) routing and floating IP network address translation.

In some cases, the use of virtual routers on the tenant networks can result in multiple default routes for a virtual machine (VM). If this happens, you can establish alternative VM access to the metadata server; see #unique_64/unique_64_Connect_42_accessing_metadata_server on page 57.

You can enable SNAT on a virtual router. For more information, see *Helion Carrier Grade 4.0 Tenant User's Guide: Configuring SNAT on a Virtual Router on page 70*.

You can also enable DNAT on a virtual router. For more information, see *Helion Carrier Grade 4.0 Tenant User's Guide: Configuring Port-based DNAT on a Virtual Router on page 71*.

To add a virtual router, see *Helion Carrier Grade 4.0 Tenant User's Guide: Adding Virtual Routers on page 65*. To create interfaces, see *Helion Carrier Grade 4.0 Cloud Administration: Adding Virtual Router Interfaces on page 66*.

Virtual Router Administration

Users with administrative privileges can review and edit router information for all tenants.

The Routers page accessed from **Admin** > **System** > **Routers** provides central control of all virtual routers. From this page, you can edit router names, change router types, change the status of routers (**Up** or **Down**), and delete routers.

Only users with administrative privileges can convert a centralized router to a distributed router.

An administrative user can delete routers on any tenant. A non-administrative user can delete routers on the tenant associated with the user.

As an alternative to the web administration interface, you can use the CLI. The following examples assume you have become the **admin** user.

- To list routers:

```
~(keystone_admin)$ neutron router-list
```

- To show details for a router:

```
~(keystone_admin)$ neutron router-show router_id
```

  where *router_id* is the name or UUID of the router.

- To update a router:

```
~(keystone_admin)$ neutron router-update router_id --distributed=True
```

  where *router_id* is the name or UUID of the router. This example updates a router to make it a distributed virtual router.

- To delete a router:

```
~(keystone_admin)$ neutron router-delete router_id
```

DiffServ-Based Priority Queuing and Scheduling

Differentiated Services, or DiffServ, is a packet-based traffic management mechanism that allows end devices to specify an expected per-hop behavior (PHB) from upstream switches and routers when handling their traffic during overload conditions.

DiffServ marking and architecture are IEEE specifications, IEEE RFC 2474 and RFC 2475.

Support for DiffServ is implemented on the AVS for all input and output ports, on all attached tenant networks. On each port, it uses eight queues associated with the CS0 to CS7 DiffServ class selectors, which are processed by a round-robin scheduler with weights of 1, 1, 2, 2, 4, 8, 16, and 32. Overflow traffic is tail-drop discarded on each queue. Guest applications in the Helion Carrier Grade 4.0 cluster can set a Differentiated Services Code Point (DSCP) value in the Differentiated Service (DS) field of the IP header to mark the desired upstream PHB.

On ingress, a packet being processed to be sent to the VM is directed to the appropriate DiffServ output queue. On overflow, that is, if the virtual machine cannot keep up with the incoming traffic rate, lower priority packets are discarded first.

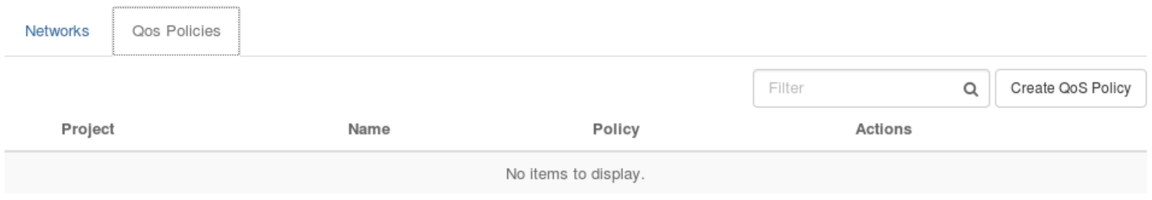
On egress, a packet sent from the virtual machine (VM) to the AVS is first enqueued into the appropriate DiffServ input queue according to the specified DSCP value, the CS0 queue being the default. On overload, that is, if the AVS cannot keep up with the incoming traffic, lower priority packets are discarded first; in this case, you should consider re-engineering the AVS, possibly assigning it more processing cores. Once serviced by the DiffServ class scheduler, the packet is processed according to the QoS policy in place for the tenant network, as described in *Helion Carrier Grade 4.0 Tenant User's Guide: Quality of Service Policies on page 63*.

Quality of Service Policies

Quality of Service (QoS) policies specify relative packet processing priorities applied by the AVS switch on each compute node to incoming tenant network's traffic during overload conditions.

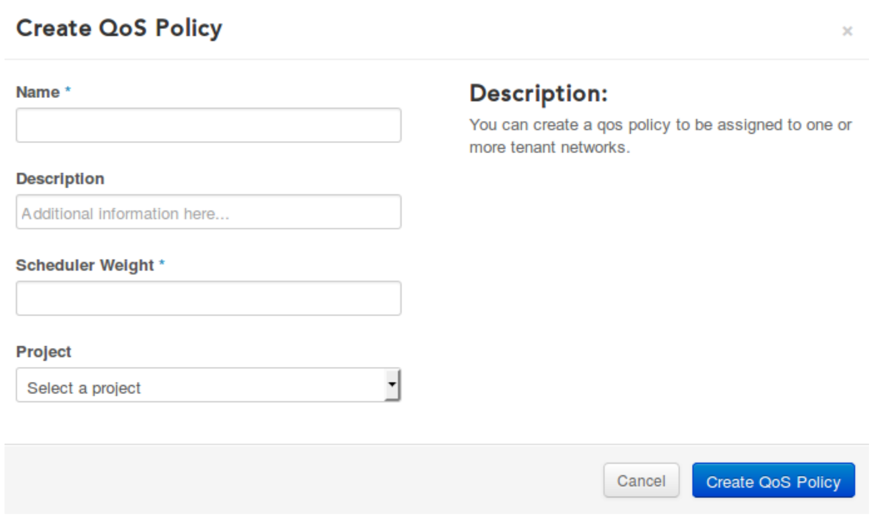
The QoS polices play no role under normal traffic loads, when no input traffic queues in the AVS are close to their overflow limits.

QoS policies are created by the cluster administrator, and selected by the tenant users to apply on a per-tenant network basis. To create a new QoS policy, log in as administrator, select **Admin** > **System** > **Networks**, and then select the **QoS Policies** tab.



Project	Name	Policy	Actions
No items to display.			

Click the button **Create QoS Policy** to display the Create QoS Policy window.


Create QoS Policy

Name *

Description

Additional information here...

Scheduler Weight *

Project

Select a project

Description:

You can create a qos policy to be assigned to one or more tenant networks.

The following fields are available:

Name

The name of the new QoS policy. This is the name that the tenant user sees as available for use.

This field is mandatory.

Description

A free form text for your reference.

Scheduler Weight

The relative weight the AVS traffic scheduler uses on overload conditions, as compared with the scheduler weight of all other QoS policies.

The scheduler weight is a positive integer number associated with each QoS policy. On overload, the AVS schedules traffic processing for each tenant network according to its assigned QoS policy. By default, with no specific QoS policies defined, traffic from all tenant networks is processed in round-robin mode, one tenant network after another. Effectively, the default behavior is similar to assigning a scheduler weight of 1 to all tenant networks.

When a specific QoS policy with a scheduler weight greater than 1 is applied to a tenant network, its traffic is scheduled with a relative higher frequency. For example, if the scheduler weight for tenant network A is 10, and the one for tenant network B is 100, then on overload, tenant network B will see its queued traffic processed 10 times as often as tenant network A.

The handling of the scheduler weight is implemented as a per-packet token bucket for each tenant network. This means that each unit in the scheduler weight counts as one packet to be processed in sequence. In the previous example, the AVS processes 10 consecutive packets from tenant network A, followed by 100 packets from tenant network B. This implementation ensures a fair-share behavior that prevents any tenant network from running into total bandwidth starvation, even if its scheduler weight is relatively low.

The range of values for the scheduler weight is arbitrary. You must however ensure that the values assigned to the different policies make sense for the intended applications.

This field is mandatory.

Project

A drop-down menu displaying the currently defined tenants (projects). The new QoS policy is available to only the selected tenant. If no tenant is selected, it is available to all tenants in the cluster.

Tenant users can select available QoS policies when the tenant networks are created from the Create Network window. They can also apply a QoS policy to an already existing tenant network from the Edit Network window.

Adding Virtual Routers

You can add virtual routers to a tenant using the web administration interface or the CLI.

To add a router to a tenant, you must be logged in as the user associated with the tenant.

NOTE: Only the **admin** tenant can add a distributed virtual router.

Procedure

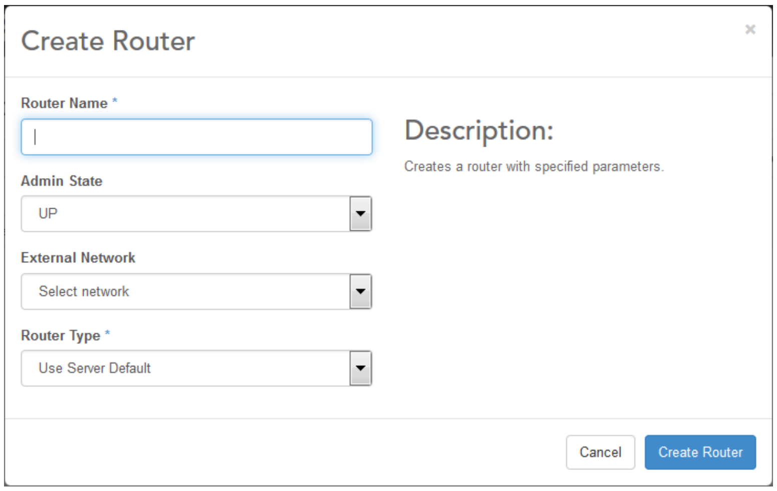
1. Log in as the user associated with the tenant.

   The following instructions assume that the **admin** tenant is logged in.

2. Select **Project** > **Network** > **Routers**.

3. On the Routers page, click **Create Router**.

   The Create Router dialog box appears.



Create Router

Router Name *

Admin State: UP

External Network: Select network

Router Type *: Use Server Default

Description:

Creates a router with specified parameters.

4. Provide a **Router Name**.

5. Optional: To add a connection to a tenant network configured for external access, select from the **External Network** drop-down menu.

   This adds a gateway for the router. Alternatively, you can add a gateway after the router is created. For more information, see Adding Virtual Router Interfaces on page 66.

6. Select the **Router Type**.

NOTE: The **Router Type** field is shown only for the **admin** user. For other tenants, a centralized router is added. The **admin** user can change this setting on behalf of another tenant after the router has been added (see Virtual Router Administration on page 62).

Use Server Default

Use the default setting specified on the controller (for a standard Helion Carrier Grade 4.0 installation, centralized).

Centralized

Add a centralized virtual router.

Distributed

Add a distributed virtual router (DVR).

For more information about these choices, see Virtual Routers on page 61.

7. To save your settings and close the dialog box, click **Create Router**.

Postrequisites

To attach router interfaces to tenant networks, see Adding Virtual Router Interfaces on page 66.

Adding Virtual Router Interfaces

You can create router interfaces to tenant networks using the web administration interface or the CLI.

For more information about virtual routers, see Virtual Routers on page 61.

Prerequisites

Before you can create interfaces to tenant networks, you must create the tenant networks and associated subnets. For information about tenant networks and subnets, see Tenant Networks on page 51 and Managed and Unmanaged Subnets on page 56.

As an alternative to the web administration interface, you can use CLI commands to add router interfaces. First, become the appropriate keystone user. The examples in this section assume you are the **admin** user.

Then use the **neutron router-interface-add** command to add interfaces. For example:

```
~(keystone_admin)$ neutron router-interface-add router_id subnet-id
```

where *router_id* is the name or UUID of the router, and *subnet_id* is the name or UUID of the subnet to which you want to attach an interface. By default, the interface is assigned the gateway address on the subnet.

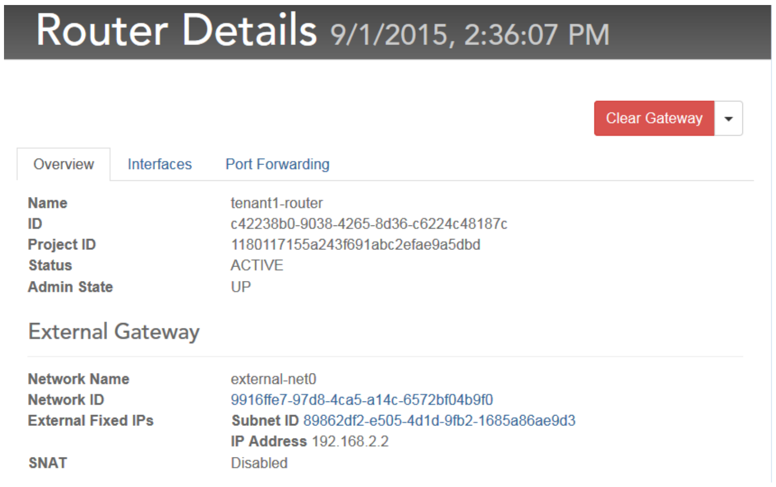
Procedure

1. Log in as the user associated with the tenant.

2. Select **Project** > **Network** > **Routers**.

3. In the list of existing routers, click the name of the router to be configured.

   The Router Details page appears.



Router Details 9/1/2015, 2:36:07 PM

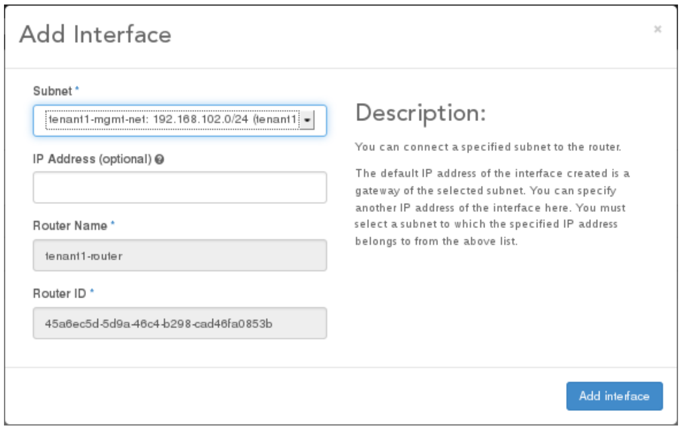
Name	tenant1-router
ID	c42238b0-9038-4265-8d36-c6224c48187c
Project ID	1180117155a243f691abc2efae9a5dbd
Status	ACTIVE
Admin State	UP

External Gateway

Network Name	external-net0
Network ID	9916ffe7-97d8-4ca5-a14c-6572bf04b9f0
External Fixed IPs	**Subnet ID** 89862df2-e505-4d1d-9fb2-1685a86ae9d3
	IP Address 192.168.2.2
SNAT	Disabled

4. On the **Interfaces** tab, click **Add Interface**.

   The Add Interface dialog box appears.



Add Interface

Subnet *	tenant1-mgmt-net: 192.168.102.0/24 (tenant1
IP Address (optional)	
Router Name *	tenant1-router
Router ID *	45a6ec5d-5d9a-46c4-b298-cad46fa0853b

Description:

You can connect a specified subnet to the router.

The default IP address of the interface created is a gateway of the selected subnet. You can specify another IP address of the interface here. You must select a subnet to which the specified IP address belongs to from the above list.

Subnet

: Use the drop-down list to select from existing subnets defined for the tenant. Each subnet is identified by a tenant network name, followed by a subnet address and mask created for the tenant network.

IP Address (optional)

: You can optionally specify an IP address for the interface. By default, the gateway address for the subnet is used (for example, 192.168.102.1).

Router Name

: This field is shown for information only.

Router ID

: This field is shown for information only.

5. To save your settings and close the dialog box, click **Add Interface**.

   The new interface is shown in the Router Details.

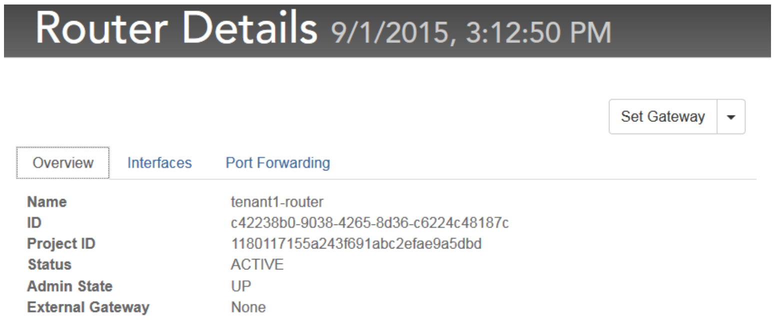
Adding a Gateway to a Virtual Router

You can configure a gateway for a virtual router as a tenant or an administrator. Users with administrator privileges have access to additional controls.

Procedure

1. Log in as the user associated with the tenant.
2. Open the Routers page.
   - To open the page with tenant privileges, select **Project** > **Network** > **Routers**.
   - To open the page with administrator privileges, select **Admin** > **System** > **Routers**.
3. In the list of existing routers, click the name of the router to be configured.

   The Router Details page appears.

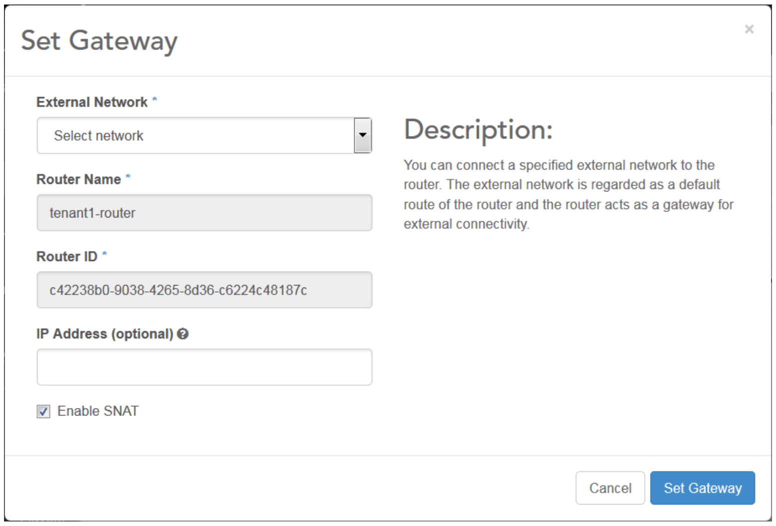


Name	tenant1-router
ID	c42238b0-9038-4265-8d36-c6224c48187c
Project ID	1180117155a243f691abc2efae9a5dbd
Status	ACTIVE
Admin State	UP
External Gateway	None

4. On the **Interfaces** tab, click **Set Gateway**.

NOTE: If a gateway has already been added, a **Clear Gateway** button is present instead.

The Set Gateway dialog box appears.



Description:

You can connect a specified external network to the router. The external network is regarded as a default route of the router and the router acts as a gateway for external connectivity.

External Network

Select a tenant network configured to provide external access.

Router Name

This field is shown for information only.

Router ID

This field is shown for information only.

IP Address (optional)

This field is shown only for users with administrator privileges. You can optionally specify an IP address for the gateway interface. By default, the server allocates the next available external IP address.

Enable SNAT

This control is shown only for users with administrator privileges. For more about the SNAT option, see Configuring SNAT on a Virtual Router on page 70.

To save the gateway gettings and close the dialog box, click **Set Gateway**.

Adding a Gateway to a Virtual Router Using the CLI

You can configure a gateway for a virtual router using CLI commands.

To use the CLI, become the appropriate keystone user. The examples in this section assume you are the **admin** user.

To add a gateway interface, use a command of the following form:

```
~(keystone_admin)$ neutron router-gateway-set router_id externalnet_id \
[--disable-snat] [--fixed-ip gateway_addr]
```

Where:

router_id

is the name or UUID of the router

externalnet_id

is the name or UUID of a tenant network configured to provide external connections

gateway_addr

is the gateway interface IP address. If *gateway_addr* is not specified, the server allocates the next available external IP address.

NOTE: The **fixed-ip** and **disable-snat** options require administrator privileges. For more about the SNAT option, see Configuring SNAT on a Virtual Router on page 70.

Configuring SNAT on a Virtual Router

You can enable or disable source network address translation (SNAT) for each individual external network connection on a virtual router. By default, SNAT is enabled for all external networks.

Both SNAT and port-based DNAT (destination network address translation) use tables to translate between IP addresses and ports used on an internal network, and an IP address and multiple ports used on an external network. However, SNAT provides translation for internally initiated UDP and TCP traffic flows only. An instance connected to an internal tenant network can initiate connections to servers on an external, usually public, network, but external servers cannot initiate connections to external addresses mapped to internal addresses. For externally initiated connections, the use of floating IP addresses or port-based DNAT is required. For more about DNAT, see Configuring Port-based DNAT on a Virtual Router on page 71.

System administrators can enable or disable SNAT using the following command:

```
~(keystone_admin)$ neutron router-update routername --external_gateway_info type=dict \
network_id=externalnetwork,enable_snat=boolean
```

where:

routername

is the name or UUID of the virtual router

externalnetwork

is the name or UUID of the external network

boolean

is **true** (to enable SNAT) or **false** (to disable SNAT)

CAUTION: For SNMP implemented on guests, the following limitations apply when SNAT is enabled:

- An SNMP agent on an internal network cannot receive requests from SNMP managers on external networks.
- An SNMP manager on an internal network cannot receive traps from SNMP agents on external networks, even if communication is initiated by the manager. This is because SNMP traps are sent to UDP port 162, not to the source port.

Only agent responses to requests from a manager on the internal network are processed appropriately.

To overcome these limitations, you can use floating IP addresses or make internal addresses public. For more information, see *Helion Carrier Grade 4.0 Cloud Administration: Connecting Virtual Machines to External Networks*.

Configuring Port-based DNAT on a Virtual Router

You can use port-based destination network address translation (DNAT), also known as port forwarding, to support externally-initiated connections to multiple VMs using a single external IP address.

As with SNAT (source network address translation), port-based DNAT uses a network address translation table to translate between IP addresses and ports used on an internal network, and the gateway IP address of a router attached to an external network. With port-based DNAT, internally initiated connections are not required to establish or maintain table entries.

As an alternative to the web administration interface, you can use the CLI. For more information, see Configuring Port-based DNAT Using the CLI on page 73.

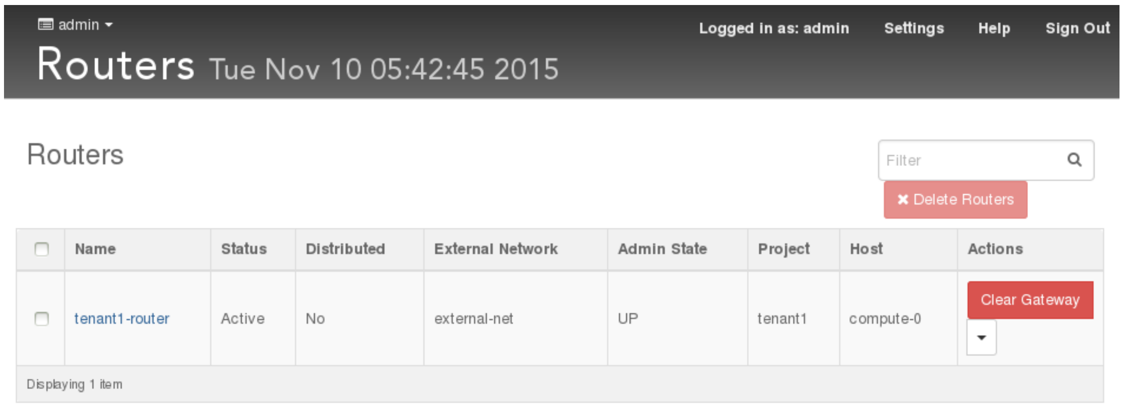
Prerequisites

Before you can use port-based DNAT, you must enable SNAT to activate support for network address translation tables. For more information, see Configuring SNAT on a Virtual Router on page 70.

Before you can add an entry, a VM IP address is required. IP addresses are assigned to instances when they are launched, and preserved until they are deleted.

Procedure

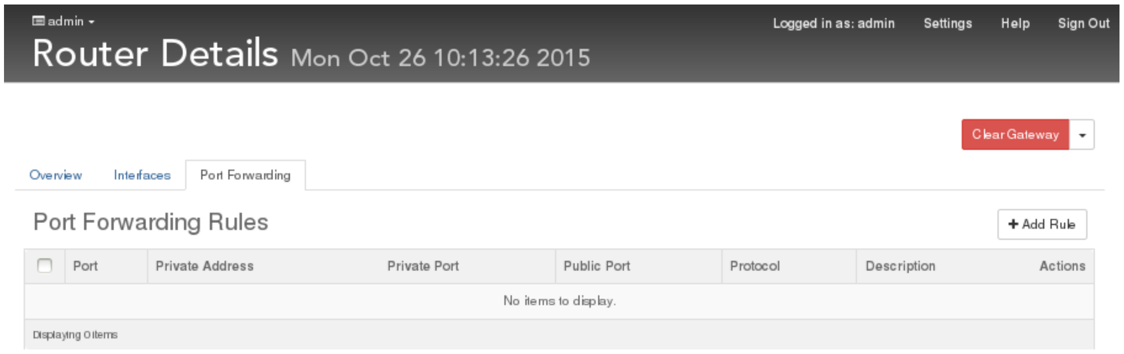
1. On the **Admin** menu of the web administration interface, in the **System** section, select **Routers**.
2. On the Routers page, click the **Name** of the router to be configured for port-based DNAT.



	Name	Status	Distributed	External Network	Admin State	Project	Host	Actions
	tenant1-router	Active	No	external-net	UP	tenant1	compute-0	Clear Gateway

Displaying 1 item

3. On the Router Details page, select the **Port Forwarding** tab.



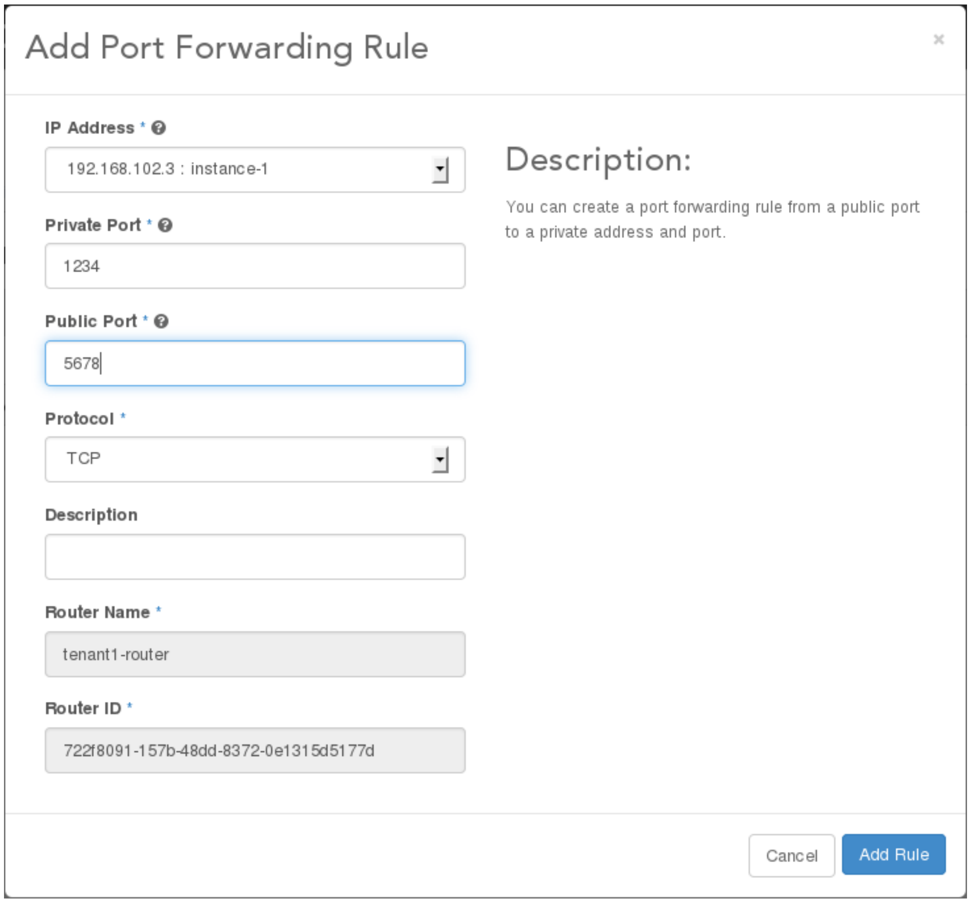
Router Details Mon Oct 26 10:13:26 2015

Port Forwarding Rules

Port	Private Address	Private Port	Public Port	Protocol	Description	Actions
No items to display.						

Displaying 0 items

- To add a rule, click **Add Rule**, and then complete the Add Port Forwarding Rule dialog box.



Add Port Forwarding Rule

IP Address *	192.168.102.3 : instance-1
Private Port *	1234
Public Port *	5678
Protocol *	TCP
Description	
Router Name *	tenant1-router
Router ID *	722f8091-157b-48dd-8372-0e1315d5177d

Description:

You can create a port forwarding rule from a public port to a private address and port.

IP Address

The IP address of the VM. You can select existing addresses from the drop-down list.

Private Port

The port to use on the VM IP address.

Public Port

The port on the external gateway interface of the router.

Protocol

The applicable protocol for the rule. Select from the drop-down list.

Description

An optional text description of the rule.

Router Name

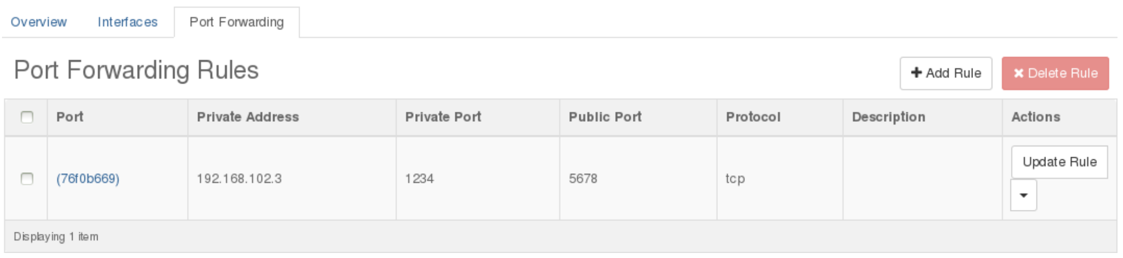
The name of the router (shown for information only).

Router ID

The UUID of the router (shown for information only).

4. Click **Add Rule** to apply the settings.

   The new rule is displayed.



	Port	Private Address	Private Port	Public Port	Protocol	Description	Actions
	(76f0b669)	192.168.102.3	1234	5678	tcp		Update Rule

Displaying 1 item

Configuring Port-based DNAT Using the CLI

If you prefer, you can use the command-line interface to configure port-based DNAT for external access to VMs.

For more about using port-based DNAT, including instructions for using the web administration interface, see Configuring Port-based DNAT on a Virtual Router on page 71.

To add a network translation table entry, use a command of the following form:

```
~(keystone_admin)$ neutron portforwarding-create router --inside-addr addr \
--inside-port inside_port --outside-port outside_port --protocol protocol_type \
[--description desc]
```

where

router

is the name of the virtual router

addr

is the IP address of the VM interface connected to the virtual router

inside_port

is a port on the VM IP address

outside_port

is the port on the external gateway interface of the router

protocol_type

is the protocol (**tcp**, **udp**, **udp-lite**, **sctp**, or **dccp**) desc

is an optional text description of the entry

To obtain the VM IP address, use the **nova show** command and provide the instance UUID.

For example:

```
~(keystone_admin)$ nova show 915ea4da-1dd8-44ad-8a5b-375b9af6237f | grep network
| internal-net network          | 10.1.1.2        |
| tenant1-mgmt-net network      | 192.168.102.3   |
| tenant1-net network           | 172.31.1.2      |
...
~(keystone_admin)$ neutron portforwarding-create tenant1-router \
--inside-addr 192.168.102.3 --inside-port 1234 \
--outside-port 5678 --protocol tcp --description port_forwarding_rule
Created a new portforwarding:
+--------------+--------------------------------------+
| Field        | Value                                |
+--------------+--------------------------------------+
| description  | port_forwarding_rule                 |
| id           | c5cbf9a8-b041-4ce4-bb55-5527722eebf8 |
| inside_addr  | 192.168.102.3                        |
| inside_port  | 1234                                 |
| outside_port | 5678                                 |
| port_id      | 76f0b669-f493-4e5a-8302-4baeadab90c6 |
| protocol     | tcp                                  |
| router_id    | 722f8091-157b-488d-8372-0e1315d5177d |
| tenant_id    | b75b40702c054c78b8cf22860e2ded51     |
+--------------+--------------------------------------+
```

To modify an entry, use a command of the following form:

```
~(keystone_admin)$ neutron portforwarding-update router [--inside-addr addr] \
[--inside-port inside_port] [--outside-port outside_port] \
[--protocol protocol_type] [--description desc]
```

To view entries for a virtual router, use a command of the following form:

```
~(keystone_admin)$ neutron portforwarding-show router_uuid
```

To view entries for all routers, use the following command:

```
~(keystone_admin)$ neutron portforwarding-list
```

Prerequisites

Before you can use port-based DNAT, you must enable SNAT to activate support for network address translation tables. For more information, see Configuring SNAT on a Virtual Router on page 70.

Adding and Maintaining Routes for a VXLAN Network

You can add or delete routing table entries for hosts on a VXLAN network using the CLI.

Prerequisites

The compute node must be locked.

To add routes, use the following command.

```
~(keystone_admin)$ system host-route-add node ifname network prefix gateway metric
```

where

node

is the name or UUID of the compute node

ifname

is the name of the interface

network

is an IPv4 or IPv6 network address

prefix

is the netmask length for the network address

gateway

is the default gateway

metric

is the cost of the route (the number of hops)

To delete routes, use the following command.

```
~(keystone_admin)$ system host-route-delete uuid ifname network prefix gateway metric
```

where **uuid** is the UUID of the route to be deleted.

To list existing routes, including their UUIDs, use the following command.

```
~(keystone_admin)$ system host-route-list compute-0
```

Address Filtering on Virtual Interfaces

The AVS on compute nodes can be configured to filter out packets based on the source MAC address.

MAC addresses for virtual network interfaces on virtual machines are dynamically allocated by the system. For most scenarios, the assigned MAC addresses are expected to be used on all outgoing packets from the virtual machine instances. However, there are scenarios where the source MAC address is not expected to match the original assignment, such as when a L2 switch is implemented internally on the virtual machine.

By default, the AVS on compute nodes accepts any source MAC address on the attached virtual network interfaces. However, it can be configured to filter out all incoming packets with non-system-generated source MAC address, if required. When evaluating the use of the filtering capability, you must consider the following:

- Source MAC address filtering can be enabled and disabled by the administrator user only, not by tenants.
- Filtering is enabled on a per-tenant basis only. Higher granularity, such as per-instance filtering, is not supported.
- When enabled, source MAC address filtering applies to all new virtual interfaces created by the Neutron service. Address filtering is not active on virtual interfaces created before filtering is enabled.

You can use the following CLI command to enable source MAC address filtering:

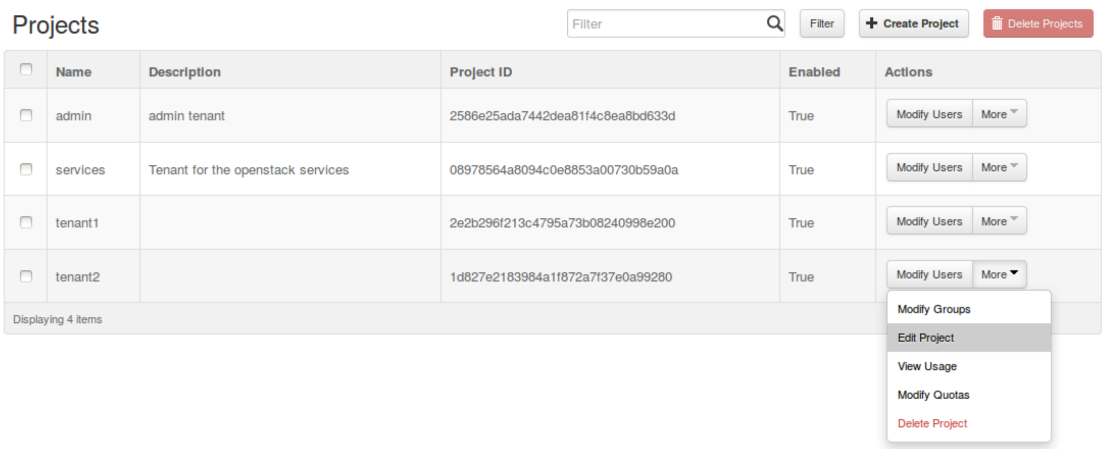
```
~(keystone_admin)$ neutron setting-update --tenant-id=<TENANTID> \
--mac-filtering={True|False}
```

Configuring Address Filtering on Virtual Interfaces

Filtering can be enabled or disabled from the web administration interface.

Procedure

1. As the administrator, select **Projects** from the **Identity** menu to display the Projects window.



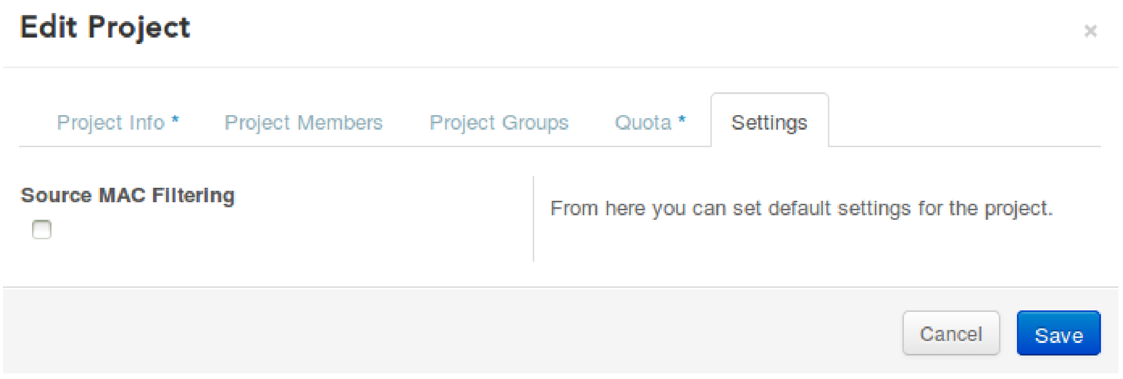
Projects

	Name	Description	Project ID	Enabled	Actions
	admin	admin tenant	2586e25ada7442dea81f4c8ea8bd633d	True	Modify Users More
	services	Tenant for the openstack services	08978564a8094c0e8853a00730b59a0a	True	Modify Users More
	tenant1		2e2b296f213c4795a73b08240998e200	True	Modify Users More
	tenant2		1d827e2183984a1f872a7f37e0a99280	True	Modify Users More

Displaying 4 items

2. Then select the option **Edit Project** from the **More** drop-down menu of the desired tenant, as illustrated below.

3. The Edit Project window is displayed. The filtering option is available from the tab **Settings**, as illustrated below.



Edit Project

Source MAC Filtering

From here you can set default settings for the project.