Red Hat OpenStack Platform 16.2

Configuring the Compute Service for Instance Creation

A guide to configuring and managing the Red Hat OpenStack Platform Compute (nova) service for creating instances

Last Updated: 2023-11-09

Red Hat OpenStack Platform 16.2 Configuring the Compute Service for Instance Creation

A guide to configuring and managing the Red Hat OpenStack Platform Compute (nova) service for creating instances

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Copyright © 2023 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

This guide provides concepts and procedures for cloud administrators to configure and manage the Red Hat OpenStack Platform Compute (nova) service using the OpenStack Client CLI.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE 5

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION 6

CHAPTER 1. COMPUTE SERVICE (NOVA) FUNCTIONALITY 7

CHAPTER 2. CONFIGURING THE COMPUTE SERVICE (NOVA) 9
2.1. CONFIGURING MEMORY FOR OVERALLOCATION 10
2.2. CALCULATING RESERVED HOST MEMORY ON COMPUTE NODES 10
2.3. CALCULATING SWAP SIZE 11

CHAPTER 3. CONFIGURING COMPUTE NODES FOR PERFORMANCE 12
3.1. CONFIGURING CPU PINNING ON COMPUTE NODES 12
3.1.1. Prerequisites 12
3.1.2. Designating Compute nodes for CPU pinning 12
3.1.3. Configuring Compute nodes for CPU pinning 16
3.1.4. Creating a dedicated CPU flavor for instances 17
3.1.5. Creating a shared CPU flavor for instances 18
3.1.6. Configuring CPU pinning on Compute nodes with simultaneous multithreading (SMT) 19
3.1.7. Additional resources 20
3.2. CONFIGURING EMULATOR THREADS 20
3.3. CONFIGURING HUGE PAGES ON COMPUTE NODES 21
3.3.1. Creating a huge pages flavor for instances 23
3.3.2. Mounting multiple huge page folders during first boot 23
3.4. CONFIGURING COMPUTE NODES TO USE FILE-BACKED MEMORY FOR INSTANCES 25
3.4.1. Changing the memory backing directory host disk 27

CHAPTER 4. CONFIGURING COMPUTE SERVICE STORAGE 28
4.1. CONFIGURATION OPTIONS FOR IMAGE CACHING 28
4.2. CONFIGURATION OPTIONS FOR INSTANCE EPHEMERAL STORAGE PROPERTIES 30
4.3. CONFIGURING SHARED INSTANCE STORAGE 33
4.4. CONFIGURING IMAGE DOWNLOADS DIRECTLY FROM RED HAT CEPH RADOS BLOCK DEVICE (RBD) 34
4.5. ADDITIONAL RESOURCES 35

CHAPTER 5. CONFIGURING PCI PASSTHROUGH 36
5.1. DESIGNATING COMPUTE NODES FOR PCI PASSTHROUGH 36
5.2. CONFIGURING A PCI PASSTHROUGH COMPUTE NODE 38
5.3. PCI PASSTHROUGH DEVICE TYPE FIELD 41
5.4. GUIDELINES FOR CONFIGURING NOVAPCIPASSTHROUGH 42

CHAPTER 6. CREATING AND MANAGING HOST AGGREGATES 44
6.1. ENABLING SCHEDULING ON HOST AGGREGATES 44
6.2. CREATING A HOST AGGREGATE 45
6.3. CREATING AN AVAILABILITY ZONE 46
6.4. DELETING A HOST AGGREGATE 47
6.5. CREATING A PROJECT-ISOLATED HOST AGGREGATE 48

CHAPTER 7. CONFIGURING INSTANCE SCHEDULING AND PLACEMENT 50
7.1. PREFILTERING USING THE PLACEMENT SERVICE 50
7.1.1. Filtering by requested image type support 51
7.1.2. Filtering by resource provider traits 51
7.1.2.1. Creating an image that requires or forbids a resource provider trait 51
7.1.2.2. Creating a flavor that requires or forbids a resource provider trait 53

7.1.3. Filtering by isolating host aggregates 54
7.1.4. Filtering by availability zone using the Placement service 56
7.2. CONFIGURING FILTERS AND WEIGHTS FOR THE COMPUTE SCHEDULER SERVICE 56
7.3. COMPUTE SCHEDULER FILTERS 57
7.4. COMPUTE SCHEDULER WEIGHTS 62

CHAPTER 8. CREATING FLAVORS FOR LAUNCHING INSTANCES 70
8.1. CREATING A FLAVOR 70
8.2. FLAVOR ARGUMENTS 71
8.3. FLAVOR METADATA 73

CHAPTER 9. ADDING METADATA TO INSTANCES 87
9.1. TYPES OF INSTANCE METADATA 87
9.2. ADDING A CONFIG DRIVE TO ALL INSTANCES 87
9.3. ADDING DYNAMIC METADATA TO INSTANCES 89

CHAPTER 10. CONFIGURING CPU FEATURE FLAGS FOR INSTANCES 91
10.1. PREREQUISITES 91
10.2. CONFIGURING CPU FEATURE FLAGS FOR INSTANCES 91

CHAPTER 11. CONFIGURING MANUAL NODE REBOOT TO DEFINE KERNELARGS 94
11.1. CONFIGURING MANUAL NODE REBOOT TO DEFINE KERNELARGS 94

CHAPTER 12. CONFIGURING AMD SEV COMPUTE NODES TO PROVIDE MEMORY ENCRYPTION FOR INSTANCES 96
12.1. SECURE ENCRYPTED VIRTUALIZATION (SEV) 96
12.2. DESIGNATING AMD SEV COMPUTE NODES FOR MEMORY ENCRYPTION 97
12.3. CONFIGURING AMD SEV COMPUTE NODES FOR MEMORY ENCRYPTION 100
12.4. CREATING AN IMAGE FOR MEMORY ENCRYPTION 102
12.5. CREATING A FLAVOR FOR MEMORY ENCRYPTION 103
12.6. LAUNCHING AN INSTANCE WITH MEMORY ENCRYPTION 103

CHAPTER 13. CONFIGURING NVDIMM COMPUTE NODES TO PROVIDE PERSISTENT MEMORY FOR INSTANCES 104
13.1. DESIGNATING COMPUTE NODES FOR PMEM 105
13.2. CONFIGURING A PMEM COMPUTE NODE 107

CHAPTER 14. CONFIGURING VIRTUAL GPUS FOR INSTANCES 109
14.1. SUPPORTED CONFIGURATIONS AND LIMITATIONS 109
14.2. CONFIGURING VGPU ON THE COMPUTE NODES 110
14.2.1. Prerequisites 110
14.2.2. Designating Compute nodes for vGPU 111
14.2.3. Configuring the Compute node for vGPU and deploying the overcloud 112
14.3. CREATING A CUSTOM GPU INSTANCE IMAGE 114
14.4. CREATING A VGPU FLAVOR FOR INSTANCES 115
14.5. LAUNCHING A VGPU INSTANCE 116
14.6. ENABLING PCI PASSTHROUGH FOR A GPU DEVICE 116

CHAPTER 15. CONFIGURING REAL-TIME COMPUTE 121
15.1. PREPARING COMPUTE NODES FOR REAL-TIME 121
15.2. DEPLOYING THE REAL-TIME COMPUTE ROLE 124
15.3. SAMPLE DEPLOYMENT AND TESTING SCENARIO 126
15.4. LAUNCHING AND TUNING REAL-TIME INSTANCES 128

CHAPTER 16. MANAGING INSTANCES 130
16.1. SECURING CONNECTIONS TO THE VNC CONSOLE OF AN INSTANCE 130

16.2. DATABASE CLEANING 131
16.2.1. Configuring database management 131
16.2.2. Configuration options for the Compute service automated database management 131
16.3. MIGRATING VIRTUAL MACHINE INSTANCES BETWEEN COMPUTE NODES 134
16.3.1. Migration types 135
16.3.2. Migration constraints 136
16.3.3. Preparing to migrate 139
16.3.4. Cold migrating an instance 139
16.3.5. Live migrating an instance 140
16.3.6. Checking migration status 141
16.3.7. Evacuating an instance 142
16.3.7.1. Evacuating one instance 143
16.3.7.2. Evacuating all instances on a host 144
16.3.8. Troubleshooting migration 145
16.3.8.1. Errors during migration 145
16.3.8.2. Never-ending live migration 146
16.3.8.3. Instance performance degrades after migration 147

MAKING OPEN SOURCE MORE INCLUSIVE

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

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Providing documentation feedback in Jira

Use the Create Issue form to provide feedback on the documentation. The Jira issue will be created in the Red Hat OpenStack Platform Jira project, where you can track the progress of your feedback.

1. Ensure that you are logged in to Jira. If you do not have a Jira account, create an account to submit feedback.
2. Click the following link to open a the **Create Issue** page: Create Issue
3. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.
4. Click **Create**.

CHAPTER 1. COMPUTE SERVICE (NOVA) FUNCTIONALITY

You use the Compute (nova) service to create, provision, and manage virtual machine instances and bare metal servers in a Red Hat OpenStack Platform (RHOSP) environment. The Compute service abstracts the underlying hardware that it runs on, rather than exposing specifics about the underlying host platforms. For example, rather than exposing the types and topologies of CPUs running on hosts, the Compute service exposes a number of virtual CPUs (vCPUs) and allows for overcommitting of these vCPUs.

The Compute service uses the KVM hypervisor to execute Compute service workloads. The libvirt driver interacts with QEMU to handle all interactions with KVM, and enables the creation of virtual machine instances. To create and provision instances, the Compute service interacts with the following RHOSP services:

- Identity (keystone) service for authentication.
- Placement service for resource inventory tracking and selection.
- Image Service (glance) for disk and instance images.
- Networking (neutron) service for provisioning the virtual or physical networks that instances connect to on boot.

The Compute service consists of daemon processes and services, named **nova-***. The following are the core Compute services:

Compute service (nova-compute)

This service creates, manages and terminates instances by using the libvirt for KVM or QEMU hypervisor APIs, and updates the database with instance states.

Compute conductor (nova-conductor)

This service mediates interactions between the Compute service and the database, which insulates Compute nodes from direct database access. Do not deploy this service on nodes where the **nova-compute** service runs.

Compute scheduler (nova-scheduler)

This service takes an instance request from the queue and determines on which Compute node to host the instance.

Compute API (nova-api)

This service provides the external REST API to users.

API database

This database tracks instance location information, and provides a temporary location for instances that are built but not scheduled. In multi-cell deployments, this database also contains cell mappings that specify the database connection for each cell.

Cell database

This database contains most of the information about instances. It is used by the API database, the conductor, and the Compute services.

Message queue

This messaging service is used by all services to communicate with each other within the cell and with the global services.

Compute metadata

This service stores data specific to instances. Instances access the metadata service at http://169.254.169.254 or over IPv6 at the link-local address fe80::a9fe:a9fe. The Networking

(neutron) service is responsible for forwarding requests to the metadata API server. You must use the **NeutronMetadataProxySharedSecret** parameter to set a secret keyword in the configuration of both the Networking service and the Compute service to allow the services to communicate. The Compute metadata service can be run globally, as part of the Compute API, or in each cell.

You can deploy more than one Compute node. The hypervisor that operates instances runs on each Compute node. Each Compute node requires a minimum of two network interfaces. The Compute node also runs a Networking service agent that connects instances to virtual networks and provides firewalling services to instances through security groups.

By default, director installs the overcloud with a single cell for all Compute nodes. This cell contains all the Compute services and databases that control and manage the virtual machine instances, and all the instances and instance metadata. For larger deployments, you can deploy the overcloud with multiple cells to accommodate a larger number of Compute nodes. You can add cells to your environment when you install a new overcloud or at any time afterwards. For more information, see Scaling Deployments with Compute Cells.

CHAPTER 2. CONFIGURING THE COMPUTE SERVICE (NOVA)

As a cloud administrator, you use environment files to customize the Compute (nova) service. Puppet generates and stores this configuration in the **/var/lib/config-data/puppet-generated/<nova_container>/etc/nova/nova.conf** file. Use the following configuration methods to customize the Compute service configuration, in the following order of precedence:

1. **Heat parameters** - as detailed in the Compute (nova) Parameters section in the *Overcloud Parameters* guide. The following example uses heat parameters to set the default scheduler filters, and configure an NFS backend for the Compute service:

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

2. **Puppet parameters** - as defined in **/etc/puppet/modules/nova/manifests/***:

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

NOTE

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

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

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

WARNING

If a heat parameter exists, use it instead of the Puppet parameter. If a Puppet parameter exists, but not a heat parameter, use the Puppet parameter instead of the manual override method. Use the manual override method only if there is no equivalent heat or Puppet parameter.

TIP

Follow the guidance in Identifying parameters that you want to modify to determine if a heat or Puppet parameter is available for customizing a particular configuration.

For more information about how to configure overcloud services, see Heat parameters in the *Advanced Overcloud Customization* guide.

2.1. CONFIGURING MEMORY FOR OVERALLOCATION

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

NOTE

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

Prerequisites

- You have calculated the swap size your node requires. For more information, see Calculating swap size.

Procedure

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

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

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

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

3. Add the **enable_swap.yaml** environment file to the stack with your other environment files and deploy the overcloud:

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

2.2. CALCULATING RESERVED HOST MEMORY ON COMPUTE NODES

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

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

- The hypervisor for each instance.

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

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

- Replace **vm_no** with the number of instances.
- Replace **avg_instance_size** with the average amount of memory each instance can use.
- Replace **overhead** with the hypervisor overhead required for each instance.
- Replace **resource1** and all resources up to **<resourcen>** with the number of a resource type on the node.
- Replace **resource_ram** with the amount of RAM each resource of this type requires.

2.3. CALCULATING SWAP SIZE

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

- overcommit_ratio = **NovaRAMAllocationRatio** - 1
- Minimum swap size (MB) = **(total_RAM * overcommit_ratio) + RHEL_min_swap**
- Recommended (maximum) swap size (MB) = **total_RAM * (overcommit_ratio + percentage_of_RAM_to_use_for_swap)**

The **percentage_of_RAM_to_use_for_swap** variable creates a buffer to account for QEMU overhead and any other resources consumed by the operating system or host services.

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

- Recommended (maximum) swap size = 64000 MB * (0 + 0.25) = 16000 MB

For information about how to calculate the **NovaReservedHostMemory** value, see Calculating reserved host memory on Compute nodes.

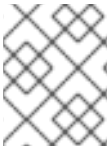
For information about how to determine the **RHEL_min_swap** value, see Recommended system swap space in the RHEL *Managing Storage Devices* guide.

CHAPTER 3. CONFIGURING COMPUTE NODES FOR PERFORMANCE

As a cloud administrator, you can configure the scheduling and placement of instances for optimal performance by creating customized flavors to target specialized workloads, including NFV and High Performance Computing (HPC).

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

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



NOTE

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

3.1. CONFIGURING CPU PINNING ON COMPUTE NODES

You can configure each instance CPU process to run on a dedicated host CPU by enabling CPU pinning on the Compute nodes. When an instance uses CPU pinning, each instance vCPU process is allocated its own host pCPU that no other instance vCPU process can use. Instances that run on Compute nodes with CPU pinning enabled have a NUMA topology. Each NUMA node of the instance NUMA topology maps to a NUMA node on the host Compute node.

You can configure the Compute scheduler to schedule instances with dedicated (pinned) CPUs and instances with shared (floating) CPUs on the same Compute node. To configure CPU pinning on Compute nodes that have a NUMA topology, you must complete the following:

1. Designate Compute nodes for CPU pinning.
2. Configure the Compute nodes to reserve host cores for pinned instance vCPU processes, floating instance vCPU processes, and host processes.
3. Deploy the overcloud.
4. Create a flavor for launching instances that require CPU pinning.
5. Create a flavor for launching instances that use shared, or floating, CPUs.

3.1.1. Prerequisites

- You know the NUMA topology of your Compute node.

3.1.2. Designating Compute nodes for CPU pinning

To designate Compute nodes for instances with pinned CPUs, you must create a new role file to configure the CPU pinning role, and configure a new overcloud flavor and CPU pinning resource class to use to tag the Compute nodes for CPU pinning.

Procedure

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

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

3. Generate a new roles data file named **roles_data_cpu_pinning.yaml** that includes the **Controller**, **Compute**, and **ComputeCPUPinning** roles:

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

4. Open **roles_data_cpu_pinning.yaml** and edit or add the following parameters and sections:

Section/Parameter	Current value	New value
Role comment	**Role: Compute**	**Role: ComputeCPUPinning**
Role name	**name: Compute**	**name: ComputeCPUPinning**
description	**Basic Compute Node role**	**CPU Pinning Compute Node role**
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputepinning-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-cpu-pinning.yaml**

5. Register the CPU pinning Compute nodes for the overcloud by adding them to your node definition template, **node.json** or **node.yaml**. For more information, see Registering nodes for the overcloud in the *Director Installation and Usage* guide.
6. Inspect the node hardware:

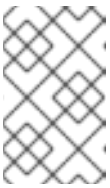
```
(undercloud)$ openstack overcloud node introspect \
 --all-manageable --provide
```

For more information, see Creating an inventory of the bare-metal node hardware in the *Director Installation and Usage* guide.

7. Create the **compute-cpu-pinning** overcloud flavor for CPU pinning Compute nodes:

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

- Replace **<ram_size_mb>** with the RAM of the bare metal node, in MB.
- Replace **<disk_size_gb>** with the size of the disk on the bare metal node, in GB.
- Replace **<no_vcpus>** with the number of CPUs on the bare metal node.



NOTE

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

8. Tag each bare metal node that you want to designate for CPU pinning with a custom CPU pinning resource class:

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

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

9. Associate the **compute-cpu-pinning** flavor with the custom CPU pinning resource class:

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_CPU_PINNING=1 \
 compute-cpu-pinning
```

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

NOTE

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

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

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

11. Optional: If the network topology of the **ComputeCPUPinning** role is different from the network topology of your **Compute** role, then create a custom network interface template. For more information, see Custom network interface templates in the *Advanced Overcloud Customization* guide.
If the network topology of the **ComputeCPUPinning** role is the same as the **Compute** role, then you can use the default network topology defined in **compute.yaml**.

12. Register the **Net::SoftwareConfig** of the **ComputeCPUPinning** role in your **network-environment.yaml** file:

```
resource_registry:
 OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
```

```
configs/compute.yaml
  OS::TripleO::ComputeCPUPinning::Net::SoftwareConfig: /home/stack/templates/nic-
configs/<cpu_pinning_net_top>.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
```

Replace **<cpu_pinning_net_top>** with the name of the file that contains the network topology of the **ComputeCPUPinning** role, for example, **compute.yaml** to use the default network topology.

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

```
parameter_defaults:
  OvercloudComputeCPUPinningFlavor: compute-cpu-pinning
  ComputeCPUPinningCount: 3
```

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

```
(undercloud)$ openstack baremetal node list --long -c "UUID" \
 -c "Instance UUID" -c "Resource Class" -c "Provisioning State" \
 -c "Power State" -c "Last Error" -c "Fault" -c "Name" -f json
```

Example output:

```
[
  {
    "Fault": null,
    "Instance UUID": "e8e60d37-d7c7-4210-acf7-f04b245582ea",
    "Last Error": null,
    "Name": "compute-0",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "baremetal.CPU-PINNING",
    "UUID": "b5a9ac58-63a7-49ba-b4ad-33d84000ccb4"
  },
  {
    "Fault": null,
    "Instance UUID": "3ec34c0b-c4f5-4535-9bd3-8a1649d2e1bd",
    "Last Error": null,
    "Name": "compute-1",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "compute",
    "UUID": "432e7f86-8da2-44a6-9b14-dfacdf611366"
  },
  {
    "Fault": null,
    "Instance UUID": "4992c2da-adde-41b3-bef1-3a5b8e356fc0",
    "Last Error": null,
    "Name": "controller-0",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "controller",
```

```
    "UUID": "474c2fc8-b884-4377-b6d7-781082a3a9c0"
  }
]
```

3.1.3. Configuring Compute nodes for CPU pinning

Configure CPU pinning on your Compute nodes based on the NUMA topology of the nodes. Reserve some CPU cores across all the NUMA nodes for the host processes for efficiency. Assign the remaining CPU cores to managing your instances.

This procedure uses the following NUMA topology, with eight CPU cores spread across two NUMA nodes, to illustrate how to configure CPU pinning:

Table 3.1. Example of NUMA Topology

NUMA Node 0		NUMA Node 1	
Core 0	Core 1	Core 2	Core 3
Core 4	Core 5	Core 6	Core 7

The procedure reserves cores 0 and 4 for host processes, cores 1, 3, 5 and 7 for instances that require CPU pinning, and cores 2 and 6 for floating instances that do not require CPU pinning.

Procedure

1. Create an environment file to configure Compute nodes to reserve cores for pinned instances, floating instances, and host processes, for example, **cpu_pinning.yaml**.

2. To schedule instances with a NUMA topology on NUMA-capable Compute nodes, add **NUMATopologyFilter** to the **NovaSchedulerDefaultFilters** parameter in your Compute environment file, if not already present:

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

   For more information on **NUMATopologyFilter**, see Compute scheduler filters .

3. To reserve physical CPU cores for the dedicated instances, add the following configuration to **cpu_pinning.yaml**:

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

4. To reserve physical CPU cores for the shared instances, add the following configuration to **cpu_pinning.yaml**:

```
parameter_defaults:
  ComputeCPUPinningParameters:
    ...
```

```
    NovaComputeCpuSharedSet: 2,6
```

5. To specify the amount of RAM to reserve for host processes, add the following configuration to **cpu_pinning.yaml**:

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

Replace **<ram>** with the amount of RAM to reserve in MB.

6. To ensure that host processes do not run on the CPU cores reserved for instances, set the parameter **IsolCpusList** to the CPU cores you have reserved for instances:

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

Specify the value of the **IsolCpusList** parameter using a list, or ranges, of CPU indices separated by a comma.

7. Add your new role and environment files to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -r /home/stack/templates/roles_data_cpu_pinning.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/cpu_pinning.yaml \
  -e /home/stack/templates/node-info.yaml
```

3.1.4. Creating a dedicated CPU flavor for instances

To enable your cloud users to create instances that have dedicated CPUs, you can create a flavor with a dedicated CPU policy for launching instances.

Prerequisites

- Simultaneous multithreading (SMT) is enabled on the host.
- The Compute node is configured to allow CPU pinning. For more information, see Configuring CPU pinning on the Compute nodes.

Procedure

1. Source the **overcloudrc** file:

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

2. Create a flavor for instances that require CPU pinning:

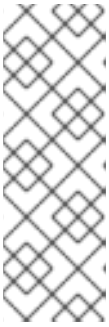
```
(overcloud)$ openstack flavor create --ram <size_mb> \
 --disk <size_gb> --vcpus <no_reserved_vcpus> pinned_cpus
```

3. To request pinned CPUs, set the **hw:cpu_policy** property of the flavor to **dedicated**:

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

4. To place each vCPU on thread siblings, set the **hw:cpu_thread_policy** property of the flavor to **require**:

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



NOTE

- If the host does not have an SMT architecture or enough CPU cores with available thread siblings, scheduling fails. To prevent this, set **hw:cpu_thread_policy** to **prefer** instead of **require**. The **prefer** policy is the default policy that ensures that thread siblings are used when available.
- If you use **hw:cpu_thread_policy=isolate**, you must have SMT disabled or use a platform that does not support SMT.

Verification

1. To verify the flavor creates an instance with dedicated CPUs, use your new flavor to launch an instance:

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

2. To verify correct placement of the new instance, enter the following command and check for **OS-EXT-SRV-ATTR:hypervisor_hostname** in the output:

```
(overcloud)$ openstack server show pinned_cpu_instance
```

3.1.5. Creating a shared CPU flavor for instances

To enable your cloud users to create instances that use shared, or floating, CPUs, you can create a flavor with a shared CPU policy for launching instances.

Prerequisites

- The Compute node is configured to reserve physical CPU cores for the shared CPUs. For more information, see Configuring CPU pinning on the Compute nodes.

Procedure

1. Source the **overcloudrc** file:

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

2. Create a flavor for instances that do not require CPU pinning:

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

3. To request floating CPUs, set the **hw:cpu_policy** property of the flavor to **shared**:

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

Verification

1. To verify the flavor creates an instance that uses the shared CPUs, use your new flavor to launch an instance:

```
(overcloud)$ openstack server create --flavor floating_cpus \
 --image <image> floating_cpu_instance
```

2. To verify correct placement of the new instance, enter the following command and check for **OS-EXT-SRV-ATTR:hypervisor_hostname** in the output:

```
(overcloud)$ openstack server show floating_cpu_instance
```

3.1.6. Configuring CPU pinning on Compute nodes with simultaneous multithreading (SMT)

If a Compute node supports simultaneous multithreading (SMT), group thread siblings together in either the dedicated or the shared set. Thread siblings share some common hardware which means it is possible for a process running on one thread sibling to impact the performance of the other thread sibling.

For example, the host identifies four logical CPU cores in a dual core CPU with SMT: 0, 1, 2, and 3. Of these four, there are two pairs of thread siblings:

- Thread sibling 1: logical CPU cores 0 and 2
- Thread sibling 2: logical CPU cores 1 and 3

In this scenario, do not assign logical CPU cores 0 and 1 as dedicated and 2 and 3 as shared. Instead, assign 0 and 2 as dedicated and 1 and 3 as shared.

The files **/sys/devices/system/cpu/cpuN/topology/thread_siblings_list**, where **N** is the logical CPU number, contain the thread pairs. You can use the following command to identify which logical CPU cores are thread siblings:

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

The following output indicates that logical CPU core 0 and logical CPU core 2 are threads on the same core:

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

3.1.7. Additional resources

- Discovering your NUMA node topology in the *Network Functions Virtualization Planning and Configuration Guide*.
- CPUs and NUMA nodes in the *Network Functions Virtualization Product Guide*.

3.2. CONFIGURING EMULATOR THREADS

Compute nodes have overhead tasks associated with the hypervisor for each instance, known as emulator threads. By default, emulator threads run on the same CPUs as the instance, which impacts the performance of the instance.

You can configure the emulator thread policy to run emulator threads on separate CPUs to those the instance uses.

NOTE

To avoid packet loss, you must never preempt the vCPUs in an NFV deployment.

Procedure

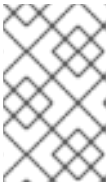
1. Log in to the undercloud as the **stack** user.
2. Open your Compute environment file.
3. To reserve physical CPU cores for instances that require CPU pinning, configure the **NovaComputeCpuDedicatedSet** parameter in the Compute environment file. For example, the following configuration sets the dedicated CPUs on a Compute node with a 32-core CPU:

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

   For more information, see Configuring CPU pinning on the Compute nodes.

4. To reserve physical CPU cores for the emulator threads, configure the **NovaComputeCpuSharedSet** parameter in the Compute environment file. For example, the following configuration sets the shared CPUs on a Compute node with a 32-core CPU:

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



NOTE

The Compute scheduler also uses the CPUs in the shared set for instances that run on shared, or floating, CPUs. For more information, see Configuring CPU pinning on Compute nodes

5. Add the Compute scheduler filter **NUMATopologyFilter** to the **NovaSchedulerDefaultFilters** parameter, if not already present.

6. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

7. Configure a flavor that runs emulator threads for the instance on a dedicated CPU, which is selected from the shared CPUs configured using **NovaComputeCpuSharedSet**:

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

For more information about configuration options for **hw:emulator_threads_policy**, see Emulator threads policy in Flavor metadata.

3.3. CONFIGURING HUGE PAGES ON COMPUTE NODES

As a cloud administrator, you can configure Compute nodes to enable instances to request huge pages.

Procedure

1. Open your Compute environment file.
2. Configure the amount of huge page memory to reserve on each NUMA node for processes that are not instances:

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

- Replace the **size** value for each node with the size of the allocated huge page. Set to one of the following valid values:
  - 2048 (for 2MB)
  - 1GB
- Replace the **count** value for each node with the number of huge pages used by OVS per NUMA node. For example, for 4096 of socket memory used by Open vSwitch, set this to 2.

3. Configure huge pages on the Compute nodes:

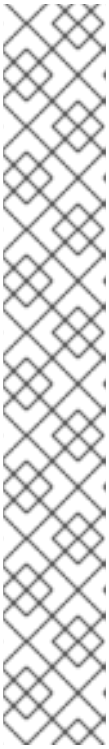
```
parameter_defaults:
  ComputeParameters:
    ...
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32"
```

NOTE

If you configure multiple huge page sizes, you must also mount the huge page folders during first boot. For more information, see Mounting multiple huge page folders during first boot.

4. Optional: To allow instances to allocate 1GB huge pages, configure the CPU feature flags, **NovaLibvirtCPUModelExtraFlags**, to include **pdpe1gb**:

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



NOTE

- CPU feature flags do not need to be configured to allow instances to only request 2 MB huge pages.
- You can only allocate 1G huge pages to an instance if the host supports 1G huge page allocation.
- You only need to set **NovaLibvirtCPUModelExtraFlags** to **pdpe1gb** when **NovaLibvirtCPUMode** is set to **host-model** or **custom**.
- If the host supports **pdpe1gb**, and **host-passthrough** is used as the **NovaLibvirtCPUMode**, then you do not need to set **pdpe1gb** as a **NovaLibvirtCPUModelExtraFlags**. The **pdpe1gb** flag is only included in Opteron_G4 and Opteron_G5 CPU models, it is not included in any of the Intel CPU models supported by QEMU.
- To mitigate for CPU hardware issues, such as Microarchitectural Data Sampling (MDS), you might need to configure other CPU flags. For more information, see RHOS Mitigation for MDS ("Microarchitectural Data Sampling") Security Flaws.

5. To avoid loss of performance after applying Meltdown protection, configure the CPU feature flags, **NovaLibvirtCPUModelExtraFlags**, to include **+pcid**:

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

TIP

For more information, see Reducing the performance impact of *Meltdown* CVE fixes for OpenStack guests with "PCID" CPU feature flag.

6. Add **NUMATopologyFilter** to the **NovaSchedulerDefaultFilters** parameter, if not already present.

7. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

-

3.3.1. Creating a huge pages flavor for instances

To enable your cloud users to create instances that use huge pages, you can create a flavor with the **hw:mem_page_size** extra spec key for launching instances.

Prerequisites

- The Compute node is configured for huge pages. For more information, see Configuring huge pages on Compute nodes.

Procedure

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

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

2. To request huge pages, set the **hw:mem_page_size** property of the flavor to the required size:

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

   Set **hw:mem_page_size** to one of the following valid values:

   - **large** - Selects the largest page size supported on the host, which may be 2 MB or 1 GB on x86_64 systems.
   - **small** - (Default) Selects the smallest page size supported on the host. On x86_64 systems this is 4 kB (normal pages).
   - **any** - Selects the largest available huge page size, as determined by the libvirt driver.
   - <pagesize>: (String) Set an explicit page size if the workload has specific requirements. Use an integer value for the page size in KB, or any standard suffix. For example: 4KB, 2MB, 2048, 1GB.

3. To verify the flavor creates an instance with huge pages, use your new flavor to launch an instance:

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

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

3.3.2. Mounting multiple huge page folders during first boot

You can configure the Compute service (nova) to handle multiple page sizes as part of the first boot process. The first boot process adds the heat template configuration to all nodes the first time you boot the nodes. Subsequent inclusion of these templates, such as updating the overcloud stack, does not run these scripts.

Procedure

1. Create a first boot template file, **hugepages.yaml**, that runs a script to create the mounts for the huge page folders. You can use the **OS::TripleO::MultipartMime** resource type to send the configuration script:

```
heat_template_version: <version>

description: >
  Huge pages configuration

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

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

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

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

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

The **config** script in this template performs the following tasks:

a. Filters the hosts to create the mounts for the huge page folders on, by specifying hostnames that match **'co?mp'**. You can update the filter grep pattern for specific computes as required.

b. Masks the default **dev-hugepages.mount systemd** unit file to enable new mounts to be created using the page size.

c. Ensures that the folders are created first.

d. Creates **systemd** mount units for each **pagesize**.

e. Runs **systemd daemon-reload** after the first loop, to include the newly created unit files.

f. Enables each mount for 2M and 1G pagesizes. You can update this loop to include additional pagesizes, as required.

2. Optional: The **/dev** folder is automatically bind mounted to the **nova_compute** and **nova_libvirt** containers. If you have used a different destination for the huge page mounts, then you need to pass the mounts to the the **nova_compute** and **nova_libvirt** containers:

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

3. Register your heat template as the **OS::TripleO::NodeUserData** resource type in your **~/templates/firstboot.yaml** environment file:

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

IMPORTANT

You can only register the **NodeUserData** resources to one heat template for each resource. Subsequent usage overrides the heat template to use.

4. Add your first boot environment file to the stack with your other environment files and deploy the overcloud:

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

3.4. CONFIGURING COMPUTE NODES TO USE FILE-BACKED MEMORY FOR INSTANCES

You can use file-backed memory to expand your Compute node memory capacity, by allocating files within the libvirt memory backing directory as instance memory. You can configure the amount of host disk that is available for instance memory, and the location on the disk of the instance memory files.

The Compute service reports the capacity configured for file-backed memory to the Placement service as the total system memory capacity. This allows the Compute node to host more instances than would normally fit within the system memory.

To use file-backed memory for instances, you must enable file-backed memory on the Compute node.

Limitations

- You cannot live migrate instances between Compute nodes that have file-backed memory enabled and Compute nodes that do not have file-backed memory enabled.
- File-backed memory is not compatible with huge pages. Instances that use huge pages cannot start on a Compute node with file-backed memory enabled. Use host aggregates to ensure that instances that use huge pages are not placed on Compute nodes with file-backed memory enabled.
- File-backed memory is not compatible with memory overcommit.
- You cannot reserve memory for host processes using **NovaReservedHostMemory**. When file-backed memory is in use, reserved memory corresponds to disk space not set aside for file-backed memory. File-backed memory is reported to the Placement service as the total system memory, with RAM used as cache memory.

Prerequisites

- **NovaRAMAllocationRatio** must be set to "1.0" on the node and any host aggregate the node is added to.
- **NovaReservedHostMemory** must be set to "0".

Procedure

1. Open your Compute environment file.
2. Configure the amount of host disk space, in MiB, to make available for instance RAM, by adding the following parameter to your Compute environment file:

```
parameter_defaults:
  NovaLibvirtFileBackedMemory: 102400
```

3. Optional: To configure the directory to store the memory backing files, set the **QemuMemoryBackingDir** parameter in your Compute environment file. If not set, the memory backing directory defaults to **/var/lib/libvirt/qemu/ram/**.

NOTE

You must locate your backing store in a directory at or above the default directory location, **/var/lib/libvirt/qemu/ram/**.

You can also change the host disk for the backing store. For more information, see Changing the memory backing directory host disk.

4. Save the updates to your Compute environment file.

5. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

3.4.1. Changing the memory backing directory host disk

You can move the memory backing directory from the default primary disk location to an alternative disk.

Procedure

1. Create a file system on the alternative backing device. For example, enter the following command to create an **ext4** filesystem on **/dev/sdb**:

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

2. Mount the backing device. For example, enter the following command to mount **/dev/sdb** on the default libvirt memory backing directory:

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

NOTE

The mount point must match the value of the **QemuMemoryBackingDir** parameter.

CHAPTER 4. CONFIGURING COMPUTE SERVICE STORAGE

You create an instance from a base image, which the Compute service copies from the Image (glance) service, and caches locally on the Compute nodes. The instance disk, which is the back end for the instance, is also based on the base image.

You can configure the Compute service to store ephemeral instance disk data locally on the host Compute node or remotely on either an NFS share or Ceph cluster. Alternatively, you can also configure the Compute service to store instance disk data in persistent storage provided by the Block Storage (Cinder) service.

You can configure image caching for your environment, and configure the performance and security of the instance disks. You can also configure the Compute service to download images directly from the RBD image repository without using the Image service API, when the Image service (glance) uses Red Hat Ceph RADOS Block Device (RBD) as the back end.

4.1. CONFIGURATION OPTIONS FOR IMAGE CACHING

Use the parameters detailed in the following table to configure how the Compute service implements and manages an image cache on Compute nodes.

Table 4.1. Compute (nova) service image cache parameters

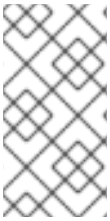
Configuration method	Parameter	Description
Puppet	**nova::compute::image_cache::manager_interval**	Specifies the number of seconds to wait between runs of the image cache manager, which manages base image caching on Compute nodes. The Compute service uses this period to perform automatic removal of unused cached images when **nova::compute::image_cache::remove_unused_base_images** is set to **True**. Set to **0** to run at the default metrics interval of 60 seconds (not recommended). Set to **-1** to disable the image cache manager. Default: **2400**

Configuration method	Parameter	Description
Puppet	**nova::compute::image_cache::precache_concurrency**	Specifies the maximum number of Compute nodes that can pre-cache images in parallel. NOTE ● Setting this parameter to a high number can cause slower pre-cache performance and might result in a DDoS on the Image service. ● Setting this parameter to a low number reduces the load on the Image service, but can cause longer runtime to completion as the pre-cache is performed as a more sequential operation. Default: **1**
Puppet	**nova::compute::image_cache::remove_unused_base_images**	Set to **True** to automatically remove unused base images from the cache at intervals configured by using **manager_interval**. Images are defined as unused if they have not been accessed during the time specified by using **NovaImageCacheTTL**. Default: **True**
Puppet	**nova::compute::image_cache::remove_unused_resized_minimum_age_seconds**	Specifies the minimum age that an unused resized base image must be to be removed from the cache, in seconds. Unused resized base images younger than this will not be removed. Set to **undef** to disable. Default: **3600**

Configuration method	Parameter	Description
Puppet	**nova::compute::image_cache::subdirectory_name**	Specifies the name of the folder where cached images are stored, relative to **$instances_path**. Default: **_base**
Heat	**NovaImageCacheTTL**	Specifies the length of time in seconds that the Compute service should continue caching an image when it is no longer used by any instances on the Compute node. The Compute service deletes images cached on the Compute node that are older than this configured lifetime from the cache directory until they are needed again. Default: 86400 (24 hours)

4.2. CONFIGURATION OPTIONS FOR INSTANCE EPHEMERAL STORAGE PROPERTIES

Use the parameters detailed in the following table to configure the performance and security of ephemeral storage used by instances.



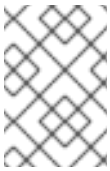
NOTE

Red Hat OpenStack Platform (RHOSP) does not support the LVM image type for instance disks. Therefore, the **[libvirt]/volume_clear** configuration option, which wipes ephemeral disks when instances are deleted, is not supported because it only applies when the instance disk image type is LVM.

Table 4.2. Compute (nova) service instance ephemeral storage parameters

Configuration method	Parameter	Description
Puppet	**nova::compute::default_ephemeral_format**	Specifies the default format that is used for a new ephemeral volume. Set to one of the following valid values: • **ext2** • **ext3** • **ext4** The **ext4** format provides much faster initialization times than ext3 for new, large disks. Default: **ext4**

Configuration method	Parameter	Description
Puppet	**nova::compute::force_raw_images**	Set to **True** to convert non-raw cached base images to raw format. The raw image format uses more space than other image formats, such as qcow2. Non-raw image formats use more CPU for compression. When set to **False**, the Compute service removes any compression from the base image during compression to avoid CPU bottlenecks. Set to **False** if you have a system with slow I/O or low available space to reduce input bandwidth. Default: **True**
Puppet	**nova::compute::use_cow_images**	Set to **True** to use CoW (Copy on Write) images in qcow2 format for instance disks. With CoW, depending on the backing store and host caching, there might be better concurrency achieved by having each instance operate on its own copy. Set to **False** to use the raw format. Raw format uses more space for common parts of the disk image. Default: **True**
Puppet	**nova::compute::libvirt::preallocate_images**	Specifies the preallocation mode for instance disks. Set to one of the following valid values: ● **none** - No storage is provisioned at instance start. ● **space** - The Compute service fully allocates storage at instance start by running **fallocate(1)** on the instance disk images. This reduces CPU overhead and file fragmentation, improves I/O performance, and helps guarantee the required disk space. Default: **none**

Configuration method	Parameter	Description
Hieradata override	**DEFAULT/resize_fs_using_block_device**	Set to **True** to enable direct resizing of the base image by accessing the image over a block device. This is only necessary for images with older versions of cloud-init that cannot resize themselves. This parameter is not enabled by default because it enables the direct mounting of images which might otherwise be disabled for security reasons. Default: **False**
Hieradata override	**[libvirt]/images_type**	Specifies the image type to use for instance disks. Set to one of the following valid values: • **raw** • **qcow2** • **flat** • **rbd** • **default**  NOTE RHOSP does not support the LVM image type for instance disks. When set to a valid value other than **default** the image type supersedes the configuration of **use_cow_images**. If **default** is specified, the configuration of **use_cow_images** determines the image type: • If **use_cow_images** is set to **True** (default) then the image type is **qcow2**. • If **use_cow_images** is set to **False** then the image type is **Flat**. The default value is determined by the configuration of **NovaEnableRbdBackend**: • **NovaEnableRbdBackend: False** Default: **default** • **NovaEnableRbdBackend: True** Default: **rbd**

4.3. CONFIGURING SHARED INSTANCE STORAGE

By default, when you launch an instance, the instance disk is stored as a file in the instance directory, **/var/lib/nova/instances**. You can configure an NFS storage backend for the Compute service to store these instance files on shared NFS storage.

Prerequisites

- You must be using NFSv4 or later. Red Hat OpenStack Platform (RHOSP) does not support earlier versions of NFS. For more information, see the Red Hat Knowledgebase solution RHOS NFSv4-Only Support Notes.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

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

3. Create an environment file to configure shared instance storage, for example, **nfs_instance_disk_backend.yaml**.
4. To configure an NFS backend for instance files, add the following configuration to **nfs_instance_disk_backend.yaml**:

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

   Replace **<nfs_share>** with the NFS share directory to mount for instance file storage, for example, **'192.168.122.1:/export/nova'** or **'192.168.24.1:/var/nfs'**. If using IPv6, use both double and single-quotes, e.g. **'"[fdd0::1]:/export/nova"'**.

5. Optional: The default mount SELinux context for NFS storage when NFS backend storage is enabled is **'context=system_u:object_r:nfs_t:s0'**. Add the following parameter to amend the mount options for the NFS instance file storage mount point:

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

   Replace **<additional_nfs_mount_options>** with a comma-separated list of the mount options you want to use for NFS instance file storage. For more information on the available mount options, see the **mount** man page:

```
$ man 8 mount.
```

6. Save the updates to your environment file.
7. Add your new environment file to the stack with your other environment files and deploy the overcloud:

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

4.4. CONFIGURING IMAGE DOWNLOADS DIRECTLY FROM RED HAT CEPH RADOS BLOCK DEVICE (RBD)

When the Image service (glance) uses Red Hat Ceph RADOS Block Device (RBD) as the back end, and the Compute service uses local file-based ephemeral storage, you can configure the Compute service to download images directly from the RBD image repository without using the Image service API. This reduces the time it takes to download an image to the Compute node image cache at instance boot time, which improves instance launch time.

Prerequisites

- The Image service back end is a Red Hat Ceph RADOS Block Device (RBD).
- The Compute service is using a local file-based ephemeral store for the image cache and instance disks.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Open your Compute environment file.
3. To download images directly from the RBD back end, add the following configuration to your Compute environment file:

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

4. Optional: If the Image service is configured to use multiple Red Hat Ceph Storage back ends, add the following configuration to your Compute environment file to identify the RBD back end to download images from:

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

   Replace **<rbd_backend_id>** with the ID used to specify the back end in the **GlanceMultistoreConfig** configuration, for example **rbd2_store**.

5. Add the following configuration to your Compute environment file to specify the Image service RBD back end, and the maximum length of time that the Compute service waits to connect to the Image service RBD back end, in seconds:

```
parameter_defaults:
```

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

6. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

7. To verify that the Compute service downloads images directly from RBD, create an instance then check the instance debug log for the entry "Attempting to export RBD image:".

4.5. ADDITIONAL RESOURCES

- Configuring the Compute service (nova)

CHAPTER 5. CONFIGURING PCI PASSTHROUGH

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

IMPORTANT

Using PCI passthrough with routed provider networks

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

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

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

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

Prerequisites

- The Compute nodes have the required PCI devices.

5.1. DESIGNATING COMPUTE NODES FOR PCI PASSTHROUGH

To designate Compute nodes for instances with physical PCI devices attached, you must create a new role file to configure the PCI passthrough role, and configure a new overcloud flavor and PCI passthrough resource class to use to tag the Compute nodes for PCI passthrough.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

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

3. Generate a new roles data file named **roles_data_pci_passthrough.yaml** that includes the **Controller**, **Compute**, and **ComputePCI** roles:

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

4. Open **roles_data_pci_passthrough.yaml** and edit or add the following parameters and sections:

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

5. Register the PCI passthrough Compute nodes for the overcloud by adding them to your node definition template, **node.json** or **node.yaml**. For more information, see Registering nodes for the overcloud in the *Director Installation and Usage* guide.

6. Inspect the node hardware:

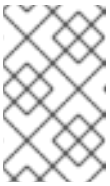
```
(undercloud)$ openstack overcloud node introspect \
 --all-manageable --provide
```

For more information, see Creating an inventory of the bare-metal node hardware in the *Director Installation and Usage* guide.

7. Create the **compute-pci-passthrough** overcloud flavor for PCI passthrough Compute nodes:

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

- Replace **<ram_size_mb>** with the RAM of the bare metal node, in MB.
- Replace **<disk_size_gb>** with the size of the disk on the bare metal node, in GB.
- Replace **<no_vcpus>** with the number of CPUs on the bare metal node.



NOTE

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

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

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

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

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

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

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

NOTE

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

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

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

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

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

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

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

5.2. CONFIGURING A PCI PASSTHROUGH COMPUTE NODE

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

Procedure

1. Create an environment file to configure the Controller node on the overcloud for PCI passthrough, for example, **pci_passthrough_controller.yaml**.

2. Add **PciPassthroughFilter** to the **NovaSchedulerDefaultFilters** parameter in **pci_passthrough_controller.yaml**:

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

3. To specify the PCI alias for the devices on the Controller node, add the following configuration to **pci_passthrough_controller.yaml**:

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

For more information about configuring the **device_type** field, see PCI passthrough device type field.

NOTE

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

4. Optional: To set a default NUMA affinity policy for PCI passthrough devices, add **numa_policy** to the **nova::pci::aliases:** configuration from step 3:

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

5. To configure the Compute node on the overcloud for PCI passthrough, create an environment file, for example, **pci_passthrough_compute.yaml**.

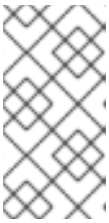
6. To specify the available PCIs for the devices on the Compute node, use the **vendor_id** and **product_id** options to add all matching PCI devices to the pool of PCI devices available for passthrough to instances. For example, to add all Intel® Ethernet Controller X710 devices to the pool of PCI devices available for passthrough to instances, add the following configuration to **pci_passthrough_compute.yaml**:

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

For more information about how to configure **NovaPCIPassthrough**, see Guidelines for configuring **NovaPCIPassthrough**.

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

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



NOTE

The Compute node aliases must be identical to the aliases on the Controller node. Therefore, if you added **numa_affinity** to **nova::pci::aliases** in **pci_passthrough_controller.yaml**, then you must also add it to **nova::pci::aliases** in **pci_passthrough_compute.yaml**.

8. To enable IOMMU in the server BIOS of the Compute nodes to support PCI passthrough, add the **KernelArgs** parameter to **pci_passthrough_compute.yaml**. For example, use the following **KernalArgs** settings to enable an Intel IOMMU:

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

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

NOTE

When you first add the **KernelArgs** parameter to the configuration of a role, the overcloud nodes are automatically rebooted. If required, you can disable the automatic rebooting of nodes and instead perform node reboots manually after each overcloud deployment. For more information, see Configuring manual node reboot to define **KernelArgs**.

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

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

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

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

11. Optional: To override the default NUMA affinity policy for PCI passthrough devices, you can add the NUMA affinity policy property key to the flavor or the image:

    - To override the default NUMA affinity policy by using the flavor, add the **hw:pci_numa_affinity_policy** property key:

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

      For more information about the valid values for **hw:pci_numa_affinity_policy**, see Flavor metadata.

    - To override the default NUMA affinity policy by using the image, add the **hw_pci_numa_affinity_policy** property key:

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

NOTE

If you set the NUMA affinity policy on both the image and the flavor then the property values must match. The flavor setting takes precedence over the image and default settings. Therefore, the configuration of the NUMA affinity policy on the image only takes effect if the property is not set on the flavor.

Verification

1. Create an instance with a PCI passthrough device:

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

2. Log in to the instance as a cloud user. For more information, see Connecting to an instance.

3. To verify that the PCI device is accessible from the instance, enter the following command from the instance:

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

5.3. PCI PASSTHROUGH DEVICE TYPE FIELD

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

type-PF

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

type-VF

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

type-PCI

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

NOTE

You must configure the Compute and Controller nodes with the same **device_type**.

5.4. GUIDELINES FOR CONFIGURING NOVAPCIPASSTHROUGH

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

Configuring the address parameter

The **address** parameter specifies the PCI address of the device. You can set the value of the **address** parameter using either a String or a **dict** mapping.

String format

If you specify the address using a string you can include wildcards (*), as shown in the following example:

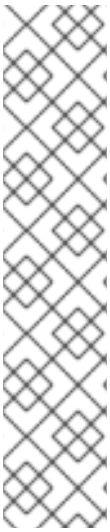
```
NovaPCIPassthrough:
  -
    address: "*:0a:00.*"
    physical_network: physnet1
```

Dictionary format

If you specify the address using the dictionary format you can include regular expression syntax, as shown in the following example:

```
NovaPCIPassthrough:
  -
    address:
      domain: ".*"
      bus: "02"
```

```
  slot: "01"
  function: "[0-2]"
physical_network: net1
```



NOTE

The Compute service restricts the configuration of **address** fields to the following maximum values:

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

The Compute service supports PCI devices with a 16-bit address domain. The Compute service ignores PCI devices with a 32-bit address domain.

CHAPTER 6. CREATING AND MANAGING HOST AGGREGATES

As a cloud administrator, you can partition a Compute deployment into logical groups for performance or administrative purposes. Red Hat OpenStack Platform (RHOSP) provides the following mechanisms for partitioning logical groups:

Host aggregate

A host aggregate is a grouping of Compute nodes into a logical unit based on attributes such as the hardware or performance characteristics. You can assign a Compute node to one or more host aggregates.
You can map flavors and images to host aggregates by setting metadata on the host aggregate, and then matching flavor extra specs or image metadata properties to the host aggregate metadata. The Compute scheduler can use this metadata to schedule instances when the required filters are enabled. Metadata that you specify in a host aggregate limits the use of that host to any instance that has the same metadata specified in its flavor or image.

You can configure weight multipliers for each host aggregate by setting the **xxx_weight_multiplier** configuration option in the host aggregate metadata.

You can use host aggregates to handle load balancing, enforce physical isolation or redundancy, group servers with common attributes, or separate classes of hardware.

When you create a host aggregate, you can specify a zone name. This name is presented to cloud users as an availability zone that they can select.

Availability zones

An availability zone is the cloud user view of a host aggregate. A cloud user cannot view the Compute nodes in the availability zone, or view the metadata of the availability zone. The cloud user can only see the name of the availability zone.
You can assign each Compute node to only one availability zone. You can configure a default availability zone where instances will be scheduled when the cloud user does not specify a zone. You can direct cloud users to use availability zones that have specific capabilities.

6.1. ENABLING SCHEDULING ON HOST AGGREGATES

To schedule instances on host aggregates that have specific attributes, update the configuration of the Compute scheduler to enable filtering based on the host aggregate metadata.

Procedure

1. Open your Compute environment file.
2. Add the following values to the **NovaSchedulerDefaultFilters** parameter, if they are not already present:
   - **AggregateInstanceExtraSpecsFilter**: Add this value to filter Compute nodes by host aggregate metadata that match flavor extra specs.

NOTE

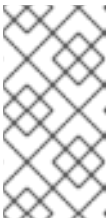
For this filter to perform as expected, you must scope the flavor extra specs by prefixing the **extra_specs** key with the **aggregate_instance_extra_specs:** namespace.

- **AggregateImagePropertiesIsolation**: Add this value to filter Compute nodes by host aggregate metadata that match image metadata properties.

NOTE

To filter host aggregate metadata by using image metadata properties, the host aggregate metadata key must match a valid image metadata property. For information about valid image metadata properties, see Image metadata.

- **AvailabilityZoneFilter**: Add this value to filter by availability zone when launching an instance.



NOTE

Instead of using the **AvailabilityZoneFilter** Compute scheduler service filter, you can use the Placement service to process availability zone requests. For more information, see Filtering by availability zone using the Placement service.

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

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

6.2. CREATING A HOST AGGREGATE

As a cloud administrator, you can create as many host aggregates as you require.

Procedure

1. To create a host aggregate, enter the following command:

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

   Replace **<aggregate_name>** with the name you want to assign to the host aggregate.

2. Add metadata to the host aggregate:

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

   - Replace **<key=value>** with the metadata key-value pair. If you are using the **AggregateInstanceExtraSpecsFilter** filter, the key can be any arbitrary string, for example, **ssd=true**. If you are using the **AggregateImagePropertiesIsolation** filter, the key must match a valid image metadata property. For more information about valid image metadata properties, see Image metadata.

- Replace **<aggregate_name>** with the name of the host aggregate.

3. Add the Compute nodes to the host aggregate:

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

- Replace **<aggregate_name>** with the name of the host aggregate to add the Compute node to.
- Replace **<host_name>** with the name of the Compute node to add to the host aggregate.

4. Create a flavor or image for the host aggregate:

- Create a flavor:

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

- Create an image:

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

5. Set one or more key-value pairs on the flavor or image that match the key-value pairs on the host aggregate.

- To set the key-value pairs on a flavor, use the scope **aggregate_instance_extra_specs**:

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

- To set the key-value pairs on an image, use valid image metadata properties as the key:

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

6.3. CREATING AN AVAILABILITY ZONE

As a cloud administrator, you can create an availability zone that cloud users can select when they create an instance.

Procedure

1. To create an availability zone, you can create a new availability zone host aggregate, or make an existing host aggregate an availability zone:

   a. To create a new availability zone host aggregate, enter the following command:

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

- Replace **<availability_zone>** with the name you want to assign to the availability zone.
- Replace **<aggregate_name>** with the name you want to assign to the host aggregate.

b. To make an existing host aggregate an availability zone, enter the following command:

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

- Replace **<availability_zone>** with the name you want to assign to the availability zone.
- Replace **<aggregate_name>** with the name of the host aggregate.

2. Optional: Add metadata to the availability zone:

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

- Replace **<key=value>** with your metadata key-value pair. You can add as many key-value properties as required.
- Replace **<aggregate_name>** with the name of the availability zone host aggregate.

3. Add Compute nodes to the availability zone host aggregate:

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

- Replace **<aggregate_name>** with the name of the availability zone host aggregate to add the Compute node to.
- Replace **<host_name>** with the name of the Compute node to add to the availability zone.

6.4. DELETING A HOST AGGREGATE

To delete a host aggregate, you first remove all the Compute nodes from the host aggregate.

Procedure

1. To view a list of all the Compute nodes assigned to the host aggregate, enter the following command:

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

2. To remove all assigned Compute nodes from the host aggregate, enter the following command for each Compute node:

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

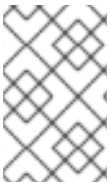
- Replace **<aggregate_name>** with the name of the host aggregate to remove the Compute node from.
- Replace **<host_name>** with the name of the Compute node to remove from the host aggregate.

3. After you remove all the Compute nodes from the host aggregate, enter the following command to delete the host aggregate:

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

6.5. CREATING A PROJECT-ISOLATED HOST AGGREGATE

You can create a host aggregate that is available only to specific projects. Only the projects that you assign to the host aggregate can launch instances on the host aggregate.



NOTE

Project isolation uses the Placement service to filter host aggregates for each project. This process supersedes the functionality of the **AggregateMultiTenancyIsolation** filter. You therefore do not need to use the **AggregateMultiTenancyIsolation** filter.

Procedure

1. Open your Compute environment file.
2. To schedule project instances on the project-isolated host aggregate, set the **NovaSchedulerLimitTenantsToPlacementAggregate** parameter to **True** in the Compute environment file.
3. Optional: To ensure that only the projects that you assign to a host aggregate can create instances on your cloud, set the **NovaSchedulerPlacementAggregateRequiredForTenants** parameter to **True**.

NOTE

NovaSchedulerPlacementAggregateRequiredForTenants is **False** by default. When this parameter is **False**, projects that are not assigned to a host aggregate can create instances on any host aggregate.

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

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

6. Create the host aggregate.
7. Retrieve the list of project IDs:

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

8. Use the **filter_tenant_id<suffix>** metadata key to assign projects to the host aggregate:

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

- Replace **<ID0>**, **<ID1>**, and all IDs up to **<IDn>** with unique values for each project filter that you want to create.
- Replace **<project_id0>**, **<project_id1>**, and all project IDs up to **<project_idn>** with the ID of each project that you want to assign to the host aggregate.
- Replace **<aggregate_name>** with the name of the project-isolated host aggregate. For example, use the following syntax to assign projects **78f1**, **9d3t**, and **aa29** to the host aggregate **project-isolated-aggregate**:

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

TIP

You can create a host aggregate that is available only to a single specific project by omitting the suffix from the **filter_tenant_id** metadata key:

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

Additional resources

- For more information on creating a host aggregate, see Creating and managing host aggregates.

CHAPTER 7. CONFIGURING INSTANCE SCHEDULING AND PLACEMENT

The Compute scheduler service determines on which Compute node or host aggregate to place an instance. When the Compute (nova) service receives a request to launch or move an instance, it uses the specifications provided in the request, the flavor, and the image to find a suitable host. For example, a flavor can specify the traits an instance requires a host to have, such as the type of storage disk, or the Intel CPU instruction set extension.

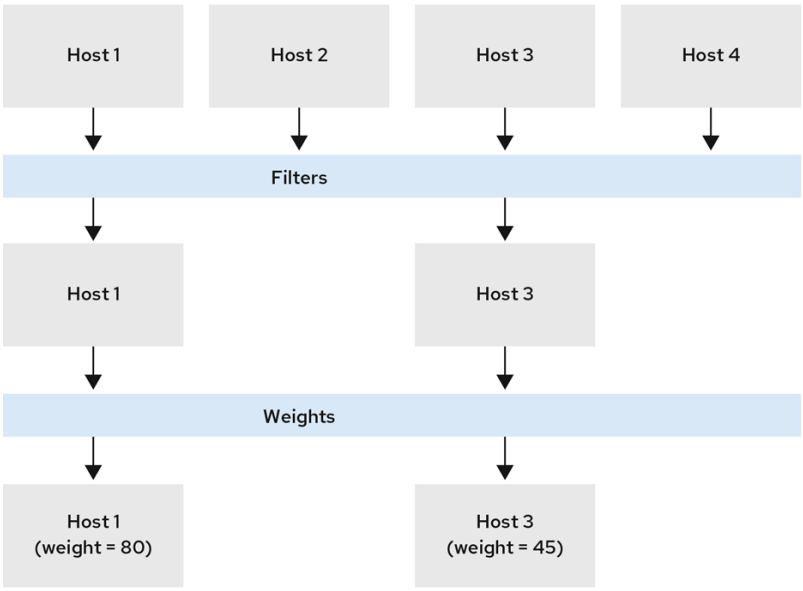
The Compute scheduler service uses the configuration of the following components, in the following order, to determine on which Compute node to launch or move an instance:

1. **Placement service prefilters**: The Compute scheduler service uses the Placement service to filter the set of candidate Compute nodes based on specific attributes. For example, the Placement service automatically excludes disabled Compute nodes.

2. **Filters**: Used by the Compute scheduler service to determine the initial set of Compute nodes on which to launch an instance.

3. **Weights**: The Compute scheduler service prioritizes the filtered Compute nodes using a weighting system. The highest weight has the highest priority.

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



7.1. PREFILTERING USING THE PLACEMENT SERVICE

The Compute service (nova) interacts with the Placement service when it creates and manages instances. The Placement service tracks the inventory and usage of resource providers, such as a Compute node, a shared storage pool, or an IP allocation pool, and their available quantitative resources, such as the available vCPUs. Any service that needs to manage the selection and consumption of resources can use the Placement service.

The Placement service also tracks the mapping of available qualitative resources to resource providers, such as the type of storage disk trait a resource provider has.

The Placement service applies prefilters to the set of candidate Compute nodes based on Placement service resource provider inventories and traits. You can create prefilters based on the following criteria:

- Supported image types
- Traits
- Projects or tenants
- Availability zone

7.1.1. Filtering by requested image type support

You can exclude Compute nodes that do not support the disk format of the image used to launch an instance. This is useful when your environment uses Red Hat Ceph Storage as an ephemeral backend, which does not support QCOW2 images. Enabling this feature ensures that the scheduler does not send requests to launch instances using a QCOW2 image to Compute nodes backed by Red Hat Ceph Storage.

Procedure

1. Open your Compute environment file.
2. To exclude Compute nodes that do not support the disk format of the image used to launch an instance, set the **NovaSchedulerQueryImageType** parameter to **True** in the Compute environment file.
3. Save the updates to your Compute environment file.
4. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

7.1.2. Filtering by resource provider traits

Each resource provider has a set of traits. Traits are the qualitative aspects of a resource provider, for example, the type of storage disk, or the Intel CPU instruction set extension.

The Compute node reports its capabilities to the Placement service as traits. An instance can specify which of these traits it requires, or which traits the resource provider must not have. The Compute scheduler can use these traits to identify a suitable Compute node or host aggregate to host an instance.

To enable your cloud users to create instances on hosts that have particular traits, you can define a flavor that requires or forbids a particular trait, and you can create an image that requires or forbids a particular trait.

For a list of the available traits, see the **os-traits** library. You can also create custom traits, as required.

7.1.2.1. Creating an image that requires or forbids a resource provider trait

You can create an instance image that your cloud users can use to launch instances on hosts that have particular traits.

Procedure

1. Create a new image:

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

2. Identify the trait you require a host or host aggregate to have. You can select an existing trait, or create a new trait:

   - To use an existing trait, list the existing traits to retrieve the trait name:

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

   - To create a new trait, enter the following command:

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

     Custom traits must begin with the prefix **CUSTOM_** and contain only the letters A through Z, the numbers 0 through 9 and the underscore "_" character.

3. Collect the existing resource provider traits of each host:

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

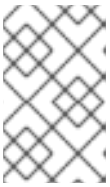
4. Check the existing resource provider traits for the traits you require a host or host aggregate to have:

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

5. If the traits you require are not already added to the resource provider, then add the existing traits and your required traits to the resource providers for each host:

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

   Replace **<TRAIT_NAME>** with the name of the trait that you want to add to the resource provider. You can use the **--trait** option more than once to add additional traits, as required.



NOTE

This command performs a full replacement of the traits for the resource provider. Therefore, you must retrieve the list of existing resource provider traits on the host and set them again to prevent them from being removed.

6. To schedule instances on a host or host aggregate that has a required trait, add the trait to the image extra specs. For example, to schedule instances on a host or host aggregate that supports AVX-512, add the following trait to the image extra specs:

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

7. To filter out hosts or host aggregates that have a forbidden trait, add the trait to the image extra specs. For example, to prevent instances from being scheduled on a host or host aggregate that supports multi-attach volumes, add the following trait to the image extra specs:

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

7.1.2.2. Creating a flavor that requires or forbids a resource provider trait

You can create flavors that your cloud users can use to launch instances on hosts that have particular traits.

Procedure

1. Create a flavor:

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

2. Identify the trait you require a host or host aggregate to have. You can select an existing trait, or create a new trait:

   - To use an existing trait, list the existing traits to retrieve the trait name:

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

   - To create a new trait, enter the following command:

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

     Custom traits must begin with the prefix **CUSTOM_** and contain only the letters A through Z, the numbers 0 through 9 and the underscore "_" character.

3. Collect the existing resource provider traits of each host:

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

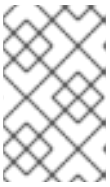
4. Check the existing resource provider traits for the traits you require a host or host aggregate to have:

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

5. If the traits you require are not already added to the resource provider, then add the existing traits and your required traits to the resource providers for each host:

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

Replace **<TRAIT_NAME>** with the name of the trait that you want to add to the resource provider. You can use the **--trait** option more than once to add additional traits, as required.



NOTE

This command performs a full replacement of the traits for the resource provider. Therefore, you must retrieve the list of existing resource provider traits on the host and set them again to prevent them from being removed.

6. To schedule instances on a host or host aggregate that has a required trait, add the trait to the flavor extra specs. For example, to schedule instances on a host or host aggregate that supports AVX-512, add the following trait to the flavor extra specs:

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

7. To filter out hosts or host aggregates that have a forbidden trait, add the trait to the flavor extra specs. For example, to prevent instances from being scheduled on a host or host aggregate that supports multi-attach volumes, add the following trait to the flavor extra specs:

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

7.1.3. Filtering by isolating host aggregates

You can restrict scheduling on a host aggregate to only those instances whose flavor and image traits match the metadata of the host aggregate. The combination of flavor and image metadata must require all the host aggregate traits to be eligible for scheduling on Compute nodes in that host aggregate.

Procedure

1. Open your Compute environment file.
2. To isolate host aggregates to host only instances whose flavor and image traits match the aggregate metadata, set the **NovaSchedulerEnableIsolatedAggregateFiltering** parameter to **True** in the Compute environment file.
3. Save the updates to your Compute environment file.
4. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

5. Identify the traits you want to isolate the host aggregate for. You can select an existing trait, or create a new trait:

   - To use an existing trait, list the existing traits to retrieve the trait name:

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

   - To create a new trait, enter the following command:

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

     Custom traits must begin with the prefix **CUSTOM_** and contain only the letters A through Z, the numbers 0 through 9 and the underscore "_" character.

6. Collect the existing resource provider traits of each Compute node:

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

7. Check the existing resource provider traits for the traits you want to isolate the host aggregate for:

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

8. If the traits you require are not already added to the resource provider, then add the existing traits and your required traits to the resource providers for each Compute node in the host aggregate:

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

   Replace **<TRAIT_NAME>** with the name of the trait that you want to add to the resource provider. You can use the **--trait** option more than once to add additional traits, as required.

   NOTE

   This command performs a full replacement of the traits for the resource provider. Therefore, you must retrieve the list of existing resource provider traits on the host and set them again to prevent them from being removed.

9. Repeat steps 6 - 8 for each Compute node in the host aggregate.

10. Add the metadata property for the trait to the host aggregate:

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

11. Add the trait to a flavor or an image:

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

7.1.4. Filtering by availability zone using the Placement service

You can use the Placement service to honor availability zone requests. To use the Placement service to filter by availability zone, placement aggregates must exist that match the membership and UUID of the availability zone host aggregates.

Procedure

1. Open your Compute environment file.
2. To use the Placement service to filter by availability zone, set the **NovaSchedulerQueryPlacementForAvailabilityZone** parameter to **True** in the Compute environment file.
3. Remove the **AvailabilityZoneFilter** filter from the **NovaSchedulerDefaultFilters** parameter.
4. Save the updates to your Compute environment file.
5. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

Additional resources

- For more information on creating a host aggregate to use as an availability zone, see Creating an availability zone.

7.2. CONFIGURING FILTERS AND WEIGHTS FOR THE COMPUTE SCHEDULER SERVICE

You need to configure the filters and weights for the Compute scheduler service to determine the initial set of Compute nodes on which to launch an instance.

Procedure

1. Open your Compute environment file.
2. Add the filters you want the scheduler to use to the **NovaSchedulerDefaultFilters** parameter, for example:

```
parameter_defaults:
  NovaSchedulerDefaultFilters:
AggregateInstanceExtraSpecsFilter,ComputeFilter,ComputeCapabilitiesFilter,ImagePropertiesF
```

```
ilter
```

3. Specify which attribute to use to calculate the weight of each Compute node, for example:

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

For more information on the available attributes, see Compute scheduler weights.

4. Optional: Configure the multiplier to apply to each weigher. For example, to specify that the available RAM of a Compute node has a higher weight than the other default weighers, and that the Compute scheduler prefers Compute nodes with more available RAM over those nodes with less available RAM, use the following configuration:

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

TIP

You can also set multipliers to a negative value. In the above example, to prefer Compute nodes with less available RAM over those nodes with more available RAM, set **ram_weight_multiplier** to **-2.0**.

5. Save the updates to your Compute environment file.

6. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

Additional resources

- For a list of the available Compute scheduler service filters, see Compute scheduler filters.
- For a list of the available weight configuration options, see Compute scheduler weights.

7.3. COMPUTE SCHEDULER FILTERS

You configure the **NovaSchedulerDefaultFilters** parameter in your Compute environment file to specify the filters the Compute scheduler must apply when selecting an appropriate Compute node to host an instance. The default configuration applies the following filters:

- **AvailabilityZoneFilter**: The Compute node must be in the requested availability zone.

- **ComputeFilter**: The Compute node can service the request.
- **ComputeCapabilitiesFilter**: The Compute node satisfies the flavor extra specs.
- **ImagePropertiesFilter**: The Compute node satisfies the requested image properties.
- **ServerGroupAntiAffinityFilter**: The Compute node is not already hosting an instance in a specified group.
- **ServerGroupAffinityFilter**: The Compute node is already hosting instances in a specified group.

You can add and remove filters. The following table describes all the available filters.

Table 7.1. Compute scheduler filters

Filter	Description
AggregateImagePropertiesIsolation	Use this filter to match the image metadata of an instance with host aggregate metadata. If any of the host aggregate metadata matches the metadata of the image, then the Compute nodes that belong to that host aggregate are candidates for launching instances from that image. The scheduler only recognises valid image metadata properties. For details on valid image metadata properties, see Image metadata properties.
AggregateInstanceExtraSpecsFilter	Use this filter to match namespaced properties defined in the flavor extra specs of an instance with host aggregate metadata. You must scope your flavor **extra_specs** keys by prefixing them with the **aggregate_instance_extra_specs:** namespace. If any of the host aggregate metadata matches the metadata of the flavor extra spec, then the Compute nodes that belong to that host aggregate are candidates for launching instances from that image.
AggregateIoOpsFilter	Use this filter to filter hosts by I/O operations with a per-aggregate **filter_scheduler/max_io_ops_per_host** value. If the per-aggregate value is not found, the value falls back to the global setting. If the host is in more than one aggregate and more than one value is found, the scheduler uses the minimum value.
AggregateMultiTenancyIsolation	Use this filter to limit the availability of Compute nodes in project-isolated host aggregates to a specified set of projects. Only projects specified using the **filter_tenant_id** metadata key can launch instances on Compute nodes in the host aggregate. For more information, see Creating a project-isolated host aggregate. NOTE The project can still place instances on other hosts. To restrict this, use the **NovaSchedulerPlacementAggregateRequiredForTenants** parameter.

Filter	Description
AggregateNumInstancesFilter	Use this filter to limit the number of instances each Compute node in an aggregate can host. You can configure the maximum number of instances per-aggregate by using the **filter_scheduler/max_instances_per_host** parameter. If the per-aggregate value is not found, the value falls back to the global setting. If the Compute node is in more than one aggregate, the scheduler uses the lowest **max_instances_per_host** value.
AggregateTypeAffinityFilter	Use this filter to pass hosts if no flavor metadata key is set, or the flavor aggregate metadata value contains the name of the requested flavor. The value of the flavor metadata entry is a string that may contain either a single flavor name or a comma-separated list of flavor names, such as **m1.nano** or **m1.nano,m1.small**.
AllHostsFilter	Use this filter to consider all available Compute nodes for instance scheduling. NOTE Using this filter does not disable other filters.
AvailabilityZoneFilter	Use this filter to launch instances on a Compute node in the availability zone specified by the instance.
ComputeCapabilitiesFilter	Use this filter to match namespaced properties defined in the flavor extra specs of an instance against the Compute node capabilities. You must prefix the flavor extra specs with the **capabilities:** namespace. A more efficient alternative to using the **ComputeCapabilitiesFilter** filter is to use CPU traits in your flavors, which are reported to the Placement service. Traits provide consistent naming for CPU features. For more information, see Filtering by using resource provider traits.
ComputeFilter	Use this filter to pass all Compute nodes that are operational and enabled. This filter should always be present.

DifferentHostFilter	Use this filter to enable scheduling of an instance on a different Compute node from a set of specific instances. To specify these instances when launching an instance, use the **--hint** argument with**different_host** as the key and the instance UUID as the value:
	`$ openstack server create --image cedef40a-ed67-4d10-800e-17455edce175 \`
	`  --flavor 1 --hint different_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 \`
	`  --hint different_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1`

Filter	Description
ImagePropertiesFilter	Use this filter to filter Compute nodes based on the following properties defined on the instance image: • **hw_architecture** - Corresponds to the architecture of the host, for example, x86, ARM, and Power. • **img_hv_type** - Corresponds to the hypervisor type, for example, KVM, QEMU, Xen, and LXC. • **img_hv_requested_version** - Corresponds to the hypervisor version the Compute service reports. • **hw_vm_mode** - Corresponds to the hyperviser type, for example hvm, xen, uml, or exe. Compute nodes that can support the specified image properties contained in the instance are passed to the scheduler. For more information on image properties, see Image metadata properties.

IsolatedHostsFilter	Use this filter to only schedule instances with isolated images on isolated Compute nodes. You can also prevent non-isolated images from being used to build instances on isolated Compute nodes by configuring **filter_scheduler/restrict_isolated_hosts_to_isolated_images**.
	To specify the isolated set of images and hosts use the **filter_scheduler/isolated_hosts** and **filter_scheduler/isolated_images** configuration options, for example:
	`parameter_defaults:`
	`  ComputeExtraConfig:`
	`  nova::config::nova_config:`
	`  filter_scheduler/isolated_hosts:`
	`  value: server1, server2`
	`  filter_scheduler/isolated_images:`
	`  value: 342b492c-128f-4a42-8d3a-c5088cf27d13, ebd267a6-ca86-4d6c-9a0e-bd132d6b7d09`

IoOpsFilter	Use this filter to filter out hosts that have concurrent I/O operations that exceed the configured **filter_scheduler/max_io_ops_per_host**, which specifies the maximum number of I/O intensive instances allowed to run on the host.

Filter	Description

MetricsFilter	Use this filter to limit scheduling to Compute nodes that report the metrics configured by using **metrics/weight_setting**.
	
	To use this filter, add the following configuration to your Compute environment file:
	
	`parameter_defaults:`
	`  ComputeExtraConfig:`
	`  nova::config::nova_config:`
	`  DEFAULT/compute_monitors:`
	`  value: 'cpu.virt_driver'`
	
	By default, the Compute scheduler service updates the metrics every 60 seconds. To ensure the metrics are up-to-date, you can increase the frequency at which the metrics data is refreshed using the **update_resources_interval** configuration option. For example, use the following configuration to refresh the metrics data every 2 seconds:
	
	`parameter_defaults:`
	`  ComputeExtraConfig:`
	`  nova::config::nova_config:`
	`  DEFAULT/update_resources_interval:`
	`  value: '2'`

NUMATopologyFilter	Use this filter to schedule instances with a NUMA topology on NUMA-capable Compute nodes. Use flavor **extra_specs** and image properties to specify the NUMA topology for an instance. The filter tries to match the instance NUMA topology to the Compute node topology, taking into consideration the over-subscription limits for each host NUMA cell.
NumInstancesFilter	Use this filter to filter out Compute nodes that have more instances running than specified by the **max_instances_per_host** option.
PciPassthroughFilter	Use this filter to schedule instances on Compute nodes that have the devices that the instance requests by using the flavor **extra_specs**. Use this filter if you want to reserve nodes with PCI devices, which are typically expensive and limited, for instances that request them.

SameHostFilter	Use this filter to enable scheduling of an instance on the same Compute node as a set of specific instances. To specify these instances when launching an instance, use the **--hint** argument with **same_host** as the key and the instance UUID as the value:
	
	`$ openstack server create --image cedef40a-ed67-4d10-800e-17455edce175 \`
	`  --flavor 1 --hint same_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 \`
	`  --hint same_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1`

Filter	Description

ServerGroupAffinityFilter	Use this filter to schedule instances in an affinity server group on the same Compute node. To create the server group, enter the following command:
	`$ openstack server group create --policy affinity <group_name>`
	To launch an instance in this group, use the **--hint** argument with **group** as the key and the group UUID as the value:
	`$ openstack server create --image <image> \`
	`  --flavor <flavor> \`
	`  --hint group=<group_uuid> <instance_name>`

ServerGroupAntiAffinityFilter	Use this filter to schedule instances that belong to an anti-affinity server group on different Compute nodes. To create the server group, enter the following command:
	`$ openstack server group create --policy anti-affinity <group_name>`
	To launch an instance in this group, use the **--hint** argument with **group** as the key and the group UUID as the value:
	`$ openstack server create --image <image> \`
	`  --flavor <flavor> \`
	`  --hint group=<group_uuid> <instance_name>`

SimpleCIDRAffinityFilter	Use this filter to schedule instances on Compute nodes that have a specific IP subnet range. To specify the required range, use the **--hint** argument to pass the keys **build_near_host_ip** and **cidr** when launching an instance:
	`$ openstack server create --image <image> \`
	`  --flavor <flavor> \`
	`  --hint build_near_host_ip=<ip_address> \`
	`  --hint cidr=<subnet_mask> <instance_name>`

7.4. COMPUTE SCHEDULER WEIGHTS

Each Compute node has a weight that the scheduler can use to prioritize instance scheduling. After the Compute scheduler applies the filters, it selects the Compute node with the largest weight from the remaining candidate Compute nodes.

The Compute scheduler determines the weight of each Compute node by performing the following tasks:

1. The scheduler normalizes each weight to a value between 0.0 and 1.0.
2. The scheduler multiplies the normalized weight by the weigher multiplier.

The Compute scheduler calculates the weight normalization for each resource type by using the lower and upper values for the resource availability across the candidate Compute nodes:

- Nodes with the lowest availability of a resource (minval) are assigned '0'.
- Nodes with the highest availability of a resource (maxval) are assigned '1'.
- Nodes with resource availability within the minval – maxval range are assigned a normalized weight calculated by using the following formula:

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

If all the Compute nodes have the same availability for a resource then they are all normalized to 0.

For example, the scheduler calculates the normalized weights for available vCPUs across 10 Compute nodes, each with a different number of available vCPUs, as follows:

Compute node	1	2	3	4	5	6	7	8	9	10
No of vCPUs	5	5	10	10	15	20	20	15	10	5
Normalized weight	0	0	0.33	0.33	0.67	1	1	0.67	0.33	0

The Compute scheduler uses the following formula to calculate the weight of a Compute node:

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

The following table describes the available configuration options for weights.

NOTE

Weights can be set on host aggregates using the aggregate metadata key with the same name as the options detailed in the following table. If set on the host aggregate, the host aggregate value takes precedence.

Table 7.2. Compute scheduler weights

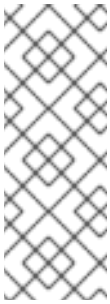
Configuration option	Type	Description

Configuration option	Type	Description
filter_scheduler/weight_classes	String	Use this parameter to configure which of the following attributes to use for calculating the weight of each Compute node: • **nova.scheduler.weights.ram.RAMWeigher** - Weighs the available RAM on the Compute node. • **nova.scheduler.weights.cpu.CPUWeigher** - Weighs the available CPUs on the Compute node. • **nova.scheduler.weights.disk.DiskWeigher** - Weighs the available disks on the Compute node. • **nova.scheduler.weights.metrics.MetricsWeigher** - Weighs the metrics of the Compute node. • **nova.scheduler.weights.affinity.ServerGroupSoftAffinityWeigher** - Weighs the proximity of the Compute node to other nodes in the given instance group. • **nova.scheduler.weights.affinity.ServerGroupSoftAntiAffinityWeigher** - Weighs the proximity of the Compute node to other nodes in the given instance group. • **nova.scheduler.weights.compute.BuildFailureWeigher** - Weighs Compute nodes by the number of recent failed boot attempts. • **nova.scheduler.weights.io_ops.IoOpsWeigher** - Weighs Compute nodes by their workload. • **nova.scheduler.weights.pci.PCIWeigher** - Weighs Compute nodes by their PCI availability. • **nova.scheduler.weights.cross_cell.CrossCellWeigher** - Weighs Compute nodes based on which cell they are in, giving preference to Compute nodes in the source cell when moving an instance. • **nova.scheduler.weights.all_weighers** - (Default) Uses all the above weighers.

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

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

Configuration option	Type	Description
filter_scheduler/pci_weight_multiplier	Positive floating point	Use this parameter to specify the multiplier to use to weigh hosts based on the number of PCI devices on the host and the number of PCI devices requested by an instance. If an instance requests PCI devices, then the more PCI devices a Compute node has the higher the weight allocated to the Compute node. For example, if there are three hosts available, one with a single PCI device, one with multiple PCI devices and one without any PCI devices, then the Compute scheduler prioritizes these hosts based on the demands of the instance. The scheduler should prefer the first host if the instance requests one PCI device, the second host if the instance requires multiple PCI devices and the third host if the instance does not request a PCI device. Configure this option to prevent non-PCI instances from occupying resources on hosts with PCI devices. Default: 1.0
filter_scheduler/host_subset_size	Integer	Use this parameter to specify the size of the subset of filtered hosts from which to select the host. You must set this option to at least 1. A value of 1 selects the first host returned by the weighing functions. The scheduler ignores any value less than 1 and uses 1 instead. Set to a value greater than 1 to prevent multiple scheduler processes handling similar requests selecting the same host, creating a potential race condition. By selecting a host randomly from the N hosts that best fit the request, the chance of a conflict is reduced. However, the higher you set this value, the less optimal the chosen host may be for a given request. Default: 1
filter_scheduler/soft_affinity_weight_multiplier	Positive floating point	Use this parameter to specify the multiplier to use to weigh hosts for group soft-affinity. **NOTE** You need to specify the microversion when creating a group with this policy: `$ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity <group_name>` Default: 1.0

Configuration option	Type	Description
filter_scheduler/soft_anti_affinity_weight_multiplier	Positive floating point	Use this parameter to specify the multiplier to use to weigh hosts for group soft-anti-affinity.  **NOTE** You need to specify the microversion when creating a group with this policy: `$ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity <group_name>` Default: 1.0
metrics/weight_multiplier	Floating point	Use this parameter to specify the multiplier to use for weighting metrics. By default, **weight_multiplier=1.0**, which spreads instances across possible hosts. Set to a number greater than 1.0 to increase the effect of the metric on the overall weight. Set to a number between 0.0 and 1.0 to reduce the effect of the metric on the overall weight. Set to 0.0 to ignore the metric value and return the value of the **weight_of_unavailable** option. Set to a negative number to prioritize the host with lower metrics, and stack instances in hosts. Default: 1.0

Configuration option	Type	Description
metrics/weight_setting	Comma-separated list of **metric=ratio** pairs	Use this parameter to specify the metrics to use for weighting, and the ratio to use to calculate the weight of each metric. Valid metric names: • **cpu.frequency** - CPU frequency • **cpu.user.time** - CPU user mode time • **cpu.kernel.time** - CPU kernel time • **cpu.idle.time** - CPU idle time • **cpu.iowait.time** - CPU I/O wait time • **cpu.user.percent** - CPU user mode percentage • **cpu.kernel.percent** - CPU kernel percentage • **cpu.idle.percent** - CPU idle percentage • **cpu.iowait.percent** - CPU I/O wait percentage • **cpu.percent** - Generic CPU use Example: **weight_setting=cpu.user.time=1.0**
metrics/required	Boolean	Use this parameter to specify how to handle configured **metrics/weight_setting** metrics that are unavailable: • **True** - Metrics are required. If the metric is unavailable, an exception is raised. To avoid the exception, use the **MetricsFilter** filter in **NovaSchedulerDefaultFilters**. • **False** - The unavailable metric is treated as a negative factor in the weighing process. Set the returned value by using the **weight_of_unavailable** configuration option.
metrics/weight_of_unavailable	Floating point	Use this parameter to specify the weight to use if any **metrics/weight_setting** metric is unavailable, and **metrics/required=False**. Default: -10000.0

CHAPTER 8. CREATING FLAVORS FOR LAUNCHING INSTANCES

An instance flavor is a resource template that specifies the virtual hardware profile for the instance. Cloud users must specify a flavor when they launch an instance.

A flavor can specify the quantity of the following resources the Compute service must allocate to an instance:

- The number of vCPUs.
- The RAM, in MB.
- The root disk, in GB.
- The virtual storage, including secondary ephemeral storage and swap disk.

You can specify who can use flavors by making the flavor public to all projects, or private to specific projects or domains.

Flavors can use metadata, also referred to as "extra specs", to specify instance hardware support and quotas. The flavor metadata influences the instance placement, resource usage limits, and performance. For a complete list of available metadata properties, see Flavor metadata.

You can also use the flavor metadata keys to find a suitable host aggregate to host the instance, by matching the **extra_specs** metadata set on the host aggregate. To schedule an instance on a host aggregate, you must scope the flavor metadata by prefixing the **extra_specs** key with the **aggregate_instance_extra_specs:** namespace. For more information, see Creating and managing host aggregates.

A Red Hat OpenStack Platform (RHOSP) deployment includes the following set of default public flavors that your cloud users can use.

Table 8.1. Default Flavors

Name	vCPUs	RAM	Root Disk Size
m1.nano	1	128 MB	1 GB
m1.micro	1	192 MB	1 GB

NOTE

Behavior set using flavor properties override behavior set using images. When a cloud user launches an instance, the properties of the flavor they specify override the properties of the image they specify.

8.1. CREATING A FLAVOR

You can create and manage specialized flavors for specific functionality or behaviors, for example:

- Change default memory and capacity to suit the underlying hardware needs.

- Add metadata to force a specific I/O rate for the instance or to match a host aggregate.

Procedure

1. Create a flavor that specifies the basic resources to make available to an instance:

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

   - Replace **<size_mb>** with the size of RAM to allocate to an instance created with this flavor.
   - Replace **<size_gb>** with the size of root disk to allocate to an instance created with this flavor.
   - Replace **<no_vcpus>** with the number of vCPUs to reserve for an instance created with this flavor.
   - Optional: Specify the **--private** and **--project** options to make the flavor accessible only by a particular project or group of users. Replace **<project_id>** with the ID of the project that can use this flavor to create instances. If you do not specify the accessibility, the flavor defaults to public, which means that it is available to all projects.

     NOTE

     You cannot make a public flavor private after it has been created.

   - Replace **<flavor_name>** with a unique name for your flavor.
     For more information about flavor arguments, see Flavor arguments.

2. Optional: To specify flavor metadata, set the required properties by using key-value pairs:

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

   - Replace **<key>** with the metadata key of the property you want to allocate to an instance that is created with this flavor. For a list of available metadata keys, see Flavor metadata.
   - Replace **<value>** with the value of the metadata key you want to allocate to an instance that is created with this flavor.
   - Replace **<flavor_name>** with the name of your flavor.
     For example, an instance that is launched by using the following flavor has two CPU sockets, each with two CPUs:

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

8.2. FLAVOR ARGUMENTS

The **openstack flavor create** command has one positional argument, **<flavor_name>**, to specify the name of your new flavor.

The following table details the optional arguments that you can specify as required when you create a new flavor.

Table 8.2. Optional flavor arguments

Optional argument	Description
--id	Unique ID for the flavor. The default value, **auto**, generates a UUID4 value. You can use this argument to manually specify an integer or UUID4 value.
--ram	(Mandatory) Size of memory to make available to the instance, in MB. Default: 256 MB
--disk	(Mandatory) Amount of disk space to use for the root (/) partition, in GB. The root disk is an ephemeral disk that the base image is copied into. When an instance boots from a persistent volume, the root disk is not used. NOTE Creation of an instance with a flavor that has **--disk** set to **0** requires that the instance boots from volume. Default: 0 GB
--ephemeral	Amount of disk space to use for the ephemeral disks, in GB. Defaults to 0 GB, which means that no secondary ephemeral disk is created. Ephemeral disks offer machine local disk storage linked to the lifecycle of the instance. Ephemeral disks are not included in any snapshots. This disk is destroyed and all data is lost when the instance is deleted. Default: 0 GB
--swap	Swap disk size in MB. Do not specify **swap** in a flavor if the Compute service back end storage is not local storage. Default: 0 GB
--vcpus	(Mandatory) Number of virtual CPUs for the instance. Default: 1
--public	The flavor is available to all projects. By default, a flavor is public and available to all projects.

Optional argument	Description
--private	The flavor is only available to the projects specified by using the **--project** option. If you create a private flavor but add no projects to it then the flavor is only available to the cloud administrator.
--property	Metadata, or "extra specs", specified by using key-value pairs in the following format: **--property <key=value>** Repeat this option to set multiple properties.
--project	Specifies the project that can use the private flavor. You must use this argument with the **--private** option. If you do not specify any projects, the flavor is visible only to the **admin** user. Repeat this option to allow access to multiple projects.
--project-domain	Specifies the project domain that can use the private flavor. You must use this argument with the **--private** option. Repeat this option to allow access to multiple project domains.
--description	Description of the flavor. Limited to 65535 characters in length. You can use only printable characters.

8.3. FLAVOR METADATA

Use the **--property** option to specify flavor metadata when you create a flavor. Flavor metadata is also referred to as *extra specs*. Flavor metadata determines instance hardware support and quotas, which influence instance placement, instance limits, and performance.

Instance resource usage

Use the property keys in the following table to configure limits on CPU, memory and disk I/O usage by instances.

Table 8.3. Flavor metadata for resource usage

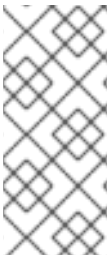
Key	Description
quota:cpu_shares	Specifies the proportional weighted share of CPU time for the domain. Defaults to the OS provided defaults. The Compute scheduler weighs this value relative to the setting of this property on other instances in the same domain. For example, an instance that is configured with **quota:cpu_shares=2048** is allocated double the CPU time as an instance that is configured with **quota:cpu_shares=1024**.

Key	Description
quota:cpu_period	Specifies the period of time within which to enforce the **cpu_quota**, in microseconds. Within the **cpu_period**, each vCPU cannot consume more than **cpu_quota** of runtime. Set to a value in the range 1000 – 1000000. Set to **0** to disable.

quota:cpu_quota	Specifies the maximum allowed bandwidth for the vCPU in each **cpu_period**, in microseconds:
	• Set to a value in the range 1000 – 18446744073709551.
	• Set to **0** to disable.
	• Set to a negative value to allow infinite bandwidth.
	You can use **cpu_quota** and **cpu_period** to ensure that all vCPUs run at the same speed. For example, you can use the following flavor to launch an instance that can consume a maximum of only 50% CPU of a physical CPU computing capability:
	`$ openstack flavor set cpu_limits_flavor \`
	`--property quota:cpu_quota=10000 \`
	`--property quota:cpu_period=20000`

Instance disk tuning

Use the property keys in the following table to tune the instance disk performance.



NOTE

The Compute service applies the following quality of service settings to storage that the Compute service has provisioned, such as ephemeral storage. To tune the performance of Block Storage (cinder) volumes, you must also configure Quality-of-Service (QOS) values for the volume type. For more information, see Use Quality-of-Service Specifications in the *Storage Guide*.

Table 8.4. Flavor metadata for disk tuning

Key	Description
quota:disk_read_bytes_sec	Specifies the maximum disk reads available to an instance, in bytes per second.
quota:disk_read_iops_sec	Specifies the maximum disk reads available to an instance, in IOPS.
quota:disk_write_bytes_sec	Specifies the maximum disk writes available to an instance, in bytes per second.

Key	Description
quota:disk_write_iops_sec	Specifies the maximum disk writes available to an instance, in IOPS.
quota:disk_total_bytes_sec	Specifies the maximum I/O operations available to an instance, in bytes per second.
quota:disk_total_iops_sec	Specifies the maximum I/O operations available to an instance, in IOPS.

Instance network traffic bandwidth

Use the property keys in the following table to configure bandwidth limits on the instance network traffic by configuring the VIF I/O options.

NOTE

The **quota :vif_*** properties are deprecated. Instead, you should use the Networking (neutron) service Quality of Service (QoS) policies. For more information about QoS policies, see Configuring Quality of Service (QoS) policies in the *Networking Guide*. The **quota:vif_*** properties are only supported when you use the ML2/OVS mechanism driver with **NeutronOVSFirewallDriver** set to **iptables_hybrid**.

Table 8.5. Flavor metadata for bandwidth limits

Key	Description
quota:vif_inbound_average	(Deprecated) Specifies the required average bit rate on the traffic incoming to the instance, in kbps.
quota:vif_inbound_burst	(Deprecated) Specifies the maximum amount of incoming traffic that can be burst at peak speed, in KB.
quota:vif_inbound_peak	(Deprecated) Specifies the maximum rate at which the instance can receive incoming traffic, in kbps.
quota:vif_outbound_average	(Deprecated) Specifies the required average bit rate on the traffic outgoing from the instance, in kbps.
quota:vif_outbound_burst	(Deprecated) Specifies the maximum amount of outgoing traffic that can be burst at peak speed, in KB.
quota:vif_outbound_peak	(Deprecated) Specifies the maximum rate at which the instance can send outgoing traffic, in kbps.

Hardware video RAM

Use the property key in the following table to configure limits on the instance RAM to use for video devices.

Table 8.6. Flavor metadata for video devices

Key	Description
hw_video:ram_max_mb	Specifies the maximum RAM to use for video devices, in MB. Use with the **hw_video_ram** image property. **hw_video_ram** must be less than or equal to **hw_video:ram_max_mb**.

Watchdog behavior

Use the property key in the following table to enable the virtual hardware watchdog device on the instance.

Table 8.7. Flavor metadata for watchdog behavior

Key	Description
hw:watchdog_action	Specify to enable the virtual hardware watchdog device and set its behavior. Watchdog devices perform the configured action if the instance hangs or fails. The watchdog uses the i6300esb device, which emulates a PCI Intel 6300ESB. If **hw:watchdog_action** is not specified, the watchdog is disabled. Set to one of the following valid values: • **disabled**: (Default) The device is not attached. • **reset**: Force instance reset. • **poweroff**: Force instance shut down. • **pause**: Pause the instance. • **none**: Enable the watchdog, but do nothing if the instance hangs or fails. NOTE Watchdog behavior that you set by using the properties of a specific image override behavior that you set by using flavors.

Random number generator (RNG)

Use the property keys in the following table to enable the RNG device on the instance.

Table 8.8. Flavor metadata for RNG

Key	Description

Key	Description
hw_rng:allowed	Set to **False** to disable the RNG device that is added to the instance through its image properties. Default: **True**
hw_rng:rate_bytes	Specifies the maximum number of bytes that the instance can read from the entropy of the host, per period.
hw_rng:rate_period	Specifies the duration of the read period in milliseconds.

Virtual Performance Monitoring Unit (vPMU)

Use the property key in the following table to enable the vPMU for the instance.

Table 8.9. Flavor metadata for vPMU

Key	Description
hw:pmu	Set to **True** to enable a vPMU for the instance. Tools such as **perf** use the vPMU on the instance to provide more accurate information to profile and monitor instance performance. For realtime workloads, the emulation of a vPMU can introduce additional latency which might be undesirable. If the telemetry it provides is not required, set **hw:pmu=False**.

Instance CPU topology

Use the property keys in the following table to define the topology of the processors in the instance.

Table 8.10. Flavor metadata for CPU topology

Key	Description
hw:cpu_sockets	Specifies the preferred number of sockets for the instance. Default: the number of vCPUs requested
hw:cpu_cores	Specifies the preferred number of cores per socket for the instance. Default: **1**
hw:cpu_threads	Specifies the preferred number of threads per core for the instance. Default: **1**

Key	Description
hw:cpu_max_sockets	Specifies the maximum number of sockets that users can select for their instances by using image properties. Example: **hw:cpu_max_sockets=2**
hw:cpu_max_cores	Specifies the maximum number of cores per socket that users can select for their instances by using image properties.
hw:cpu_max_threads	Specifies the maximum number of threads per core that users can select for their instances by using image properties.

Serial ports

Use the property key in the following table to configure the number of serial ports per instance.

Table 8.11. Flavor metadata for serial ports

Key	Description
hw:serial_port_count	Maximum serial ports per instance.

CPU pinning policy

By default, instance virtual CPUs (vCPUs) are sockets with one core and one thread. You can use properties to create flavors that pin the vCPUs of instances to the physical CPU cores (pCPUs) of the host. You can also configure the behavior of hardware CPU threads in a simultaneous multithreading (SMT) architecture where one or more cores have thread siblings.

Use the property keys in the following table to define the CPU pinning policy of the instance.

Table 8.12. Flavor metadata for CPU pinning

Key	Description
hw:cpu_policy	Specifies the CPU policy to use. Set to one of the following valid values: • **shared**: (Default) The instance vCPUs float across host pCPUs. • **dedicated**: Pin the instance vCPUs to a set of host pCPUs. This creates an instance CPU topology that matches the topology of the CPUs to which the instance is pinned. This option implies an overcommit ratio of 1.0.

Key	Description

hw:cpu_thread_policy	Specifies the CPU thread policy to use when **hw:cpu_policy=dedicated**. Set to one of the following valid values:
	• **prefer**: (Default) The host might or might not have an SMT architecture. If an SMT architecture is present, the Compute scheduler gives preference to thread siblings.
	• **isolate**: The host must not have an SMT architecture or must emulate a non-SMT architecture. This policy ensures that the Compute scheduler places the instance on a host without SMT by requesting hosts that do not report the **HW_CPU_HYPERTHREADING** trait. It is also possible to request this trait explicitly by using the following property:
	`--property trait:HW_CPU_HYPERTHREADING=forbidden`
	If the host does not have an SMT architecture, the Compute service places each vCPU on a different core as expected. If the host does have an SMT architecture, then the behaviour is determined by the configuration of the **[workarounds]/disable_fallback_pcpu_query** parameter:
	◦ **True**: The host with an SMT architecture is not used and scheduling fails.
	◦ **False**: The Compute service places each vCPU on a different physical core. The Compute service does not place vCPUs from other instances on the same core. All but one thread sibling on each used core is therefore guaranteed to be unusable.
	• **require**: The host must have an SMT architecture. This policy ensures that the Compute scheduler places the instance on a host with SMT by requesting hosts that report the **HW_CPU_HYPERTHREADING** trait. It is also possible to request this trait explicitly by using the following property:
	`--property trait:HW_CPU_HYPERTHREADING=required`
	The Compute service allocates each vCPU on thread siblings. If the host does not have an SMT architecture, then it is not used. If the host has an SMT architecture, but not enough cores with free thread siblings are available, then scheduling fails.

Instance PCI NUMA affinity policy

Use the property key in the following table to create flavors that specify the NUMA affinity policy for PCI passthrough devices and SR-IOV interfaces.

Table 8.13. Flavor metadata for PCI NUMA affinity policy

Key	Description

hw:pci_numa_affinity_policy	Specifies the NUMA affinity policy for PCI passthrough devices and SR-IOV interfaces. Set to one of the following valid values:
	• **required**: The Compute service creates an instance that requests a PCI device only when at least one of the NUMA nodes of the instance has affinity with the PCI device. This option provides the best performance.
	• **preferred**: The Compute service attempts a best effort selection of PCI devices based on NUMA affinity. If this is not possible, then the Compute service schedules the instance on a NUMA node that has no affinity with the PCI device.
	• **legacy**: (Default) The Compute service creates instances that request a PCI device in one of the following cases:
	  ◦ The PCI device has affinity with at least one of the NUMA nodes.
	  ◦ The PCI devices do not provide information about their NUMA affinities.

Instance NUMA topology

You can use properties to create flavors that define the host NUMA placement for the instance vCPU threads, and the allocation of instance vCPUs and memory from the host NUMA nodes.

Defining a NUMA topology for the instance improves the performance of the instance OS for flavors whose memory and vCPU allocations are larger than the size of NUMA nodes in the Compute hosts.

The Compute scheduler uses these properties to determine a suitable host for the instance. For example, a cloud user launches an instance by using the following flavor:

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

The Compute scheduler searches for a host that has two NUMA nodes, one with 3GB of RAM and the ability to run six CPUs, and the other with 1GB of RAM and two CPUS. If a host has a single NUMA node with capability to run eight CPUs and 4GB of RAM, the Compute scheduler does not consider it a valid match.

NOTE

NUMA topologies defined by a flavor cannot be overridden by NUMA topologies defined by the image. The Compute service raises an **ImageNUMATopologyForbidden** error if the image NUMA topology conflicts with the flavor NUMA topology.

CAUTION

You cannot use this feature to constrain instances to specific host CPUs or NUMA nodes. Use this feature only after you complete extensive testing and performance measurements. You can use the **hw:pci_numa_affinity_policy** property instead.

Use the property keys in the following table to define the instance NUMA topology.

Table 8.14. Flavor metadata for NUMA topology

Key	Description
hw:numa_nodes	Specifies the number of host NUMA nodes to restrict execution of instance vCPU threads to. If not specified, the vCPU threads can run on any number of the available host NUMA nodes.
hw:numa_cpus.N	A comma-separated list of instance vCPUs to map to instance NUMA node N. If this key is not specified, vCPUs are evenly divided among available NUMA nodes. N starts from 0. Use *.N values with caution, and only if you have at least two NUMA nodes. This property is valid only if you have set **hw:numa_nodes**, and is required only if the NUMA nodes of the instance have an asymmetrical allocation of CPUs and RAM, which is important for some NFV workloads.
hw:numa_mem.N	The number of MB of instance memory to map to instance NUMA node N. If this key is not specified, memory is evenly divided among available NUMA nodes. N starts from 0. Use *.N values with caution, and only if you have at least two NUMA nodes. This property is valid only if you have set **hw:numa_nodes**, and is required only if the NUMA nodes of the instance have an asymmetrical allocation of CPUs and RAM, which is important for some NFV workloads.

WARNING

If the combined values of **hw:numa_cpus.N** or **hw:numa_mem.N** are greater than the available number of CPUs or memory respectively, the Compute service raises an exception.

Instance memory encryption

Use the property key in the following table to enable encryption of instance memory.

Table 8.15. Flavor metadata for memory encryption

Key	Description
hw:mem_encryption	Set to **True** to request memory encryption for the instance. For more information, see Configuring AMD SEV Compute nodes to provide memory encryption for instances.

CPU real-time policy

Use the property keys in the following table to define the real-time policy of the processors in the instance.

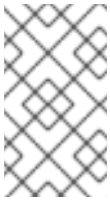
NOTE

- Although most of your instance vCPUs can run with a real-time policy, you must mark at least one vCPU as non-real-time to use for both non-real-time guest processes and emulator overhead processes.
- To use this extra spec, you must enable pinned CPUs.

Table 8.16. Flavor metadata for CPU real-time policy

Key	Description
hw:cpu_realtime	Set to **yes** to create a flavor that assigns a real-time policy to the instance vCPUs. Default: **no**

Key	Description

hw:cpu_realtime_mask	Specifies the vCPUs to not assign a real-time policy to. You must prepend the mask value with a caret symbol (^). The following example indicates that all vCPUs except vCPUs 0 and 1 have a real-time policy:
	`$ openstack flavor set <flavor> \`
	`--property hw:cpu_realtime="yes" \`
	`--property hw:cpu_realtime_mask=^0-1`
	 
	NOTE
	If the **hw_cpu_realtime_mask** property is set on the image then it takes precedence over the **hw:cpu_realtime_mask** property set on the flavor.

Emulator threads policy

You can assign a pCPU to an instance to use for emulator threads. Emulator threads are emulator processes that are not directly related to the instance. A dedicated emulator thread pCPU is required for real-time workloads. To use the emulator threads policy, you must enable pinned CPUs by setting the following property:

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

Use the property key in the following table to define the emulator threads policy of the instance.

Table 8.17. Flavor metadata for the emulator threads policy

Key	Description
hw:emulator_threads_policy	Specifies the emulator threads policy to use for instances. Set to one of the following valid values: • **share**: The emulator thread floats across the pCPUs defined in the **NovaComputeCpuSharedSet** heat parameter. If **NovaComputeCpuSharedSet** is not configured, then the emulator thread floats across the pinned CPUs that are associated with the instance. • **isolate**: Reserves an additional dedicated pCPU per instance for the emulator thread. Use this policy with caution, as it is prohibitively resource intensive. • unset: (Default) The emulator thread policy is not enabled, and the emulator thread floats across the pinned CPUs associated with the instance.

Instance memory page size

Use the property keys in the following table to create an instance with an explicit memory page size.

Table 8.18. Flavor metadata for memory page size

Key	Description
hw:mem_page_size	Specifies the size of large pages to use to back the instances. Use of this option creates an implicit NUMA topology of 1 NUMA node unless otherwise specified by **hw:numa_nodes**. Set to one of the following valid values: • **large**: Selects a page size larger than the smallest page size supported on the host, which can be 2 MB or 1 GB on x86_64 systems. • **small**: Selects the smallest page size supported on the host. On x86_64 systems this is 4 kB (normal pages). • **any**: Selects the largest available huge page size, as determined by the libvirt driver. • **<pagesize>**: (String) Sets an explicit page size if the workload has specific requirements. Use an integer value for the page size in KB, or any standard suffix. For example: **4KB**, **2MB**, **2048**, **1GB**. • unset: (Default) Large pages are not used to back instances and no implicit NUMA topology is generated.

PCI passthrough

Use the property key in the following table to attach a physical PCI device, such as a graphics card or a network device, to an instance. For more information about using PCI passthrough, see Configuring PCI passthrough.

Table 8.19. Flavor metadata for PCI passthrough

Key	Description
pci_passthrough:alias	Specifies the PCI device to assign to an instance by using the following format: `<alias>:<count>` • Replace **<alias>** with the alias that corresponds to a particular PCI device class. • Replace **<count>** with the number of PCI devices of type **<alias>** to assign to the instance.

Hypervisor signature

Use the property key in the following table to hide the hypervisor signature from the instance.

Table 8.20. Flavor metadata for hiding hypervisor signature

Key	Description
hide_hypervisor_id	Set to **True** to hide the hypervisor signature from the instance, to allow all drivers to load and work on the instance.

Instance resource traits

Each resource provider has a set of traits. Traits are the qualitative aspects of a resource provider, for example, the type of storage disk, or the Intel CPU instruction set extension. An instance can specify which of these traits it requires.

The traits that you can specify are defined in the **os-traits** library. Example traits include the following:

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

For details about how to use the **os-traits** library, see https://docs.openstack.org/os-traits/latest/user/index.html.

Use the property key in the following table to define the resource traits of the instance.

Table 8.21. Flavor metadata for resource traits

Key	Description
trait:<trait_name>	Specifies Compute node traits. Set the trait to one of the following valid values: • **required**: The Compute node selected to host the instance must have the trait. • **forbidden**: The Compute node selected to host the instance must not have the trait. Example: `$ openstack flavor set --property trait:HW_CPU_X86_AVX512BW=required avx512-flavor`

Instance bare-metal resource class

Use the property key in the following table to request a bare-metal resource class for an instance.

Table 8.22. Flavor metadata for bare-metal resource class

Key	Description

resources:<resource_class_name>	Use this property to specify standard bare-metal resource classes to override the values of, or to specify custom bare-metal resource classes that the instance requires.
	
	The standard resource classes that you can override are **VCPU**, **MEMORY_MB** and **DISK_GB**. To prevent the Compute scheduler from using the bare-metal flavor properties for scheduling instance, set the value of the standard resource classes to **0**.
	
	The name of custom resource classes must start with **CUSTOM_**. To determine the name of a custom resource class that corresponds to a resource class of a Bare Metal service node, convert the resource class to uppercase, replace all punctuation with an underscore, and prefix with CUSTOM_.
	
	For example, to schedule instances on a node that has **--resource-class baremetal.SMALL**, create the following flavor:
	
	`$ openstack flavor set \`
	`--property resources:CUSTOM_BAREMETAL_SMALL=1 \`
	`--property resources:VCPU=0 --property resources:MEMORY_MB=0 \`
	`--property resources:DISK_GB=0 compute-small`

CHAPTER 9. ADDING METADATA TO INSTANCES

The Compute (nova) service uses metadata to pass configuration information to instances on launch. The instance can access the metadata by using a config drive or the metadata service.

Config drive

Config drives are special drives that you can attach to an instance when it boots. The config drive is presented to the instance as a read-only drive. The instance can mount this drive and read files from it to get information that is normally available through the metadata service.

Metadata service

The Compute service provides the metadata service as a REST API, which can be used to retrieve data specific to an instance. Instances access this service at **169.254.169.254** or at **fe80::a9fe:a9fe**.

9.1. TYPES OF INSTANCE METADATA

Cloud users, cloud administrators, and the Compute service can pass metadata to instances:

Cloud user provided data

Cloud users can specify additional data to use when they launch an instance, such as a shell script that the instance runs on boot. The cloud user can pass data to instances by using the user data feature, and by passing key-value pairs as required properties when creating or updating an instance.

Cloud administrator provided data

The RHOSP administrator uses the vendordata feature to pass data to instances. The Compute service provides the vendordata modules **StaticJSON** and **DynamicJSON** to allow administrators to pass metadata to instances:

- **StaticJSON**: (Default) Use for metadata that is the same for all instances.
- **DynamicJSON**: Use for metadata that is different for each instance. This module makes a request to an external REST service to determine what metadata to add to an instance.

Vendordata configuration is located in one of the following read-only files on the instance:

- **/openstack/{version}/vendor_data.json**
- **/openstack/{version}/vendor_data2.json**

Compute service provided data

The Compute service uses its internal implementation of the metadata service to pass information to the instance, such as the requested hostname for the instance, and the availability zone the instance is in. This happens by default and requires no configuration by the cloud user or administrator.

9.2. ADDING A CONFIG DRIVE TO ALL INSTANCES

As an administrator, you can configure the Compute service to always create a config drive for instances, and populate the config drive with metadata that is specific to your deployment. For example, you might use a config drive for the following reasons:

- To pass a networking configuration when your deployment does not use DHCP to assign IP addresses to instances. You can pass the IP address configuration for the instance through the config drive, which the instance can mount and access before you configure the network settings for the instance.

- To pass data to an instance that is not known to the user starting the instance, for example, a cryptographic token to be used to register the instance with Active Directory post boot.
- To create a local cached disk read to manage the load of instance requests, which reduces the impact of instances accessing the metadata servers regularly to check in and build facts.

Any instance operating system that is capable of mounting an ISO 9660 or VFAT file system can use the config drive.

Procedure

1. Open your Compute environment file.
2. To always attach a config drive when launching an instance, set the following parameter to **True**:

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::force_config_drive: 'true'
```

3. Optional: To change the format of the config drive from the default value of **iso9660** to **vfat**, add the **config_drive_format** parameter to your configuration:

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::force_config_drive: 'true'
    nova::compute::config_drive_format: vfat
```

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

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

Verification

1. Create an instance:

```
(overcloud)$ openstack server create --flavor m1.tiny \
 --image cirros test-config-drive-instance
```

2. Log in to the instance.
3. Mount the config drive:
   - If the instance OS uses **udev**:

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

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

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

4. Inspect the files in the mounted config drive directory, **mnt/config/openstack/{version}/**, for your metadata.

9.3. ADDING DYNAMIC METADATA TO INSTANCES

You can configure your deployment to create instance-specific metadata, and make the metadata available to that instance through a JSON file.

TIP

You can use dynamic metadata on the undercloud to integrate director with a Red Hat Identity Management (IdM) server. An IdM server can be used as a certificate authority and manage the overcloud certificates when SSL/TLS is enabled on the overcloud. For more information, see Add the undercloud to IdM.

Procedure

1. Open your Compute environment file.
2. Add **DynamicJSON** to the vendordata provider module:

```
parameter_defaults:
  ControllerExtraConfig:
   nova::vendordata::vendordata_providers:
     - DynamicJSON
```

3. Specify the REST services to contact to generate the metadata. You can specify as many target REST services as required, for example:

```
parameter_defaults:
  ControllerExtraConfig:
  nova::vendordata::vendordata_providers:
    - DynamicJSON
    nova::vendordata::vendordata_dynamic_targets:
      "target1@http://127.0.0.1:125"
    nova::vendordata::vendordata_dynamic_targets:
      "target2@http://127.0.0.1:126"
```

The Compute service generates the JSON file, **vendordata2.json**, to contain the metadata retrieved from the configured target services, and stores it in the config drive directory.

NOTE

Do not use the same name for a target service more than once.

4. Save the updates to your Compute environment file.

5. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

CHAPTER 10. CONFIGURING CPU FEATURE FLAGS FOR INSTANCES

You can enable or disable CPU feature flags for an instance without changing the settings on the host Compute node and rebooting the Compute node. By configuring the standard set of CPU feature flags that are applied to instances, you are helping to achieve live migration compatibility across Compute nodes. You are also helping to manage the performance and security of the instances, by disabling flags that have a negative impact on the security or performance of the instances with a particular CPU model, or enabling flags that provide mitigation from a security problem or alleviates performance problems.

10.1. PREREQUISITES

- The CPU model and feature flags must be supported by the hardware and software of the host Compute node:
  - To check the hardware your host supports, enter the following command on the Compute node:

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

  - To check the CPU models supported on your host, enter the following command on the Compute node:

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

    Replace **<arch>** with the name of the architecture, for example, **x86_64**.

10.2. CONFIGURING CPU FEATURE FLAGS FOR INSTANCES

Configure the Compute service to apply CPU feature flags to instances with specific vCPU models.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

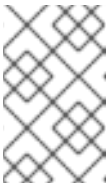
   ```
   [stack@director ~]$ source ~/stackrc
   ```

3. Open your Compute environment file.
4. Configure the instance CPU mode:

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

   Replace **<cpu_mode>** with the CPU mode of each instance on the Compute node. Set to one of the following valid values:

- **host-model**: (Default) Use the CPU model of the host Compute node. Use this CPU mode to automatically add critical CPU flags to the instance to provide mitigation from security flaws.
- **custom**: Use to configure the specific CPU models each instance should use.



NOTE

You can also set the CPU mode to **host-passthrough** to use the same CPU model and feature flags as the Compute node for the instances hosted on that Compute node.

5. Optional: If you set **NovaLibvirtCPUMode** to **custom**, configure the instance CPU models that you want to customise:

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

Replace **<cpu_model>** with a comma-separated list of the CPU models that the host supports. List the CPU models in order, placing the more common and less advanced CPU models first in the list, and the more feature-rich CPU models last, for example, **SandyBridge,IvyBridge,Haswell,Broadwell**. For a list of model names, see **/usr/share/libvirt/cpu_map.xml**, or enter the following command on the host Compute node:

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

Replace **<arch>** with the name of the architecture of the Compute node, for example, **x86_64**.

6. Configure the CPU feature flags for instances with the specified CPU models:

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

Replace **<cpu_feature_flags>** with a comma-separated list of feature flags to enable or disable. Prefix each flag with "+" to enable the flag, or "-" to disable it. If a prefix is not specified, the flag is enabled. For a list of the available feature flags for a given CPU model, see **/usr/share/libvirt/cpu_map/*.xml**.

The following example enables the CPU feature flags **pcid** and **ssbd** for the **IvyBridge** and **Cascadelake-Server** models, and disables the feature flag **mtrr**.

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

7. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

CHAPTER 11. CONFIGURING MANUAL NODE REBOOT TO DEFINE KERNELARGS

Overcloud nodes are automatically rebooted when the overcloud deployment includes setting the **KernelArgs** for the first time. Rebooting nodes can be an issue for existing workloads if you are adding **KernelArgs** to a deployment that is already in production. You can disable the automatic rebooting of nodes when updating a deployment, and instead perform node reboots manually after each overcloud deployment.

NOTE

If you disable automatic reboot and then add new Compute nodes to your deployment, the new nodes will not be rebooted during their initial provisioning. This might cause deployment errors because the configuration of **KernelArgs** is applied only after a reboot.

11.1. CONFIGURING MANUAL NODE REBOOT TO DEFINE KERNELARGS

You can disable the automatic rebooting of nodes when you configure **KernelArgs** for the first time, and instead reboot the nodes manually.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

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

3. Enable the **KernelArgsDeferReboot** role parameter in a custom environment file, for example, **kernelargs_manual_reboot.yaml**:

```
parameter_defaults:
  <Role>Parameters:
    KernelArgsDeferReboot: True
```

4. Add your custom environment file to the stack with your other environment files and deploy the overcloud:

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

5. Retrieve a list of your Compute nodes to identify the host name of the node that you want to reboot:

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

6. Disable the Compute service on the Compute node you want to reboot, to prevent the Compute scheduler from assigning new instances to the node:

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

Replace **<node>** with the host name of the node you want to disable the Compute service on.

7. Retrieve a list of the instances hosted on the Compute node that you want to migrate:

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

8. Migrate the instances to another Compute node. For information on migrating instances, see Migrating virtual machine instances between Compute nodes .
9. Log in to the node that you want to reboot.
10. Reboot the node:

```
[heat-admin@overcloud-compute-0 ~]$ sudo reboot
```

11. Wait until the node boots.
12. Re-enable the Compute node:

```
(overcloud)$ openstack compute service set <node_UUID>  nova-compute --enable
```

13. Check that the Compute node is enabled:

```
(overcloud)$ openstack compute service list
```

CHAPTER 12. CONFIGURING AMD SEV COMPUTE NODES TO PROVIDE MEMORY ENCRYPTION FOR INSTANCES

As a cloud administrator, you can provide cloud users the ability to create instances that run on SEV-capable Compute nodes with memory encryption enabled.

This feature is available to use from the 2nd Gen AMD EPYC™ 7002 Series ("Rome").

To enable your cloud users to create instances that use memory encryption, you must perform the following tasks:

1. Designate the AMD SEV Compute nodes for memory encryption.
2. Configure the Compute nodes for memory encryption.
3. Deploy the overcloud.
4. Create a flavor or image for launching instances with memory encryption.

TIP

If the AMD SEV hardware is limited, you can also configure a host aggregate to optimize scheduling on the AMD SEV Compute nodes. To schedule only instances that request memory encryption on the AMD SEV Compute nodes, create a host aggregate of the Compute nodes that have the AMD SEV hardware, and configure the Compute scheduler to place only instances that request memory encryption on the host aggregate. For more information, see Creating and managing host aggregates and Filtering by isolating host aggregates.

12.1. SECURE ENCRYPTED VIRTUALIZATION (SEV)

Secure Encrypted Virtualization (SEV), provided by AMD, protects the data in DRAM that a running virtual machine instance is using. SEV encrypts the memory of each instance with a unique key.

SEV increases security when you use non-volatile memory technology (NVDIMM), because an NVDIMM chip can be physically removed from a system with the data intact, similar to a hard drive. Without encryption, any stored information such as sensitive data, passwords, or secret keys can be compromised.

For more information, see the *AMD Secure Encrypted Virtualization (SEV)* documentation.

Limitations of instances with memory encryption

- You cannot live migrate, or suspend and resume instances with memory encryption.
- You cannot use PCI passthrough to directly access devices on instances with memory encryption.
- You cannot use **virtio-blk** as the boot disk of instances with memory encryption with Red Hat Enterprise Linux (RHEL) kernels earlier than kernel-4.18.0-115.el8 (RHEL-8.1.0).

NOTE

You can use **virtio-scsi** or **SATA** as the boot disk, or **virtio-blk** for non-boot disks.

- The operating system that runs in an encrypted instance must provide SEV support. For more information, see the Red Hat Knowledgebase solution Enabling AMD Secure Encrypted Virtualization in RHEL 8.
- Machines that support SEV have a limited number of slots in their memory controller for storing encryption keys. Each running instance with encrypted memory consumes one of these slots. Therefore, the number of instances with memory encryption that can run concurrently is limited to the number of slots in the memory controller. For example, on 1st Gen AMD EPYC™ 7001 Series ("Naples") the limit is 16, and on 2nd Gen AMD EPYC™ 7002 Series ("Rome") the limit is 255.
- Instances with memory encryption pin pages in RAM. The Compute service cannot swap these pages, therefore you cannot overcommit memory on a Compute node that hosts instances with memory encryption.
- You cannot use memory encryption with instances that have multiple NUMA nodes.

12.2. DESIGNATING AMD SEV COMPUTE NODES FOR MEMORY ENCRYPTION

To designate AMD SEV Compute nodes for instances that use memory encryption, you must create a new role file to configure the AMD SEV role, and configure a new overcloud flavor and AMD SEV resource class to use to tag the Compute nodes for memory encryption.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

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

3. Generate a new roles data file that includes the **ComputeAMDSEV** role, along with any other roles that you need for the overcloud. The following example generates the roles data file **roles_data_amd_sev.yaml**, which includes the roles **Controller** and **ComputeAMDSEV**:

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

4. Open **roles_data_amd_sev.yaml** and edit or add the following parameters and sections:

Section/Parameter	Current value	New value
Role comment	**Role: Compute**	**Role: ComputeAMDSEV**
Role name	**name: Compute**	**name: ComputeAMDSEV**
description	**Basic Compute Node role**	**AMD SEV Compute Node role**

Section/Parameter	Current value	New value
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputeamdsev-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-amd-sev.yaml**

5. Register the AMD SEV Compute nodes for the overcloud by adding them to your node definition template, **node.json** or **node.yaml**. For more information, see Registering nodes for the overcloud in the *Director Installation and Usage* guide.

6. Inspect the node hardware:

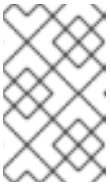
```
(undercloud)$ openstack overcloud node introspect \
 --all-manageable --provide
```

For more information, see Creating an inventory of the bare-metal node hardware in the *Director Installation and Usage* guide.

7. Create the **compute-amd-sev** overcloud flavor for AMD SEV Compute nodes:

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

- Replace **<ram_size_mb>** with the RAM of the bare metal node, in MB.
- Replace **<disk_size_gb>** with the size of the disk on the bare metal node, in GB.
- Replace **<no_vcpus>** with the number of CPUs on the bare metal node.



NOTE

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

8. Retrieve a list of your nodes to identify their UUIDs:

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

9. Tag each bare metal node that you want to designate for memory encryption with a custom AMD SEV resource class:

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

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

10. Associate the **compute-amd-sev** flavor with the custom AMD SEV resource class:

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_AMD_SEV=1 \
  compute-amd-sev
```

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

NOTE

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

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

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

12. Optional: If the network topology of the **ComputeAMDSEV** role is different from the network topology of your **Compute** role, then create a custom network interface template. For more information, see Custom network interface templates in the *Advanced Overcloud Customization* guide.
If the network topology of the **ComputeAMDSEV** role is the same as the **Compute** role, then you can use the default network topology defined in **compute.yaml**.

13. Register the **Net::SoftwareConfig** of the **ComputeAMDSEV** role in your **network-environment.yaml** file:

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-configs/compute.yaml
  OS::TripleO::ComputeCPUPinning::Net::SoftwareConfig: /home/stack/templates/nic-configs/<amd_sev_net_top>.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-configs/controller.yaml
```

Replace **<amd_sev_net_top>** with the name of the file that contains the network topology of the **ComputeAMDSEV** role, for example, **compute.yaml** to use the default network topology.

14. Add the following parameters to the **node-info.yaml** file to specify the number of AMD SEV Compute nodes, and the flavor that you want to use for the AMD SEV designated Compute nodes:

```
parameter_defaults:
  OvercloudComputeAMDSEVFlavor: compute-amd-sev
  ComputeAMDSEVCount: 3
```

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

```
(undercloud)$ openstack baremetal node list --long -c "UUID" \
 -c "Instance UUID" -c "Resource Class" -c "Provisioning State" \
 -c "Power State" -c "Last Error" -c "Fault" -c "Name" -f json
```

Example output:

```
[
  {
    "Fault": null,
    "Instance UUID": "e8e60d37-d7c7-4210-acf7-f04b245582ea",
    "Last Error": null,
    "Name": "compute-0",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "baremetal.AMD-SEV",
    "UUID": "b5a9ac58-63a7-49ba-b4ad-33d84000ccb4"
  },
  {
    "Fault": null,
    "Instance UUID": "3ec34c0b-c4f5-4535-9bd3-8a1649d2e1bd",
    "Last Error": null,
    "Name": "compute-1",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "compute",
    "UUID": "432e7f86-8da2-44a6-9b14-dfacdf611366"
  },
  {
    "Fault": null,
    "Instance UUID": "4992c2da-adde-41b3-bef1-3a5b8e356fc0",
    "Last Error": null,
    "Name": "controller-0",
    "Power State": "power on",
    "Provisioning State": "active",
    "Resource Class": "controller",
    "UUID": "474c2fc8-b884-4377-b6d7-781082a3a9c0"
  }
]
```

12.3. CONFIGURING AMD SEV COMPUTE NODES FOR MEMORY ENCRYPTION

To enable your cloud users to create instances that use memory encryption, you must configure the Compute nodes that have the AMD SEV hardware.

Prerequisites

- Your deployment must include a Compute node that runs on AMD hardware capable of supporting SEV, such as an AMD EPYC CPU. You can use the following command to determine if your deployment is SEV-capable:

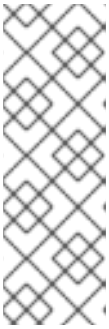
```
$ lscpu | grep sev
```

Procedure

1. Open your Compute environment file.

2. Optional: Add the following configuration to your Compute environment file to specify the maximum number of memory-encrypted instances that the AMD SEV Compute nodes can host concurrently:

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



NOTE

The default value of the **libvirt/num_memory_encrypted_guests** parameter is **none**. If you do not set a custom value, the AMD SEV Compute nodes do not impose a limit on the number of memory-encrypted instances that the nodes can host concurrently. Instead, the hardware determines the maximum number of memory-encrypted instances that the AMD SEV Compute nodes can host concurrently, which might cause some memory-encrypted instances to fail to launch.

3. Optional: To specify that all x86_64 images use the q35 machine type by default, add the following configuration to your Compute environment file:

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

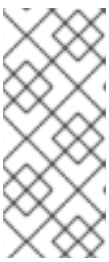
If you specify this parameter value, you do not need to set the **hw_machine_type** property to **q35** on every AMD SEV instance image.

4. To ensure that the AMD SEV Compute nodes reserve enough memory for host-level services to function, add 16MB for each potential AMD SEV instance:

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

5. Configure the kernel parameters for the AMD SEV Compute nodes:

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



NOTE

When you first add the **KernelArgs** parameter to the configuration of a role, the overcloud nodes are automatically rebooted. If required, you can disable the automatic rebooting of nodes and instead perform node reboots manually after each overcloud deployment. For more information, see Configuring manual node reboot to define **KernelArgs**.

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

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

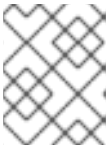
12.4. CREATING AN IMAGE FOR MEMORY ENCRYPTION

When the overcloud contains AMD SEV Compute nodes, you can create an AMD SEV instance image that your cloud users can use to launch instances that have memory encryption.

Procedure

1. Create a new image for memory encryption:

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



NOTE

If you use an existing image, the image must have the **hw_firmware_type** property set to **uefi**.

2. Optional: Add the property **hw_mem_encryption=True** to the image to enable AMD SEV memory encryption on the image:

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

TIP

You can enable memory encryption on the flavor. For more information, see Creating a flavor for memory encryption.

3. Optional: Set the machine type to **q35**, if not already set in the Compute node configuration:

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

4. Optional: To schedule memory-encrypted instances on a SEV-capable host aggregate, add the following trait to the image extra specs:

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

TIP

You can also specify this trait on the flavor. For more information, see Creating a flavor for memory encryption.

12.5. CREATING A FLAVOR FOR MEMORY ENCRYPTION

When the overcloud contains AMD SEV Compute nodes, you can create one or more AMD SEV flavors that your cloud users can use to launch instances that have memory encryption.

NOTE

An AMD SEV flavor is necessary only when the **hw_mem_encryption** property is not set on an image.

Procedure

1. Create a flavor for memory encryption:

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

2. To schedule memory-encrypted instances on a SEV-capable host aggregate, add the following trait to the flavor extra specs:

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

12.6. LAUNCHING AN INSTANCE WITH MEMORY ENCRYPTION

To verify that you can launch instances on an AMD SEV Compute node with memory encryption enabled, use a memory encryption flavor or image to create an instance.

Procedure

1. Create an instance by using an AMD SEV flavor or image. The following example creates an instance by using the flavor created in Creating a flavor for memory encryption and the image created in Creating an image for memory encryption :

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

2. Log in to the instance as a cloud user.

3. To verify that the instance uses memory encryption, enter the following command from the instance:

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

CHAPTER 13. CONFIGURING NVDIMM COMPUTE NODES TO PROVIDE PERSISTENT MEMORY FOR INSTANCES

A non-volatile dual in-line memory module (NVDIMM) is a technology that provides DRAM with persistent memory (PMEM). Standard computer memory loses its data after a loss of electrical power. The NVDIMM maintains its data even after a loss of electrical power. Instances that use PMEM can provide applications with the ability to load large contiguous segments of memory that persist the application data across power cycles. This is useful for high performance computing (HPC), which requests huge amounts of memory.

As a cloud administrator, you can make the PMEM available to instances as virtual PMEM (vPMEM) by creating and configuring PMEM namespaces on Compute nodes that have NVDIMM hardware. Cloud users can then create instances that request vPMEM when they need the instance content to be retained after it is shut down.

To enable your cloud users to create instances that use PMEM, you must complete the following procedures:

1. Designate Compute nodes for PMEM.
2. Configure the Compute nodes for PMEM that have the NVDIMM hardware.
3. Deploy the overcloud.
4. Create PMEM flavors for launching instances that have vPMEM.

TIP

If the NVDIMM hardware is limited, you can also configure a host aggregate to optimize scheduling on the PMEM Compute nodes. To schedule only instances that request vPMEM on the PMEM Compute nodes, create a host aggregate of the Compute nodes that have the NVDIMM hardware, and configure the Compute scheduler to place only PMEM instances on the host aggregate. For more information, see Creating and managing host aggregates and Filtering by isolating host aggregates .

Prerequisites

- Your Compute node has persistent memory hardware, such as Intel® Optane™ DC Persistent Memory.
- You have configured backend NVDIMM regions on the PMEM hardware device to create PMEM namespaces. You can use the ipmctl tool provided by Intel to configure the PMEM hardware.

Limitations when using PMEM devices

- You cannot cold migrate, live migrate, resize or suspend and resume instances that use vPMEM.
- Only instances running RHEL8 can use vPMEM.
- When rebuilding a vPMEM instance, the persistent memory namespaces are removed to restore the initial state of the instance.
- When resizing an instance using a new flavor, the content of the original virtual persistent memory will not be copied to the new virtual persistent memory.
- Virtual persistent memory hotplugging is not supported.

- When creating a snapshot of a vPMEM instance, the virtual persistent images are not included.

13.1. DESIGNATING COMPUTE NODES FOR PMEM

To designate Compute nodes for PMEM workloads, you must create a new role file to configure the PMEM role, and configure a new overcloud flavor and resource class for PMEM to use to tag the NVDIMM Compute nodes.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

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

3. Generate a new roles data file named **roles_data_pmem.yaml** that includes the **Controller**, **Compute**, and **ComputePMEM** roles:

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

4. Open **roles_data_pmem.yaml** and edit or add the following parameters and sections:

Section/Parameter	Current value	New value
Role comment	**Role: Compute**	**Role: ComputePMEM**
Role name	**name: Compute**	**name: ComputePMEM**
description	**Basic Compute Node role**	**PMEM Compute Node role**
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputepmem-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-pmem.yaml**

5. Register the NVDIMM Compute nodes for the overcloud by adding them to your node definition template, **node.json** or **node.yaml**. For more information, see Registering nodes for the overcloud in the *Director Installation and Usage* guide.
6. Inspect the node hardware:

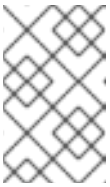
```
(undercloud)$ openstack overcloud node introspect --all-manageable --provide
```

   For more information, see Creating an inventory of the bare-metal node hardware in the *Director Installation and Usage* guide.

7. Create the **compute-pmem** overcloud flavor for PMEM Compute nodes:

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

- Replace **<ram_size_mb>** with the RAM of the bare metal node, in MB.
- Replace **<disk_size_gb>** with the size of the disk on the bare metal node, in GB.
- Replace **<no_vcpus>** with the number of CPUs on the bare metal node.



NOTE

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

8. Tag each bare metal node that you want to designate for PMEM workloads with a custom PMEM resource class:

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

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

9. Associate the **compute-pmem** flavor with the custom PMEM resource class:

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_PMEM=1 \
  compute-pmem
```

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

NOTE

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

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

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

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

```
parameter_defaults:
  OvercloudComputePMEMFlavor: compute-pmem
  ComputePMEMCount: 3 #set to the no of NVDIMM devices you have
```

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

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

13.2. CONFIGURING A PMEM COMPUTE NODE

To enable your cloud users to create instances that use vPMEM, you must configure the Compute nodes that have the NVDIMM hardware.

Procedure

1. Create a new Compute environment file for configuring NVDIMM Compute nodes, for example, **env_pmem.yaml**.
2. To partition the NVDIMM regions into PMEM namespaces that the instances can use, add the **NovaPMEMNamespaces** role-specific parameter to the PMEM role in your Compute environment file, and set the value using the following format:

   ```
   <size>:<namespace_name>[,<size>:<namespace_name>]
   ```

   Use the following suffixes to represent the size:

   - "k" or "K" for KiB
   - "m" or "M" for MiB
   - "G" or "G" for GiB
   - "t" or "T" for TiB

     For example, the following configuration creates four namespaces, three of size 6 GiB, and one of size 100 GiB:

     ```
     parameter_defaults:
       ComputePMEMParameters:
         NovaPMEMNamespaces: "6G:ns0,6G:ns1,6G:ns2,100G:ns3"
     ```

3. To map the PMEM namespaces to labels that can be used in a flavor, add the **NovaPMEMMappings** role-specific parameter to the PMEM role in your Compute environment file, and set the value using the following format:

   ```
   <label>:<namespace_name>[|<namespace_name>][,<label>:<namespace_name>[|
   <namespace_name>]].
   ```

   For example, the following configuration maps the three 6 GiB namespaces to the label "6GB", and the 100 GiB namespace to the label "LARGE":

   ```
   parameter_defaults:
     ComputePMEMParameters:
       NovaPMEMNamespaces: "6G:ns0,6G:ns1,6G:ns2,100G:ns3"
       NovaPMEMMappings: "6GB:ns0|ns1|ns2,LARGE:ns3"
   ```

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

   ```
   -
   ```

```
(undercloud)$ openstack overcloud deploy --templates \
 -r /home/stack/templates/roles_data_pmem.yaml \
 -e /home/stack/templates/node-info.yaml \
 -e [your environment files] \
 -e /home/stack/templates/env_pmem.yaml
```

6. Create and configure the flavors that your cloud users can use to launch instances that have vPMEM. The following example creates a flavor that requests a small PMEM device, 6GB, as mapped in Step 3:

```
(overcloud)$ openstack flavor create --vcpus 1 --ram 512 --disk 2  \
 --property hw:pmem='6GB' small_pmem_flavor
```

Verification

1. Create an instance using one of the PMEM flavors:

```
(overcloud)$ openstack flavor list
(overcloud)$ openstack server create --flavor small_pmem_flavor \
 --image rhel8 pmem_instance
```

2. Log in to the instance as a cloud user. For more information, see Connecting to an instance.

3. List all the disk devices attached to the instance:

```
$ sudo fdisk -l /dev/pmem0
```

The instance has vPMEM if one of the devices listed is NVDIMM.

CHAPTER 14. CONFIGURING VIRTUAL GPUS FOR INSTANCES

To support GPU-based rendering on your instances, you can define and manage virtual GPU (vGPU) resources according to your available physical GPU devices and your hypervisor type. You can use this configuration to divide the rendering workloads between all your physical GPU devices more effectively, and to have more control over scheduling your vGPU-enabled instances.

To enable vGPU in the Compute (nova) service, create flavors that your cloud users can use to create Red Hat Enterprise Linux (RHEL) instances with vGPU devices. Each instance can then support GPU workloads with virtual GPU devices that correspond to the physical GPU devices.

The Compute service tracks the number of vGPU devices that are available for each GPU profile you define on each host. The Compute service schedules instances to these hosts based on the flavor, attaches the devices, and monitors usage on an ongoing basis. When an instance is deleted, the Compute service adds the vGPU devices back to the available pool.

IMPORTANT

Red Hat enables the use of NVIDIA vGPU in RHOSP without the requirement for support exceptions. However, Red Hat does not provide technical support for the NVIDIA vGPU drivers. The NVIDIA vGPU drivers are shipped and supported by NVIDIA. You require an NVIDIA Certified Support Services subscription to obtain NVIDIA Enterprise Support for NVIDIA vGPU software. For issues that result from the use of NVIDIA vGPUs where you are unable to reproduce the issue on a supported component, the following support policies apply:

- When Red Hat does not suspect that the third-party component is involved in the issue, the normal Scope of Support and Red Hat SLA apply.
- When Red Hat suspects that the third-party component is involved in the issue, the customer will be directed to NVIDIA in line with the Red Hat third party support and certification policies. For more information, see the Knowledge Base article Obtaining Support from NVIDIA .

14.1. SUPPORTED CONFIGURATIONS AND LIMITATIONS

Supported GPU cards

For a list of supported NVIDIA GPU cards, see Virtual GPU Software Supported Products on the NVIDIA website.

Limitations when using vGPU devices

- You can enable only one vGPU type on each Compute node.
- Each instance can use only one vGPU resource.
- Live migration of vGPU instances between hosts is not supported.
- Evacuation of vGPU instances is not supported.
- If you need to reboot the Compute node that hosts the vGPU instances, the vGPUs are not automatically reassigned to the recreated instances. You must either cold migrate the instances before you reboot the Compute node, or manually allocate each vGPU to the correct instance

after reboot. To manually allocate each vGPU, you must retrieve the **mdev** UUID from the instance XML for each vGPU instance that runs on the Compute node before you reboot. You can use the following command to discover the **mdev** UUID for each instance:

```
# virsh dumpxml <instance_name> | grep mdev
```

Replace **<instance_name>** with the libvirt instance name, **OS-EXT-SRV-ATTR:instance_name**, returned in a **/servers** request to the Compute API.

- Suspend operations on a vGPU-enabled instance is not supported due to a libvirt limitation. Instead, you can snapshot or shelve the instance.
- By default, vGPU types on Compute hosts are not exposed to API users. To grant access, add the hosts to a host aggregate. For more information, see Creating and managing host aggregates.
- If you use NVIDIA accelerator hardware, you must comply with the NVIDIA licensing requirements. For example, NVIDIA vGPU GRID requires a licensing server. For more information about the NVIDIA licensing requirements, see NVIDIA License Server Release Notes on the NVIDIA website.

14.2. CONFIGURING VGPU ON THE COMPUTE NODES

To enable your cloud users to create instances that use a virtual GPU (vGPU), you must configure the Compute nodes that have the physical GPUs:

1. Designate Compute nodes for vGPU.
2. Configure the Compute node for vGPU.
3. Deploy the overcloud.
4. Create a vGPU flavor for launching instances that have vGPU.

TIP

If the GPU hardware is limited, you can also configure a host aggregate to optimize scheduling on the vGPU Compute nodes. To schedule only instances that request vGPUs on the vGPU Compute nodes, create a host aggregate of the vGPU Compute nodes, and configure the Compute scheduler to place only vGPU instances on the host aggregate. For more information, see Creating and managing host aggregates and Filtering by isolating host aggregates .

NOTE

To use an NVIDIA GRID vGPU, you must comply with the NVIDIA GRID licensing requirements and you must have the URL of your self-hosted license server. For more information, see the NVIDIA License Server Release Notes web page.

14.2.1. Prerequisites

- You have downloaded the NVIDIA GRID host driver RPM package that corresponds to your GPU device from the NVIDIA website. To determine which driver you need, see the NVIDIA Driver Downloads Portal. You must be a registered NVIDIA customer to download the drivers from the portal.

- You have built a custom overcloud image that has the NVIDIA GRID host driver installed.

14.2.2. Designating Compute nodes for vGPU

To designate Compute nodes for vGPU workloads, you must create a new role file to configure the vGPU role, and configure a new overcloud flavor and resource class to use to tag the GPU-enabled Compute nodes.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

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

3. Generate a new roles data file named **roles_data_gpu.yaml** that includes the **Controller**, **Compute**, and **ComputeGpu** roles:

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

4. Open **roles_data_gpu.yaml** and edit or add the following parameters and sections:

Section/Parameter	Current value	New value
Role comment	**Role: Compute**	**Role: ComputeGpu**
Role name	**name: Compute**	**name: ComputeGpu**
description	**Basic Compute Node role**	**GPU Compute Node role**
ImageDefault	n/a	**overcloud-full-gpu**
HostnameFormatDefault	**-compute-**	**-computegpu-**
deprecated_nic_config_name	**compute.yaml**	**compute-gpu.yaml**

5. Register the GPU-enabled Compute nodes for the overcloud by adding them to your node definition template, **node.json** or **node.yaml**. For more information, see Registering nodes for the overcloud in the *Director Installation and Usage* guide.
6. Inspect the node hardware:

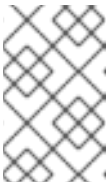
```
(undercloud)$ openstack overcloud node introspect --all-manageable \
 --provide
```

For more information, see Creating an inventory of the bare-metal node hardware in the *Director Installation and Usage* guide.

7. Create the **compute-vgpu-nvidia** overcloud flavor for vGPU Compute nodes:

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

   - Replace **<ram_size_mb>** with the RAM of the bare metal node, in MB.
   - Replace **<disk_size_gb>** with the size of the disk on the bare metal node, in GB.
   - Replace **<no_vcpus>** with the number of CPUs on the bare metal node.



NOTE

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

8. Tag each bare metal node that you want to designate for GPU workloads with a custom GPU resource class:

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

   Replace **<node>** with the ID of the baremetal node.

9. Associate the **compute-vgpu-nvidia** flavor with the custom GPU resource class:

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_GPU=1 \
 compute-vgpu-nvidia
```

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

NOTE

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

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

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

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

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

14.2.3. Configuring the Compute node for vGPU and deploying the overcloud

You need to retrieve and assign the vGPU type that corresponds to the physical GPU device in your environment, and prepare the environment files to configure the Compute node for vGPU.

Procedure

1. Install Red Hat Enterprise Linux and the NVIDIA GRID driver on a temporary Compute node and launch the node.

2. On the Compute node, locate the vGPU type of the physical GPU device that you want to enable. For libvirt, virtual GPUs are mediated devices, or **mdev** type devices. To discover the supported **mdev** devices, enter the following command:

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

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

3. Register the **Net::SoftwareConfig** of the **ComputeGpu** role in your **network-environment.yaml** file:

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::ComputeGpu::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute-gpu.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
```

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

```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeFlavor: compute
  OvercloudComputeGpuFlavor: compute-vgpu-nvidia
  ControllerCount: 1
  ComputeCount: 0
  ComputeGpuCount: 1
```

5. Create a **gpu.yaml** file to specify the vGPU type of your GPU device:

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

NOTE

Each physical GPU supports only one virtual GPU type. If you specify multiple vGPU types in this property, only the first type is used.

6. Save the updates to your Compute environment file.
7. Add your new role and environment files to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -r /home/stack/templates/roles_data_gpu.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/gpu.yaml \
  -e /home/stack/templates/node-info.yaml
```

14.3. CREATING A CUSTOM GPU INSTANCE IMAGE

To enable your cloud users to create instances that use a virtual GPU (vGPU), you can create a custom vGPU-enabled image for launching instances. Use the following procedure to create a custom vGPU-enabled instance image with the NVIDIA GRID guest driver and license file.

Prerequisites

- You have configured and deployed the overcloud with GPU-enabled Compute nodes.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **overcloudrc** credential file:

```
$ source ~/overcloudrc
```

3. Create an instance with the hardware and software profile that your vGPU instances require:

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

   - Replace **<flavor>** with the name or ID of the flavor that has the hardware profile that your vGPU instances require. For information about creating a vGPU flavor, see Creating a vGPU flavor for instances.
   - Replace **<image>** with the name or ID of the image that has the software profile that your vGPU instances require. For information about downloading RHEL cloud images, see Image service.

4. Log in to the instance as a cloud-user.
5. Create the **gridd.conf** NVIDIA GRID license file on the instance, following the NVIDIA guidance: Licensing an NVIDIA vGPU on Linux by Using a Configuration File .

6. Install the GPU driver on the instance. For more information about installing an NVIDIA driver, see Installing the NVIDIA vGPU Software Graphics Driver on Linux .

NOTE

Use the **hw_video_model** image property to define the GPU driver type. You can choose **none** if you want to disable the emulated GPUs for your vGPU instances. For more information about supported drivers, see Image metadata.

7. Create an image snapshot of the instance:

```
(overcloud)$ openstack server image create \
 --name vgpu_image temp_vgpu_instance
```

8. Optional: Delete the instance.

14.4. CREATING A VGPU FLAVOR FOR INSTANCES

To enable your cloud users to create instances for GPU workloads, you can create a GPU flavor that can be used to launch vGPU instances, and assign the vGPU resource to that flavor.

Prerequisites

- You have configured and deployed the overcloud with GPU-designated Compute nodes.

Procedure

1. Create an NVIDIA GPU flavor, for example:

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

2. Assign a vGPU resource to the flavor that you created. You can assign only one vGPU for each instance.

```
(overcloud)$ openstack flavor set m1.small-gpu \
 --property "resources:VGPU=1"
```

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

14.5. LAUNCHING A VGPU INSTANCE

You can create a GPU-enabled instance for GPU workloads.

Procedure

1. Create an instance using a GPU flavor and image, for example:

```
(overcloud)$ openstack server create --flavor m1.small-gpu \
 --image vgpu_image --security-group web --nic net-id=internal0 \
 --key-name lambda vgpu-instance
```

2. Log in to the instance as a cloud-user.

3. To verify that the GPU is accessible from the instance, enter the following command from the instance:

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

14.6. ENABLING PCI PASSTHROUGH FOR A GPU DEVICE

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

Prerequisites

- The **pciutils** package is installed on the physical servers that have the PCI cards.
- The driver for the GPU device must be installed on the instance that the device is passed through to. Therefore, you need to have created a custom instance image that has the required GPU driver installed. For more information about how to create a custom instance image with the GPU driver installed, see Creating a custom GPU instance image .

Procedure

1. To determine the vendor ID and product ID for each passthrough device type, enter the following command on the physical server that has the PCI cards:

```
# lspci -nn | grep -i <gpu_name>
```

   For example, to determine the vendor and product ID for an NVIDIA GPU, enter the following command:

```
# lspci -nn | grep -i nvidia
3b:00.0 3D controller [0302]: NVIDIA Corporation TU104GL [Tesla T4] [10de:1eb8] (rev a1)
d8:00.0 3D controller [0302]: NVIDIA Corporation TU104GL [Tesla T4] [10de:1db4] (rev a1)
```

2. To determine if each PCI device has Single Root I/O Virtualization (SR-IOV) capabilities, enter the following command on the physical server that has the PCI cards:

```
# lspci -v -s 3b:00.0
3b:00.0 3D controller: NVIDIA Corporation TU104GL [Tesla T4] (rev a1)
        ...
        Capabilities: [bcc] Single Root I/O Virtualization (SR-IOV)
        ...
```

3. To configure the Controller node on the overcloud for PCI passthrough, create an environment file, for example, **pci_passthru_controller.yaml**.

4. Add **PciPassthroughFilter** to the **NovaSchedulerDefaultFilters** parameter in **pci_passthru_controller.yaml**:

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

5. To specify the PCI alias for the devices on the Controller node, add the following configuration to **pci_passthru_controller.yaml**:

   - If the PCI device has SR-IOV capabilities:

```
ControllerExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
      device_type: "type-PF"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
      device_type: "type-PF"
```

   - If the PCI device does not have SR-IOV capabilities:

```
ControllerExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
```

```
    vendor_id: "10de"
  - name: "v100"
    product_id: "1db4"
    vendor_id: "10de"
```

For more information on configuring the **device_type** field, see PCI passthrough device type field.

NOTE

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

6. To configure the Compute node on the overcloud for PCI passthrough, create an environment file, for example, **pci_passthru_compute.yaml**.

7. To specify the available PCIs for the devices on the Compute node, add the following to **pci_passthru_compute.yaml**:

```
parameter_defaults:
  NovaPCIPassthrough:
    - vendor_id: "10de"
      product_id: "1eb8"
```

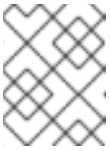
8. You must create a copy of the PCI alias on the Compute node for instance migration and resize operations. To specify the PCI alias for the devices on the Compute node, add the following to **pci_passthru_compute.yaml**:

   - If the PCI device has SR-IOV capabilities:

```
ComputeExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
      device_type: "type-PF"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
      device_type: "type-PF"
```

   - If the PCI device does not have SR-IOV capabilities:

```
ComputeExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
```



NOTE

The Compute node aliases must be identical to the aliases on the Controller node.

9. To enable IOMMU in the server BIOS of the Compute nodes to support PCI passthrough, add the **KernelArgs** parameter to **pci_passthru_compute.yaml**:

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

NOTE

When you first add the **KernelArgs** parameter to the configuration of a role, the overcloud nodes are automatically rebooted. If required, you can disable the automatic rebooting of nodes and instead perform node reboots manually after each overcloud deployment. For more information, see Configuring manual node reboot to define **KernelArgs**.

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

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

11. Configure a flavor to request the PCI devices. The following example requests two devices, each with a vendor ID of **10de** and a product ID of **13f2**:

```
# openstack flavor set m1.large \
  --property "pci_passthrough:alias"="t4:2"
```

Verification

1. Create an instance with a PCI passthrough device:

```
# openstack server create --flavor m1.large \
  --image <custom_gpu> --wait test-pci
```

Replace **<custom_gpu>** with the name of your custom instance image that has the required GPU driver installed.

2. Log in to the instance as a cloud user.

3. To verify that the GPU is accessible from the instance, enter the following command from the instance:

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

4. To check the NVIDIA System Management Interface status, enter the following command from the instance:

```
$ nvidia-smi
```

Example output:

```
-----------------------------------------------------------------------------
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================
====|
|   0  Tesla T4            Off  | 00000000:01:00.0 Off |                    0 |
| N/A   43C    P0    20W /  70W |      0MiB / 15109MiB |      0%      Default |
-------------------------------+----------------------+----------------------


-----------------------------------------------------------------------------
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================
==|
|  No running processes found                                                 |
-----------------------------------------------------------------------------
```

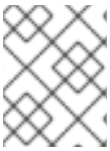
CHAPTER 15. CONFIGURING REAL-TIME COMPUTE

As a cloud administrator, you might need instances on your Compute nodes to adhere to low-latency policies and perform real-time processing. Real-time Compute nodes include a real-time capable kernel, specific virtualization modules, and optimized deployment parameters, to facilitate real-time processing requirements and minimize latency.

The process to enable Real-time Compute includes:

- configuring the BIOS settings of the Compute nodes
- building a real-time image with real-time kernel and Real-Time KVM (RT-KVM) kernel module
- assigning the **ComputeRealTime** role to the Compute nodes

For a use-case example of real-time Compute deployment for NFV workloads, see the Example: Configuring OVS-DPDK with ODL and VXLAN tunnelling section in the *Network Functions Virtualization Planning and Configuration Guide*.



NOTE

Real-time Compute nodes are supported only with Red Hat Enterprise Linux version 7.5 or later.

15.1. PREPARING COMPUTE NODES FOR REAL-TIME

Before you can deploy Real-time Compute in your overcloud, you must enable Red Hat Enterprise Linux Real-Time KVM (RT-KVM), configure your BIOS to support real-time, and build the real-time overcloud image.

Prerequisites

- You must use Red Hat certified servers for your RT-KVM Compute nodes. See Red Hat Enterprise Linux for Real Time 7 certified servers for details.
- You need a separate subscription to *Red Hat OpenStack Platform for Real Time* to access the **rhel-8-for-x86_64-nfv-rpms** repository. For details on managing repositories and subscriptions for your undercloud, see Registering the undercloud and attaching subscriptions in the *Director Installation and Usage* guide.

Procedure

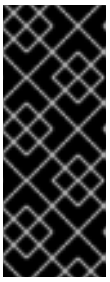
1. To build the real-time overcloud image, you must enable the **rhel-8-for-x86_64-nfv-rpms** repository for RT-KVM. To check which packages will be installed from the repository, enter the following command:

```
$ dnf repo-pkgs rhel-8-for-x86_64-nfv-rpms list
Loaded plugins: product-id, search-disabled-repos, subscription-manager
Available Packages
kernel-rt.x86_64                4.18.0-80.7.1.rt9.153.el8_0          rhel-8-for-x86_64-nfv-rpms
kernel-rt-debug.x86_64          4.18.0-80.7.1.rt9.153.el8_0          rhel-8-for-x86_64-nfv-
rpms
kernel-rt-debug-devel.x86_64    4.18.0-80.7.1.rt9.153.el8_0          rhel-8-for-x86_64-nfv-
rpms
kernel-rt-debug-kvm.x86_64      4.18.0-80.7.1.rt9.153.el8_0          rhel-8-for-x86_64-nfv-
```

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

2. To build the overcloud image for Real-time Compute nodes, install the **libguestfs-tools** package on the undercloud to get the **virt-customize** tool:

```
(undercloud)$ sudo dnf install libguestfs-tools
```



IMPORTANT

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

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

3. Extract the images:

```
(undercloud)$ tar -xf /usr/share/rhosp-director-images/overcloud-full.tar
(undercloud)$ tar -xf /usr/share/rhosp-director-images/ironic-python-agent.tar
```

4. Copy the default image:

```
(undercloud)$ cp overcloud-full.qcow2 overcloud-realtime-compute.qcow2
```

5. Register the image and configure the required subscriptions:

```
(undercloud)$ virt-customize -a overcloud-realtime-compute.qcow2 --run-command
'subscription-manager register --username=<username> --password=<password>'
[  0.0] Examining the guest ...
[ 10.0] Setting a random seed
[ 10.0] Running: subscription-manager register --username=<username> --password=
<password>
[ 24.0] Finishing off
```

Replace the **username** and **password** values with your Red Hat customer account details.

For general information about building a Real-time overcloud image, see the knowledgebase article Modifying the Red Hat Enterprise Linux OpenStack Platform Overcloud Image with virt-customize.

6. Find the SKU of the *Red Hat OpenStack Platform for Real Time* subscription. The SKU might be located on a system that is already registered to the Red Hat Subscription Manager with the same account and credentials:

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

7. Attach the *Red Hat OpenStack Platform for Real Time* subscription to the image:

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

8. Create a script to configure **rt** on the image:

```
(undercloud)$ cat rt.sh
 #!/bin/bash

 set -eux

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

 # END OF SCRIPT
```

9. Run the script to configure the real-time image:

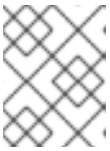
```
(undercloud)$ virt-customize -a overcloud-realtime-compute.qcow2 -v --run rt.sh 2>&1 | tee
virt-customize.log
```

10. Re-label SELinux:

```
(undercloud)$ virt-customize -a overcloud-realtime-compute.qcow2 --selinux-relabel
```

11. Extract **vmlinuz** and **initrd**. For example:

```
(undercloud)$ mkdir image
(undercloud)$ guestmount -a overcloud-realtime-compute.qcow2 -i --ro image
(undercloud)$ cp image/boot/vmlinuz-4.18.0-80.7.1.rt9.153.el8_0.x86_64 ./overcloud-
realtime-compute.vmlinuz
(undercloud)$ cp image/boot/initramfs-4.18.0-80.7.1.rt9.153.el8_0.x86_64.img ./overcloud-
realtime-compute.initrd
(undercloud)$ guestunmount image
```



NOTE

The software version in the **vmlinuz** and **initramfs** filenames vary with the kernel version.

12. Upload the image:

```
(undercloud)$ openstack overcloud image upload \
 --update-existing --os-image-name
 overcloud-realtime-compute.qcow2
```

You now have a real-time image you can use with the **ComputeRealTime** composable role on select Compute nodes.

13. To reduce latency on your Real-time Compute nodes, you must modify the BIOS settings in the Compute nodes. You should disable all options for the following components in your Compute node BIOS settings:

- Power Management

- Power Management
- Hyper-Threading
- CPU sleep states
- Logical processors

  See Setting BIOS parameters for descriptions of these settings and the impact of disabling them. See your hardware manufacturer documentation for complete details on how to change BIOS settings.

15.2. DEPLOYING THE REAL-TIME COMPUTE ROLE

Red Hat OpenStack Platform (RHOSP) director provides the template for the **ComputeRealTime** role, which you can use to deploy real-time Compute nodes. You must perform additional steps to designate Compute nodes for real-time.

Procedure

1. Based on the **/usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-example.yaml** file, create a **compute-real-time.yaml** environment file that sets the parameters for the **ComputeRealTime** role.

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

   The file must include values for the following parameters:

   - **IsolCpusList** and **NovaComputeCpuDedicatedSet**: List of isolated CPU cores and virtual CPU pins to reserve for real-time workloads. This value depends on the CPU hardware of your real-time Compute nodes.
   - **NovaComputeCpuSharedSet**: List of host CPUs to reserve for emulator threads.
   - **KernelArgs**: Arguments to pass to the kernel of the Real-time Compute nodes. For example, you can use **default_hugepagesz=1G hugepagesz=1G hugepages=<number_of_1G_pages_to_reserve> hugepagesz=2M hugepages=<number_of_2M_pages>** to define the memory requirements of guests that have huge pages with multiple sizes. In this example, the default size is 1GB but you can also reserve 2M huge pages.
   - **NovaComputeDisableIrqBalance**: Ensure that this parameter is set to **true** for Real-time Compute nodes, because the **tuned** service manages IRQ balancing for real-time deployments instead of the **irqbalance** service.

2. Add the **ComputeRealTime** role to your roles data file and regenerate the file. For example:

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

   This command generates a **ComputeRealTime** role with contents similar to the following example, and also sets the **ImageDefault** option to **overcloud-realtime-compute**.

   ```
   - name: ComputeRealTime
     description: |
       Compute role that is optimized for real-time behaviour. When using this role
   ```

```
    it is mandatory that an overcloud-realtime-compute image is available and
    the role specific parameters IsolCpusList, NovaComputeCpuDedicatedSet and
    NovaComputeCpuSharedSet are set accordingly to the hardware of the real-time compute
nodes.
  CountDefault: 1
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Tenant:
      subnet: tenant_subnet
    Storage:
      subnet: storage_subnet
  HostnameFormatDefault: '%stackname%-computerealtime-%index%'
  ImageDefault: overcloud-realtime-compute
  RoleParametersDefault:
    TunedProfileName: "realtime-virtual-host"
    KernelArgs: ""      # these must be set in an environment file
    IsolCpusList: ""    # or similar according to the hardware
    NovaComputeCpuDedicatedSet: ""  # of real-time nodes
    NovaComputeCpuSharedSet: ""     #
    NovaLibvirtMemStatsPeriodSeconds: 0
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::BootParams
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::ComputeCeilometerAgent
    - OS::TripleO::Services::ComputeNeutronCorePlugin
    - OS::TripleO::Services::ComputeNeutronL3Agent
    - OS::TripleO::Services::ComputeNeutronMetadataAgent
    - OS::TripleO::Services::ComputeNeutronOvsAgent
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::IpaClient
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MetricsQdr
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronBgpVpnBagpipe
    - OS::TripleO::Services::NeutronLinuxbridgeAgent
    - OS::TripleO::Services::NeutronVppAgent
    - OS::TripleO::Services::NovaCompute
    - OS::TripleO::Services::NovaLibvirt
    - OS::TripleO::Services::NovaLibvirtGuests
    - OS::TripleO::Services::NovaMigrationTarget
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::OpenDaylightOvs
    - OS::TripleO::Services::Podman
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::RsyslogSidecar
```

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

For general information about custom roles and about the **roles-data.yaml**, see Roles.

3. Create the **compute-realtime** flavor to tag nodes that you want to designate for real-time workloads. For example:

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

4. Tag each node that you want to designate for real-time workloads with the **compute-realtime** profile.

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

5. Map the **ComputeRealTime** role to the **compute-realtime** flavor by creating an environment file with the following content:

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

6. Add your environment files and the new roles file to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -r /home/stack/templates/rt~/my_roles_data.yaml \
 -e home/stack/templates/compute-real-time.yaml
```

15.3. SAMPLE DEPLOYMENT AND TESTING SCENARIO

The following example procedure uses a simple single-node deployment to test that the environment variables and other supporting configuration is set up correctly. Actual performance results might vary, depending on the number of nodes and instances that you deploy in your cloud.

Procedure

1. Create the **compute-real-time.yaml** file with the following parameters:

```
parameter_defaults:
  ComputeRealTimeParameters:
    IsolCpusList: "1"
    NovaComputeCpuDedicatedSet: "1"
    NovaComputeCpuSharedSet: "0"
    KernelArgs: "default_hugepagesz=1G hugepagesz=1G hugepages=16"
    NovaComputeDisableIrqBalance: true
```

2. Create a new **rt_roles_data.yaml** file with the **ComputeRealTime** role:

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

3. Add **compute-real-time.yaml** and **rt_roles_data.yaml** to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
  -r /home/stack/rt_roles_data.yaml \
  -e [your environment files] \
  -e /home/stack/templates/compute-real-time.yaml
```

This command deploys one Controller node and one Real-time Compute node.

4. Log into the Real-time Compute node and check the following parameters:

```
[root@overcloud-computerealtime-0 ~]# uname -a
Linux overcloud-computerealtime-0 4.18.0-80.7.1.rt9.153.el8_0.x86_64 #1 SMP PREEMPT
RT Wed Dec 13 13:37:53 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
[root@overcloud-computerealtime-0 ~]# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.18.0-80.7.1.rt9.153.el8_0.x86_64 root=UUID=45ae42d0-
58e7-44fe-b5b1-993fe97b760f ro console=tty0 crashkernel=auto console=ttyS0,115200
default_hugepagesz=1G hugepagesz=1G hugepages=16
[root@overcloud-computerealtime-0 ~]# tuned-adm active
Current active profile: realtime-virtual-host
[root@overcloud-computerealtime-0 ~]# grep ^isolated_cores /etc/tuned/realtime-virtual-host-
variables.conf
isolated_cores=1
[root@overcloud-computerealtime-0 ~]# cat /usr/lib/tuned/realtime-virtual-
host/lapic_timer_adv_ns
4000 # The returned value must not be 0
[root@overcloud-computerealtime-0 ~]# cat
/sys/module/kvm/parameters/lapic_timer_advance_ns
4000 # The returned value must not be 0
# To validate hugepages at a host level:
[root@overcloud-computerealtime-0 ~]# cat /proc/meminfo | grep -E
HugePages_Total|Hugepagesize
HugePages_Total:      64
Hugepagesize:    1048576 kB
# To validate hugepages on a per NUMA level (below example is a two NUMA compute
host):
[root@overcloud-computerealtime-0 ~]# cat
/sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
32
[root@overcloud-computerealtime-0 ~]# cat
/sys/devices/system/node/node1/hugepages/hugepages-1048576kB/nr_hugepages
```

```
32
[root@overcloud-computerealtime-0 ~]# crudini --get /var/lib/config-data/puppet-
generated/nova_libvirt/etc/nova/nova.conf compute cpu_dedicated_set
1
[root@overcloud-computerealtime-0 ~]# crudini --get /var/lib/config-data/puppet-
generated/nova_libvirt/etc/nova/nova.conf compute cpu_shared_set
0
[root@overcloud-computerealtime-0 ~]# systemctl status irqbalance
● irqbalance.service - irqbalance daemon
   Loaded: loaded (/usr/lib/systemd/system/irqbalance.service; enabled; vendor preset:
enabled)
   Active: inactive (dead) since Tue 2021-03-30 13:36:31 UTC; 2s ago
```

15.4. LAUNCHING AND TUNING REAL-TIME INSTANCES

After you deploy and configure Real-time Compute nodes, you can launch real-time instances on those nodes. You can further configure these real-time instances with CPU pinning, NUMA topology filters, and huge pages.

Prerequisites

- The **compute-realtime** flavor exists on the overcloud, as described in Deploying the Real-time Compute role.

Procedure

1. Launch the real-time instance:

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

2. Optional: Verify that the instance uses the assigned emulator threads:

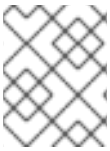
```
# virsh dumpxml <instance_id> | grep vcpu -A1
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='3'/>
  <vcpupin vcpu='2' cpuset='5'/>
  <vcpupin vcpu='3' cpuset='7'/>
  <emulatorpin cpuset='0-1'/>
  <vcpusched vcpus='2-3' scheduler='fifo'
  priority='1'/>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

Pinning CPUs and setting emulator thread policy

To ensure that there are enough CPUs on each Real-time Compute node for real-time workloads, you need to pin at least one virtual CPU (vCPU) for an instance to a physical CPU (pCPUs) on the host. The emulator threads for that vCPU then remain dedicated to that pCPU.

Configure your flavor to use a dedicated CPU policy. To do so, set the **hw:cpu_policy** parameter to **dedicated** on the flavor. For example:

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



NOTE

Make sure that your resources quota has enough pCPUs for the Real-time Compute nodes to consume.

Optimizing your network configuration

Depending on the needs of your deployment, you might need to set parameters in the **network-environment.yaml** file to tune your network for certain real-time workloads.

To review an example configuration optimized for OVS-DPDK, see the Configuring the OVS-DPDK parameters section of the *Network Functions Virtualization Planning and Configuration Guide* .

Configuring huge pages

It is recommended to set the default huge pages size to 1GB. Otherwise, TLB flushes might create jitter in the vCPU execution. For general information about using huge pages, see the Running DPDK applications web page.

Disabling Performance Monitoring Unit (PMU) emulation

Instances can provide PMU metrics by specifying an image or flavor with a vPMU. Providing PMU metrics introduces latency.

NOTE

The vPMU defaults to enabled when **NovaLibvirtCPUMode** is set to **host-passthrough**.

If you do not need PMU metrics, then disable the vPMU to reduce latency by setting the PMU property to "False" in the image or flavor used to create the instance:

- Image: **hw_pmu=False**
- Flavor: **hw:pmu=False**

CHAPTER 16. MANAGING INSTANCES

As a cloud administrator, you can monitor and manage the instances running on your cloud.

16.1. SECURING CONNECTIONS TO THE VNC CONSOLE OF AN INSTANCE

You can secure connections to the VNC console for an instance by configuring the allowed TLS ciphers and the minimum protocol version to enforce for incoming client connections to the VNC proxy service.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

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

3. Open your Compute environment file.
4. Configure the minimum protocol version to use for VNC console connections to instances:

```
parameter_defaults:
  ...
  NovaVNCProxySSLMinimumVersion: <version>
```

   Replace **<version>** with the minimum allowed SSL/TLS protocol version. Set to one of the following valid values:

   - **default**: Uses the underlying system OpenSSL defaults.
   - **tlsv1_1**: Use if you have clients that do not support a later version.

NOTE

TLS 1.0 and TLS 1.1 are deprecated in RHEL 8, and not supported in RHEL 9.

   - **tlsv1_2**: Use if you want to configure the SSL/TLS ciphers to use for VNC console connections to instances.

5. If you set the minimum allowed SSL/TLS protocol version to **tlsv1_2**, then configure the SSL/TLS ciphers to use for VNC console connections to instances:

```
parameter_defaults:
  NovaVNCProxySSLCiphers: <ciphers>
```

   Replace **<ciphers>** with a colon-delimited list of the cipher suites to allow. Retrieve the list of available ciphers from **openssl**.

6. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

16.2. DATABASE CLEANING

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

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

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

16.2.1. Configuring database management

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

Procedure

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

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

   For a complete list of the heat parameters to manage database cron jobs, see Configuration options for the Compute service automated database management.
3. Save the updates to your Compute environment file.
4. Add your Compute environment file to the stack with your other environment files and deploy the overcloud:

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

16.2.2. Configuration options for the Compute service automated database management

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

Table 16.1. Compute (nova) service cron parameters

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

Parameter	Description
NovaCronArchiveDeleteRowsMonth	Use this parameter to configure in which month to run the cron command to move deleted instance records to another table. Default: ***** (every month)
NovaCronArchiveDeleteRowsPurge	Set this parameter to "True" to purge shadow tables immediately after scheduled archiving. Default: **False**
NovaCronArchiveDeleteRowsUntilComplete	Set this parameter to "True" to continue to move deleted instance records to another table until all records are moved. Default: **True**
NovaCronArchiveDeleteRowsUser	Use this parameter to configure the user that owns the crontab that archives deleted instance records and that has access to the log file the crontab uses. Default: **nova**
NovaCronArchiveDeleteRowsWeekday	Use this parameter to configure on which day of the week to run the cron command to move deleted instance records to another table. Default: ***** (every day)
NovaCronPurgeShadowTablesAge	Use this parameter to purge shadow tables based on their age in days. Set to **0** to purge shadow tables older than today. Default: **14**
NovaCronPurgeShadowTablesAllCells	Set this parameter to "True" to purge shadow tables from all cells. Default: **True**
NovaCronPurgeShadowTablesDestination	Use this parameter to configure the file for logging purged shadow tables. Default: **/var/log/nova/nova-rowspurge.log**
NovaCronPurgeShadowTablesHour	Use this parameter to configure the hour at which to run the cron command to purge shadow tables. Default: **5**

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

16.3. MIGRATING VIRTUAL MACHINE INSTANCES BETWEEN COMPUTE NODES

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

Compute node maintenance

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

Failing Compute node

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

Failed Compute nodes

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

Workload rebalancing

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

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

NOTE

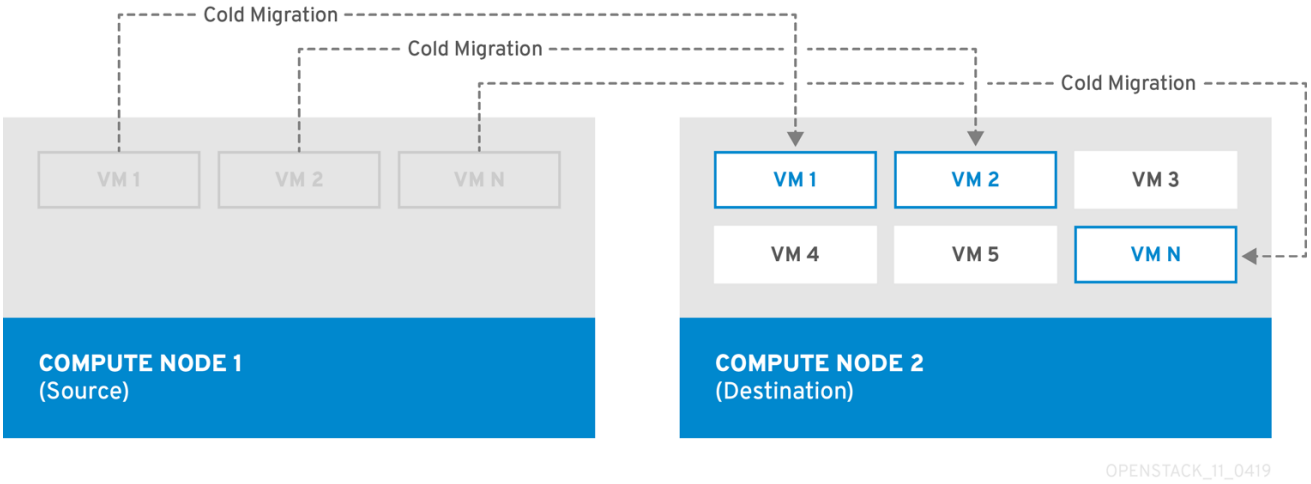
If you have a functioning Compute node, and you want to make a copy of an instance for backup purposes, or to copy the instance to a different environment, follow the procedure in Importing virtual machines into the overcloud in the *Director Installation and Usage* guide.

16.3.1. Migration types

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

Cold migration

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



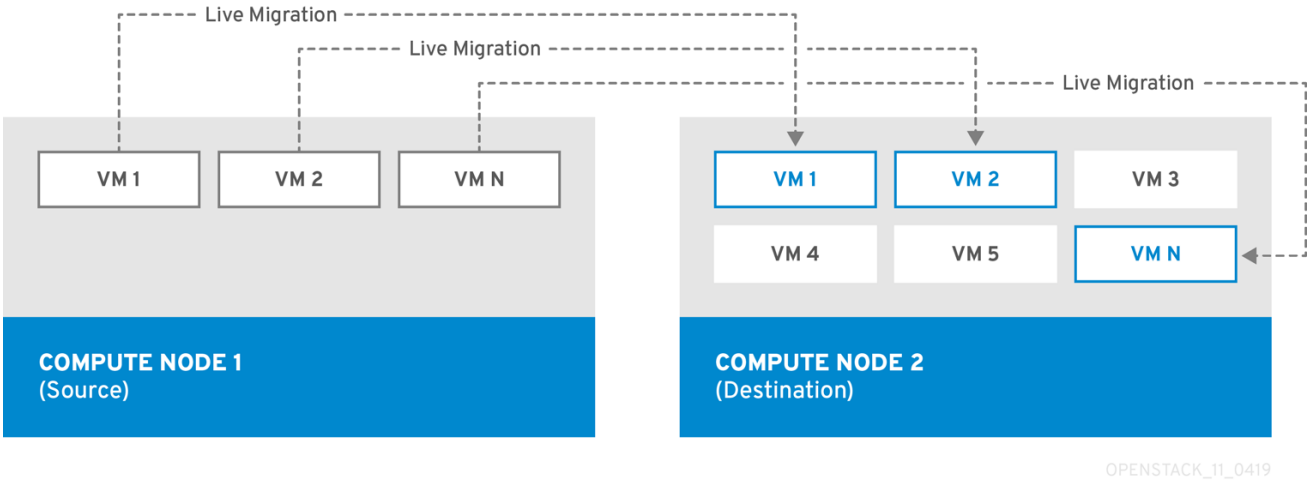
Cold migration involves some downtime for the instance. The migrated instance maintains access to the same volumes and IP addresses.

NOTE

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

Live migration

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



Live migrating an instance involves little or no perceptible downtime. However, live migration does impact performance for the duration of the migration operation. Therefore, instances should be taken out of the critical path while being migrated.

IMPORTANT

Live migration impacts the performance of the workload being moved. Red Hat does not provide support for increased packet loss, network latency, memory latency or a reduction in network bandwith, memory bandwidth, storage IO, or CPU peformance during live migration.

NOTE

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

In some cases, instances cannot use live migration. For more information, see *Migration constraints*.

Evacuation

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

16.3.2. Migration constraints

Migration constraints typically arise with block migration, configuration disks, or when one or more instances access physical hardware on the Compute node.

CPU constraints

The source and destination Compute nodes must have the same CPU architecture. For example, Red Hat does not support migrating an instance from an **x86_64** CPU to a **ppc64le** CPU.

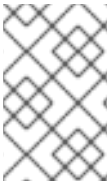
Migration between different CPU models is not supported. In some cases, the CPU of the source and destination Compute node must match exactly, such as instances that use CPU host passthrough. In all cases, the CPU features of the destination node must be a superset of the CPU features on the source node.

Memory constraints

The destination Compute node must have sufficient available RAM. Memory oversubscription can cause migration to fail.

Block migration constraints

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



NOTE

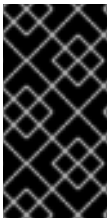
Network congestion in the control plane network caused by migrating local disks or instances that consume large amounts of RAM might impact the performance of other systems that use the control plane network, such as RabbitMQ.

Read-only drive migration constraints

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

Live migration constraints

In some cases, live migrating instances involves additional constraints.



IMPORTANT

Live migration impacts the performance of the workload being moved. Red Hat does not provide support for increased packet loss, network latency, memory latency or a reduction in network bandwidth, memory bandwidth, storage IO, or CPU performance during live migration.

No new operations during migration

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

CPU pinning with NUMA

NovaSchedulerDefaultFilters parameter in the Compute configuration must include the values **AggregateInstanceExtraSpecsFilter** and **NUMATopologyFilter**.

Multi-cell clouds

In a multi-cell cloud, instances can be live migrated to a different host in the same cell, but not across cells.

Floating instances

When live migrating floating instances, if the configuration of **NovaComputeCpuSharedSet** on the destination Compute node is different from the configuration of **NovaComputeCpuSharedSet** on the source Compute node, the instances will not be allocated to the CPUs configured for shared (unpinned) instances on the destination Compute node. Therefore, if you need to live migrate floating instances, you must configure all the Compute nodes with the same CPU mappings for dedicated (pinned) and shared (unpinned) instances, or use a host aggregate for the shared instances.

Destination Compute node capacity

The destination Compute node must have sufficient capacity to host the instance that you want to migrate.

SR-IOV live migration

Instances with SR-IOV-based network interfaces can be live migrated. Live migrating instances with direct mode SR-IOV network interfaces incurs network downtime. This is because the direct mode interfaces need to be detached and re-attached during the migration.

Packet loss on ML2/OVN deployments

ML2/OVN does not support live migration without packet loss. This is because OVN cannot handle multiple port bindings and therefore does not know when a port is being migrated.
To minimize package loss during live migration, configure your ML2/OVN deployment to announce the instance on the destination host once migration is complete:

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      workarounds/enable_qemu_monitor_announce_self:
        value: 'True'
```

Live migration on ML2/OVS deployments

During the live migration process, when the virtual machine is unpaused in the destination host, the metadata service might not be available because the metadata server proxy has not yet spawned. This unavailability is brief. The service becomes available momentarily and the live migration succeeds.

Constraints that preclude live migration

You cannot live migrate an instance that uses the following features.

PCI passthrough

QEMU/KVM hypervisors support attaching PCI devices on the Compute node to an instance. Use PCI passthrough to give an instance exclusive access to PCI devices, which appear and behave as if they are physically attached to the operating system of the instance. However, because PCI passthrough involves direct access to the physical devices, QEMU/KVM does not support live migration of instances using PCI passthrough.

Port resource requests

You cannot live migrate an instance that uses a port that has resource requests, such as a guaranteed minimum bandwidth QoS policy. Use the following command to check if a port has resource requests:

```
$ openstack port show <port_name/port_id>
```

16.3.3. Preparing to migrate

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

Procedure

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

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

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

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

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

3. Optional: If you are migrating instances from a source Compute node to perform maintenance on the node, you must disable the node to prevent the scheduler from assigning new instances to the node during maintenance:

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

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

You are now ready to perform the migration. Follow the required procedure detailed in Cold migrating an instance or Live migrating an instance .

16.3.4. Cold migrating an instance

Cold migrating an instance involves stopping the instance and moving it to another Compute node. Cold migration facilitates migration scenarios that live migrating cannot facilitate, such as migrating instances that use PCI passthrough. The scheduler automatically selects the destination Compute node. For more information, see Migration constraints.

Procedure

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

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

   - Replace **<instance>** with the name or ID of the instance to migrate.
   - Specify the **--block-migration** flag if migrating a locally stored volume.

2. Wait for migration to complete. While you wait for the instance migration to complete, you can check the migration status. For more information, see Checking migration status.

3. Check the status of the instance:

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

A status of "VERIFY_RESIZE" indicates you need to confirm or revert the migration:

- If the migration worked as expected, confirm it:

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

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

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

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

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

4. Restart the instance:

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

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

5. Optional: If you disabled the source Compute node for maintenance, you must re-enable the node so that new instances can be assigned to it:

```
(overcloud)$ openstack compute service set <source> nova-compute --enable
```

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

16.3.5. Live migrating an instance

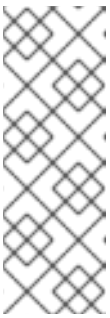
Live migration moves an instance from a source Compute node to a destination Compute node with a minimal amount of downtime. Live migration might not be appropriate for all instances. For more information, see Migration constraints.

Procedure

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

```
(overcloud)$ openstack server migrate <instance> --live-migration [--host <dest>] --wait
```

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



NOTE

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

```
(overcloud)$ openstack server migrate <instance> --live-migration [--host
<dest>] --wait --block-migration
```

2. Confirm that the instance is migrating:

```
(overcloud)$ openstack server show <instance>

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

3. Wait for migration to complete. While you wait for the instance migration to complete, you can check the migration status. For more information, see Checking migration status.

4. Check the status of the instance to confirm if the migration was successful:

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

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

5. Optional: If you disabled the source Compute node for maintenance, you must re-enable the node so that new instances can be assigned to it:

```
(overcloud)$ openstack compute service set <source> nova-compute --enable
```

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

16.3.6. Checking migration status

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

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

Procedure

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

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

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

```
+----+-------------+-----------+ (...)
| 2  | -           | -         | (...)
+----+-------------+-----------+ (...)
```

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

2. Show the status of the migration:

```
$ nova server-migration-show <instance> <migration_id>
```

- Replace **<instance>** with the name or ID of the instance.
- Replace **<migration_id>** with the ID of the migration.
  Running the **nova server-migration-show** command returns the following example output:

```
+------------------------+--------------------------------------+
| Property               | Value                                |
+------------------------+--------------------------------------+
| created_at             | 2017-03-08T02:53:06.000000           |
| dest_compute           | controller                           |
| dest_host              | -                                    |
| dest_node              | -                                    |
| disk_processed_bytes   | 0                                    |
| disk_remaining_bytes   | 0                                    |
| disk_total_bytes       | 0                                    |
| id                     | 2                                    |
| memory_processed_bytes | 65502513                             |
| memory_remaining_bytes | 786427904                            |
| memory_total_bytes     | 1091379200                           |
| server_uuid            | d1df1b5a-70c4-4fed-98b7-423362f2c47c |
| source_compute         | compute2                             |
| source_node            | -                                    |
| status                 | running                              |
| updated_at             | 2017-03-08T02:53:47.000000           |
+------------------------+--------------------------------------+
```

TIP

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

Sometimes instance migration can take a long time or encounter errors. For more information, see Troubleshooting migration.

16.3.7. Evacuating an instance

If you want to move an instance from a dead or shut-down Compute node to a new host in the same environment, you can evacuate it.

The evacuate process destroys the original instance and rebuilds it on another Compute node using the original image, instance name, UUID, network addresses, and any other resources the original instance had allocated to it.

If the instance uses shared storage, the instance root disk is not rebuilt during the evacuate process, as the disk remains accessible by the destination Compute node. If the instance does not use shared storage, then the instance root disk is also rebuilt on the destination Compute node.

NOTE

- You can only perform an evacuation when the Compute node is fenced, and the API reports that the state of the Compute node is "down" or "forced-down". If the Compute node is not reported as "down" or "forced-down", the **evacuate** command fails.
- To perform an evacuation, you must be a cloud administrator.

16.3.7.1. Evacuating one instance

You can evacuate instances one at a time.

Procedure

1. Confirm that the instance is not running:

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

   - Replace **<node>** with the name or UUID of the Compute node that hosts the instance.

2. Confirm that the host Compute node is fenced or shut down:

```
(overcloud)[stack@director ~]$ openstack baremetal node show <node>
```

   - Replace **<node>** with the name or UUID of the Compute node that hosts the instance to evacuate. To perform an evacuation, the Compute node must have a status of **down** or **forced-down**.

3. Disable the Compute node:

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

   - Replace **<node>** with the name of the Compute node to evacuate the instance from.
   - Replace **<disable_host_reason>** with details about why you disabled the Compute node.

4. Evacuate the instance:

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

   - Optional: Replace **<pass>** with the administrative password required to access the evacuated instance. If a password is not specified, a random password is generated and output when the evacuation is complete.

NOTE

The password is changed only when ephemeral instance disks are stored on the local hypervisor disk. The password is not changed if the instance is hosted on shared storage or has a Block Storage volume attached, and no error message is displayed to inform you that the password was not changed.

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

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

5. Optional: Enable the Compute node when it is recovered:

```
(overcloud)[stack@director ~]$ openstack compute service set \
 <node> nova-compute --enable
```

- Replace **<node>** with the name of the Compute node to enable.

16.3.7.2. Evacuating all instances on a host

You can evacuate all instances on a specified Compute node.

Procedure

1. Confirm that the instances to evacuate are not running:

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

- Replace **<node>** with the name or UUID of the Compute node that hosts the instances to evacuate.

2. Confirm that the host Compute node is fenced or shut down:

```
(overcloud)[stack@director ~]$ openstack baremetal node show <node>
```

- Replace **<node>** with the name or UUID of the Compute node that hosts the instances to evacuate. To perform an evacuation, the Compute node must have a status of **down** or **forced-down**.

3. Disable the Compute node:

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

- Replace **<node>** with the name of the Compute node to evacuate the instances from.
- Replace **<disable_host_reason>** with details about why you disabled the Compute node.

4. Evacuate all instances on a specified Compute node:

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

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

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

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

5. Optional: Enable the Compute node when it is recovered:

```
(overcloud)[stack@director ~]$ openstack compute service set \
<node> nova-compute --enable
```

- Replace **<node>** with the name of the Compute node to enable.

16.3.8. Troubleshooting migration

The following issues can arise during instance migration:

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

16.3.8.1. Errors during migration

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

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

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

- A destination Compute host is not available.
- A scheduler exception occurs.
- The rebuild process fails due to insufficient computing resources.
- A server group check fails.

- The instance on the source Compute node gets deleted before migration to the destination Compute node is complete.

16.3.8.2. Never-ending live migration

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

Use one of the following methods to address this situation:

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

Aborting live migration

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

Procedure

1. Retrieve the list of migrations for the instance:

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

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

2. Abort the live migration:

```
$ nova live-migration-abort <instance> <migration_id>
```

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

Forcing live migration to complete

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

IMPORTANT

Forcing live migration to complete might lead to perceptible downtime.

Procedure

1. Retrieve the list of migrations for the instance:

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

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

2. Force the live migration to complete:

```
$ nova live-migration-force-complete <instance> <migration_id>
```

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

16.3.8.3. Instance performance degrades after migration

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