



Red Hat OpenStack Services on OpenShift 18.0

Configuring the Compute service for instance creation

Configuring and managing the Compute service (nova) for creating instances

Red Hat OpenStack Services on OpenShift 18.0 Configuring the Compute service for instance creation

Configuring and managing the Compute service (nova) for creating instances

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Copyright © 2024 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

Configure and manage the Compute service (nova) in a Red Hat OpenStack Services on OpenShift deployment.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	4
ABOUT THIS GUIDE	5
CHAPTER 1. COMPUTE SERVICE (NOVA) FUNCTIONALITY	6
CHAPTER 2. CONFIGURING THE COMPUTE SERVICE (NOVA)	8
CHAPTER 3. CREATING FLAVORS FOR LAUNCHING INSTANCES	9
3.1. CREATING A FLAVOR	9
3.2. FLAVOR ARGUMENTS	10
3.3. FLAVOR METADATA	12
CHAPTER 4. CONFIGURING CPUS ON COMPUTE NODES	28
4.1. CONFIGURING CPU PINNING ON COMPUTE NODES	28
4.1.1. Prerequisites	29
4.1.2. Designating and configuring Compute nodes for CPU pinning	29
4.1.3. Creating a dedicated CPU flavor for instances	32
4.1.4. Creating a shared CPU flavor for instances	33
4.1.5. Creating a mixed CPU flavor for instances	34
4.1.6. Configuring CPU pinning on Compute nodes with simultaneous multithreading (SMT)	35
4.1.7. Additional resources	35
CHAPTER 5. CONFIGURING MEMORY ON COMPUTE NODES	36
5.1. CONFIGURING MEMORY FOR OVERALLOCATION	36
5.2. CALCULATING RESERVED HOST MEMORY ON COMPUTE NODES	38
5.3. CALCULATING SWAP SIZE	39
5.4. CONFIGURING HUGE PAGES ON COMPUTE NODES	39
5.4.1. Creating a huge pages flavor for instances	43
5.4.2. Mounting multiple huge page folders during first boot	44
5.5. CONFIGURING COMPUTE NODES TO USE FILE-BACKED MEMORY FOR INSTANCES	46
5.5.1. Changing the memory backing directory host disk	49
5.6. CONFIGURING AMD SEV COMPUTE NODES TO PROVIDE MEMORY ENCRYPTION FOR INSTANCES	49
5.6.1. Secure Encrypted Virtualization (SEV)	50
5.6.2. Designating AMD SEV Compute nodes for memory encryption	51
5.6.3. Configuring AMD SEV Compute nodes for memory encryption	51
5.6.4. Creating an image for memory encryption	54
5.6.5. Creating a flavor for memory encryption	55
5.6.6. Launching an instance with memory encryption	56
CHAPTER 6. CONFIGURING INSTANCE SCHEDULING AND PLACEMENT	57
6.1. PREFILTERING USING THE PLACEMENT SERVICE	57
6.1.1. Filtering by requested image type support	58
6.1.2. Filtering by resource provider traits	59
6.1.2.1. Creating an image that requires or forbids a resource provider trait	59
6.1.2.2. Creating a flavor that requires or forbids a resource provider trait	61
6.1.3. Filtering by isolating host aggregates	62
6.2. CONFIGURING FILTERS AND WEIGHTS FOR THE COMPUTE SCHEDULER SERVICE	64
6.3. COMPUTE SCHEDULER FILTERS	66
6.4. COMPUTE SCHEDULER WEIGHTS	71
6.5. DECLARING CUSTOM TRAITS AND RESOURCE CLASSES	77
6.6. CREATING AND MANAGING HOST AGGREGATES	83
6.6.1. Enabling scheduling on host aggregates	83

6.6.2. Creating a host aggregate	85
6.6.3. Creating an availability zone	86
6.6.4. Deleting a host aggregate	87
6.6.5. Creating a project-isolated host aggregate	88
CHAPTER 7. ADDING METADATA TO INSTANCES	91
7.1. TYPES OF INSTANCE METADATA	91
CHAPTER 8. CONFIGURING INSTANCE SECURITY	92
CHAPTER 9. DATABASE CLEANING	93
CHAPTER 10. MIGRATING VIRTUAL MACHINE INSTANCES BETWEEN COMPUTE NODES	94
10.1. MIGRATION TYPES	94

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 for Red Hat OpenStack Services on OpenShift (RHOSO) or earlier releases of Red Hat OpenStack Platform (RHOSP). When you create an issue for RHOSO or RHOSP documents, the issue is recorded in the RHOSO Jira project, where you can track the progress of your feedback.

To complete the [Create Issue](#) form, ensure that you are logged in to Jira. If you do not have a Red Hat Jira account, you can create an account at <https://issues.redhat.com>.

1. Click the following link to open a **Create Issue** page: [Create Issue](#)
2. 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.
3. Click **Create**.

ABOUT THIS GUIDE



IMPORTANT

Red Hat is currently reviewing the information and procedures provided in this guide for Red Hat OpenStack Services on OpenShift (RHOSO). This guide will be updated when the reviewed content is available for enterprise use. If you require assistance for the current RHOSO release, please contact Red Hat Support.

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 Services on OpenShift (RHOSO) 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 RHOSO 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.

CHAPTER 2. CONFIGURING THE COMPUTE SERVICE (NOVA)

To designate and configure all node sets for a particular feature or workload, the Compute service (nova) provides a default **ConfigMap** CR named **nova-extra-config**, where you can add generic configuration that applies to all the node sets that use the default **nova** service. If you use this default **nova-extra-config** ConfigMap to add generic configuration to be applied to all the node sets, then you do not need to create a custom service.

Example of a generic feature configuration:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nova-extra-config
  namespace: openstack
data:
  <integer>-<service>-<feature>.conf: |
  [<section>]
  <config_option>=<value>
```

- Replace **<integer>** with an integer that indicates when the configuration should be applied in the series of configuration files that are applied to etc/<service>/<service>.conf.d/ in the <service> container when the service is deployed. Numbers below 25 are reserved for the OpenStack services and Ansible configuration files.
- Replace **<service>** with the name of the service.
- Replace **<feature>** with a string that identifies the feature.
- Replace **<section>** with the section to which you want to add the configuration.

You can configure only whole node sets. Reconfiguring a subset of the nodes within a node set is not supported. If you need to reconfigure a subset of nodes within a node set, you must split the node set via scaling in the current node set, remove the scaled in nodes from the node set, and add them to a new node set.

If your deployment has more than one node set, changes to the **nova-extra-config ConfigMap** might directly affect more than one node set, depending on how the node sets and the DataPlaneServices are configured.

Procedure

1. Create or update the default **ConfigMap** CR named **nova-extra-config**.
2. Create a new **OpenStackDataPlaneDeployment** CR to configure the services on the data plane nodes.
3. Specify **nodeSets** to include all the **OpenStackDataPlaneNodeSet** CRs you want to deploy.
4. Deploy the data plane.

Additional resources

- [Customizing the data plane](#) in *Customizing the Red Hat OpenStack Services on OpenShift deployment*

CHAPTER 3. 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.

There are no default flavors in Red Hat OpenStack Services on OpenShift (RHOSO). To create a flavor, you must use the **openstack flavor create** command, for example:

```
openstack --os-compute-api=2.86 flavor create --ram 128 --disk 1 --vcpus 1 m1.nano
```

This command creates a public flavor called m1.nano with 128MB RAM and 1GB disk size. The API micro version enables flavor extra spec validation. Flavor extra spec validation prevents common typos and similar errors when defining flavors. You specify the micro version by using **--os-compute-api=2.86**.

```
openstack --os-compute-api=2.86 flavor create --ram 196 --disk 1 --vcpus 1 m1.micro
```

This command creates a public flavor called m1.micro with 196MB RAM and 1GB disk size.

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](#).



NOTE

Behaviors set using flavor properties override behaviors 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.

3.1. 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:

```
$ openstack --os-compute-api=2.86 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:

```
$ openstack --os-compute-api=2.86 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:


```
$ openstack --os-compute-api=2.86 flavor set \
--property hw:cpu_sockets=2 \
--property hw:cpu_cores=2 processor_topology_flavor
```

3.2. 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 3.1. 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	<p>(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.</p> <div data-bbox="687 907 798 1075" style="display: inline-block; vertical-align: middle;">  </div> <div data-bbox="874 913 1410 1070" style="display: inline-block; vertical-align: middle; margin-left: 10px;"> <p>NOTE</p> <p>Creation of an instance with a flavor that has --disk set to 0 requires that the instance boots from volume.</p> </div> <p>Default: 0 GB</p>
--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.

3.3. 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.



NOTE

The extra specs for limiting instance CPU resource usage are host-specific tunable properties that are passed directly to libvirt, which then passes the limits onto the host OS. Therefore, the supported instance CPU resource limits configurations are dependent on the underlying host OS.

For more information on how to configure instance CPU resource usage for the Compute nodes in your RHOSO deployment, see [Understanding cgroups](#) in the RHEL 9 documentation, and [CPU Tuning](#) in the Libvirt documentation.

Table 3.2. 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 .
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	<p>Specifies the maximum allowed bandwidth for the vCPU in each cpu_period, in microseconds:</p> <ul style="list-style-type: none"> ● Set to a value in the range 1000 – 18446744073709551. ● Set to 0 to disable. ● Set to a negative value to allow infinite bandwidth. <p>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:</p> <pre>\$ openstack flavor set cpu_limits_flavor \ --property quota:cpu_quota=10000 \ --property quota:cpu_period=20000</pre>

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 and associate a Quality of Service (QoS) specification for the volume type.

Table 3.3. 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.

Key	Description
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.
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 *Configuring Red Hat OpenStack Platform networking* guide. The **quota:vif_*** properties are only supported when you use the ML2/OVS mechanism driver with **NeutronOVSFirewallDriver** set to **iptables_hybrid**.

Table 3.4. 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.

Key	Description
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.

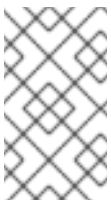
Table 3.5. Flavor metadata for video devices

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 3.6. Flavor metadata for watchdog behavior

Key	Description
hw:watchdog_action	<p>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.</p> <p>Set to one of the following valid values:</p> <ul style="list-style-type: none"> ● 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. <div style="display: flex; align-items: flex-start; margin-top: 20px;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p>NOTE</p> <p>Watchdog behavior that you set by using the properties of a specific image override behavior that you set by using flavors.</p> </div> </div>

Random number generator (RNG)

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

Table 3.7. Flavor metadata for RNG

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 3.8. 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 .

Virtual Trusted Platform Module (vTPM) devices

Use the property keys in the following table to enable a vTPM device for the instance.

Table 3.9. Flavor metadata for vPMU

Key	Description
hw:tpm_version	Set to the version of TPM to use. TPM version 2.0 is the only supported version.

Key	Description
hw:tpm_model	<p>Set to the model of TPM device to use. Ignored if hw:tpm_version is not configured. Set to one of the following valid values:</p> <ul style="list-style-type: none"> ● tpm-tis: (Default) TPM Interface Specification. ● tpm-crb: Command-Response Buffer. Compatible only with TPM version 2.0.

Instance CPU topology

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

Table 3.10. Flavor metadata for CPU topology

Key	Description
hw:cpu_sockets	<p>Specifies the preferred number of sockets for the instance.</p> <p>Default: the number of vCPUs requested</p>
hw:cpu_cores	<p>Specifies the preferred number of cores per socket for the instance.</p> <p>Default: 1</p>
hw:cpu_threads	<p>Specifies the preferred number of threads per core for the instance.</p> <p>Default: 1</p>
hw:cpu_max_sockets	<p>Specifies the maximum number of sockets that users can select for their instances by using image properties.</p> <p>Example: hw:cpu_max_sockets=2</p>
hw:cpu_max_cores	<p>Specifies the maximum number of cores per socket that users can select for their instances by using image properties.</p>
hw:cpu_max_threads	<p>Specifies the maximum number of threads per core that users can select for their instances by using image properties.</p>

Serial ports

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

Table 3.11. 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 3.12. Flavor metadata for CPU pinning

Key	Description
hw:cpu_policy	<p>Specifies the CPU policy to use. Set to one of the following valid values:</p> <ul style="list-style-type: none"> ● 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. ● mixed: The instance vCPUs use a mix of dedicated (pinned) host pCPUs and shared (unpinned) host pCPUs.

Key	Description
<p>hw:cpu_thread_policy</p>	<p>Specifies the CPU thread policy to use when hw:cpu_policy=dedicated. Set to one of the following valid values:</p> <ul style="list-style-type: none"> ● 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: <ul style="list-style-type: none"> ● <code>--property trait:HW_CPU_HYPERTHREADING=forbidden</code> <p>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:</p> <ul style="list-style-type: none"> ○ 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: <ul style="list-style-type: none"> ● <code>--property trait:HW_CPU_HYPERTHREADING=required</code> <p>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.</p>

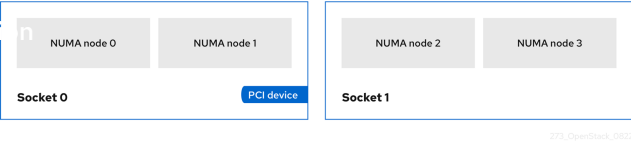
Key	Description
hw:cpu_dedicated_mask	<p>Specifies which CPUs are dedicated (pinned) or shared (unpinned/floating).</p> <ul style="list-style-type: none"> ● To specify dedicated CPUs, specify the CPU number or CPU range. For example, set the property to 2-3 to specify that CPUs 2 and 3 are dedicated and all the remaining CPUs are shared. ● To specify shared CPUs, prepend the CPU number or CPU range with a caret (^). For example, set the property to ^0-1 to specify that CPUs 0 and 1 are shared and all the remaining CPUs are dedicated.

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 3.13. Flavor metadata for PCI NUMA affinity policy

Key	Description
hw:pci_numa_affinity_policy	<p>Specifies the NUMA affinity policy for PCI passthrough devices and SR-IOV interfaces. Set to one of the following valid values:</p> <ul style="list-style-type: none"> ● 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: <ul style="list-style-type: none"> ○ The PCI device has affinity with at least one of the NUMA nodes. ○ The PCI devices do not provide information about their NUMA affinities. ● socket: The Compute service creates an instance that requests a PCI device only when at least one of the instance NUMA nodes has affinity with a NUMA node in the same host socket as the PCI device. For example, the following host architecture has two sockets, each socket has two NUMA nodes, and a PCI device is connected to one of the nodes in one of the sockets.

Key	Description
	 <p>The Compute service can pin an instance with two NUMA nodes and the socket PCI NUMA affinity policy only to the following combinations of host nodes because they all have at least one instance NUMA node pinned to the PCI device's socket:</p> <ul style="list-style-type: none"> ○ node 0 and node 1 ○ node 0 and node 2 ○ node 0 and node 3 ○ node 1 and node 2 ○ node 1 and node 3 <p>The only combination of host nodes that the instance cannot be pinned to is node 2 and node 3, as neither of those nodes are on the same socket as the PCI device. If the other nodes are consumed by other instances and only nodes 2 and 3 are available, the instance does not boot.</p>

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 3.14. 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	<p>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.</p> <p>N starts from 0. Use *.N values with caution, and only if you have at least two NUMA nodes.</p> <p>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.</p>
hw:numa_mem.N	<p>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.</p> <p>N starts from 0. Use *.N values with caution, and only if you have at least two NUMA nodes.</p> <p>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.</p>

**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.

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 3.15. Flavor metadata for CPU real-time policy

Key	Description
hw:cpu_realtime	<p>Set to yes to create a flavor that assigns a real-time policy to the instance vCPUs.</p> <p>Default: no</p>
hw:cpu_realtime_mask	<p>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:</p> <pre>\$ openstack flavor set <flavor> \ --property hw:cpu_realtime="yes" \ --property hw:cpu_realtime_mask=^0-1</pre> <div style="display: flex; align-items: flex-start; margin-top: 10px;"> <div> <p>NOTE</p> <p>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.</p> </div> </div>

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 3.16. Flavor metadata for the emulator threads policy

Key	Description
-----	-------------

Key	Description
hw:emulator_threads_policy	<p>Specifies the emulator threads policy to use for instances. Set to one of the following valid values:</p> <ul style="list-style-type: none"> ● 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 3.17. Flavor metadata for memory page size

Key	Description
hw:mem_page_size	<p>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:</p> <ul style="list-style-type: none"> ● 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.

Table 3.18. Flavor metadata for PCI passthrough

Key	Description
pci_passthrough:alias	<p>Specifies the PCI device to assign to an instance by using the following format:</p> <pre><alias>:<count></pre> <ul style="list-style-type: none"> • 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 3.19. 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.

UEFI Secure Boot

Use the property key in the following table to create an instance that is protected with UEFI Secure Boot.



NOTE

Instances with UEFI Secure Boot must support UEFI and the GUID Partition Table (GPT) standard, and include an EFI system partition.

Table 3.20. Flavor metadata for UEFI Secure Boot

Key	Description
os:secure_boot	Set to required to enable Secure Boot for instances launched with this flavor. Disabled by default.

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 3.21. Flavor metadata for resource traits

Key	Description
trait:<trait_name>	<p>Specifies Compute node traits. Set the trait to one of the following valid values:</p> <ul style="list-style-type: none"> • 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. <p>Example:</p> <pre>\$ openstack flavor set --property trait:HW_CPU_X86_AVX512BW=required avx512- flavor</pre>

Instance bare-metal resource class

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

Table 3.22. Flavor metadata for bare-metal resource class

Key	Description
-----	-------------

Key	Description
<p>resources:<resource_class_name></p>	<p>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.</p> <p>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.</p> <p>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_.</p> <p>For example, to schedule instances on a node that has --resource-class baremetal.SMALL, create the following flavor:</p> <pre data-bbox="687 909 1350 1144"> \$ 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 </pre>

CHAPTER 4. CONFIGURING CPUS ON COMPUTE NODES



WARNING

The content for this feature is available in this release as a *Documentation Preview*, and therefore is not fully verified by Red Hat. Use it only for testing, and do not use in a production environment.

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 CPU performance:

- **CPU pinning:** Pin virtual CPUs to physical CPUs.
- **Emulator threads:** Pin emulator threads associated with the instance to physical CPUs.
- **CPU feature flags** Configure the standard set of CPU feature flags that are applied to instances to improve live migration compatibility across Compute nodes.

4.1. 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 data plane.
4. Create a flavor for launching instances that require CPU pinning.
5. Create a flavor for launching instances that use shared, or floating, CPUs.



NOTE

Configuring CPU pinning creates an implicit NUMA topology on the instance even if a NUMA topology is not requested. Do not run NUMA and non-NUMA virtual machines (VMs) on the same hosts.

4.1.1. Prerequisites

- You know the NUMA topology of your Compute node.
- The **oc** command line tool is installed on your workstation.
- You are logged in to Red Hat OpenStack Services on OpenShift (RHOSO) as a user with **cluster-admin** privileges.

4.1.2. Designating and configuring Compute nodes for CPU pinning

To designate Compute nodes for instances with pinned CPUs, you must create and configure a new **OpenStackDataPlaneNodeSet** custom resource (CR) to configure the nodes that are designated 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 4.1. Example of NUMA Topology

NUMA Node 0		NUMA Node 1	
Core 0	Core 1	Core 4	Core 5
Core 2	Core 3	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.



NOTE

The following procedure applies to new **OpenStackDataPlaneNodeSet** CRs that have not yet been provisioned. To reconfigure an existing **OpenStackDataPlaneNodeSet** that has already been provisioned, you must first drain the guest instances from all the nodes in the **OpenStackDataPlaneNodeSet**.



NOTE

Configuring CPU pinning creates an implicit NUMA topology on the instance even if a NUMA topology is not requested. Do not run NUMA and non-NUMA virtual machines (VMs) on the same hosts.

Prerequisites

- You have selected the **OpenStackDataPlaneNodeSet** CR that defines the nodes for which you want to designate and configure CPU pinning. For more information about creating an **OpenStackDataPlaneNodeSet** CR, see [Creating the data plane](#) in the *Deploying Red Hat OpenStack Services on OpenShift* guide.

Procedure

1. Create or update the **ConfigMap** CR named **nova-extra-config.yaml** and set the values of the parameters under `[compute]` and `[default]`:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: nova-extra-config
  namespace: openstack
data:
  25-nova-cpu-pinning.conf: | 1
    [compute]
    cpu_shared_set = 2,6 2
    cpu_dedicated_set = 1,3,5,7 3
    [DEFAULT]
    reserved_huge_pages = node:0,size:4,count:131072 4
    reserved_huge_pages = node:1,size:4,count:131072

```

- 1** The name of the new Compute configuration file. The **nova-operator** generates the default configuration file with the name **01-nova.conf**. Do not use the default name, because it would override the infrastructure configuration, such as the **transport_url**. The **nova-compute** service applies every file under `/etc/nova/nova.conf.d/` in lexicographical order, therefore configurations defined in later files override the same configurations defined in an earlier file.
- 2** Reserves physical CPU cores for the shared instances.
- 3** Reserves physical CPU cores for the dedicated instances.
- 4** Specifies the amount memory to reserve per NUMA node.

For more information about creating **ConfigMap** objects, see [Creating and using config maps](#).

2. Create a new **OpenStackDataPlaneDeployment** CR to configure the services on the data plane nodes and deploy the data plane, and save it to a file named **compute_cpu_pinning_deploy.yaml** on your workstation:

```

apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: openstack-edpm-cpu-pinning

```

For more information about creating an **OpenStackDataPlaneDeployment** CR, see [Deploying the data plane](#) in the *Deploying Red Hat OpenStack Services on OpenShift* guide.

3. In the **compute_cpu_pinning_deploy.yaml**, specify **nodeSets** to include all the **OpenStackDataPlaneNodeSet** CRs you want to deploy. Ensure that you include the **OpenStackDataPlaneNodeSet** CR that you selected as a prerequisite. That **OpenStackDataPlaneNodeSet** CR defines the nodes you want to designate for CPU pinning.

**WARNING**

You can configure only whole node sets. Reconfiguring a subset of the nodes within a node set is not supported. If you need to reconfigure a subset of nodes within a node set, you must scale the node set down, and create a new node set from the previously removed nodes.

**WARNING**

If your deployment has more than one node set, changes to the **nova-extra-config.yaml ConfigMap** might directly affect more than one node set, depending on how the node sets and the **DataPlaneServices** are configured. To check if a node set uses the **nova-extra-config ConfigMap** and therefore will be affected by the reconfiguration, complete the following steps:

1. Check the services list of the node set and find the name of the **DataPlaneService** that points to nova.
2. Ensure that the value of the **edpmServiceType** field of the **DataPlaneService** is set to **nova**.

If the **dataSources** list of the **DataPlaneService** contains a **configMapRef** named **nova-extra-config**, then this node set uses this **ConfigMap** and therefore will be affected by the configuration changes in this **ConfigMap**. If some of the node sets that are affected should not be reconfigured, you must create a new **DataPlaneService** pointing to a separate **ConfigMap** for these node sets.

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: openstack-edpm-cpu-pinning
spec:
  nodeSets:
    - openstack-edpm
    - compute-cpu-pinning
    - ...
    - <nodeSet_name>
```

- Replace **<nodeSet_name>** with the names of the **OpenStackDataPlaneNodeSet** CRs that you want to include in your data plane deployment.
4. Save the **compute_cpu_pinning_deploy.yaml** deployment file.
 5. Deploy the data plane:

```
$ oc create -f compute_cpu_pinning_deploy.yaml
```

- Verify that the data plane is deployed:

```
$ oc get openstackdataplanenodeset
NAME          STATUS MESSAGE
compute-cpu-pinning True   Deployed
```

- Access the remote shell for **openstackclient** and verify that the deployed Compute nodes are visible on the control plane:

```
$ oc rsh -n openstack openstackclient
$ openstack hypervisor list
```

4.1.3. 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 configured on the host if you intend to use the **required cpu_thread_policy**. You can have a mix of SMT and non-SMT Compute hosts. Flavors with the **require cpu_thread_policy** will land on SMT hosts, and flavors with **isolate** will land on non-SMT.
- The Compute node is configured to allow CPU pinning. For more information, see [Configuring CPU pinning on the Compute nodes](#).

Procedure

- Create a flavor for instances that require CPU pinning:

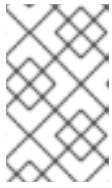
```
$ openstack flavor create --ram <size_mb> \
  --disk <size_gb> --vcpus <num_guest_vcpus> pinned_cpus
```

- If you are not using file-backed memory, set the **hw:mem_page_size** property of the flavor to enable NUMA-aware memory allocation:

```
$ openstack --os-compute-api=2.86 flavor set \
  --property hw:mem_page_size=<page_size> pinned_cpus
```

- Replace **<page_size>** with 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 page size by using the **hw_mem_page_size** set on the image. If the page size is not specified by the image, selects the largest available page size, as determined by the libvirt driver.

- **<pagesize>**: 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.



NOTE

To set **hw:mem_page_size** to **small** or **any**, you must have configured the amount of memory pages to reserve on each NUMA node for processes that are not instances.

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

```
$ openstack --os-compute-api=2.86 flavor set \
  --property hw:cpu_policy=dedicated pinned_cpus
```

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

```
$ openstack --os-compute-api=2.86 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.

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

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

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

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

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

```
$ openstack --os-compute-api=2.86 flavor set \
--property hw:cpu_policy=shared floating_cpus
```

4.1.5. Creating a mixed CPU flavor for instances

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

Procedure

- Create a flavor for instances that require a mix of dedicated and shared CPUs:

```
$ openstack flavor create --ram <size_mb> \
--disk <size_gb> --vcpus <number_of_reserved_vcpus> \
--property hw:cpu_policy=mixed mixed_CPUs_flavor
```

- Specify which CPUs must be dedicated or shared:

```
$ openstack --os-compute-api=2.86 flavor set \
--property hw:cpu_dedicated_mask=<CPU_MASK> \
mixed_CPUs_flavor
```

- Replace **<CPU_MASK>** with the CPUs that must be either dedicated or shared:
 - To specify dedicated CPUs, specify the CPU number or CPU range. For example, set the property to **2-3** to specify that CPUs 2 and 3 are dedicated and all the remaining CPUs are shared.
 - To specify shared CPUs, prepend the CPU number or CPU range with a caret (^). For example, set the property to **^0-1** to specify that CPUs 0 and 1 are shared and all the remaining CPUs are dedicated.
- If you are not using file-backed memory, set the **hw:mem_page_size** property of the flavor to enable NUMA-aware memory allocation:

```
$ openstack --os-compute-api=2.86 flavor set \
--property hw:mem_page_size=<page_size> mixed_CPUs_flavor
```

- Replace **<page_size>** with 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 page size by using the **hw_mem_page_size** set on the image. If the page size is not specified by the image, selects the largest available page size, as determined by the libvirt driver.

- **<pagesize>**: 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.



NOTE

To set **hw:mem_page_size** to **small** or **any**, you must have configured the amount of memory pages to reserve on each NUMA node for processes that are not instances.

4.1.6. 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
```

4.1.7. Additional resources

- [Discovering your NUMA node topology](#)

CHAPTER 5. CONFIGURING MEMORY ON COMPUTE NODES



WARNING

The content for this feature is available in this release as a *Documentation Preview*, and therefore is not fully verified by Red Hat. Use it only for testing, and do not use in a production environment.

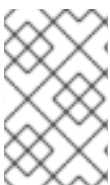
As a cloud administrator, you can configure the scheduling and placement of instances for optimal performance by creating customized flavors to target specialized workloads, including NFV and High Performance Computing (HPC).

Use the following features to tune your instances:

- **Overallocation:** Tune the virtual RAM to physical RAM allocation ratio.
- **Swap:** Tune the allocated swap size to handle memory overcommit.
- **Huge pages:** Tune instance memory allocation policies both for normal memory (4k pages) and huge pages (2 MB or 1 GB pages).
- **File-backed memory:** Use to expand your Compute node memory capacity.
- **SEV:** Use to enable your cloud users to create instances that use memory encryption.

5.1. CONFIGURING MEMORY FOR OVERALLOCATION

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



NOTE

If your **ram_allocation_ratio** 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. Open the **OpenStackDataPlaneNodeSet** CR definition file for the node set you want to update, for example, **my_data_plane_node_set.yaml**.
2. Add the required configuration or modify the existing configuration under **ansibleVars**:

```
apiVersion: dataplane.openstack.org/v1beta1
```



```

kind: OpenStackDataPlaneNodeSet
metadata:
  name: my-data-plane-node-set
spec:
  ...
  nodeTemplate:
    ...
    ansible:
      ansibleVars:
        edpm_bootstrap_swap_size_megabytes: 1024
        edpm_bootstrap_swap_path: /swap
        edpm_bootstrap_swap_partition_enabled: false
        edpm_bootstrap_swap_partition_label: swap1
    ...

```

3. Save the **OpenStackDataPlaneNodeSet** CR definition file.
4. Apply the updated **OpenStackDataPlaneNodeSet** CR configuration:

```
$ oc apply -f my_data_plane_node_set.yaml -n openstack
```

5. Verify that the data plane resource has been updated:

```
$ oc get openstackdataplanenodeset
```

Sample output:

```

NAME                STATUS MESSAGE
my-data-plane-node-set  False  Deployment not started

```

6. Create a file on your workstation to define the **OpenStackDataPlaneDeployment** CR, for example, **my_data_plane_deploy.yaml**:

```

apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: my-data-plane-deploy

```

TIP

Give the definition file and the **OpenStackDataPlaneDeployment** CR a unique and descriptive name that indicates the purpose of the modified node set.

7. Add the **OpenStackDataPlaneNodeSet** CR that you modified:

```

spec:
  nodeSets:
    - my-data-plane-node-set

```

8. Save the **OpenStackDataPlaneDeployment** CR deployment file.
9. Deploy the modified **OpenStackDataPlaneNodeSet** CR:

```
$ oc create -f my_data_plane_deploy.yaml -n openstack
```

10. You can view the Ansible logs while the deployment executes:

```
$ oc get pod -l app=openstackansibleee -n openstack -w
$ oc logs -l app=openstackansibleee -n openstack -f \
--max-log-requests 10
```

11. Verify that the modified **OpenStackDataPlaneNodeSet** CR is deployed:

Example:

```
$ oc get openstackdataplanedeployment -n openstack
```

Sample output

```
NAME                STATUS MESSAGE
my-data-plane-node-set True    Setup Complete
```

12. Repeat the **oc get** command until you see the NodeSet Ready message:

Example

```
$ oc get openstackdataplanenodeset -n openstack
```

Sample output:

```
NAME                STATUS MESSAGE
my-data-plane-node-set True    NodeSet Ready
```

For information on the meaning of the returned status, see [Data plane conditions and states](#) in the *Deploying Red Hat OpenStack Services on OpenShift* .

5.2. 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, Ceph Object Storage Daemon (OSD) consumes 3 GB of memory.
- The emulator overhead required to host instances.
- The hypervisor for each instance.

You can use the following formula to calculate the amount of memory to reserve for host processes on each node:

```
reserved_host_memory_mb = 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.



NOTE

If this host will run workloads with a guest NUMA topology, for example, instances with CPU pinning, huge pages, or an explicit NUMA topology specified in the flavor, you must use the **reserved_huge_pages** configuration option to reserve the memory per NUMA node as 4096 pages.

For information about how to calculate the **reserved_host_memory_mb** value, see [Calculating reserved host memory on Compute nodes](#).

5.3. CALCULATING SWAP SIZE

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

- $\text{overcommit_ratio} = \text{ram_allocation_ratio} - 1$
- Minimum swap size (MB) = $(\text{total_RAM} * \text{overcommit_ratio}) + \text{RHEL_min_swap}$
- Recommended (maximum) swap size (MB) = $\text{total_RAM} * (\text{overcommit_ratio} + \text{percentage_of_RAM_to_use_for_swap})$

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

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

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

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

5.4. CONFIGURING HUGE PAGES ON COMPUTE NODES

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



NOTE

Configuring huge pages creates an implicit NUMA topology on the instance even if a NUMA topology is not requested. Do not run NUMA and non-NUMA virtual machines (VMs) on the same hosts.

Prerequisites

- The **oc** command line tool is installed on your workstation.

- You are logged in to Red Hat OpenStack Services on OpenShift (RHOSO) as a user with **cluster-admin** privileges.
- You have selected the **OpenStackDataPlaneNodeSet** CR that defines the nodes for which you want to enable instances to request huge pages. For more information about creating an **OpenStackDataPlaneNodeSet** CR, see [Creating a set of data plane nodes](#) in *Deploying Red Hat OpenStack Services on OpenShift*.

Procedure

1. Create or update the **ConfigMap** CR named **nova-extra-config.yaml** and set the values of the parameters under `[default]` and `[libvirt]`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nova-extra-config
  namespace: openstack
data:
  28-nova-huge-pages.conf: | 1
    [default]
      reserved_huge_pages = node:0,size:2048,count:64
      reserved_huge_pages = node:1,size:1GB,count:1
    [libvirt]
      cpu_mode = custom
      cpu_models = Haswell-noTSX
      cpu_model_extra_flags = vmx, pdpe1gb, +pcid
```

- 1** The name of the new Compute configuration file. The **nova-operator** generates the default configuration file with the name **01-nova.conf**. Do not use the default name, because it would override the infrastructure configuration, such as the **transport_url**. The **nova-compute** service applies every file under `/etc/nova/nova.conf.d/` in lexicographical order, therefore configurations defined in later files override the same configurations defined in an earlier file.

**NOTE**

- Do not configure CPU feature flags to allow instances to only request 2 MB huge pages.
- You can only allocate 1G huge pages to an instance if the host supports 1G huge page allocation.
- You only need to set **cpu_model_extra_flags** to **pdpe1gb** when **cpu_mode** is set to **host-model** or **custom**.
- If the host supports **pdpe1gb**, and **host-passthrough** is used as the **cpu_mode**, then you do not need to set **pdpe1gb** as a **cpu_model_extra_flags**.

**NOTE**

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 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.

For more information about creating **ConfigMap** objects, see [Creating and using config maps](#).

2. Create a new **OpenStackDataPlaneDeployment** CR to configure the services on the data plane nodes and deploy the data plane, and save it to a file named **compute_huge_pages_deploy.yaml** on your workstation:

```
$ openstack baremetal node set \
kind: OpenStackDataPlaneDeployment
metadata:
name: openstack-edpm-huge-pages
```

3. In the **compute_huge_pages_deploy.yaml**, specify **nodeSets** to include all the **OpenStackDataPlaneNodeSet** CRs you want to deploy. Ensure that you include the **OpenStackDataPlaneNodeSet** CR that you selected as a prerequisite. That **OpenStackDataPlaneNodeSet** CR defines the nodes you want to designate for huge pages.

**WARNING**

You can configure only whole node sets. Reconfiguring a subset of the nodes within a node set is not supported. If you need to reconfigure a subset of nodes within a node set, you must scale the node set down, and create a new node set from the previously removed nodes.

**WARNING**

If your deployment has more than one node set, changes to the **nova-extra-config.yaml ConfigMap** might directly affect more than one node set. To check if a node set uses the **nova-extra-config.yaml ConfigMap** and therefore will be affected by the reconfiguration, complete the following steps:

1. Navigate to the services list of the node set and find the name of the **DataPlaneService** that points to nova.
2. Ensure that the value of the **edpmServiceType** field of the **DataPlaneService** is set to **nova**.
If the **dataSources** list of the **DataPlaneService** contains a **configMapRef** named **nova-extra-config**, then this node set uses this **ConfigMap** and therefore will be affected by the configuration changes in this **ConfigMap**. If some of the node sets that are affected should not be reconfigured, you must create a new **DataPlaneService** pointing to a separate **ConfigMap** for these node sets.

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: openstack-edpm-huge-pages
spec:
  nodeSets:
    - openstack-edpm
    - compute-huge-pages
    - ...
    - <nodeSet_name>
```

- Replace **<nodeSet_name>** with the names of the **OpenStackDataPlaneNodeSet** CRs that you want to include in your data plane deployment.
4. Save the **compute_huge_pages_deploy.yaml** deployment file.
 5. Deploy the data plane:

```
$ oc create -f compute_huge_pages_deploy.yaml
```

6. Verify that the data plane is deployed:

```
$ oc get openstackdataplanenodeset

NAME          STATUS MESSAGE
compute-huge-pages True   Deployed
```

7. Access the remote shell for **openstackclient** and verify that the deployed Compute nodes are visible on the control plane:

```
$ oc rsh -n openstack openstackclient
```

```
$ openstack hypervisor list
```

5.4.1. 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.



NOTE

To execute **openstack** client commands on the cloud you must specify the name of the cloud detailed in your **clouds.yaml** file. You can specify the name of the cloud by using one of the following methods:

- Use the **--os-cloud** option with each command:

```
$ openstack flavor list --os-cloud <cloud_name>
```

Use this option if you access more than one cloud.

- Create an environment variable for the cloud name in your **bashrc** file:

```
`export OS_CLOUD=<cloud_name>`
```

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 <num_reserved_vcpus> huge_pages
```

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

```
$ openstack --os-compute-api=2.86 flavor set huge_pages --property hw:mem_page_size=
<page_size>
```

- Replace **<page_size>** with 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 page size by using the **hw_mem_page_size** set on the image. If the page size is not specified by the image, selects the largest available page size, as determined by the libvirt driver.

- **<pagesize>**: 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.

5.4.2. 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.

Procedure

1. Open the **OpenStackDataPlaneNodeSet** CR definition file for the node set you want to update, for example, **my_data_plane_node_set.yaml**. Add the required configuration or modify the existing configuration in the **edpm_default_mounts** template under **ansibleVars**:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: my-data-plane-node-set
spec:
  ...
  nodeTemplate:
    ...
    ansible:
      ansibleVars:
        edpm_default_mounts: |
          [
            {
              "name": "hugepages1G",
              "path": "/dev/hugepages1G",
              "opts": "pagesize=1G",
              "fstype": "hugetlbfs",
              "group": "hugetlbfs"
            },
            {
              "name": "hugepages2M",
              "path": "/dev/hugepages2M",
              "opts": "pagesize=2M",
              "fstype": "hugetlbfs",
              "group": "hugetlbfs"
            }
          ]
      ...
```


2. Save the **OpenStackDataPlaneNodeSet** CR definition file.
3. Apply the updated **OpenStackDataPlaneNodeSet** CR configuration:

```
$ oc apply -f my_data_plane_node_set.yaml
```

4. Verify that the data plane resource has been updated:

```
$ oc get openstackdataplanenodeset
```

Sample output:

```
NAME                STATUS MESSAGE
my-data-plane-node-set  False  Deployment not started
```

5. Create a file on your workstation to define the **OpenStackDataPlaneDeployment** CR, for example, **my_data_plane_deploy.yaml**:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: my-data-plane-deploy
```

TIP

Give the definition file and the **OpenStackDataPlaneDeployment** CR a unique and descriptive name that indicates the purpose of the modified node set.

6. Add the **OpenStackDataPlaneNodeSet** CR that you modified:

```
spec:
  nodeSets:
    - my-data-plane-node-set
```

7. Save the **OpenStackDataPlaneDeployment** CR deployment file.
8. Deploy the modified **OpenStackDataPlaneNodeSet** CR:

```
$ oc create -f my_data_plane_deploy.yaml -n openstack
```

9. To view the Ansible logs while the deployment executes, enter the following command:

```
$ oc get pod -l app=openstackansibleeee -n openstack -w
$ oc logs -l app=openstackansibleeee -n openstack -f \
--max-log-requests 10
```

10. Verify that the modified **OpenStackDataPlaneNodeSet** CR is deployed: Example:

```
$ oc get openstackdataplanedeployment -n openstack
```

Sample output:

```

NAME                STATUS MESSAGE
my-data-plane-node-set True    Setup Complete

```

- Repeat the **oc get** command until you see the **NodeSet Ready** message:
Example:

```
$ oc get openstackdataplanenodeset -n openstack
```

Sample output:

```

NAME                STATUS MESSAGE
my-data-plane-node-set True    NodeSet Ready

```

For information on the meaning of the returned status, see [Data plane conditions and states](#) in the *Deploying Red Hat OpenStack Services on OpenShift* .

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

You can use file-backed memory to expand your Compute node memory capacity. In this case, you allocate files within the libvirt memory backing directory to be instance memory. You can configure the amount of the host disk that is available for instance memory, and the location on the disk of the instance memory files.

The Compute (nova) 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 **reserved_host_memory_mb**. 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

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

- The **oc** command line tool is installed on your workstation.
- You are logged in to Red Hat OpenStack Services on OpenShift (RHOSO) as a user with **cluster-admin** privileges.
- You have selected the **OpenStackDataPlaneNodeSet** CR that defines which nodes use file-backed memory for instances. For more information about creating an **OpenStackDataPlaneNodeSet** CR, see [Creating a set of data plane nodes](#) in the *Deploying Red Hat OpenStack Services on OpenShift* guide.

Procedure

1. Create or update the **ConfigMap** CR named **nova-extra-config.yaml** and set the values of the parameters under `[libvirt]`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nova-extra-config
  namespace: openstack
data:
  30-nova-file-backed-memory.conf: |
    [libvirt]
    file_backed_memory = 1048576
```

For more information about creating **ConfigMap** objects, see [Creating and using config maps](#).

2. Optional: To configure the directory to store the memory backing files, set the **memory_backing_dir** parameter. The default memory backing directory is `/var/lib/libvirt/qemu/ram/`:

```
[libvirt]
file_backed_memory = 1048576
memory_backing_dir = <new_directory_location>
```

- Replace `<new_directory_location>` with the location of the memory backing directory.



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](#).

3. Create a new **OpenStackDataPlaneDeployment** CR to configure the services on the data plane nodes and deploy the data plane, and save it to a file named **compute_file_backed_memory_deploy.yaml** on your workstation:

```
$ openstack baremetal node set \
kind: OpenStackDataPlaneDeployment
metadata:
  name: compute-file-backed-memory
```

4. In the **compute_file_backed_memory_deploy.yaml**, specify **nodeSets** to include all the

OpenStackDataPlaneNodeSet CRs you want to deploy. Ensure that you include the **OpenStackDataPlaneNodeSet** CR that you selected as a prerequisite. That **OpenStackDataPlaneNodeSet** CR defines the nodes you want to designate for file-backed memory.



WARNING

You can configure only whole node sets. Reconfiguring a subset of the nodes within a node set is not supported. If you need to reconfigure a subset of nodes within a node set, you must scale the node set down, and create a new node set from the previously removed nodes.



WARNING

If your deployment has more than one node set, changes to the **nova-extra-config.yaml ConfigMap** might directly affect more than one node set, depending on how the node sets and the **DataPlaneServices** are configured. To check if a node set uses the **nova-extra-config ConfigMap** and therefore will be affected by the reconfiguration, complete the following steps:

1. Check the services list of the node set and find the name of the **DataPlaneService** that points to nova.
2. Ensure that the value of the **edpmServiceType** field of the **DataPlaneService** is set to **nova**.
If the **dataSources** list of the **DataPlaneService** contains a **configMapRef** named **nova-extra-config**, then this node set uses this **ConfigMap** and therefore will be affected by the configuration changes in this **ConfigMap**. If some of the node sets that are affected should not be reconfigured, you must create a new **DataPlaneService** pointing to a separate **ConfigMap** for these node sets.

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: compute-file-backed-memory
spec:
  nodeSets:
    - openstack-edpm
    - compute-file-backed-memory
    - ...
    - <nodeSet_name>
```

- Replace `<nodeSet_name>` with the names of the OpenStackDataPlaneNodeSet CRs that you want to include in your data plane deployment.
5. Save the `compute_file_backed_memory_deploy.yaml` deployment file.
 6. Deploy the data plane:

```
$ oc create -f compute_file_backed_memory_deploy.yaml
```

7. Verify that the data plane is deployed:

```
$ oc get openstackdataplanenodeset

NAME                STATUS MESSAGE
compute-file-backed-memory True   Deployed
```

8. Access the remote shell for `openstackclient` and verify that the deployed Compute nodes are visible on the control plane:

```
$ oc rsh -n openstack openstackclient

$ openstack hypervisor list
```

5.5.1. 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.

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

Secure Encrypted Virtualization (SEV) hardware, 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.

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 data plane.
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](#).

5.6.1. 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.

5.6.2. 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 node set to configure the AMD SEV role, and configure the bare metal nodes with an AMD SEV resource class to use to tag the Compute nodes for memory encryption.



NOTE

The following procedure applies to new overcloud nodes that have not yet been provisioned. To assign a resource class to an existing overcloud node that has already been provisioned, you must use the scale down procedure to unprovision the node, then use the scale up procedure to reprovision the node with the new resource class assignment. For more information, see [Scaling overcloud nodes](#).

For more information, see [Configuring a node set for a feature or workload](#) in *Customizing the Red Hat OpenStack Services on OpenShift deployment*.

5.6.3. 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

- The **oc** command line tool is installed on your workstation.
- You are logged in to Red Hat OpenStack Services on OpenShift (RHOSO) as a user with **cluster-admin** privileges.
- You have selected the **OpenStackDataPlaneNodeSet** CR that defines the nodes for which you want to designate and configure CPU pinning. For more information about creating an **OpenStackDataPlaneNodeSet** CR, see *Creating the data plane in the Deploying Red Hat OpenStack Services on OpenShift guide*.
- 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. Create or update the ConfigMap CR named `nova-extra-config.yaml` and set the values of the parameters under `[libvirt]`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nova-extra-config
  namespace: openstack
data:
  30-nova-amd-sev.conf: |
    [libvirt]
    num_memory_encrypted_guests = 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.



NOTE

Q35 machine type is the default machine type and is required for SEV.

2. To configure the kernel parameters for the AMD SEV Compute nodes, open the **OpenStackDataPlaneNodeSet** CR definition file for the node set you want to update, for example, `my_data_plane_node_set.yaml`.
3. Add the required network configuration or modify the existing configuration. Place the configuration in the under **ansibleVars**:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: my-data-plane-node-set
spec:
  ...
  nodeTemplate:
    ...
    ansible:
      ansibleVars:
        edpm_kernel_args: "default_hugepagesz=1GB hugepagesz=1G hugepages=64 iommu=pt
        intel_iommu=on tsx=off isolcpus=2-11,14-23"
```


4. Create a new **OpenStackDataPlaneDeployment** CR to configure the services on the data plane nodes and deploy the data plane, and save it to a file named **compute_amd_sev_deploy.yaml** on your workstation:

```
$ openstack baremetal node set \
kind: OpenStackDataPlaneDeployment
metadata:
name: openstack-edpm-amd_sev
```

5. In the **compute_amd_sev_deploy.yaml**, specify **nodeSets** to include all the **OpenStackDataPlaneNodeSet** CRs you want to deploy. Ensure that you include the **OpenStackDataPlaneNodeSet** CR that you selected as a prerequisite. That **OpenStackDataPlaneNodeSet** CR defines the nodes you want to designate for memory encryption.



WARNING

You can configure only whole node sets. Reconfiguring a subset of the nodes within a node set is not supported. If you need to reconfigure a subset of nodes within a node set, you must scale the node set down, and create a new node set from the previously removed nodes.



WARNING

If your deployment has more than one node set, changes to the **nova-extra-config.yaml** ConfigMap might directly affect more than one node set, depending on how the NodeSets and the DataPlaneServices are configured. To check if a node set uses the **nova-extra-config.yaml** ConfigMap and therefore will be affected by the reconfiguration, complete the following steps:

1. Check the services list of the node set and find the name of the DataPlaneService that points to nova.
2. Ensure that the value of the **edpmServiceType** field of the DataPlaneService is set to **nova**.
If the dataSources list of the DataPlaneService contains a configMapRef named **nova-extra-config**, then this node set uses this ConfigMap and therefore will be affected by the configuration changes in this ConfigMap. If some of the node sets that are affected should not be reconfigured, you must create a new DataPlaneService pointing to a separate ConfigMap for these node sets.

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
```

```

metadata:
  name: openstack-edpm-amd-sev
spec:
  nodeSets:
    - openstack-edpm
    - compute-amd-sev
    - my-data-plane-node-set
    - ...
    - <nodeSet_name>

```

- Replace **<nodeSet_name>** with the names of the OpenStackDataPlaneNodeSet CRs that you want to include in your data plane deployment.

6. Save the **compute_amd_sev_deploy.yaml** deployment file.

7. Deploy the data plane:

```
$ oc create -f compute_amd_sev_deploy.yaml
```

8. Verify that the data plane is deployed:

```
$ oc get openstackdataplanenodeset
NAME          STATUS MESSAGE
openstack-edpm True   Deployed
```

9. Access the remote shell for **openstackclient** and verify that the deployed Compute nodes are visible on the control plane:

```
$ oc rsh -n openstack openstackclient
$ openstack hypervisor list
```

5.6.4. Creating an image for memory encryption

When the data plane 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.



NOTE

To execute **openstack** client commands on the cloud you must specify the name of the cloud detailed in your **clouds.yaml** file. You can specify the name of the cloud by using one of the following methods:

- Use the **--os-cloud** option with each command:

```
$ openstack flavor list --os-cloud <cloud_name>
```

Use this option if you access more than one cloud.

- Create an environment variable for the cloud name in your **bashrc** file:

```
`export OS_CLOUD=<cloud_name>`
```

Prerequisites

- The administrator has created a project for you and they have provided you with a **clouds.yaml** file for you to access the cloud.
- You have installed the **python-openstackclient** package.

Procedure

1. Create a new image for memory encryption:

```
$ 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. Add the property **hw_mem_encryption=True** to the image to enable AMD SEV memory encryption on the image:

```
$ 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](#).

5.6.5. Creating a flavor for memory encryption

When the data plane 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.

Prerequisites

- You have the **oc** command line tool installed on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.



NOTE

An AMD SEV flavor is necessary only when the **hw_mem_encryption** property is not set on an image.

Procedure

1. Access the remote shell for the OpenStackClient pod from your workstation:

```
$ oc rsh -n openstack openstackclient
```

2. Create a flavor for memory encryption:

-

```
$ openstack flavor create --vcpus 1 --ram 512 --disk 2 \
--property hw:mem_encryption=True m1.small-amd-sev
```

3. Exit the openstackclient pod:

```
$ exit
```

5.6.6. 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.



NOTE

To execute **openstack** client commands on the cloud you must specify the name of the cloud detailed in your **clouds.yaml** file. You can specify the name of the cloud by using one of the following methods:

- Use the **--os-cloud** option with each command:

```
$ openstack flavor list --os-cloud <cloud_name>
```

Use this option if you access more than one cloud.

- Create an environment variable for the cloud name in your **bashrc** file:

```
`export OS_CLOUD=<cloud_name>`
```

Prerequisites

- The administrator has created a project for you and they have provided you with a **clouds.yaml** file for you to access the cloud.
- You have installed the **python-openstackclient** package.

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](#):

```
$ 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
```

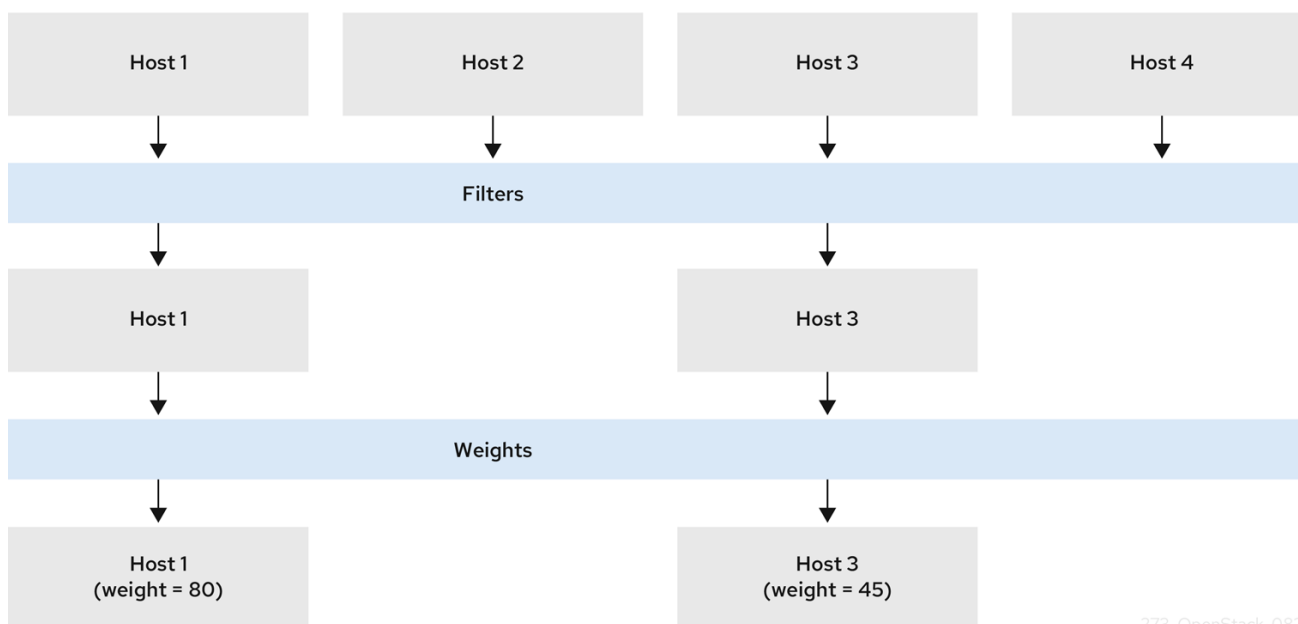
CHAPTER 6. CONFIGURING INSTANCE SCHEDULING AND PLACEMENT

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, hosts 1 and 3 are eligible after filtering. Host 1 has the highest weight and therefore has the highest priority for scheduling.



273_OpenStack_0822

6.1. 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 use 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

6.1.1. 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 **OpenStackControlPlane** custom resource (CR) file, **openstack_control_plane.yaml**, on your workstation.
2. Add the **customServiceConfig** parameter to the Compute scheduler (**nova-scheduler**) template, **schedulerServiceTemplate**, to configure the Compute scheduler service to filter by requested image type support:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
spec:
  extraMounts:
    ...
  nova:
    template:
      schedulerServiceTemplate:
        customServiceConfig: |
          [scheduler]
          query_placement_for_image_type_support = true
```

3. Update the control plane:

```
$ oc apply -f openstack_control_plane.yaml -n openstack
```

4. Wait until RHOCP creates the resources related to the **OpenStackControlPlane** CR. Run the following command to check the status:

```
$ oc get openstackcontrolplane -n openstack
```

The **OpenStackControlPlane** resources are created when the status is "Setup complete".

TIP

Append the **-w** option to the end of the **get** command to track deployment progress.

- Optional: Confirm that the control plane is deployed by reviewing the pods in the **openstack** namespace for each of the cells you created. The control plane is deployed when all the pods are either completed or running.

6.1.2. 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.

Additional resources

- [Section 6.5, “Declaring custom traits and resource classes”](#)

6.1.2.1. 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.

Prerequisites

- You installed the **oc** and **podman** command line tools on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with cluster-admin privileges.

Procedure

1. Access the remote shell for the OpenStackClient pod from your workstation:

```
$ oc rsh -n openstack openstackclient
```

2. Change to the cloud-admin home directory:

```
$ cd /home/cloud-admin
```

3. Create a new image:

```
$ openstack image create ... trait-image
```

4. 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:

```
$ openstack --os-placement-api-version 1.6 trait list
```

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

```
$ 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.

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

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

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

```
$ echo $existing_traits
```

7. 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:

```
$ 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.

8. 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:

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

9. 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:

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


10. Exit the openstackclient pod:

```
$ exit
```

6.1.2.2. 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.

Prerequisites

- You installed the **oc** and **podman** command line tools on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with cluster-admin privileges.

Procedure

1. Access the remote shell for the OpenStackClient pod from your workstation:

```
$ oc rsh -n openstack openstackclient
```

2. Change to the cloud-admin home directory:

```
$ cd /home/cloud-admin
```

3. Create a flavor:

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

4. 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:

```
$ openstack --os-placement-api-version 1.6 trait list
```

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

```
$ 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.

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

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

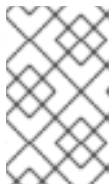
6. Check the existing resource provider traits for the traits you require a host or host aggregate to have:

```
$ echo $existing_traits
```

- 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:

```
$ 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.

- 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:

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

- 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:

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

- Exit the openstackclient pod:

```
$ exit
```

6.1.3. 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.

Prerequisites

- You installed **oc** and **podman** command line tools on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.

Procedure

1. Access the remote shell for the OpenStackClient pod from your workstation:

```
$ oc rsh -n openstack openstackclient
```

2. Change to the cloud-admin home directory:

```
$ cd /home/cloud-admin
```

3. Open your Compute environment file.
4. 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.
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 data plane:

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

7. 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:

```
$ openstack --os-placement-api-version 1.6 trait list
```

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

```
$ 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.

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

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

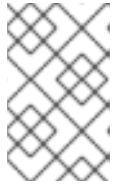
9. Check the existing resource provider traits for the traits you want to isolate the host aggregate for:

```
$ echo $existing_traits
```

10. 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:

```
$ 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.

11. Repeat steps 6 - 8 for each Compute node in the host aggregate.
12. Add the metadata property for the trait to the host aggregate:

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

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

```
$ openstack --os-compute-api=2.86 flavor set \
  --property trait:<TRAIT_NAME>=required <flavor>
$ openstack image set \
  --property trait:<TRAIT_NAME>=required <image>
```

14. Exit the openstackclient pod:

```
$ exit
```

6.2. 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. On your workstation, open your OpenStackControlPlane custom resource (CR) file, **openstack_control_plane.yaml**.
2. Add the filters that you want the scheduler to use to the **[filter_scheduler] enabled_filters** parameter, for example:

```
spec:
  nova:
    template:
      schedulerServiceTemplate:
        customServiceConfig: |
```

```
[filter_scheduler]
enabled_filters = AggregateInstanceExtraSpecsFilter, ComputeFilter,
ComputeCapabilitiesFilter, ImagePropertiesFilter
```

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

```
spec:
  nova:
    template:
      schedulerServiceTemplate:
        customServiceConfig: |
          [filter_scheduler]
          weight_classes = nova.scheduler.weights.all_weighers
```

For more information on the available attributes, see [Compute scheduler weights](#).

- 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:

```
spec:
  nova:
    template:
      schedulerServiceTemplate:
        customServiceConfig: |
          [filter_scheduler]
          weight_classes = nova.scheduler.weights.all_weighers
          [filter_scheduler]
          ram_weight_multiplier = 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**.

- Update the control plane:

```
$ oc apply -f openstack_control_plane.yaml -n openstack
```

- After the RHOCP creates the resources related to the OpenStackControlPlane CR, run the following command to check the status:

```
$ oc get openstackcontrolplane -n openstack
```

The OpenStackControlPlane resources are created when the status is "Setup complete".

TIP

Append the **-w** option to the end of the get command to track deployment progress.

- Optional: Confirm that the control plane is deployed by reviewing the pods in the openstack namespace for each of the cells that you created.

```
$ oc get pods -n openstack
```

The control plane is deployed when all the pods are either completed or running.

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](#).

6.3. COMPUTE SCHEDULER FILTERS

You configure the **enabled_filters** 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:

- 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.
- SameHostFilter:** The Compute node can schedule an instance on the same Compute node as a set of specific instances.
- DifferentHostFilter:** The Compute host can schedule an instance on a different Compute node from a set of specific instances.
- PciPassthroughFilter:** The Compute host can schedule instances on Compute nodes that have the devices that the instance requests by using the flavor extra_specs.
- NUMATopologyFilter:** The Compute host can schedule instances with a NUMA topology on NUMA-capable Compute nodes.

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

Table 6.1. 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.

Filter	Description
AggregateInstanceExtraSpecsFilter	<p>Use this filter to match namespaced properties defined in the flavor extra specs of an instance with host aggregate metadata.</p> <p>You must scope your flavor extra_specs keys by prefixing them with the aggregate_instance_extra_specs: namespace.</p> <p>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.</p>
AggregateIOPSFilter	<p>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.</p>
AggregateMultiTenancyIsolation	<p>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.</p> <div data-bbox="528 1003 639 1205" style="float: left; margin-right: 10px;"> </div> <p>NOTE</p> <p>The project can still place instances on other hosts. To restrict this, use the NovaSchedulerPlacementAggregateRequiredForTenants parameter.</p>
AggregateNumInstancesFilter	<p>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.</p>
AggregateTypeAffinityFilter	<p>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.</p>
AllHostsFilter	<p>Use this filter to consider all available Compute nodes for instance scheduling.</p> <div data-bbox="528 1854 639 1966" style="float: left; margin-right: 10px;"> </div> <p>NOTE</p> <p>Using this filter does not disable other filters.</p>
AvailabilityZoneFilter	<p>Use this filter to launch instances on a Compute node in the availability zone specified by the instance.</p>

Filter	Description
ComputeCapabilitiesFilter	<p>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.</p> <p>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</p>
ComputeFilter	Use this filter to pass all Compute nodes that are operational and enabled. This filter should always be present.
DifferentHostFilter	<p>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:</p> <pre>\$ 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</pre>
ImagePropertiesFilter	<p>Use this filter to filter Compute nodes based on the following properties defined on the instance image:</p> <ul style="list-style-type: none"> ● 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 hypervisor type, for example hvm, xen, uml, or exe. <p>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 configuration parameters.</p>

Filter	Description
IsolatedHostsFilter	<p>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.</p> <p>To specify the isolated set of images and hosts use the filter_scheduler/isolated_hosts and filter_scheduler/isolated_images configuration options, for example:</p> <pre>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</pre>
IoOpsFilter	<p>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.</p>
MetricsFilter	<p>Use this filter to limit scheduling to Compute nodes that report the metrics configured by using metrics/weight_setting.</p> <p>To use this filter, add the following configuration to your Compute environment file:</p> <pre>parameter_defaults: ComputeExtraConfig: nova::config::nova_config: DEFAULT/compute_monitors: value: 'cpu.virt_driver'</pre> <p>By default, the Compute scheduler service updates the metrics every 60 seconds.</p>
NUMATopologyFilter	<p>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.</p>
NumInstancesFilter	<p>Use this filter to filter out Compute nodes that have more instances running than specified by the max_instances_per_host option.</p>

Filter	Description
PciPassthroughFilter	<p>Use this filter to schedule instances on Compute nodes that have the devices that the instance requests by using the flavor extra_specs.</p> <p>Use this filter if you want to reserve nodes with PCI devices, which are typically expensive and limited, for instances that request them.</p>
SameHostFilter	<p>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:</p> <pre>\$ 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</pre>
ServerGroupAffinityFilter	<p>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:</p> <pre>\$ openstack server group create --policy affinity <group_name></pre> <p>To launch an instance in this group, use the --hint argument with group as the key and the group UUID as the value:</p> <pre>\$ openstack server create --image <image> \ --flavor <flavor> \ --hint group=<group_uuid> <instance_name></pre>
ServerGroupAntiAffinityFilter	<p>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:</p> <pre>\$ openstack server group create --policy anti-affinity <group_name></pre> <p>To launch an instance in this group, use the --hint argument with group as the key and the group UUID as the value:</p> <pre>\$ openstack server create --image <image> \ --flavor <flavor> \ --hint group=<group_uuid> <instance_name></pre>

Filter	Description
SimpleCIDRAffinityFilter	<p>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:</p> <pre>\$ openstack server create --image <image> \ --flavor <flavor> \ --hint build_near_host_ip=<ip_address> \ --hint cidr=<subnet_mask> <instance_name></pre>

6.4. 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:

$$\text{(node_resource_availability - minval) / (maxval - minval)}$$

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

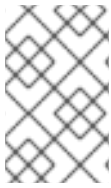
For example, the scheduler calculates the normalized weights for available vCPUs across 10 Compute nodes, each with a different number of available vCPUs, as follows:

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:

$$(w1_multiplier * \text{norm}(w1)) + (w2_multiplier * \text{norm}(w2)) + \dots$$

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 6.2. Compute scheduler weights

Configuration option	Type	Description
filter_scheduler/w eight_classes	String	<p>Use this parameter to configure which of the following attributes to use for calculating the weight of each Compute node:</p> <ul style="list-style-type: none"> ● 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	<p>Use this parameter to specify the multiplier to use to weigh hosts based on the available RAM.</p> <p>Set to a positive value to prefer hosts with more available RAM, which spreads instances across many hosts.</p> <p>Set to a negative value to prefer hosts with less available RAM, which fills up (stacks) hosts as much as possible before scheduling to a less-used host.</p> <p>The absolute value, whether positive or negative, controls how strong the RAM weigher is relative to other weighers.</p> <p>Default: 1.0 - The scheduler spreads instances across all hosts evenly.</p>
filter_scheduler/disk_weight_multiplier	Floating point	<p>Use this parameter to specify the multiplier to use to weigh hosts based on the available disk space.</p> <p>Set to a positive value to prefer hosts with more available disk space, which spreads instances across many hosts.</p> <p>Set to a negative value to prefer hosts with less available disk space, which fills up (stacks) hosts as much as possible before scheduling to a less-used host.</p> <p>The absolute value, whether positive or negative, controls how strong the disk weigher is relative to other weighers.</p> <p>Default: 1.0 - The scheduler spreads instances across all hosts evenly.</p>
filter_scheduler/cpu_weight_multiplier	Floating point	<p>Use this parameter to specify the multiplier to use to weigh hosts based on the available vCPUs.</p> <p>Set to a positive value to prefer hosts with more available vCPUs, which spreads instances across many hosts.</p> <p>Set to a negative value to prefer hosts with less available vCPUs, which fills up (stacks) hosts as much as possible before scheduling to a less-used host.</p> <p>The absolute value, whether positive or negative, controls how strong the vCPU weigher is relative to other weighers.</p> <p>Default: 1.0 - The scheduler spreads instances across all hosts evenly.</p>

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

Configuration option	Type	Description
filter_scheduler/p ci_weight_multipli er	Positive floating point	<p>Use this parameter to specify the multiplier to use to weigh hosts based on the number of PCI devices on the host and the number of PCI devices requested by an instance. If an instance requests PCI devices, then the more PCI devices a Compute node has the higher the weight allocated to the Compute node.</p> <p>For example, if there are three hosts available, one with a single PCI device, one with multiple PCI devices and one without any PCI devices, then the Compute scheduler prioritizes these hosts based on the demands of the instance. The 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.</p> <p>Configure this option to prevent non-PCI instances from occupying resources on hosts with PCI devices.</p> <p>Default: 1.0</p>
filter_scheduler/h ost_subset_size	Integer	<p>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.</p> <p>Set to a value greater than 1 to prevent multiple scheduler processes handling similar requests selecting the same host, creating a potential race condition. By selecting a host randomly from the N hosts that best fit the request, the chance of a conflict is reduced. However, the higher you set this value, the less optimal the chosen host may be for a given request.</p> <p>Default: 1</p>
filter_scheduler/s oft_affinity_weigh t_multiplier	Positive floating point	<p>Use this parameter to specify the multiplier to use to weigh hosts for group soft-affinity.</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p>NOTE</p> <p>You need to specify the microversion when creating a group with this policy:</p> <pre>\$ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity <group_name></pre> </div> </div> <p>Default: 1.0</p>

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

Configuration option	Type	Description
metrics/weight_setting	Comma-separated list of metric=ratio pairs	<p>Use this parameter to specify the metrics to use for weighting, and the ratio to use to calculate the weight of each metric. Valid metric names:</p> <ul style="list-style-type: none"> ● 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 <p>Example: weight_setting=cpu.user.time=1.0</p>
metrics/required	Boolean	<p>Use this parameter to specify how to handle configured metrics/weight_setting metrics that are unavailable:</p> <ul style="list-style-type: none"> ● True - Metrics are required. If the metric is unavailable, an exception is raised. To avoid the exception, use the MetricsFilter filter in NovaSchedulerEnabledFilters. ● 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	<p>Use this parameter to specify the weight to use if any metrics/weight_setting metric is unavailable, and metrics/required=False.</p> <p>Default: -10000.0</p>

6.5. DECLARING CUSTOM TRAITS AND RESOURCE CLASSES

As an administrator, you can declare which custom physical features and consumable resources are available on data plane nodes by defining a custom inventory of resources in a YAML file, **provider.yaml**.

You can declare the availability of physical host features by defining custom traits, such as **CUSTOM_DIESEL_BACKUP_POWER**, **CUSTOM_FIPS_COMPLIANT**, and **CUSTOM_HPC_OPTIMIZED**. You can also declare the availability of consumable resources by defining

resource classes, such as **CUSTOM_DISK_IOPS**, and **CUSTOM_POWER_WATTS**.



NOTE

You can use flavor metadata to request custom resources and custom traits. For more information, see [Instance bare-metal resource class](#) and [Instance resource traits](#).

Prerequisites

- You installed the **oc** and **podman** command line tools on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.

Procedure

1. Create a file in **/home/stack/templates/** called **provider.yaml**.
2. To configure the resource provider, add the following configuration to your **provider.yaml** file:

```
meta:
  schema_version: '1.0'
providers:
  - identification:
      uuid: <node_uuid>
```

- Replace **<node_uuid>** with the UUID for the node, for example, **'5213b75d-9260-42a6-b236-f39b0fd10561'**. Alternatively, you can use the **name** property to identify the resource provider: **name: 'EXAMPLE_RESOURCE_PROVIDER'**.
3. To configure the available custom resource classes for the resource provider, add the following configuration to your **provider.yaml** file:

```
meta:
  schema_version: '1.0'
providers:
  - identification:
      uuid: <node_uuid>
    inventories:
      additional:
        - CUSTOM_EXAMPLE_RESOURCE_CLASS:
            total: <total_available>
            reserved: <reserved>
            min_unit: <min_unit>
            max_unit: <max_unit>
            step_size: <step_size>
            allocation_ratio: <allocation_ratio>
```

- Replace **CUSTOM_EXAMPLE_RESOURCE_CLASS** with the name of the resource class. Custom resource classes must begin with the prefix **CUSTOM_** and contain only the letters A through Z, the numbers 0 through 9 and the underscore **"_"** character.
- Replace **<total_available>** with the number of available **CUSTOM_EXAMPLE_RESOURCE_CLASS** for this resource provider.

- Replace **<reserved>** with the number of available **CUSTOM_EXAMPLE_RESOURCE_CLASS** for this resource provider.
 - Replace **<min_unit>** with the minimum units of resources a single instance can consume.
 - Replace **<max_unit>** with the maximum units of resources a single instance can consume.
 - Replace **<step_size>** with the number of available **CUSTOM_EXAMPLE_RESOURCE_CLASS** for this resource provider.
 - Replace **<allocation_ratio>** with the value to set the allocation ratio. If `allocation_ratio` is set to 1.0, then no overallocation is allowed. But if `allocation_ratio` is greater than 1.0, then the total available resource is more than the physically existing one.
4. To configure the available traits for the resource provider, add the following configuration to your **provider.yaml** file:

```
meta:
  schema_version: '1.0'
providers:
  - identification:
      uuid: <node_uuid>
    inventories:
      additional:
        ...
    traits:
      additional:
        - 'CUSTOM_EXAMPLE_TRAIT'
```

- Replace **CUSTOM_EXAMPLE_TRAIT** with the name of the trait. 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.

Example provider.yaml file

The following example declares one custom resource class and one custom trait for a resource provider.

```
meta:
  schema_version: 1.0
providers:
  - identification:
      uuid: $COMPUTE_NODE
    inventories:
      additional:
        CUSTOM_LLC:
          # Describing LLC on this compute node
          # max_unit indicates maximum size of single LLC
          # total indicates sum of sizes of all LLC
          total: 22 1
          reserved: 2 2
          min_unit: 1 3
          max_unit: 11 4
          step_size: 1 5
          allocation_ratio: 1.0 6
```

```

traits:
  additional:
    # Describing that this compute node enables support for
    # P-state control
    - CUSTOM_P_STATE_ENABLED

```

- 1 This hypervisor has 22 units of last level cache (LLC).
- 2 Two of the units of LLC are reserved for the host.
- 3 4 The min_unit and max_unit values define how many units of resources a single VM can consume.
- 5 The step size defines the increments of consumption.
- 6 The allocation ratio configures the overallocation of resources.

5. Save and close the **provider.yaml** file.
6. Create a ConfigMap CR that configures the Compute nodes to use the provider.yaml file for the declaration of the custom traits and resources, and save it to a file named **compute-provider.yaml** on your workstation:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: compute-provider
  namespace: openstack
data:
  provider.yaml: |

```

For more information about creating **ConfigMap** objects, see [Creating and using config maps](#).

7. Create the **ConfigMap** object:

```
$ oc create -f compute-provider.yaml
```

8. Create a new custom service, **compute-provider**, that includes the **compute-provider ConfigMap** object, and save it to a file named **compute-provider-service.yaml** on your workstation:

```

apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
  name: compute-provider
  namespace: openstack
spec:
  label: dataplane-deployment-compute
  playbook: osp.edpm.nova
  secrets: []
  dataSources:
    - secretRef:
        name: nova-cell1-compute-config
    - secretRef:
        name: nova-migration-ssh-key
    - configMapRef:

```

```

name: compute-provider
- configMapRef:
  name: nova-extra-config
  optional: true

```

9. Create the **compute-provider** service:

```
$ oc apply -f compute-provider-service.yaml
```

10. Create a new **OpenStackDataPlaneNodeSet** CR that defines the nodes that you want to use the provider.yaml file for the declaration of the custom traits and resources, and save it to a file named compute-provider.yaml on your workstation:

```

apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: compute-provider

```

For information about how to create an **OpenStackDataPlaneNodeSet** CR, see [Creating a set of data plane nodes](#).

11. Modify your **compute-provider** OpenStackDataPlaneNodeSet CR to use your **compute-provider-service** service instead of the default Compute service:

```

apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: compute-provider
spec:
  services:
    - download-cache
    - configure-network
    - validate-network
    - install-os
    - configure-os
    - run-os
    - ovn
    - libvirt
    - compute-provider-service #replaced the nova service
    - telemetry

```

12. Save the compute-provider.yaml **OpenStackDataPlaneNodeSet** CR definition file.

13. Create the data plane resources:

```
$ oc create -f compute-provider.yaml
```

14. Verify the data plane resources have been created:

```

$ oc get openstackdataplanenodeset
NAME          STATUS MESSAGE
compute-provider False  Deployment not started

```

15. Verify the services were created:

```
$ oc get openstackdataplaneservice
NAME          AGE
download-cache 6d7h
configure-network 6d7h
configure-os    6d6h
install-os     6d6h
run-os         6d6h
validate-network 6d6h
ovn            6d6h
libvirt        6d6h
compute-provider 6d6h
telemetry      6d6h
```

16. Create a new **OpenStackDataPlaneDeployment** CR to configure the services on the data plane nodes and deploy the nodes, and save it to a file named `compute-provider_deploy.yaml` on your workstation:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: compute-provider
```

For information about how to create an `OpenStackDataPlaneDeployment` CR, see [Deploying the data plane](#).

17. Specify **nodeSets** to include all the `OpenStackDataPlaneNodeSet` CRs that you want to deploy:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: compute-provider
spec:
  nodeSets:
    - openstack-edpm
    - compute-provider
    - ...
    - <nodeSet_name>
```

- Replace `<nodeSet_name>` with the names of the `OpenStackDataPlaneNodeSet` CRs that you want to include in your data plane deployment.

18. Save the `compute-provider_deploy.yaml` deployment file.
19. Deploy the data plane:

```
$ oc create -f compute-provider_deploy.yaml
```

20. Verify that the data plane is deployed:

```
$ oc get openstackdataplanedeployment
NAME          STATUS  MESSAGE
compute-provider True    Deployment Completed

$ oc get openstackdataplanenodeset
```

NAME	STATUS	MESSAGE
openstack-edpm	True	Deployed
compute-provider	True	Deployed

21. Ensure that the deployed Compute nodes are visible on the control plane:

```
$ oc rsh nova-cell0-conductor-0 nova-manage cell_v2 discover_hosts --verbose
```

22. Access the remote shell for openstackclient and verify that the deployed Compute nodes are visible on the control plane:

```
$ oc rsh -n openstack openstackclient
$ openstack hypervisor list
```

6.6. 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 Services on OpenShift (RHOSO) 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.

6.6.1. 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.

Prerequisites

- You installed **oc** and **podman** command line tools on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.

Procedure

1. Access the remote shell for the OpenStackClient pod from your workstation:

```
$ oc rsh -n openstack openstackclient
```

2. Change to the cloud-admin home directory:

```
$ cd /home/cloud-admin
```

3. Open your **OpenStackControlPlane** custom resource (CR) file, **openstack_control_plane.yaml**, on your workstation.
4. Add the following values to the **enabled_filters** 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 configuration parameters](#).

- **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.

5. Update the control plane:

```
$ oc apply -f openstack_control_plane.yaml -n openstack
```


6. Exit the openstackclient pod:

```
$ exit
```

6.6.2. Creating a host aggregate

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

Prerequisites

- You have the **oc** and **podman** command line tools installed on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.

Procedure

1. Access the remote shell for the OpenStackClient pod from your workstation:

```
$ oc rsh -n openstack openstackclient
```

2. Change to the cloud-admin home directory:

```
$ cd /home/cloud-admin
```

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

```
# openstack aggregate create <aggregate_name>
```

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

4. Add metadata to the host aggregate:

```
# 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 configuration parameters](#).
- Replace **<aggregate_name>** with the name of the host aggregate.

5. Add the Compute nodes to the host aggregate:

```
# 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.
6. Create a flavor or image for the host aggregate:

- Create a flavor:

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

- Create an image:

```
$ openstack image create host-agg-image
```

7. 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**:

```
# 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:

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

8. Exit the openstackclient pod:

```
$ exit
```

6.6.3. Creating an availability zone

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

Prerequisites

- You installed the **oc** and **podman** command line tools on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.

Procedure

1. Access the remote shell for the OpenStackClient pod from your workstation:

```
$ oc rsh -n openstack openstackclient
```

2. Change to the cloud-admin home directory:

```
$ cd /home/cloud-admin
```

3. 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:

```
# 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:

```
# 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.

4. Optional: Add metadata to the availability zone:

```
# 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.

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

```
# 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. Exit the openstackclient pod:

```
$ exit
```

6.6.4. Deleting a host aggregate

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

Prerequisites

- You installed **oc** and **podman** command line tools on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.

Procedure

1. Access the remote shell for the OpenStackClient pod from your workstation:

```
$ oc rsh -n openstack openstackclient
```

2. Change to the cloud-admin home directory:

```
$ cd /home/cloud-admin
```

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

```
# openstack aggregate show <aggregate_name>
```

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

```
# 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.

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

```
# openstack aggregate delete <aggregate_name>
```

6. Exit the openstackclient pod:

```
$ exit
```

6.6.5. 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.

Prerequisites

- You installed the **oc** and **podman** command line tools on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.

Procedure

1. Access the remote shell for the OpenStackClient pod from your workstation:

```
$ oc rsh -n openstack openstackclient
```

2. Change to the cloud-admin home directory:

```
$ cd /home/cloud-admin
```

3. On your workstation, open your **OpenStackControlPlane** custom resource (CR) file, **openstack_control_plane.yaml**.

4. To schedule project instances on the project-isolated host aggregate, set the value of the **query_placement_for_image_type_support** parameter to **True**:

```
[scheduler]
query_placement_for_image_type_support = True
```

5. Optional: To ensure that only the projects that you assign to a host aggregate can create instances on your cloud, set the value of the **placement_aggregate_required_for_tenants** parameter to **True**.

**NOTE**

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

6. Save the updates to your Compute environment file.

7. Update the control plane:

```
$ oc apply -f openstack_control_plane.yaml -n openstack
```

8. Create the host aggregate.

9. Retrieve the list of project IDs:

```
# openstack project list
```

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

```
# 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**:

```
# 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:

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

11. Exit the openstackclient pod:

```
$ exit
```

Additional resources

- For more information on creating a host aggregate, see [Creating and managing host aggregates](#).

CHAPTER 7. ADDING METADATA TO INSTANCES



WARNING

The content for this feature is available in this release as a *Documentation Preview*, and therefore is not fully verified by Red Hat. Use it only for testing, and do not use in a production environment.

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

By default, every instance has a 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**.

7.1. 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 Red Hat OpenStack Services on OpenShift (RHOSO) 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.

CHAPTER 8. CONFIGURING INSTANCE SECURITY



WARNING

The content for this feature is available in this release as a *Documentation Preview*, and therefore is not fully verified by Red Hat. Use it only for testing, and do not use in a production environment.

As a cloud administrator, you can configure the following security features for the instances that run on your cloud:

- **UEFI Secure boot** You can create a UEFI Secure Boot flavor with the property key **os:secure_boot** enabled. Cloud users can use this flavor to create instances that are protected with UEFI Secure Boot.
- **Emulated virtual Trusted Platform Module (vTPM)** You can provide cloud users the ability to create instances that have emulated vTPM devices.
- **SEV**: Use to enable your cloud users to create instances that use memory encryption.

CHAPTER 9. DATABASE CLEANING



WARNING

The content for this feature is available in this release as a *Documentation Preview*, and therefore is not fully verified by Red Hat. Use it only for testing, and do not use in a production environment.

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.

The following database management tasks are performed by default:

- 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.

CHAPTER 10. MIGRATING VIRTUAL MACHINE INSTANCES BETWEEN COMPUTE NODES



WARNING

The content for this feature is available in this release as a *Documentation Preview*, and therefore is not fully verified by Red Hat. Use it only for testing, and do not use in a production environment.

You sometimes need to migrate instances from one Compute node to another Compute node in the data plane, 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.

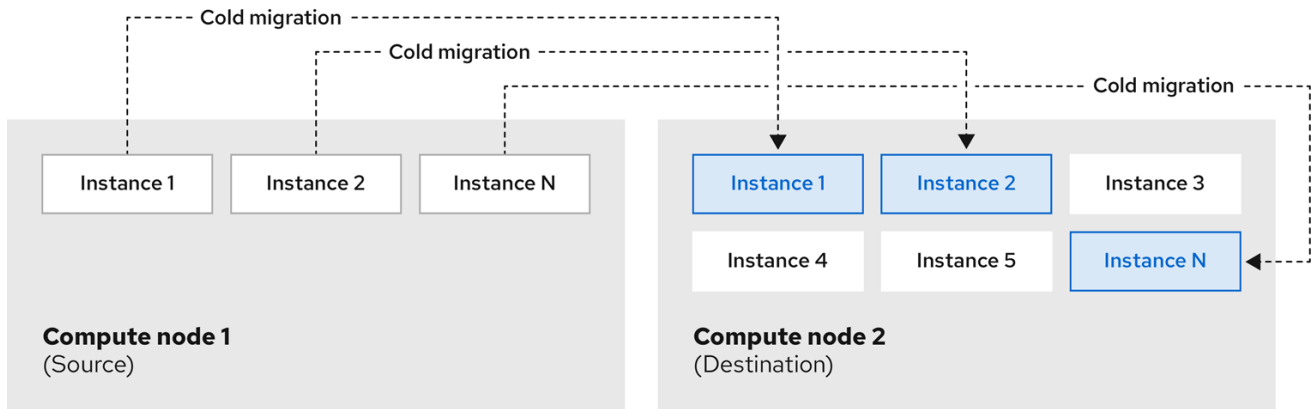
All Compute nodes provide secure migration. All Compute nodes also require a shared SSH key to provide the users of each host with access to other Compute nodes during the migration process.

10.1. MIGRATION TYPES

Red Hat OpenStack Services on OpenShift (RHOSO) 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.



273_OpenStack_0822

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

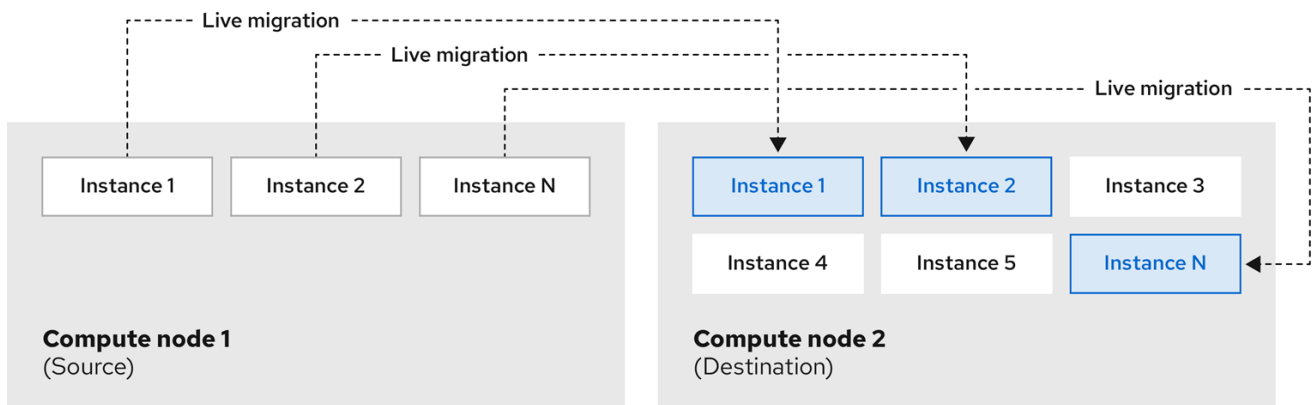


NOTE

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.



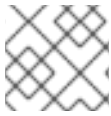
273_OpenStack_0822

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 bandwidth, memory bandwidth, storage IO, or CPU performance during live migration.



NOTE

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

Evacuation

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