Red Hat OpenStack Platform 16.2

为实例创建配置 Compute 服务

配置和管理用于创建实例的 Red Hat OpenStack Platform Compute (nova)服务的指南

Last Updated: 2024-05-21

Red Hat OpenStack Platform 16.2 为实例创建配置 Compute 服务

配置和管理用于创建实例的 Red Hat OpenStack Platform Compute (nova)服务的指南

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

本指南为云管理员提供了使用 OpenStack 客户端 CLI 配置和管理 Red Hat OpenStack Platform Compute (nova)服务的概念和流程。

目录

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

对红帽文档提供反馈

我们感谢您对文档提供反馈信息。与我们分享您的成功秘诀。

在 JIRA 中提供文档反馈

使用 Create Issue 表单对文档提供反馈。JIRA 问题将在 Red Hat OpenStack Platform Jira 项目中创建，您可以在其中跟踪您的反馈进度。

1. 确保您已登录到 JIRA。如果您没有 JIRA 帐户，请创建一个帐户来提交反馈。
2. 点击以下链接打开 **Create Issue** 页面： Create Issue
3. 完成 **Summary** 和 **Description** 字段。在 **Description** 字段中，包含文档 URL、章节或章节号以及问题的详细描述。不要修改表单中的任何其他字段。
4. 点 **Create**。

第 1 章 COMPUTE 服务(NOVA)功能

您可以使用 Compute (nova)服务在 Red Hat OpenStack Platform (RHOSP)环境中创建、置备和管理虚拟机实例和裸机服务器。Compute 服务提取其上运行的底层硬件，而不是公开有关底层主机平台的特定硬件。例如，计算服务不会公开主机上运行的 CPU 的类型和拓扑，而是公开多个虚拟 CPU (vCPU)，并允许过度使用这些 vCPU。

Compute 服务使用 KVM 管理程序来执行计算服务工作负载。libvirt 驱动程序与 QEMU 交互来处理与 KVM 的所有交互，并允许创建虚拟机实例。要创建并置备实例，计算服务与以下 RHOSP 服务交互：

- 用于身份验证的身份(keystone)服务。
- 资源清单跟踪和选择的放置服务。
- 用于磁盘和实例镜像的镜像服务(glance)。
- 用于置备实例在引导时连接的虚拟网络或物理网络的网络(neutron)服务。

Compute 服务由守护进程进程和服务组成，名为 **nova the**。以下是 Compute 服务的核心：

Compute 服务(nova-compute)

此服务使用 libvirt 作为 KVM 或 QEMU hypervisor API 创建、管理和终止实例，并更新具有实例状态的数据库。

计算编排器(nova-conductor)

此服务在计算服务和数据库之间划分交互，这会使 Compute 节点无法直接访问数据库。不要在运行 **nova-compute** 服务的节点上部署此服务。

计算调度程序(nova-scheduler)

此服务从队列获取实例请求，并确定托管实例的 Compute 节点上。

计算 API (nova-api)

此服务为用户提供外部 REST API。

API 数据库

此数据库跟踪实例位置信息，并为构建但未调度的实例提供临时位置。在多单元部署中，此数据库还包含单元映射，用于为每个单元指定数据库连接。

cell 数据库

此数据库包含有关实例的大多数信息。它供 API 数据库、编排器和计算服务使用。

消息队列

所有服务使用此消息传递服务与单元格和全局服务相互通信。

计算元数据

此服务存储特定于实例的数据。实例通过 http://169.254.169.254 或 IPv6 的本地链路地址 fe80::a9fe:a9fe 访问元数据服务。Networking (neutron)服务负责将请求转发到元数据 API 服务器。您必须使用 **NeutronMetadataProxySharedSecret** 参数在网络服务和计算服务配置中设置 secret 关键字，以允许服务进行通信。Compute 元数据服务可以全局运行，作为计算 API 的一部分，也可以在每个单元中运行。

您可以部署多个 Compute 节点。运行实例的虚拟机监控程序在每个 Compute 节点上运行。每个 Compute 节点至少需要两个网络接口。Compute 节点还运行一个网络服务代理，它将实例连接到虚拟网络，并通过安全组向实例提供防火墙服务。

默认情况下，director 为所有 Compute 节点使用一个单元（cell）安装 overcloud。此单元包含所有控制和管理虚拟机实例的计算服务和数据库，以及所有实例和实例元数据。对于较大的部署，您可以使用多单元部署 overcloud，以适应大量 Compute 节点。在安装新的 overcloud 或之后的任何时间，您可以在环境

中添加单元格。如需更多信息，请参阅使用 Compute Cells 扩展部署。

第 2 章 配置计算服务(NOVA)

作为云管理员，您可以使用环境文件来自定义计算(nova)服务。Puppet 生成此配置并将其存储在 **/var/lib/config-data/puppet-generated/<nova_container>/etc/nova/nova.conf** 文件中。使用以下配置方法按以下优先级顺序自定义 Compute 服务配置：

1. **Heat 参数** - 如 *Overcloud 参数指南中的计算(nova) 参数* 一节中所述。以下示例使用 heat 参数设置默认调度程序过滤器，并为计算服务配置 NFS 后端：

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

2. puppet **参数** - 如 **/etc/puppet/modules/nova/manifests** the 中定义的：

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

注意

只有等同的 heat 参数不存在时才使用此方法。

3. **手动 hieradata 覆盖** - 在不存在 heat 或 Puppet 参数时自定义参数。例如，以下命令在 Compute 角色的 **[DEFAULT]** 部分中设置 **timeout_nbd** ：

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

警告

如果存在 heat 参数，则使用它而不是 Puppet 参数。如果存在 Puppet 参数，但没有 heat 参数，则使用 Puppet 参数而不是手动覆盖方法。仅在没有等同的 heat 或 Puppet 参数时使用手动覆盖方法。

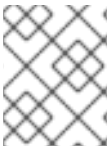
提示

按照 您要修改的参数中的指导， 以确定是否有 heat 或 Puppet 参数来自定义特定的配置。

有关如何配置 overcloud 服务的更多信息，*请参阅高级 Overcloud 自定义指南中的* Heat 参数。

2.1. 为超分配配置内存

当您使用内存过量使用(**NovaRAMAllocationRatio** >= 1.0)时，您需要使用足够的交换空间部署 overcloud 来支持分配比率。



注意

如果您的 **NovaRAMAllocationRatio** 参数设置为 < **1**，请按照 RHEL 交换大小的建议进行操作。如需更多信息，请参阅 RHEL *管理存储设备* 指南中的 推荐的系统交换空间。

先决条件

- 您已计算了节点所需的 swap 大小。如需更多信息，请参阅 捕获 swap 大小。

流程

1. 将 **/usr/share/openstack-tripleo-heat-templates/environments/enable-swap.yaml** 文件复制到环境文件目录中：

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

2. 通过在 **enable-swap.yaml** 文件中添加以下参数来配置 swap 大小：

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

3. 使用其他环境文件将 **enable_swap.yaml** 环境文件添加到堆栈中，并部署 overcloud：

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

2.2. 计算 COMPUTE 节点上的保留主机内存

要确定要为主机进程保留的 RAM 总量，您需要为每个进程分配足够的内存：

- 在主机上运行的资源，例如 OSD 消耗 3 GB 内存。
- 主机实例所需的仿真开销。
- 每个实例的虚拟机监控程序。

计算内存上的额外需求后，使用以下公式来帮助确定为每个节点中的主机进程保留的内存量：

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

- 将 **vm_no** 替换为实例数量。
- 将 **avg_instance_size** 替换为每个实例可以使用的平均内存量。

- 使用每个实例所需的虚拟机监控程序开销替换 **overhead**。
- 将 **resource1** 和所有资源（直到 **<resourcen** >）替换为节点上的资源类型数量。
- 将 **resource_ram** 替换为每个类型资源所需的 RAM 量。

2.3. 计算 SWAP 大小

分配的 swap 大小必须足够大，才能处理任何内存过量使用。您可以使用以下公式来计算节点所需的 swap 大小：

- overcommit_ratio = **NovaRAMAllocationRatio** - 1
- Minimum swap size (MB) = **(total_RAM * overcommit_ratio) + RHEL_min_swap**
- 建议（最大）交换大小(MB)= **total_RAM the (overcommit_ratio + percentage_of_RAM_to_use_for_swap)**

百分比_of_RAM_to_use_for_swap 变量创建一个缓冲区，以考虑 QEMU 开销以及操作系统或主机服务消耗的任何其他资源。

例如，要将 25% 的可用 RAM 用于 swap，并将 64GB RAM 总量，并将 **NovaRAMAllocationRatio** 设置为 **1**：

- 建议（最大）交换大小 = 64000 MB114 (0 + 0.25)= 16000 MB

有关如何计算 **NovaReservedHostMemory** 值的详情，请参考 Compute 节点上的 Calculating reserved host memory。

有关如何确定 **RHEL_min_swap** 值的详情，请参考 RHEL *管理存储设备* 指南中的 推荐的系统交换空间。

第 3 章 配置 COMPUTE 节点以提高性能

作为云管理员，您可以通过创建自定义类别来针对于专用工作负载（包括 NFV 和高性能计算）配置实例的调度和放置，以实现最佳性能。

使用以下功能调整您的实例以获得最佳性能：

- **CPU 固定** ：将虚拟 CPU 分配给物理 CPU。
- **模拟器线程** ：与实例关联的仿真程序线程到物理 CPU。
- **巨页** ：用于普通内存(4k 页面)和巨页(2 MB 或 1 GB 页)的实例内存分配策略。

注意

如果还没有 NUMA 拓扑，配置这些功能会在实例上创建一个隐式 NUMA 拓扑。

3.1. 在 COMPUTE 节点上配置 CPU 固定

您可以通过在 Compute 节点上启用 CPU 固定，将每个实例 CPU 进程配置为在专用主机 CPU 上运行。当实例使用 CPU 固定时，每个实例 vCPU 进程都会被分配自己的主机 pCPU，无需其他实例 vCPU 进程可以使用。在启用了 CPU 固定的 Compute 节点上运行的实例具有 NUMA 拓扑。实例 NUMA 拓扑的每个 NUMA 节点都会映射到主机 Compute 节点上的 NUMA 节点。

您可以将计算调度程序配置为使用专用（固定）CPU 和有共享(floating) CPU 的实例调度到同一 Compute 节点上。要在具有 NUMA 拓扑的 Compute 节点上配置 CPU 固定，您必须完成以下内容：

1. 为 CPU 固定指定 Compute 节点。
2. 配置 Compute 节点，为固定实例 vCPU 进程、浮动实例 vCPU 进程和主机进程保留主机内核。
3. 部署 overcloud。
4. 创建类别以启动需要 CPU 固定的实例。
5. 创建类别，以启动使用共享或浮动 CPU 的实例。

3.1.1. 先决条件

- 您知道 Compute 节点的 NUMA 拓扑。

3.1.2. 为 CPU 固定设计 Compute 节点

要为带有固定 CPU 的实例指定 Compute 节点，您必须创建一个新角色文件来配置 CPU 固定角色，并配置一个新的 overcloud 类别和 CPU 固定资源类，以用于标记 Compute 节点以进行 CPU 固定。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

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

3. 生成一个名为 **roles_data_cpu_pinning.yaml** 的新角色数据文件，其中包含 **Controller**, **Compute**, 和 **ComputeCPUPinning** 角色：

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

4. 打开 **roles_data_cpu_pinning.yaml** 并编辑或添加以下参数和部分：

section/Parameter	当前值	新值
角色注释	**Role: Compute**	**Role: ComputeCPUPinning**
角色名称	**名称 ：Compute**	**名称 ：ComputeCPUPinning**
description	**基本 Compute 节点角色**	**CPU 固定 Compute 节点角色**
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputepinning-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-cpu-pinning.yaml**

5. 通过将 overcloud 添加到节点定义模板、**node.json** 或 **node.yaml**，注册 overcloud 的 CPU 固定 Compute 节点。有关更多信息，请参阅 *Director 安装和使用* 指南中的 为 overcloud 注册节点。

6. 检查节点硬件：

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

如需更多信息，请参阅 *Director 安装和使用指南中的* 创建裸机节点硬件清单。

7. 为 CPU 固定 Compute 节点创建 **compute-cpu-pinning** overcloud 类别：

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

- 将 **<ram_size_mb>** 替换为裸机节点的 RAM，以 MB 为单位。
- 将 **<disk_size_gb>** 替换为裸机节点中的磁盘大小（以 GB 为单位）。
- 将 **<no_vcpus>** 替换为裸机节点中的 CPU 数量。

注意

这些属性不可用于调度实例。但是，计算调度程序使用磁盘大小来确定根分区大小。

8. 使用自定义 CPU 固定资源类标记您要为 CPU 固定指定的每个裸机节点：

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

将 **<node>** 替换为裸机节点的 ID。

9. 将 **compute-cpu-pinning** 类别与自定义 CPU 固定资源类关联：

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

要确定与 Bare Metal 服务节点的资源类型对应的自定义资源类的名称，请将资源类转换为大写，将每个 punctuation 标记替换为下划线，并使用 **CUSTOM_** 前缀。

注意

类别只能请求一个裸机资源类实例。

10. 设置以下类别属性，以防止计算调度程序使用裸机类别属性来调度实例：

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

11. 可选：如果 **ComputeCPUPinning** 角色的网络拓扑与 **Compute** 角色的网络拓扑不同，则创建自定义网络接口模板。有关更多信息，请参阅*高级 Overcloud 自定义指南*中的自定义网络接口模板。
如果 **ComputeCPUPinning** 角色的网络拓扑与 **Compute** 角色相同，您可以使用 **compute.yaml** 中定义的默认网络拓扑。

12. 在 **network-environment.yaml** 文件中注册 **ComputeCPUPinning** 角色的 **Net::SoftwareConfig** ：

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

将 < **cpu_pinning_net_top** > 替换为包含 **ComputeCPUPinning** 角色的网络拓扑的文件名称，如 **compute.yaml** 以使用默认网络拓扑。

13. 在 **node-info.yaml** 文件中添加以下参数，以指定 CPU 固定 Compute 节点的数量，以及用于指定 Compute 节点的 CPU 固定类别：

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

14. 要验证角色是否已创建，请输入以下命令：

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

输出示例：

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

3.1.3. 为 CPU 固定配置 Compute 节点

根据节点的 NUMA 拓扑，在 Compute 节点上配置 CPU 固定。在所有 NUMA 节点中保留一些 CPU 内核，以便主机进程效率。分配剩余的 CPU 内核来管理您的实例。

此流程使用以下 NUMA 拓扑，其中 8 个 CPU 内核分布到两个 NUMA 节点中，以说明如何配置 CPU 固定：

表 3.1. NUMA 拓扑示例

NUMA 节点 0		NUMA 节点 1	
Core 0	核心 1	核心 2	Core 3

Core 4	Core 5	Core 6	Core 7

此流程为主机进程保留了内核 0 和 4，为需要 CPU 固定的实例保留了内核 1，3，5 和7，为于不需要 CPU 固定的浮动实例保留了内核 2 和6。

流程

1. 创建一个环境文件来配置 Compute 节点，以便为固定实例、浮动实例和主机进程保留内核，如 **cpu_pinning.yaml**。
2. 要在支持 NUMA 的 Compute 节点上调度带有 NUMA 拓扑的实例，请将 **NUMATopologyFilter** 添加到 Compute 环境文件中的 **NovaSchedulerDefaultFilters** 参数中（如果不存在）：

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

   有关 **NUMATopologyFilter** 的更多信息，请参阅 计算调度程序过滤器。

3. 要为专用实例保留物理 CPU 内核，请在 **cpu_pinning.yaml** 中添加以下配置：

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

4. 要为共享实例保留物理 CPU 内核，请在 **cpu_pinning.yaml** 中添加以下配置：

```
parameter_defaults:
  ComputeCPUPinningParameters:
    ...
    NovaComputeCpuSharedSet: 2,6
```

5. 要指定为主机进程保留的 RAM 量，请在 **cpu_pinning.yaml** 中添加以下配置：

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

   将 **<ram** > 替换为以 MB 为单位保留的 RAM 数量。

6. 要确保主机进程不在为实例保留的 CPU 内核上运行，请将参数 **IsolCpusList** 设置为您为实例保留的 CPU 内核：

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

   使用以逗号分隔的 CPU 索引的列表或范围来指定 **IsolCpusList** 参数的值。

7. 使用其他环境文件将新角色和环境文件添加到堆栈中，并部署 overcloud：

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

3.1.4. 为实例创建专用 CPU 类别

要让您的云用户创建具有专用 CPU 的实例，您可以创建一个类别，并创建一个专用的 CPU 策略来启动实例。

先决条件

- 主机上启用了并发多线程(SMT)。
- Compute 节点被配置为允许 CPU 固定。如需更多信息，请参阅在 Compute 节点上配置 CPU 固定。

流程

1. 获取 **overcloudrc** 文件：

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

2. 为需要 CPU 固定的实例创建类别：

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

3. 要请求固定 CPU，请将类别的 **hw:cpu_policy** 属性设置为 **专用** ：

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

4. 要将每个 vCPU 放在线程同级上，请将类别的 **hw:cpu_thread_policy** 属性设置为 **需要** ：

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

注意

- 如果主机没有 SMT 架构或足够 CPU 内核，且有可用线程同级的 CPU 内核，则调度会失败。要防止这种情况，将 **hw:cpu_thread_policy** 设置为 **prefer** 而不是 **require**。**prefer** 策略是默认策略，可确保在可用时使用线程同级。
- 如果使用 **hw:cpu_thread_policy=isolate**，则必须禁用 SMT，或使用不支持 SMT 的平台。

验证

1. 要验证类别是否创建带有专用 CPU 的实例，请使用您的新类别来启动实例：

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

2. 要验证新实例的正确位置，请输入以下命令并检查输出中的 **OS-EXT-SRV-ATTR:hypervisor_hostname** ：

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

3.1.5. 为实例创建共享 CPU 类别

要让您的云用户创建使用共享或浮动 CPU 的实例，您可以创建一个具有共享 CPU 策略的类别来启动实例。

先决条件

- Compute 节点被配置为为共享 CPU 保留物理 CPU 内核。如需更多信息，请参阅在 Compute 节点上配置 CPU 固定。

流程

1. 获取 **overcloudrc** 文件：

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

2. 为不需要 CPU 固定的实例创建类别：

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

3. 要请求浮动 CPU，请将类别的 **hw:cpu_policy** 属性设置为 **shared** ：

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

验证

1. 要验证类别创建了使用共享 CPU 的实例，请使用您的新类别来启动实例：

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

2. 要验证新实例的正确位置，请输入以下命令并检查输出中的 **OS-EXT-SRV-ATTR:hypervisor_hostname** ：

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

3.1.6. 在具有并发多线程(SMT)的 Compute 节点上配置 CPU 固定

如果 Compute 节点支持并发多线程(SMT)，组线程在专用或共享集中将线程连接在一起。线程同级共享一些常见硬件，这意味着在一个线程运行的进程可能会影响其他线程同级的性能。

例如，主机使用 SMT: 0, 1, 2, 和 3 识别双核 CPU 中的四个逻辑 CPU 内核。在这四个中，线程使用两对线程：

- 线程同级 1：逻辑 CPU 内核 0 和 2
- 线程同级 2：逻辑 CPU 内核 1 和 3

在这种情况下，请勿将逻辑 CPU 内核 0 和 1 分配为专用，2 和 3 作为共享。相反，将 0 和 2 分配为专用，并将 1 和 3 分配为共享。

文件 **/sys/devices/system/cpu/cpuN/topology/thread_siblings_list**，其中 **N** 是逻辑 CPU 号，包含线程对。您可以使用以下命令识别哪些逻辑 CPU 内核是线程同级的：

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

以下输出表示逻辑 CPU 内核 0 和逻辑 CPU 内核 2 是同一内核的线程：

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

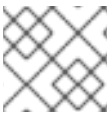
3.1.7. 其他资源

- 网络功能虚拟化规划和配置指南中的发现您的 NUMA 节点 *拓扑*。
- *网络功能虚拟化产品指南* 中的 CPU 和 NUMA 节点。

3.2. 配置仿真程序线程

Compute 节点具有与每个实例的虚拟机监控程序关联的开销任务，称为仿真程序线程。默认情况下，仿真程序线程与实例在同一 CPU 上运行，这会影响实例的性能。

您可以将仿真程序线程策略配置为在独立 CPU 上运行仿真程序线程，以用于实例使用的 CPU。



注意

为避免数据包丢失，不得在 NFV 部署中抢占 vCPU。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. 打开您的计算环境文件。
3. 要为需要 CPU 固定的实例保留物理 CPU 内核，请在计算环境文件中配置 **NovaComputeCpuDedicatedSet** 参数。例如，以下配置在带有 32 核 CPU 的 Compute 节点上设置专用 CPU：

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

   如需更多信息，请参阅在 Compute 节点上配置 CPU 固定。

4. 要为仿真程序线程保留物理 CPU 内核，请在计算环境文件中配置 **NovaComputeCpuSharedSet** 参数。例如，以下配置在带有 32 核 CPU 的 Compute 节点上设置共享 CPU：

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

注意

计算调度程序也对在共享或浮动 CPU 上运行的实例使用共享集合中的 CPU。如需更多信息，请参阅在 Compute 节点上配置 CPU 固定

5. 将 Compute 调度程序过滤器 **NUMATopologyFilter** 添加到 **NovaSchedulerDefaultFilters** 参数中（如果不存在）。

6. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

7. 配置一个类别，其在专用 CPU 上运行实例仿真程序线程，该 CPU 从使用 **NovaComputeCpuSharedSet** 配置的共享 CPU 中选择：

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

有关 **hw:emulator_threads_policy** 的配置选项的更多信息，请参阅 类别元数据中的 仿真程序线程 策略。

3.3. 在 COMPUTE 节点上配置巨页

作为云管理员，您可以配置 Compute 节点，使实例能够请求巨页。

流程

1. 打开您的计算环境文件。

2. 配置巨页内存量，以在每个 NUMA 节点上为不是实例的进程保留：

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

- 将每个节点的 **大小** 值替换为分配的巨页的大小。设置为以下有效值之一：
  - 2048（用于 2MB）
  - 1GB
- 将每个节点的 **计数** 值替换为每个 NUMA 节点 OVS 使用的巨页数。例如，对于 Open vSwitch 使用的插槽内存 4096，将其设置为 2。

3. 在 Compute 节点上配置巨页：

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

注意

如果配置多个巨页大小，还必须在第一次引导过程中挂载巨页文件夹。如需更多信息，请参阅 第一次引导过程中挂载多个巨页文件夹。

4. 可选：要允许实例分配 1GB 巨页，请配置 CPU 功能标记 **NovaLibvirtCPUModelExtraFlags**，使其包含 **pdpe1gb** ：

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

注意

- CPU 功能标志不需要配置为仅允许实例请求 2 MB 巨页。
- 只有主机支持 1G 巨页分配时，您只能为实例分配 1G 巨页。
- 当将 **NovaLibvirtCPUModelExtraFlags** 设置为 **host-model** 或 **custom** 时，您只需要将 **NovaLibvirtCPUModelExtraFlags** 设置为 **pdpe1gb**。
- 如果主机支持 **pdpe1gb**，并且 **host-passthrough** 用作 **NovaLibvirtCPUMode**，那么您不需要将 **pdpe1gb** 设置为 **NovaLibvirtCPUModelExtraFlags**。**pdpe1gb** 标志仅包含在 Exit_G4 和 the_G5 CPU 模型中，它不包含在 QEMU 支持的任何 Intel CPU 模型中。
- 要缓解 CPU 硬件问题，如 Microarchitectural Data Sampling (MDS)，您可能需要配置其他 CPU 标志。如需更多信息，请参阅 MDS 的 RHOS Mitigation for MDS ("Microarchitectural Data Sampling") Security Flaws。

5. 为了避免应用 Meltdown 保护后性能丢失，请将 CPU 功能标记 **NovaLibvirtCPUModelExtraFlags** 配置为包含 **+pcid** ：

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

提示

如需更多信息，请参阅 使用 "PCID" CPU 功能标记为 OpenStack 客户端降低 *Meltdown* CVE 修复的性能影响。

6. 将 **NUMATopologyFilter** 添加到 **NovaSchedulerDefaultFilters** 参数中（如果不存在）。

7. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

3.3.1. 为实例创建巨页类型

要让您的云用户创建使用巨页的实例，您可以创建一个带有 **hw:mem_page_size** 额外 spec 键的类别来启动实例。

先决条件

- Compute 节点是为巨页配置的。如需更多信息，请参阅在 Compute 节点上配置巨页。

流程

1. 为需要巨页的实例创建类别：

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

2. 要请求巨页，请将类别的 **hw:mem_page_size** 属性设置为所需的大小：

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

将 **hw:mem_page_size** 设置为以下有效值之一：

- **large** - 选择主机上支持的最大页面大小，在 x86_64 系统中可能为 2 MB 或 1 GB。
- **small** - （默认）选择主机上支持的最小页面大小。在 x86_64 系统中，这是 4 kB （普通页）。
- **any** - 选择最大可用的巨页大小，具体由 libvirt 驱动程序决定。
- <pageSize>：如果工作负载具有特定要求，则设置显式页面大小。对页大小（以 KB 为单位）或任何标准后缀使用整数值。例如： 4KB、2MB、2048、1GB。

3. 要验证类别是否创建带有巨页的实例，请使用您的新类别来启动实例：

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

计算调度程序标识有足够可用巨页的主机，以便支持实例的内存。如果调度程序无法找到主机和具有足够页面的 NUMA 节点，则请求将失败，并显示 **NoValidHost** 错误。

3.3.2. 在第一次引导过程中挂载多个巨页文件夹

作为第一个引导过程的一部分，您可以配置计算服务(nova)来处理多个页面大小。第一次引导过程在您第一次引导节点时将 heat 模板配置添加到所有节点中。后续包含这些模板（如更新 overcloud 堆栈）不会运行这些脚本。

流程

1. 创建一个第一个引导模板文件 **hugepages.yaml**，它将运行脚本来为巨页文件夹创建挂载。您可以使用 **OS::TripleO::MultipartMime** 资源类型来发送配置脚本：

```
heat_template_version: <version>

description: >
  Huge pages configuration

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

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

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

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

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

此 模板中的配置脚本执行以下任务：

a. 通过指定与 **'co?mp'** 匹配的主机名，过滤主机来为巨页文件夹创建挂载。您可以根据需要更新特定计算的过滤器 grep 模式。

b. 屏蔽默认的 **dev-hugepages.mount systemd** 单元文件，以便使用页面大小创建新挂载。

c. 确保首先创建文件夹。

d. 为每个 **pagesize** 创建 **systemd** 挂载单元。

e. 在第一个循环后运行 **systemd 守护进程-reload**，使其包含新创建的单元文件。

f. 为 2M 和 1G pagesize 启用每个挂载。您可以根据需要更新此循环，使其包含其他 pagesize。

2. 可选： **/dev** 文件夹会自动绑定到 **nova_compute** 和 **nova_libvirt** 容器。如果您已为巨页挂载使用不同的目的地，则需要将挂载传递给 **nova_compute** 和 **nova_libvirt** 容器：

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

3. 将您的 heat 模板注册为 **~/templates/firstboot.yaml** 环境文件中的 **OS::TripleO::NodeUserData** 资源类型：

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

重要

您只能将 **NodeUserData** 资源注册到每个资源的 heat 模板。后续用法会覆盖要使用的 heat 模板。

4. 使用其他环境文件将第一个引导环境文件添加到堆栈中，并部署 overcloud：

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

3.4. 配置 COMPUTE 节点，为实例使用由文件支持的内存

您可以通过将 libvirt 内存后备目录中的文件分配为实例内存内存内存，使用文件支持的内存来扩展 Compute 节点内存容量。您可以配置可用于实例内存的主机磁盘量，以及实例内存文件磁盘上的位置。

Compute 服务向放置服务报告为文件支持的内存配置的容量，作为系统内存容量总量。这允许 Compute 节点托管比系统内存通常适合更多的实例。

要将文件支持的内存用于实例，您必须在 Compute 节点上启用由文件支持的内存。

限制

- 您无法在启用了文件支持内存的 Compute 节点和没有启用文件支持的内存的 Compute 节点间实时迁移实例。
- 文件支持的内存与巨页不兼容。使用巨页的实例无法在启用了文件支持的内存的 Compute 节点上启动。使用主机聚合来确保使用巨页的实例不会放在启用了文件支持内存的 Compute 节点上。
- 文件支持的内存与内存过量使用不兼容。
- 您不能使用 **NovaReservedHostMemory** 为主机进程保留内存。当使用文件支持的内存时，保留内存对应于没有为文件支持的内存设置磁盘空间。文件支持的内存作为系统内存总量报告给放置服务，RAM 用作缓存内存。

先决条件

- **NovaRAMAllocationRatio** 必须在节点上设置为 "1.0"，并且节点添加到的任何主机聚合。
- **NovaReservedHostMemory** 必须设为 "0"。

流程

1. 打开您的计算环境文件。
2. 通过在 Compute 环境文件中添加以下参数，将主机磁盘空间（以 MiB 为单位）配置为可用于实例 RAM：

```
parameter_defaults:
  NovaLibvirtFileBackedMemory: 102400
```

3. 可选：要将目录配置为存储内存后备文件，请在 Compute 环境文件中设置 **QemuMemoryBackingDir** 参数。如果没有设置，内存后备目录默认为 **/var/lib/libvirt/qemu/ram/**。

注意

您必须将后备存储位于位于默认目录位置(**/var/lib/libvirt/qemu/ram/**)的目录中。

您还可以更改后备存储的主机磁盘。如需更多信息，请参阅 更改内存后备目录主机磁盘。

4. 保存对 Compute 环境文件的更新。
5. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

3.4.1. 更改内存后备目录主机磁盘

您可以将内存后备目录从默认主磁盘位置移到替代磁盘。

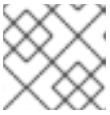
流程

1. 在替代的后备设备中创建文件系统。例如，输入以下命令在 **/dev/sdb** 上创建 **ext4** 文件系统：

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

2. 挂载后备设备。例如，输入以下命令将 **/dev/sdb** 挂载到默认 libvirt 内存后备目录中：

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



注意

挂载点必须与 **QemuMemoryBackingDir** 参数的值匹配。

第 4 章 配置计算服务存储

您可以从基础镜像创建一个实例，计算服务从镜像(glance)服务复制，并在 Compute 节点上本地缓存。实例磁盘（实例的后端）也基于基础镜像。

您可以配置 Compute 服务，以在主机 Compute 节点上本地存储临时实例磁盘数据，或者在 NFS 共享或 Ceph 集群中远程存储。或者，您还可以配置计算服务，将实例磁盘数据存储在块存储(Cinder)服务提供的持久性存储中。

您可以为环境配置镜像缓存，并配置实例磁盘的性能和安全性。当镜像服务(glance)使用 Red Hat Ceph RADOS Block Device (RBD)作为后端时，您还可以将计算服务配置为直接从 RBD 镜像存储库下载镜像。

4.1. 镜像缓存的配置选项

使用下表中详述的参数配置计算服务如何在 Compute 节点上实施和管理镜像缓存。

表 4.1. Compute (nova)服务镜像缓存参数

配置方法	参数	描述
puppet	**nova::compute::image_cache::manager_interval**	指定镜像缓存管理器运行之间等待的秒数，后者管理 Compute 节点上的基础镜像缓存。当 **nova::compute::image_cache::remove_unused_base_images** 设置为 **True** 时，计算服务使用这个周期执行自动移除未使用的镜像。 设置为 **0，** 以默认指标间隔为 60 秒运行（不推荐）。设置为 **-1** 以禁用镜像缓存管理器。 默认： **2400**
puppet	**nova::compute::image_cache::precache_concurrency**	指定并行预缓存镜像的 Compute 节点的最大数量。  **注意** ● 将此参数设置为高数量可能会导致较慢的预缓存性能，并可能导致镜像服务上的 DDoS。 ● 将此参数设置为低数量可减少镜像服务的负载，但可能导致较长的运行时完成，因为预缓存是作为后续操作执行的。 默认： **1**

配置方法	参数	描述
puppet	**nova::compute::image_cache::remove_unused_base_images**	设置为 **True**，以使用 **manager_interval** 在缓存中自动删除未使用的基础镜像。如果在使用 **NovaImageCacheTTL** 指定期间没有访问镜像，则镜像将定义为未使用。 Default: **True**
puppet	**nova::compute::image_cache::remove_unused_resized_minimum_age_seconds**	指定未使用的基础镜像大小必须从缓存中删除的最小期限（以秒为单位）。未使用的已调整的基础镜像的大小不少于此镜像被删除。设置为 **undef** 以禁用。 默认： **3600**
puppet	**nova::compute::image_cache::subdirectory_name**	指定保存缓存镜像的文件夹名称，相对于 **$instances_path**。 默认： **_base**
heat	**NovaImageCacheTTL**	指定 Compute 服务在 Compute 节点上的任何实例不再使用镜像时应继续缓存镜像的时间长度（以秒为单位）。Compute 服务从缓存目录中删除比此配置生命周期旧的镜像，直到再次需要它们。 默认： 86400 (24 小时)

4.2. 实例临时存储属性的配置选项

使用下表中详述的参数配置实例使用的临时存储的性能和安全性。

注意

Red Hat OpenStack Platform (RHOSP)不支持实例类型的 LVM 镜像类型。因此，不支持 **[libvirt]/volume_clear** 配置选项，该选项在实例被删除时擦除临时磁盘，因为它仅在实例磁盘镜像类型是 LVM 时才适用。

表 4.2. Compute (nova)服务实例临时存储参数

配置方法	参数	描述

配置方法	参数	描述
puppet	**nova::compute::default_ephemeral_format**	指定用于新临时卷的默认格式。设置为以下有效值之一： • **ext2** • **ext3** • **ext4** **ext4** 格式为新的大型磁盘提供比 ext3 快的初始化时间。 默认： **ext4**
puppet	**nova::compute::force_raw_images**	设置为 **True** 可将非原始缓存的基础镜像转换为 raw 格式。raw 镜像格式使用超过其他镜像格式的空间，如 qcow2。非原始镜像格式使用更多 CPU 进行压缩。当设置为 **False** 时，计算服务会在压缩过程中从基础镜像中删除任何压缩，以避免 CPU 瓶颈。如果您有一个 I/O 或低可用空间的系统来减少输入带宽，则设置为 **False**。 Default: **True**
puppet	**nova::compute::use_cow_images**	设置为 **True**，以在实例磁盘中使用 CoW (Write 上的Copy)镜像（以 qcow2 格式）。使用 CoW 时，根据后备存储和主机缓存，让每个实例在其自己的副本上运行可能会更好地并发实现。 设置为 **False** 以使用 raw 格式。原始格式在磁盘镜像的通用部分使用更多空间。 Default: **True**
puppet	**nova::compute::libvirt::preallocate_images**	指定实例磁盘的预分配模式。设置为以下有效值之一： • **none** - 实例启动时没有置备任何存储。 • **space** - Compute 服务首先在实例磁盘镜像上运行 **fallocate (1)** 来完全分配存储。这可减少 CPU 开销和文件碎片，提高 I/O 性能，并有助于保证所需的磁盘空间。 默认： **none**

配置方法	参数	描述
hieradata 覆盖	**DEFAULT/resize_fs_using_block_device**	设置为 **True**，通过块设备访问镜像来启用基础镜像的直接调整大小。这只适用于旧版本 cloud-init 的镜像无法调整自身大小。 默认情况下不启用此参数，因为它启用了直接挂载可能因为安全原因禁用的镜像。 默认 ： **False**
hieradata 覆盖	**[libvirt]/images_type**	指定用于实例磁盘的镜像类型。设置为以下有效值之一： ● **raw** ● **qcow2** ● **flat** ● **rbd** ● **default**  **注意** RHOSP 不支持实例类型的 LVM 镜像类型。 当设置了一个不是 **default** 的有效值时，镜像列席会取代 **use_cow_images** 的配置。如果指定了 **default**，**use_cow_images** 的配置决定了镜像类型： ● 如果 **use_cow_images** 设为 **True**（默认），则镜像类型是 **qcow2**。 ● 如果将 **use_cow_images** 设置为 **False**，则镜像类型是 **Flat**。 默认值由 **NovaEnableRbdBackend** 的配置决定： ● **NovaEnableRbdBackend: False** 默认值：**default** ● **NovaEnableRbdBackend: True** 默认： **rbd**

4.3. 配置共享实例存储

默认情况下，当您启动实例时，实例磁盘作为文件存储在实例目录中，**/var/lib/nova/instances**。您可以为 Compute 服务配置 NFS 存储后端，将这些实例文件存储在共享 NFS 存储中。

先决条件

- 您必须使用 NFSv4 或更高版本。Red Hat OpenStack Platform (RHOSP)不支持早期版本的 NFS。如需更多信息，请参阅红帽知识库解决方案 RHOS NFSv4 支持说明。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

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

3. 创建一个环境文件来配置共享实例存储，如 **nfs_instance_disk_backend.yaml**。
4. 要为实例文件配置 NFS 后端，请在 **nfs_instance_disk_backend.yaml** 中添加以下配置：

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

将 **<nfs_share** > 替换为要挂载实例文件存储的 NFS 共享目录，例如 **'192.168.122.1:/export/nova'** 或 **'192.168.24.1:/var/nfs'**。如果使用 IPv6，则同时使用双和单引号，例如 **'"[fdd0::1]:/export/nova'** "。

5. 可选：当 NFS 后端存储被启用时，NFS 存储的默认挂载 SELinux 上下文是 **'context=system_u:object_r:nfs_t:nfs_t:s0'**。添加以下参数以修改 NFS 实例文件存储挂载点的挂载选项：

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

将 < **additional_nfs_mount_options** > 替换为您要用于 NFS 实例文件存储的挂载选项的逗号分隔列表。有关可用挂载选项的详情，请查看 **mount** man page：

```
$ man 8 mount.
```

6. 保存对环境文件的更新。
7. 使用其他环境文件将您的新环境文件添加到堆栈中，并部署 overcloud：

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

4.4. 配置直接从 RED HAT CEPH RADOS 块设备(RBD)下载镜像

当镜像服务 (glance) 将 Red Hat Ceph RADOS Block Device (RBD) 用作后端，并且计算服务使用基于文件的本地临时存储时，无需使用镜像服务 API 即可配置计算服务以直接从 RBD 镜像软件仓库下载镜像。这可减少在实例引导时将镜像下载到 Compute 节点镜像缓存所需的时间，从而缩短实例启动时间。

先决条件

- 镜像服务后端是一个 Red Hat Ceph RADOS 块设备(RBD)。
- Compute 服务将基于文件的本地存储用于镜像缓存和实例磁盘。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. 打开您的计算环境文件。
3. 要直接从 RBD 后端下载镜像，请在计算环境文件中添加以下配置：

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

4. 可选：如果镜像服务配置为使用多个 Red Hat Ceph Storage 后端，请在 Compute 环境文件中添加以下配置来标识 RBD 后端以下载镜像：

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

   将 **<rbd_backend_id** > 替换为用于在 **GlanceMultistoreConfig** 配置中指定后端的 ID，如 **rbd2_store**。

5. 在您的计算环境文件中添加以下配置，以指定镜像服务 RBD 后端，以及计算服务等待连接到镜像服务 RBD 后端的最长时间，以秒为单位：

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

6. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

7. 要验证计算服务是否直接从 RBD 下载镜像，请创建一个实例，然后检查条目"Attemping to export RBD image"的实例调试日志：

4.5. 其他资源

- 配置计算服务(nova)

第 5 章 配置 PCI 透传

您可以使用 PCI 透传将物理 PCI 设备（如图形卡或网络设备）附加到实例。如果您将 PCI 透传用于设备，则实例会对该设备进行独占访问来执行任务，且设备不适用于主机。

重要

在路由的提供商网络中使用 PCI 透传

Compute 服务不支持跨越多个提供商网络的单个网络。当网络包含多个物理网络时，计算服务仅使用第一个物理网络。因此，如果您使用路由的提供者网络，则必须在所有 Compute 节点上使用相同的 **physical_network** 名称。

如果将路由的提供者网络与 VLAN 或扁平网络搭配使用，则必须对所有片段使用相同的 **physical_network** 名称。然后，您可以为网络创建多个片段，并将片段映射到适当的子网。

要让您的云用户创建附加 PCI 设备的实例，您必须完成以下操作：

1. 为 PCI 透传指定 Compute 节点。
2. 为具有所需 PCI 设备的 PCI 透传配置 Compute 节点。
3. 部署 overcloud。
4. 创建类别，以启动附加有 PCI 设备的实例。

先决条件

- Compute 节点具有所需的 PCI 设备。

5.1. 为 PCI 透传设计 COMPUTE 节点

要为附加物理 PCI 设备的实例指定 Compute 节点，您必须创建一个新角色文件来配置 PCI 透传角色，并配置一个新的 overcloud 类别和 PCI 透传资源类，以用于标记 PCI 透传的 Compute 节点。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

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

3. 生成一个名为 **roles_data_pci_passthrough.yaml** 的新角色数据文件，其中包含 **Controller**, **Compute**, 和 **ComputeCPI** 角色：

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

4. 打开 **roles_data_pci_passthrough.yaml** 并编辑或添加以下参数和部分：

section/Parameter	当前值	新值
角色注释	**Role: Compute**	**Role: ComputePCI**
角色名称	**名称 ：Compute**	**名称 ：ComputePCI**
description	**基本 Compute 节点角色**	**PCI Passthrough Compute 节点角色**
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputepci-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-pci-passthrough.yaml**

5. 通过将 overcloud 添加到节点定义模板 **node.json** 或 **node.yaml**，注册 overcloud 的 PCI 透传 Compute 节点。有关更多信息，请参阅 *Director 安装和使用* 指南中的 为 overcloud 注册节点。

6. 检查节点硬件：

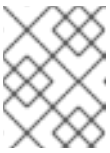
```
(undercloud)$ openstack overcloud node introspect \
 --all-manageable --provide
```

如需更多信息，请参阅 *Director 安装和使用指南中的* 创建裸机节点硬件清单。

7. 为 PCI 透传 Compute 节点创建 **compute-pci-passthrough** overcloud 类别：

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

- 将 **<ram_size_mb>** 替换为裸机节点的 RAM，以 MB 为单位。
- 将 **<disk_size_gb>** 替换为裸机节点中的磁盘大小（以 GB 为单位）。
- 将 **<no_vcpus>** 替换为裸机节点中的 CPU 数量。



注意

这些属性不可用于调度实例。但是，计算调度程序使用磁盘大小来确定根分区大小。

8. 使用自定义 PCI 透传资源类标记您要为 PCI 透传指定的每个裸机节点：

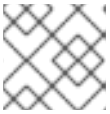
```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.PCI-PASSTHROUGH <node>
```

将 **<node>** 替换为裸机节点的 ID。

9. 将 **compute-pci-passthrough** 类别与自定义 PCI passthrough 资源类关联：

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

要确定与 Bare Metal 服务节点的资源类对应的自定义资源类的名称，请将资源类转换为大写，请将所有 punctuation 替换为下划线，并使用 **CUSTOM_** 前缀。



注意

类别只能请求一个裸机资源类实例。

10. 设置以下类别属性，以防止计算调度程序使用裸机类别属性来调度实例：

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

11. 在 **node-info.yaml** 文件中添加以下参数，以指定 PCI 透传 Compute 节点的数量，以及用于指定 PCI 透传的 Compute 节点的类别：

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

12. 要验证角色是否已创建，请输入以下命令：

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

5.2. 配置 PCI 透传 COMPUTE 节点

要让您的云用户创建附加 PCI 设备的实例，您必须配置具有 PCI 设备和 Controller 节点的 Compute 节点。

流程

1. 创建环境文件，以在 overcloud 上为 PCI 透传配置 Controller 节点，如 **pci_passthrough_controller.yaml**。
2. 将 **PciPassthroughFilter** 添加到 **pci_passthrough_controller.yaml** 中的 **NovaSchedulerDefaultFilters** 参数中：

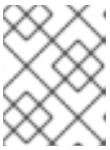
```
parameter_defaults:
  NovaSchedulerDefaultFilters:
['AvailabilityZoneFilter','ComputeFilter','ComputeCapabilitiesFilter','ImagePropertiesFilter','Serve
rGroupAntiAffinityFilter','ServerGroupAffinityFilter','PciPassthroughFilter','NUMATopologyFilter']
```

3. 要为 Controller 节点上的设备指定 PCI 别名，请在 **pci_passthrough_controller.yaml** 中添加以下配置：

```
parameter_defaults:
  ...
  ControllerExtraConfig:
```

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

有关配置 **device_type** 字段的更多信息，请参阅 PCI passthrough 设备类型字段。



注意

如果 **nova-api** 服务在不同于 **Controller** 角色的角色中运行时，将 **Controller ExtraConfig** 替换为用户角色，格式为 **<Role>ExtraConfig**。

4. 可选：要为 PCI 透传设备设置默认 NUMA 关联性策略，将步骤 3 中的 **numa_policy** 添加到 **nova::pci::aliases:** 配置：

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

5. 要在 overcloud 上为 PCI 透传配置 Compute 节点，请创建一个环境文件，如 **pci_passthrough_compute.yaml**。

6. 要为 Compute 节点上的设备指定可用的 PCI，请使用 **vendor_id** 和 **product_id** 选项将所有匹配的 PCI 设备添加到可用于透传到实例的 PCI 设备池中。例如，要将所有 Intel® Ethernet Controller X710 设备添加到可用于透传到实例的 PCI 设备池中，请将以下配置添加到 **pci_passthrough_compute.yaml** 中：

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

有关如何配置 **NovaPCIPassthrough** 的更多信息，请参阅配置 **NovaPCIPassthrough**指南。

7. 您必须在 Compute 节点上为实例迁移和调整大小操作创建 PCI 别名副本。要为 PCI 透传 Compute 节点上的设备指定 PCI 别名，请将以下内容添加到 **pci_passthrough_compute.yaml** 中：

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

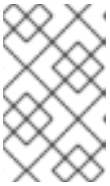
注意

Compute 节点别名必须与 Controller 节点上的别名相同。因此，如果您将 **numa_affinity** 添加到 **pci_passthrough_controller.yaml** 中的 **nova::pci::aliases** 中，则必须将其添加到 **pci_passthrough_compute.yaml** 中的 **nova::pci::aliases** 中。

8. 要在 Compute 节点的服务器 BIOS 中启用 IOMMU 来支持 PCI 透传，请将 **KernelArgs** 参数添加到 **pci_passthrough_compute.yaml** 中。例如，使用以下 **KernalArgs** 设置来启用 Intel IOMMU：

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

要启用 AMD IOMMU，将 **KernelArgs** 设置为 **"amd_iommu=on iommu=pt** "。



注意

首次将 **KernelArgs** 参数添加到角色的配置中时，overcloud 节点会自动重启。如果需要，您可以禁用节点自动重新引导，而是在每次 overcloud 部署后手动执行节点重启。如需更多信息，请参阅配置手动节点重新引导以定义 **KernelArgs**。

9. 使用其他环境文件将自定义环境文件添加到堆栈中，并部署 overcloud：

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

10. 创建并配置您的云用户可用于请求 PCI 设备的类别。以下示例请求两个设备，每个设备 ID 为 **8086**，产品 ID 为 **1572**，使用第 7 步中定义的别名：

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

11. 可选： 要覆盖 PCI 透传设备的默认 NUMA 关联性策略，您可以将 NUMA 关联性策略属性键添加到 flavor 或镜像中：

- 要使用类别覆盖默认 NUMA 关联性策略，请添加 **hw:pci_numa_affinity_policy** 属性键：

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

有关 **hw:pci_numa_affinity_policy** 的有效值的更多信息，请参阅 类别元数据。

- 要使用镜像覆盖默认 NUMA 关联性策略，请添加 **hw_pci_numa_affinity_policy** 属性键：

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

注意

如果您在镜像和类别上设置 NUMA 关联性策略，则属性值必须匹配。flavor 设置优先于镜像和默认设置。因此，只有在类别上未设置属性时，镜像上的 NUMA 关联性策略配置才会生效。

验证

1. 使用 PCI 透传设备创建实例：

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

2. 以云用户身份登录实例。如需更多信息，请参阅 连接到实例。

3. 要验证是否可从实例访问 PCI 设备，请从实例输入以下命令：

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

5.3. PCI PASSTHROUGH 设备类型字段

根据设备报告的功能，计算服务将 PCI 设备分为三种类型之一。以下列表可将 **device_type** 字段设置为的有效值：

type-PF

该设备支持 SR-IOV，它是父设备或 root 设备。指定此设备类型来传递支持整个 SR-IOV 的设备。

type-VF

该设备是支持 SR-IOV 的设备的子设备。

type-PCI

该设备不支持 SR-IOV。如果没有设置 **device_type** 字段，则这是默认设备类型。

注意

您必须使用相同的 **device_type** 配置 Compute 和 Controller 节点。

5.4. 配置 NOVAPCIPASSTHROUGH的指南

- 在配置 PCI 透传时不要使用 **devname** 参数，因为 NIC 的设备名称可能会改变。使用 **vendor_id** 和 **product_id**，因为它们更为稳定，**或使用** NIC 的地址。
- 要通过特定的物理功能(PF)，您可以使用 **address** 参数，因为 PCI 地址对每个设备都是唯一的。或者，您可以使用 **product_id** 参数通过 PF，但如果您有相同类型的多个 PF，还必须指定 PF **的地址**。
- 要传递所有虚拟功能(VF)，仅指定您要用于 PCI 透传的 VF 的 **product_id** 和 **vendor_id**。如果您要将 SRIOV 用于 NIC 分区并且您在 VF 上运行 OVS，则还必须指定 VF 的**地址**。

- 要只为 PF 而不是 PF 本身传递 VF，您可以使用 **address** 参数指定 PF 和 **product_id** 的 PCI 地址，以指定 VF 的产品 ID。

配置 address 参数

address 参数指定设备的 PCI 地址。您可以使用 String 或 **字典映射来设置 address 参数的值**。

字符串格式

如果您使用字符串指定地址，您可以包含通配符 prompt，如下例所示：

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

字典格式

如果使用字典格式指定地址，您可以包含正则表达式语法，如下例所示：

```
NovaPCIPassthrough:
 -
   address:
     domain: ".*"
     bus: "02"
     slot: "01"
     function: "[0-2]"
   physical_network: net1
```

注意

Compute 服务将 **地址** 字段的配置限制为以下最大值：

- 域 - 0xFFFF
- bus - 0xFF
- 插槽 - 0x1F
- function - 0x7

Compute 服务支持具有 16 位地址域的 PCI 设备。Compute 服务忽略具有 32 位地址域的 PCI 设备。

第 6 章 创建和管理主机聚合

作为云管理员，您可以将计算部署分区为逻辑组，以满足性能或管理需要。Red Hat OpenStack Platform (RHOSP)为分区逻辑组提供以下机制：

主机聚合

主机聚合是根据硬件或性能特性等属性将 Compute 节点分组到逻辑单元中。您可以将 Compute 节点分配给一个或多个主机聚合。
您可以通过在主机聚合上设置元数据将类别和镜像映射到主机聚合，然后将类别额外规格或镜像元数据属性与主机聚合元数据属性匹配。在启用所需过滤器时，计算调度程序可以使用此元数据来调度实例。在主机聚合中指定的元数据将该主机的使用限制为具有其类别或镜像中指定的相同元数据的任何实例。

您可以通过在主机聚合元数据中设置 **xxx_weight_multiplier** 配置选项，为每个主机聚合配置权重倍数。

您可以使用主机聚合来处理负载平衡、实施物理隔离或冗余、具有通用属性的组服务器或独立硬件类。

在创建主机聚合时，您可以指定区名称。此名称将呈现给云用户，作为他们可以选择的可用区。

可用区

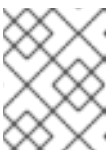
可用域是主机聚合的云用户视图。云用户无法查看可用区中的 Compute 节点，或者查看可用区的元数据。云用户只能看到可用区的名称。
您可以将每个 Compute 节点分配给只有一个可用区。您可以配置一个默认可用区，其中当云用户没有指定区时实例将被调度。您可以指示云用户使用具有特定功能的可用区。

6.1. 在主机聚合上启用调度

要将实例调度到具有特定属性的主机聚合上，请更新计算调度程序的配置，以根据主机聚合元数据启用过滤。

流程

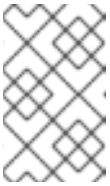
1. 打开您的计算环境文件。
2. 在 **NovaSchedulerDefaultFilters** 参数中添加以下值（如果尚不存在）：
   - **AggregateInstanceExtraSpecsFilter** ：添加此值，以根据与类别额外规格匹配的主机聚合元数据过滤 Compute 节点。

     注意

     要使此过滤器按预期执行，您需要限制类型额外规格的范围，使用 **aggregate_instance_extra_specs:** 命名空间作为 **extra_specs** 键的前缀。

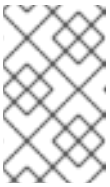
   - **AggregateImagePropertiesIsolation** ：添加此值来根据与镜像元数据属性匹配的聚合元数据过滤 Compute 节点。



注意

要使用镜像元数据属性过滤主机聚合元数据，主机聚合元数据密钥必须与有效的镜像元数据属性匹配。有关有效镜像元数据属性的详情，请参考 镜像元数据。

- **AvailabilityZoneFilter** ：在启动实例时添加这个值以根据可用区过滤。



注意

您可以使用放置服务来处理可用区请求，而不使用 **AvailabilityZoneFilter** Compute 调度程序服务过滤器。如需更多信息，请参阅使用 放置服务根据可用区过滤。

3. 保存对 Compute 环境文件的更新。
4. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

6.2. 创建主机聚合

作为云管理员，您可以根据需要创建任意数量的主机聚合。

流程

1. 运行以下命令来创建主机聚合：

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

将 **<aggregate_name>** 替换为您要分配给主机聚合的名称。

2. 在主机聚合中添加元数据：

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

- 将 **<key=value>** 替换为元数据键值对。如果您使用 **AggregateInstanceExtraSpecsFilter** 过滤器，则键可以是任意字符串，例如 **ssd=true**。如果使用 **AggregateImagePropertiesIsolation** 过滤器，键必须与有效的镜像 metadata 属性匹配。有关有效镜像元数据属性的更多信息，请参阅 镜像元数据。
- 将 **<aggregate_name** > 替换为主机聚合的名称。

3. 将 Compute 节点添加到主机聚合中：

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

- 将 **<aggregate_name** > 替换为要将 Compute 节点添加到的主机聚合的名称。
- 将 **<host_name>** 替换为要添加到主机聚合中的 Compute 节点的名称。

4. 为主机聚合创建类别或镜像：
   - 创建类别：

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

   - 创建镜像：

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

5. 对与主机聚合上的键值对匹配的类别或镜像设置一个或多个键值对。
   - 要在类别上设置键值对，请使用范围 **aggregate_instance_extra_specs** ：

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

   - 要在镜像上设置键值对，请使用有效的镜像元数据属性作为键：

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

6.3. 创建可用区

作为云管理员，您可以创建云用户可在创建实例时选择的可用区。

流程

1. 要创建可用区，您可以创建新的可用区主机聚合，或者使现有主机聚合成为可用区：
   a. 要创建新可用区主机聚合，请输入以下命令：

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

   - 将 **<availability_zone>** 替换为您要分配给可用区的名称。
   - 将 **<aggregate_name>** 替换为您要分配给主机聚合的名称。

   b. 要使现有主机聚合到一个可用区，请输入以下命令：

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

- 将 **<availability_zone>** 替换为您要分配给可用区的名称。
- 将 **<aggregate_name** > 替换为主机聚合的名称。

2. 可选：在可用区中添加元数据：

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

- 将 **<key=value>** 替换为您的原始键值对。您可以根据需要添加任意数量的键值属性。
- 将 **<aggregate_name>** 替换为可用区主机聚合的名称。

3. 将 Compute 节点添加到可用区主机聚合中：

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

- 将 **<aggregate_name>** 替换为可用区主机聚合的名称，以将 Compute 节点添加到其中。
- 将 **<host_name>** 替换为添加到可用区的 Compute 节点的名称。

6.4. 删除主机聚合

要删除主机聚合，您首先从主机聚合中删除所有 Compute 节点。

流程

1. 要查看分配给主机聚合的所有 Compute 节点列表，请输入以下命令：

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

2. 要从主机聚合中删除所有分配的 Compute 节点，为每个 Compute 节点输入以下命令：

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

- 将 **<aggregate_name** > 替换为用于从中删除 Compute 节点的主机聚合的名称。
- 将 **<host_name** > 替换为要从主机聚合中删除的 Compute 节点的名称。

3. 从主机聚合中删除所有 Compute 节点后，输入以下命令删除主机聚合：

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

6.5. 创建项目隔离主机聚合

您可以创建一个仅适用于特定项目的主机聚合。只有您分配给主机聚合的项目才能在主机聚合中启动实例。

注意

项目隔离使用放置服务为每个项目过滤主机聚合。此过程取代了 **AggregateMultiTenancyIsolation** 过滤器的功能。因此，您不需要使用 **AggregateMultiTenancyIsolation** 过滤器。

流程

1. 打开您的计算环境文件。
2. 要将项目实例调度到项目隔离主机聚合上，请在 Compute 环境文件中将 **NovaSchedulerLimitTenantsToPlacementAggregate** 参数设置为 **True**。
3. 可选：为确保您分配给主机聚合的项目才能在云中创建实例，请将 **NovaSchedulerPlacementAggregateRequiredForTenants** 参数设置为 **True**。

注意

NovaSchedulerPlacementAggregateRequiredForTenants 默认为 **False**。当此参数为 **False** 时，未分配给主机聚合的项目可以在任何主机聚合上创建实例。

4. 保存对 Compute 环境文件的更新。
5. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

6. 创建主机聚合。
7. 检索项目 ID 列表：

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

8. 使用 **filter_tenant_id<suffix** > 元数据键将项目分配给主机聚合：

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

- 将 **<ID0>**, **<ID1>**，以及直到 **<IDn>** 的所有 ID 替换为您要创建的每个项目过滤器的唯一值。
- 将 **<project_id** 0& **gt;、<project_id** 1> 以及直到 &lt **;project_idn** > 的所有项目 ID 替换为您要分配给主机聚合的每个项目 ID。
- 将 **<aggregate_name** > 替换为项目隔离主机聚合的名称。
  例如，使用以下语法将项目 **78f1、 9d3t** 和 **aa29** 分配给主机聚合 **project-isolated-aggregate** ：

```
(overcloud)# openstack aggregate set \
```

```
  --property filter_tenant_id0=78f1 \
  --property filter_tenant_id1=9d3t \
  --property filter_tenant_id2=aa29 \
  project-isolated-aggregate
```

提示

您可以通过省略 **filter_tenant_id** 元数据键中的后缀来创建仅适用于单个特定项目的主机聚合：

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

其他资源

- 有关创建主机聚合的更多信息，请参阅 创建和管理主机聚合。

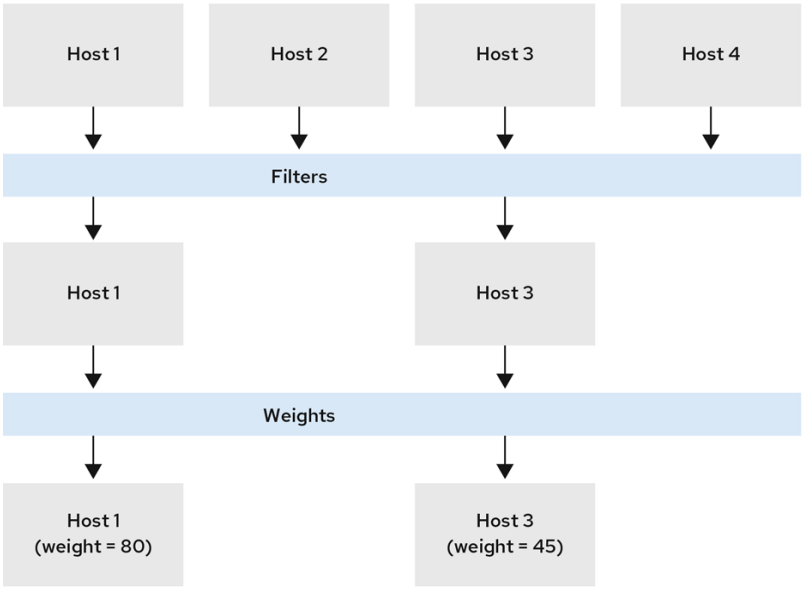
第 7 章 配置实例调度和放置

计算调度程序服务决定在哪个 Compute 节点或主机聚合上放置实例。当计算(nova)服务收到启动或移动实例的请求时，它会使用请求中提供的规格、类别和镜像来查找合适的主机。例如，类别可以指定实例需要具有主机的特征，如存储磁盘类型或 Intel CPU 指令集扩展。

计算调度程序服务使用以下组件的配置，按以下顺序决定在哪个 Compute 节点上启动或移动实例：

1. **Placement service prefilters** ：计算调度程序服务使用放置服务根据特定属性过滤候选 Compute 节点集合。例如，Placement 服务会自动排除禁用的 Compute 节点。
2. **过滤器** ：计算调度程序服务用来决定在其上启动实例的初始 Compute 节点集合。
3. **Weights**：计算调度程序服务通过权重系统来优先选择过滤的 Compute 节点。最高权重具有最高优先级。

在下图中，主机 1 和 3 在过滤后有资格使用。主机 1 具有最高权重，因此调度具有最高优先级。



7.1. 使用放置服务过滤

Compute 服务(nova)在创建和管理实例时与放置服务交互。放置服务跟踪资源提供程序的清单和使用，如 Compute 节点、共享存储池或 IP 分配池，以及其可用数量资源，如可用的 vCPU。任何需要管理资源的选择和消耗的服务都可以使用放置服务。

放置服务还跟踪可用限定资源到资源提供程序的映射，如资源供应商具有的存储磁盘特征。

放置服务会根据放置服务资源供应商清单和特征，将 prefilters 应用到 candidate Compute 节点集合。您可以基于以下条件创建 prefilters：

- 支持的镜像类型
- 特征
- 项目或租户
- 可用区

7.1.1. 根据请求的镜像类型支持进行过滤

您可以排除不支持启动实例镜像的磁盘格式的 Compute 节点。当您的环境使用 Red Hat Ceph Storage 作为临时后端时，这非常有用，它不支持 QCOW2 镜像。启用此功能可确保调度程序不会将使用 QCOW2 镜像启动实例的请求发送到由 Red Hat Ceph Storage 支持的 Compute 节点。

流程

1. 打开您的计算环境文件。
2. 要排除不支持启动实例的镜像的磁盘格式的 Compute 节点，请在 Compute 环境文件中将 **NovaSchedulerQueryImageType** 参数设置为 **True**。
3. 保存对 Compute 环境文件的更新。
4. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

7.1.2. 根据资源供应商特征进行过滤

每个资源供应商都有一组特征。特征是资源提供程序的限定方面，如存储类型或 Intel CPU 指令集扩展名。

Compute 节点将放置服务的功能报告为特征。一个实例可以指定需要哪些特征，或者资源供应商不能有哪些特征。计算调度程序可使用这些特征来识别适合的 Compute 节点或主机聚合来托管实例。

要让您的云用户在具有特定特征的主机上创建实例，您可以定义需要或禁止特定特征的类别，您可以创建需要或禁止特定特征的镜像。

有关可用特征的列表，请参阅 **os-traits** 库。您还可以根据需要创建自定义特征。

7.1.2.1. 创建需要或禁止资源供应商特征的镜像

您可以创建云用户可用于在特定特征的主机上启动实例的镜像。

流程

1. 创建新镜像：

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

2. 识别需要主机或主机聚合具有的特征。您可以选择现有的特征，或创建新的特征：
   - 要使用现有的特征，请列出现有特征以检索特征名称：

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

   - 要创建新特征，请输入以下命令：

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

自定义特征必须以前缀 **CUSTOM_** 开头，仅包含字母 A 到 Z、数字 0 到 9，以及下划线 "_" 字符。

3. 收集每个主机的现有资源供应商特征：

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

4. 检查现有资源供应商特征是否有您需要主机或主机聚合的特征：

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

5. 如果您所需的特征还没有添加到资源供应商中，请将现有的特征和所需的特征添加到每个主机的资源供应商中：

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

将 < **TRAIT_NAME** > 替换为您要添加到资源供应商的特征的名称。您可以根据需要，使用 **--trait** 选项一次添加额外的特征。

注意

此命令对资源提供程序执行特征的完整替换。因此，您必须检索主机上现有资源提供程序特征的列表，并再次设置它们，以防止它们被删除。

6. 要将实例调度到具有所需特征的主机或主机聚合上，请将特征添加到镜像额外规格中。例如，要将实例调度到支持 AVX-512 的主机或主机聚合上，请在镜像额外规格中添加以下特征：

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

7. 要过滤出具有禁止特征的主机或主机聚合，请将特征添加到镜像额外规格中。例如，为了避免将实例调度到支持多附加卷的主机或主机聚合上，请在镜像额外规格中添加以下特征：

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

7.1.2.2. 创建需要或禁止资源供应商特征的类别

您可以创建云用户可用于在特定特征的主机上启动实例的类别。

流程

1. 创建类别：

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

2. 识别需要主机或主机聚合具有的特征。您可以选择现有的特征，或创建新的特征：

   - 要使用现有的特征，请列出现有特征以检索特征名称：

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

   - 要创建新特征，请输入以下命令：

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

     自定义特征必须以前缀 **CUSTOM_** 开头，仅包含字母 A 到 Z、数字 0 到 9，以及下划线 "_" 字符。

3. 收集每个主机的现有资源供应商特征：

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

4. 检查现有资源供应商特征是否有您需要主机或主机聚合的特征：

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

5. 如果您所需的特征还没有添加到资源供应商中，请将现有的特征和所需的特征添加到每个主机的资源供应商中：

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

   将 < **TRAIT_NAME** > 替换为您要添加到资源供应商的特征的名称。您可以根据需要，使用 **--trait** 选项一次添加额外的特征。

   注意

   此命令对资源提供程序执行特征的完整替换。因此，您必须检索主机上现有资源提供程序特征的列表，并再次设置它们，以防止它们被删除。

6. 要将实例调度到具有所需特征的主机或主机聚合上，请将特征添加到类别额外规格中。例如，要将实例调度到支持 AVX-512 的主机或主机聚合上，请在类别额外规格中添加以下特征：

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

7. 要过滤出具有禁止特征的主机或主机聚合，请将特征添加到类别额外规格中。例如，为了避免将实例调度到支持多附加卷的主机或主机聚合上，请将以下特征添加到类别额外规格中：

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

7.1.3. 通过隔离主机聚合进行过滤

您可以将主机聚合上的调度限制为只有类别和镜像特征与主机聚合的元数据匹配的实例。类别和镜像元数据的组合必须要求所有主机聚合特征都有资格调度到该主机聚合中的 Compute 节点上。

流程

1. 打开您的计算环境文件。
2. 要将主机聚合隔离到仅其类别和镜像特征与聚合元数据匹配的实例，请在计算环境文件中将 **NovaSchedulerEnableIsolatedAggregateFiltering** 参数设置为 **True**。
3. 保存对 Compute 环境文件的更新。
4. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

5. 识别您要隔离主机聚合的特征。您可以选择现有的特征，或创建新的特征：
   - 要使用现有的特征，请列出现有特征以检索特征名称：

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

   - 要创建新特征，请输入以下命令：

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

     自定义特征必须以前缀 **CUSTOM_** 开头，仅包含字母 A 到 Z、数字 0 到 9，以及下划线 "_" 字符。

6. 收集每个 Compute 节点的现有资源供应商特征：

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

7. 检查您要隔离主机聚合的现有资源供应商特征：

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

8. 如果您所需的特征还没有添加到资源提供程序中，请在主机聚合中将现有的特征和所需的特征添加到每个 Compute 节点的资源供应商中：

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

   将 < **TRAIT_NAME** > 替换为您要添加到资源供应商的特征的名称。您可以根据需要，使用 **--trait** 选项一次添加额外的特征。

注意

此命令对资源提供程序执行特征的完整替换。因此，您必须检索主机上现有资源提供程序特征的列表，并再次设置它们，以防止它们被删除。

9. 对主机聚合中的每个 Compute 节点重复步骤 6 - 8。
10. 将特征的 metadata 属性添加到主机聚合中：

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

11. 将特征添加到类别或镜像中：

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

7.1.4. 使用放置服务根据可用区进行过滤

您可以使用放置服务来遵循可用区请求。要使用放置服务根据可用区过滤，放置聚合必须存在与可用区主机聚合的成员资格和 UUID 匹配。

流程

1. 打开您的计算环境文件。
2. 要使用放置服务根据可用区过滤，请在计算环境文件中将 **NovaSchedulerQueryPlacementForAvailabilityZone** 参数设置为 **True**。
3. 从 **NovaSchedulerDefaultFilters** 参数中删除 **AvailabilityZoneFilter** 过滤器。
4. 保存对 Compute 环境文件的更新。
5. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

其他资源

- 有关创建用作可用区的主机聚合的更多信息，请参阅创建可用区。

7.2. 为计算调度程序服务配置过滤器和权重

您需要为计算调度程序服务配置过滤器和权重，以确定在其上启动实例的初始 Compute 节点集合。

流程

1. 打开您的计算环境文件。

2. 将您希望调度程序使用的过滤器添加到 **NovaSchedulerDefaultFilters** 参数中，例如：

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

3. 指定要计算每个 Compute 节点的权重的属性，例如：

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

有关可用属性的更多信息，请参阅 计算调度程序权重。

4. 可选：将倍数配置为适用于每个 weig。例如，要指定 Compute 节点的可用 RAM 的权重高于其他默认权重，并且计算调度程序首选在可用 RAM 的那些节点上具有更多可用 RAM 的 Compute 节点，使用以下配置：

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

提示

您还可以将倍数设置为负值。在上例中，若要首选在具有更多可用 RAM 的节点上具有较少可用 RAM 的 Compute 节点，请将 **ram_weight_multiplier** 设置为 **-2.0**。

5. 保存对 Compute 环境文件的更新。

6. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

其他资源

- 有关可用的计算调度程序服务过滤器列表，请参阅 计算调度程序过滤器。
- 有关可用权重配置选项的列表，请参阅 计算调度程序权重。

7.3. 计算调度程序过滤器

您可以在 Compute 环境文件中配置 **NovaSchedulerDefaultFilters** 参数，以指定在选择适当的 Compute 节点托管实例时必须应用计算调度程序的过滤器。默认配置应用以下过滤器：

- **AvailabilityZoneFilter** ： Compute 节点必须位于请求的可用区中。
- **ComputeFilter** ： Compute 节点可以服务请求。
- **ComputeCapabilitiesFilter** ： Compute 节点满足类别额外规格。
- **ImagePropertiesFilter** ： Compute 节点满足请求的镜像属性。
- **ServerGroupAntiAffinityFilter** ： Compute 节点还没有在指定组中托管实例。
- **ServerGroupAffinityFilter** ： Compute 节点已经托管指定组中的实例。

您可以添加和删除过滤器。下表描述了所有可用的过滤器。

表 7.1. 计算调度程序过滤器

Filter	描述
AggregateImagePropertiesIsolation	使用此过滤器将实例的镜像元数据与主机聚合元数据匹配。如果有任何主机聚合元数据与镜像的元数据匹配，则应候选属于该主机聚合的 Compute 节点，以便从该镜像启动实例。调度程序仅识别有效的镜像元数据属性。有关有效镜像元数据属性的详情，请参阅 镜像元数据属性。
AggregateInstanceExtraSpecsFilter	使用此过滤器与带有主机聚合元数据的实例类别额外规格中定义的命名空间属性匹配。 您需要限制类型 **extra_specs** 键的范围，使用 **aggregate_instance_extra_specs:** 命名空间作为前缀。 如果有任何主机聚合元数据与类别额外规格的元数据匹配，则应候选属于该主机聚合的 Compute 节点，以便从该镜像启动实例。
AggregateIoOpsFilter	使用此过滤器根据每个聚合 **filter_scheduler/max_io_ops_per_host** 值的 I/O 操作过滤主机。如果没有找到每个聚合值，则该值会返回全局设置。如果主机处于多个聚合中，并且找到多个值，调度程序将使用最小值。
AggregateMultiTenancyIsolation	使用此过滤器将项目隔离主机聚合中的 Compute 节点可用性限制为指定的项目集合。只有使用 **filter_tenant_id** 元数据键指定的项目才能在主机聚合中的 Compute 节点上启动实例。如需更多信息，请参阅 创建项目隔离主机聚合。  **注意** 该项目仍然可以将实例放在其他主机上。要限制这一点，请使用 **NovaSchedulerPlacementAggregateRequiredForTenants** 参数。
AggregateNumInstancesFilter	使用此过滤器限制聚合中每个 Compute 节点的实例数量可以托管。您可以使用 **filter_scheduler/max_instances_per_host** 参数配置每个聚合的最大实例数量。如果没有找到每个聚合值，则该值会返回全局设置。如果 Compute 节点在多个聚合中，调度程序将使用最低的 **max_instances_per_host** 值。

Filter	描述
AggregateTypeAffinityFilter	如果没有设置类别元数据键，则使用此过滤器传递主机，或者类别聚合元数据值包含请求的类别的名称。类别元数据条目的值是一个字符串，可能包含单个类别名称或以逗号分隔的类别名称列表，如 **m1.nano** 或 **m1.nano,m1.small**。
AllHostsFilter	使用此过滤器考虑所有可用的 Compute 节点进行实例调度。  **注意** 使用此过滤器不会禁用其他过滤器。
AvailabilityZoneFilter	使用此过滤器在实例指定的可用区中的 Compute 节点上启动实例。
ComputeCapabilitiesFilter	使用此过滤器匹配实例的类别额外规格中定义的命名空间属性，具体取决于 Compute 节点功能。您必须使用 **capabilities:** namespace 为 flavor 额外规格添加前缀。 使用 **ComputeCapabilitiesFilter** 过滤器的有效替代方案是，在您的类别中使用 CPU 特征，该特征会报告给放置服务。特征为 CPU 功能提供一致的命名。如需更多信息，请参阅使用资源供应商特征过滤。
ComputeFilter	使用此过滤器传递所有运行并启用的 Compute 节点。此过滤器应始终存在。

DifferentHostFilter	使用此过滤器从一组特定实例启用在不同 Compute 节点上调度实例。要在启动实例时指定这些实例，请使用带有 **different_host** 作为键的 **--hint** 参数，并将实例 UUID 用作值：
	`$ 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`

ImagePropertiesFilter	使用此过滤器根据实例镜像中定义的以下属性过滤 Compute 节点： • **hw_architecture** - 与主机的构架相关的 Corresponds，如 x86、ARM 和 Power。 • **img_hv_type** - 与 hypervisor 类型相关的 Corresponds，例如 KVM、QEMU、xen 和 LXC。 • **img_hv_requested_version** - 与计算服务报告的 hypervisor 版本相对应。 • **hw_vm_mode** - Corresponds 到 hyperviser 类型，如 hvm、xen、uml 或 exe。 支持实例中包含的指定镜像属性的计算节点将传递到调度程序。有关镜像属性的更多信息，请参阅 镜像元数据属性。

Filter	描述
IsolatedHostsFilter	使用此过滤器仅调度隔离 Compute 节点上的实例。您还可以通过配置 **filter_scheduler/restrict_isolated_hosts_to_isolated_images** 来防止非隔离镜像在隔离的 Compute 节点上构建实例。 要指定隔离的镜像和主机集合，请使用 **filter_scheduler/isolated_hosts** 和 **filter_scheduler/isolated_images** 配置选项，例如： parameter_defaults: ComputeExtraConfig: nova::config::nova_config: filter_scheduler/isolated_hosts: value: server1, server2 filter_scheduler/isolated_images: value: 342b492c-128f-4a42-8d3a-c5088cf27d13, ebd267a6-ca86-4d6c-9a0e-bd132d6b7d09
IoOpsFilter	使用此过滤器过滤掉具有并发 I/O 操作的主机，其超过配置的 **filter_scheduler/max_io_ops_per_host**，用于指定主机上允许的最大 I/O 密集型实例数量。
MetricsFilter	使用此过滤器将调度限制为使用 **metrics/weight_setting** 报告指标的 Compute 节点。 要使用此过滤器，请在 Compute 环境文件中添加以下配置： parameter_defaults: ComputeExtraConfig: nova::config::nova_config: DEFAULT/compute_monitors: value: 'cpu.virt_driver' 默认情况下，计算调度程序服务每 60 秒更新指标。为确保指标为最新版本，您可以增加使用 **update_resources_interval** 配置选项刷新指标数据的频率。例如，使用以下配置每 2 秒刷新指标数据： parameter_defaults: ComputeExtraConfig: nova::config::nova_config: DEFAULT/update_resources_interval: value: '2'
NUMATopologyFilter	使用此过滤器将具有 NUMA 拓扑的实例调度到支持 NUMA 的 Compute 节点上。使用类别 **extra_specs** 和镜像属性来指定实例的 NUMA 拓扑。过滤器尝试将实例 NUMA 拓扑与 Compute 节点拓扑匹配，考虑每个主机 NUMA 单元的超订阅限值。
NumInstancesFilter	使用此过滤器过滤运行的实例超过 **max_instances_per_host** 选项的 Compute 节点。

Filter	描述
PciPassthroughFilter	使用此过滤器将实例调度到具有实例请求的设备的 Compute 节点上，使用类别 **extra_specs**。 如果要为请求它们的实例保留 PCI 设备（通常比较高且有限）的节点，请使用此过滤器。

SameHostFilter	使用此过滤器在与一组特定实例相同的 Compute 节点上启用实例调度。要在启动实例时指定这些实例，请使用 **--hint** 参数，其与 **key_host** 以及实例 UUID 与值相同：
	
	`$ 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`

ServerGroupAffinityFilter	使用此过滤器将关联性组中的实例调度到同一 Compute 节点上。运行以下命令来创建服务器组：
	
	`$ openstack server group create --policy affinity <group_name>`
	
	要在此组中启动实例，请使用带有 **group** 的 **--hint** 参数作为键，组 UUID 作为值：
	
	`$ openstack server create --image <image> \`
	`  --flavor <flavor> \`
	`  --hint group=<group_uuid> <instance_name>`

ServerGroupAntiAffinityFilter	使用此过滤器调度属于不同 Compute 节点上的反关联性服务器组的实例。运行以下命令来创建服务器组：
	
	`$ openstack server group create --policy anti-affinity <group_name>`
	
	要在此组中启动实例，请使用带有 **group** 的 **--hint** 参数作为键，组 UUID 作为值：
	
	`$ openstack server create --image <image> \`
	`  --flavor <flavor> \`
	`  --hint group=<group_uuid> <instance_name>`

Filter	描述

SimpleCIDRAffinityFilter	使用此过滤器将实例调度到具有特定 IP 子网范围的 Compute 节点上。要指定所需的范围，在启动实例时使用 **--hint** 参数传递密钥 **build_near_host_ip** 和 **cidr** ：
	`$ openstack server create --image <image> \`
	`--flavor <flavor> \`
	`--hint build_near_host_ip=<ip_address> \`
	`--hint cidr=<subnet_mask> <instance_name>`

7.4. 计算调度程序权重

每个 Compute 节点都有一个权重，调度程序可用于确定实例调度优先级。在计算调度程序应用过滤器后，它会从剩余的候选 Compute 节点选择具有最大权重的 Compute 节点。

计算调度程序通过执行以下任务来确定每个 Compute 节点的权重：

1. 调度程序将每个权重规范化为 0.0 和 1.0 之间的值。
2. 调度程序将规范化权重乘以 weigher multiplier。

计算调度程序使用候选 Compute 节点的资源可用性的下限值计算每种资源类型的权重规范化：

- 为资源(minval)提供最低可用性的节点被分配为 '0'。
- 分配了资源最多可用性的节点(maxval)会被分配 '1'。
- 在 minval 范围内具有资源可用性的节点使用以下公式分配一个规范化权重计算：

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

如果所有 Compute 节点都具有相同的资源可用性，则它们都规范化为 0。

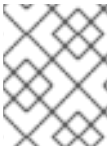
例如，调度程序计算 10 个 Compute 节点之间可用 vCPU 的规范化权重，每个节点都有不同的可用 vCPU 数量，如下所示：

Compute 节点	1	2	3	4	5	6	7	8	9	10
没有 vCPU	5	5	10	10	15	20	20	15	10	5
规范化权重	0	0	0.33	0.33	0.67	1	1	0.67	0.33	0

计算调度程序使用以下公式来计算 Compute 节点的权重：

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

下表描述了可用于权重的配置选项。



注意

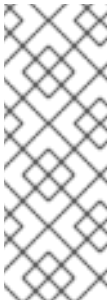
可以使用与下表中详述的选项相同的聚合元数据键在主机聚合上设置权重。如果在主机聚合上设置，则主机聚合值将具有优先权。

表 7.2. 计算调度程序权重

配置选项	类型	描述
filter_scheduler/weight_classes	字符串	使用此参数配置用于计算每个 Compute 节点的权重的以下属性： • **nova.scheduler.weights.ram.RAMWeigher** - 探测上可用的 RAM。 • **nova.scheduler.weights.cpu.CPUWeigher** - 探测上可用的 CPU。 • **nova.scheduler.weights.disk.DiskWeigher** - Weighs 上可用的磁盘。 • **nova.scheduler.weights.metrics.MetricsWeigher** - Weighs Compute 节点的指标。 • **nova.scheduler.weights.affinity.ServerGroup SoftAffinityWeigher** - 将 Compute 节点的代理分配到给定实例组中的其他节点。 • **nova.scheduler.weights.affinity.ServerGroup SoftAntiAffinityWeigher** - 将 Compute 节点的代理分配到给定实例组中的其他节点。 • **nova.scheduler.weights.compute.BuildFailureWeigher** - Weighs Compute 节点通过最近失败的引导尝试次数进行。 • **nova.scheduler.weights.io_ops.IoOpsWeigher** - 按工作负载分配 Compute 节点。 • **nova.scheduler.weights.pci.PCIWeigher** - 按 PCI 可用性创建 Compute 节点。 • **nova.scheduler.weights.cross_cell.CrossCellWeigher** - 根据它们所在的单元，优先选择源单元中的 Compute 节点。 • **nova.scheduler.weights.all_weighers** - （默认）使用上述所有权重。

配置选项	类型	描述
filter_scheduler/ram_weight_multiplier	浮点	使用此参数指定基于可用 RAM 的权重主机使用的倍数。 设置为正值，以优先选择具有更多可用 RAM 的主机，这会将实例分散到多个主机上。 设置为负值，以首选使用较少可用 RAM 的主机，这会在调度到较少使用的主机前填充（堆栈）主机。 绝对值（无论是正还是负）控制 RAM weigher 相对于其他 Weighers 的强度。 Default: 1.0 - 调度程序均匀地将实例分散到所有主机中。
filter_scheduler/disk_weight_multiplier	浮点	使用此参数指定基于可用磁盘空间的权重主机使用的倍数。 设置为正值，以优先选择具有更多可用磁盘空间的主机，这会将实例分散到多个主机上。 设置为负值，以首选使用较少可用磁盘空间的主机，这会在调度到较少使用的主机前填充（堆栈）主机。 绝对值（无论是正还是负）控制磁盘的速度相对于其他权重。 Default: 1.0 - 调度程序均匀地将实例分散到所有主机中。
filter_scheduler/cpu_weight_multiplier	浮点	使用此参数指定基于可用 vCPU 的权重主机使用的倍数。 设置为正值，以优先选择具有更多可用 vCPU 的主机，这会将实例分散到多个主机上。 设置为负值，以首选使用较少可用 vCPU 的主机，这会在调度到较少使用的主机前填充（堆栈）主机。 绝对值（无论是正还是负）控制 vCPU weigher 相对于其他 Weighers 的强度。 Default: 1.0 - 调度程序均匀地将实例分散到所有主机中。
filter_scheduler/io_ops_weight_multiplier	浮点	使用此参数指定基于主机工作负载的权重主机使用的倍数。 设置为负值，以首选使用轻量级工作负载的主机，这会在更多主机上分发工作负载。 设置为正值，以首选具有 heavier 工作负载的主机，它将实例调度到已经忙碌的主机上。 绝对值（无论是正还是负）控制我们自己相对于其他 Weighers 的 I/O 操作强度。 Default: -1.0 - 调度程序在更多主机之间分发工作负载。

配置选项	类型	描述
filter_scheduler/build_failure_weight_multiplier	浮点	使用此参数指定基于最新构建失败的主机使用的倍数。 设置为正值，以增加主机最近报告的构建故障的意义。然后，选择构建失败的主机不太可能被选择。 设置为 **0，** 通过最近失败的数量禁用计算主机。 默认：1000000.0
filter_scheduler/cross_cell_move_weight_multiplier	浮点	使用此参数指定跨单元移动期间用于 weigh 主机的倍数。此选项决定了在移动实例时放置在同一源单元的主机上放置多少个权重。默认情况下，调度程序在迁移实例时首选同一源单元中的主机。 设置为正值，以优先选择实例当前运行的同一单元格中的主机。设置为负值，以优先选择位于实例当前运行的单元中的主机。 默认：1000000.0
filter_scheduler/pci_weight_multiplier	正浮动点	使用此参数根据主机上的 PCI 设备数量以及实例请求的 PCI 设备数量来指定用于权重主机的倍数。如果实例请求 PCI 设备，则计算节点的 PCI 设备具有更高的权重为 Compute 节点。 例如，如果有三个可用的主机，一个主机只有一个 PCI 设备，一个具有多个 PCI 设备和一个没有 PCI 设备的主机，则计算调度程序会根据实例的需求优先选择这些主机。如果实例请求一个 PCI 设备，调度程序应首选第一个主机，如果实例需要多个 PCI 设备和第三个主机，则第二个主机（如果实例不请求 PCI 设备）。 配置这个选项，以防止非 PCI 实例在带有 PCI 设备的主机上占据资源。 默认：1.0
filter_scheduler/host_subset_size	整数	使用此参数指定过滤的主机子集的大小，以便从中选择主机。您必须将这个选项设置为至少 1。1 代表选择由权重函数返回的第一个主机。调度程序忽略任何小于 1 的值，而是使用 1。 设置为大于 1 的值，以防止多个调度程序进程处理类似的请求，创建潜在的竞争条件。通过从最适合请求的 N 主机随机选择主机，可能会减少冲突的机会。但是，您设置这个值越大，所选主机对给定请求可能最最佳的。 默认：1

配置选项	类型	描述
filter_scheduler/soft_affinity_weight_multiplier	正浮动点	使用此参数指定用于为组软关联性分配主机使用的倍数。  **注意** 使用此策略创建组时，您需要指定 microversion： $ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity <group_name> 默认：1.0
filter_scheduler/soft_anti_affinity_weight_multiplier	正浮动点	使用此参数指定用于为组 soft-anti-affinity 分配到主机的倍数。  **注意** 使用此策略创建组时，您需要指定 microversion： $ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity <group_name> 默认：1.0
metrics/weight_multiplier	浮点	使用此参数指定用于权重指标的倍数。默认情况下，**weight_multiplier=1.0**，将实例分散到可能的主机中。 设置为大于 1.0 的数字，以提高指标对总权重的影响。 设置为 0.0 到 1.0 之间的数字，以减少指标对总权重的影响。 设置为 0.0 以忽略指标值并返回 **weight_of_unavailable** 选项的值。 设置为负数，以优先选择指标较低的主机，并在主机上堆栈实例。 默认：1.0

配置选项	类型	描述
metrics/weight_setting	以逗号分隔的 **metric=ratio** 对列表	使用此参数指定用于权重的指标，以及用于计算每个指标的权重的比率。有效的指标名称： ● **cpu.frequency** - CPU 频率 ● **cpu.user.time** - CPU 用户模式时间 ● **cpu.kernel.time** - CPU 内核时间 ● **cpu.idle.time** - CPU 空闲时间 ● **cpu.iowait.time** - CPU I/O 等待时间 ● **cpu.user.percent** - CPU 用户模式百分比 ● **cpu.kernel.percent** - CPU 内核百分比 ● **cpu.idle.percent** - CPU 空闲百分比 ● **cpu.iowait.percent** - CPU I/O 等待百分比 ● **cpu.percent** - 通用 CPU 使用 示例： **weight_setting=cpu.user.time=1.0**
metrics/required	布尔值	使用此参数指定如何处理不可用的 **metrics/weight_setting** 指标： ● **true** - 需要指标数据。如果指标不可用，则会引发异常。为避免异常，请在 **NovaSchedulerDefaultFilters** 中使用 **MetricsFilter** 过滤器。 ● **False** - 不可用的指标在 weighing 进程中被视为一个负因素。使用 **weight_of_unavailable** 配置选项设置返回的值。
metrics/weight_of_unavailable	浮点	如果任何 **metrics/weight_setting** 指标不可用，则使用此参数指定要使用的权重，以及 **metrics/required=False**。 Default: -10000.0

第 8 章 为启动实例创建类别

实例类别是一个资源模板，用于指定实例的虚拟硬件配置文件。云用户必须在启动实例时指定类别。

类别可以指定 Compute 服务必须分配给实例的以下资源数量：

- vCPU 数量。
- RAM，以 MB 为单位。
- 根磁盘（以 GB 为单位）。
- 虚拟存储，包括辅助临时存储和交换磁盘。

您可以通过使类别公开到所有项目或私有到特定的项目或域来指定谁可以使用类别。

类别可以使用元数据（也称为"额外规格"）来指定实例硬件支持和配额。类别元数据会影响实例放置、资源使用量限值和性能。有关可用元数据属性的完整列表，请参阅 类别元数据。

您还可以使用类别元数据键，通过匹配主机聚合中设置的 **extra_specs** 元数据来查找适合的主机聚合来托管实例。要将实例调度到主机聚合上，必须通过使用 aggregate_instance_ **extra_specs：命名空间前缀 extra_specs** 键来限制类别元数据。如需更多信息，请参阅创建和管理主机聚合。

Red Hat OpenStack Platform (RHOSP)部署包括您的云用户可以使用的以下默认公共类型集合。

表 8.1. 默认类别

名称	VCPU	RAM	根磁盘大小
m1.nano	1	128 MB	1 GB
m1.micro	1	192 MB	1 GB

注意

使用类别属性设置的行为覆盖使用镜像设置的行为。当云用户启动实例时，它们指定的类别的属性会覆盖它们指定的镜像的属性。

8.1. 创建类别

您可以为特定功能或行为创建和管理专用类别，例如：

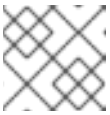
- 更改默认内存和容量以满足底层硬件需求。
- 添加元数据来强制实例的特定 I/O 速率或与主机聚合匹配。

流程

1. 创建一个类别，用于指定实例可用的基本资源：

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

- 将 **<size_mb** > 替换为分配给使用此类别创建的实例的 RAM 大小。
- 将 < **size_gb** > 替换为要分配给使用此类别创建的实例的根磁盘大小。
- 将 **<no_** the> 替换为为此类别创建的实例保留的 vCPU 数量。
- 可选：指定 **--private** 和 **--project** 选项，使类别只能被特定的项目或用户组访问。将 **<project_id** > 替换为可以使用此类别创建实例的项目的 ID。如果没有指定可访问性，则类别默认为 public，这意味着它可供所有项目使用。



注意

您不能在创建后将公共类别设置为私有。

- 将 **<flavor_name** > 替换为您的类别的唯一名称。
  有关类别参数的更多信息，请参阅类别参数。

2. 可选： 要指定类别元数据，请使用键值对设置所需的属性：

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

- 将 **&lt**;key> 替换为您要分配给使用此类别创建的实例的属性的元数据键。有关可用元数据密钥的列表，请参阅 类别元数据。
- 使用您要分配给使用此类别创建的实例的元数据密钥值替换 **<value>**。
- 将 **<flavor_name** > 替换为您的类型的名称。
  例如，使用以下类别启动的实例有两个 CPU 套接字，各自有两个 CPU 套接字：

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

8.2. 类别参数

openstack flavor create 命令有一个位置参数 &lt **;flavor_name** >，用于指定您的新类别的名称。

下表详细介绍了创建新类别时可根据需要指定的可选参数。

表 8.2. 可选类别参数

可选参数	描述
--id	类别的唯一 ID。默认值 **auto** 生成一个 UUID4 值。您可以使用此参数手动指定整数或 UUID4 值。
--ram	（必需）实例可用的内存大小（以 MB 为单位）。 默认： 256 MB

可选参数	描述
--disk	（必需）用于根(/)分区的磁盘空间挂载，以 GB 为单位。根磁盘是基础镜像复制到的临时磁盘。当实例从持久性卷引导时，不使用根磁盘。 **注意** 创建将 **--disk** 设置为 **0** 的类别的实例要求实例从卷引导。 默认： 0 GB
--ephemeral	用于临时磁盘的磁盘空间量（以 GB 为单位）。默认值为 0 GB，这意味着不会创建辅助临时磁盘。临时磁盘提供链接到实例生命周期的机器本地磁盘存储。临时磁盘不包含在任何快照中。此磁盘将被销毁，在实例被删除时会丢失所有数据。 默认： 0 GB
--swap	交换磁盘大小（以 MB 为单位）。如果计算服务后端存储不是本地存储，则不要在类别中指定 **交换**。 默认： 0 GB
--vcpus	（必需）实例的虚拟 CPU 数量。 默认：1
--public	该类别可供所有项目使用。默认情况下，一个类别是公共的，适用于所有项目。
--private	该类别仅适用于通过 **--project** 选项指定的项目。如果您创建了私有类别，但没有向其添加项目，则类别仅可供云管理员使用。
--property	元数据或"额外 specs"，使用以下格式使用键值对指定： **--property <key=value>** 重复这个选项来设置多个属性。
--project	指定可以使用私有类别的项目。您必须将此参数与 **--private** 选项一起使用。如果没有指定任何项目，则类别仅对 **admin** 用户可见。 重复此选项以允许访问多个项目。
--project-domain	指定可以使用私有类别的项目域。您必须将此参数与 **--private** 选项一起使用。 重复此选项以允许访问多个项目域。

可选参数	描述
--description	类别的描述。长度限制为 65535 个字符。您只能使用可打印的字符。

8.3. 类别元数据

在创建类别时，使用 **--property** 选项指定类别元数据。类别元数据也称为*额外规格*。类别元数据决定了实例硬件支持和配额，这会影响实例放置、实例限值和性能。

实例资源使用量

使用下表中的属性键配置实例的 CPU、内存和磁盘 I/O 使用量的限制。

表 8.3. 资源使用量的类别元数据

键	描述
quota:cpu_shares	指定域的 CPU 时间的比例权重共享。默认为操作系统提供的默认值。计算调度程序相对于同一域中的其他实例设置此属性的权重。例如，配置有 **quota:cpu_shares=2048** 的实例将 CPU 时间加倍为配置了 **quota:cpu_shares=1024** 的实例。
quota:cpu_period	指定以微秒为单位强制执行 **cpu_quota** 的时间期限。在 **cpu_period** 中，每个 vCPU 无法消耗超过 **cpu_quota** 的运行时。设置为范围 1000 - 1000000 中的值。设置为 **0** 以禁用。

quota:cpu_quota	指定每个 **cpu_period** 中 vCPU 允许的最大带宽（以微秒为单位）：
	• 设置为范围 1000 - 18446744073709551 中的值。
	• 设置为 **0** 以禁用。
	• 设置为负值以允许无限带宽。
	您可以使用 **cpu_quota** 和 **cpu_period** 来确保所有 vCPU 都以相同的速度运行。例如，您可以使用以下类别来启动可消耗最多 50% 物理 CPU 能力的实例：
	`$ openstack flavor set cpu_limits_flavor \`
	`  --property quota:cpu_quota=10000 \`
	`  --property quota:cpu_period=20000`

实例磁盘调整

使用下表中的属性键调整实例磁盘性能。

注意

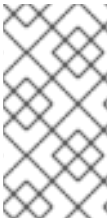
Compute 服务对 Compute 服务调配的存储（如临时存储）应用下列服务质量设置。要调整 Block Storage (cinder)卷的性能，还必须为卷类型配置服务质量(QOS)值。如需更多信息，请参阅 *存储指南*中的使用服务质量规格。

表 8.4. 磁盘调整的类别元数据

键	描述
quota:disk_read_bytes_sec	指定实例可用的最大磁盘读取，以字节/秒为单位。
quota:disk_read_iops_sec	指定实例可用的最大磁盘读取，以 IOPS 为单位。
quota:disk_write_bytes_sec	指定实例可用的最大磁盘写入，以字节/秒为单位。
quota:disk_write_iops_sec	指定实例可用的最大磁盘写入，以 IOPS 为单位。
quota:disk_total_bytes_sec	指定实例可用的最大 I/O 操作，以字节/秒为单位。
quota:disk_total_iops_sec	指定实例可用的最大 I/O 操作，以 IOPS 为单位。

实例网络流量带宽

通过配置 VIF I/O 选项，使用下表中的属性键配置实例网络流量的带宽限制。



注意

配额 :vif 114 属性已弃用。取而代之，您应该使用 Networking (neutron)服务质量(QoS)策略。如需有关 QoS 策略的更多信息，请参阅*网络指南*中的配置服务质量(QoS)策略。只有在使用 ML2/OVS 机制驱动 **NeutronOVSFirewallDriver** 设置为 **iptables_hybrid** 时，才会支持 **quota:vif_*** 属性。

表 8.5. 带宽限制的类别元数据

键	描述
quota:vif_inbound_average	（已弃用）指定进入实例的流量所需的平均位率（以 kbps 为单位）。
quota:vif_inbound_burst	（已弃用）以 KB 为单位指定可能以峰值速度突发的最大传入流量数量。
quota:vif_inbound_peak	（已弃用）指定实例可以接收传入流量的最大速率，以 kbps 为单位。
quota:vif_outbound_average	（已弃用）指定从实例传出的流量所需的平均位率（以 kbps 为单位）。
quota:vif_outbound_burst	（已弃用）以 KB 为单位指定可突发的最大传出流量数量。
quota:vif_outbound_peak	（已弃用）指定实例可以发送传出流量的最大速率，以 kbps 为单位。

硬件视频 RAM

使用下表中的属性键配置用于视频设备的实例 RAM 的限制。

表 8.6. 视频设备的类别元数据

键	描述
hw_video:ram_max_mb	以 MB 为单位指定视频设备使用的最大 RAM。与 **hw_video_ram** 镜像属性一起使用。**hw_video_ram** 必须小于或等于 **hw_video:ram_max_mb**。

watchdog 行为

使用下表中的属性键，在实例上启用虚拟硬件 watchdog 设备。

表 8.7. watchdog 行为的类别元数据

键	描述
hw:watchdog_action	指定以启用虚拟硬件 watchdog 设备并设置其行为。如果实例挂起或失败，watchdog 设备会执行配置的操作。watchdog 使用 i6300esb 设备，它模拟 PCI Intel 6300ESB。如果没有指定 **hw:watchdog_action**，则禁用 watchdog。 设置为以下有效值之一： ● **禁用** ：（默认）设备不会被附加。 ● **重置** ：强制实例重置。 ● **poweroff**: 强制实例关闭。 ● **暂停**: 暂停实例。 ● **none** ：启用 watchdog，但当实例挂起或失败时不做任何操作。 **注意** 使用特定镜像的属性设置的 watchdog 行为会覆盖您使用类别设置的行为。

随机数生成器(RNG)

使用下表中的属性键在实例上启用 RNG 设备。

表 8.8. RNG 的类别元数据

键	描述
hw_rng:allowed	设置为 **False**，以禁用通过其镜像属性添加到实例的 RNG 设备。 Default: **True**

键	描述
hw_rng:rate_bytes	指定实例可在每个期间从主机的熵中读取的最大字节数。
hw_rng:rate_period	以毫秒为单位指定读取周期的持续时间。

虚拟性能监控单元(vPMU)

使用下表中的属性键为实例启用 vPMU。

表 8.9. vPMU 的类别元数据

键	描述
hw:pmu	设置为 **True**，为实例启用 vPMU。 **perf** 等工具使用实例上的 vPMU 工具，为配置集和监控实例性能提供更准确的信息。对于实时工作负载，vPMU 的模拟可能会引入额外的延迟。如果不需要提供的遥测，请设置 **hw:pmu=False**。

实例 CPU 拓扑

使用下表中的属性键来定义实例中处理器的拓扑。

表 8.10. CPU 拓扑的类别元数据

键	描述
hw:cpu_sockets	指定实例的首选插槽数。 默认：请求的 vCPU 数量
hw:cpu_cores	指定实例每个插槽的首选内核数。 默认：**1**
hw:cpu_threads	指定实例每个内核的首选线程数量。 默认：**1**
hw:cpu_max_sockets	使用镜像属性指定用户可以为其实例选择的最大插槽数。 示例：**hw:cpu_max_sockets=2**
hw:cpu_max_cores	使用镜像属性指定用户可以为其实例选择的最大内核数。
hw:cpu_max_threads	使用镜像属性指定用户可以为其实例选择的最大线程数量。

串行端口

使用下表中的属性键配置每个实例的串行端口数。

表 8.11. 串行端口的类别元数据

键	描述
hw:serial_port_count	每个实例的最大串行端口。

CPU 固定策略

默认情况下，实例虚拟 CPU (vCPU)是带有一个内核的插槽，以及一个线程。您可以使用属性创建将实例 vCPU 固定到主机的物理 CPU 内核(pCPU)的类别。您还可以在并发多线程(SMT)架构中配置硬件 CPU 线程行为，其中一个或多个内核有线程同级。

使用下表中的属性键来定义实例的 CPU 固定策略。

表 8.12. CPU 固定的类别元数据

键	描述
hw:cpu_policy	指定要使用的 CPU 策略。设置为以下有效值之一： ● **共享** ：（默认）主机 pCPU 之间实例 vCPU 浮点数。 ● **专用** ：在实例 vCPU 中到一组主机 pCPU。这会创建一个与实例固定 CPU 拓扑匹配的实例 CPU 拓扑。这个选项表示过量使用比率为 1.0。

键	描述

hw:cpu_thread_policy	指定 **hw:cpu_policy=dedicated** 时要使用的 CPU 线程策略。设置为以下有效值之一：
	• **首选** ：（默认）主机可能或没有 SMT 架构。如果存在 SMT 架构，计算调度程序会首选线程同级。
	• **隔离** ：主机不能具有 SMT 架构，或者必须模拟非 SMT 架构。此策略通过请求没有报告 **HW_CPU_HYPERTHREADING** 特征的主机来确保计算调度程序将实例放置在没有 SMT 的主机上。也可以使用以下属性明确请求此特征：
	`--property trait:HW_CPU_HYPERTHREADING=forbidden`
	如果主机没有 SMT 架构，计算服务会将每个 vCPU 放置到不同的内核中。如果主机具有 SMT 架构，则行为由 **[workarounds]/disable_fallback_pcpu_query** 参数的配置决定：
	◦ **true** ：不使用 SMT 架构的主机，调度失败。
	◦ **false** ：计算服务将每个 vCPU 放置到不同的物理内核中。Compute 服务不会将来自其他实例的 vCPU 放置到同一内核中。因此，每个使用的内核中有一个线程的同级都保证不可用。
	• **require** ：主机必须具有 SMT 架构。此策略通过请求报告 **HW_CPU_HYPERTHREADING** 特征的主机来确保计算调度程序将实例放置到具有 SMT 的主机上。也可以使用以下属性明确请求此特征：
	`--property trait:HW_CPU_HYPERTHREADING=required`
	Compute 服务在线程同级时分配每个 vCPU。如果主机没有 SMT 架构，则不会使用它。如果主机具有 SMT 架构，但没有足够内核，且没有足够的空闲线程的内核可用，则调度会失败。

实例 PCI NUMA 关联性策略

使用下表中的属性键创建类别，为 PCI 透传设备和 SR-IOV 接口指定 NUMA 关联性策略。

表 8.13. PCI NUMA 关联性策略的类别元数据

键	描述

键	描述

hw:pci_numa_affinity_policy	指定 PCI 透传设备和 SR-IOV 接口的 NUMA 关联性策略。设置为以下有效值之一：
	• **必需** ：计算服务会创建一个实例，它仅在实例的至少一个 NUMA 节点与 PCI 设备具有关联性时才请求 PCI 设备。这个选项提供最佳性能。
	• **preferred** ：计算服务会尝试根据 NUMA 关联性选择 PCI 设备。如果无法做到这一点，则计算服务会将实例调度到没有 PCI 设备关联性的 NUMA 节点上。
	• **旧** ：（默认）计算服务在以下情况下创建请求 PCI 设备的实例：
	  ◦ PCI 设备与至少一个 NUMA 节点具有关联性。
	  ◦ PCI 设备不提供有关其 NUMA 关联性的信息。

实例 NUMA 拓扑

您可以使用属性创建类别，为实例 vCPU 线程定义主机 NUMA 放置，以及从主机 NUMA 节点分配实例 vCPU 和内存。

为实例定义 NUMA 拓扑可提高实例操作系统的性能，用于内存和 vCPU 分配大于 Compute 主机中的 NUMA 节点大小。

计算调度程序使用这些属性来确定适合实例的主机。例如，一个云用户使用以下类别启动实例：

```
$ 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
```

计算调度程序搜索有两个 NUMA 节点的主机，一个有 3GB RAM，另一个运行 6 个 CPU，另一个具有 1GB RAM 和两个 CPU。如果主机只有一个 NUMA 节点，其能够运行 8 个 CPU 和 4GB RAM，则计算调度程序不会认为它是一个有效的匹配项。

注意

由类别定义的 NUMA 拓扑无法被镜像定义的 NUMA 拓扑覆盖。如果镜像 NUMA 拓扑与类别 NUMA 拓扑冲突，计算服务会引发 **ImageNUMATopologyForbidden** 错误。

小心

您不能使用此功能将实例限制到特定的主机 CPU 或 NUMA 节点。只有在完成广泛的测试和性能测量后，才使用此功能。您可以使用 **hw:pci_numa_affinity_policy** 属性替代。

使用下表中的属性键来定义实例 NUMA 拓扑。

表 8.14. NUMA 拓扑的类别元数据

键	描述
hw:numa_nodes	指定限制实例 vCPU 线程执行的主机 NUMA 节点数量。如果没有指定，vCPU 线程可以在任意数量的可用主机 NUMA 节点上运行。
hw:numa_cpus.N	以逗号分隔的实例 vCPU 列表，用于映射到实例 NUMA 节点 N。如果没有指定此密钥，则 vCPU 会均匀地划分到可用的 NUMA 节点。 N 从 0 开始。请谨慎使用 sVirtN 值，只有在您至少有两个 NUMA 节点时才使用。 只有在设置了 **hw:numa_nodes** 时才有效，只有在实例的 NUMA 节点具有非对称分配 CPU 和 RAM 时，才需要它，这对某些 NFV 工作负载非常重要。
hw:numa_mem.N	映射到实例 NUMA 节点 N 的 MB 实例内存量。如果没有指定此密钥，内存将平均划分到可用的 NUMA 节点。 N 从 0 开始。请谨慎使用 sVirtN 值，只有在您至少有两个 NUMA 节点时才使用。 只有在设置了 **hw:numa_nodes** 时才有效，只有在实例的 NUMA 节点具有非对称分配 CPU 和 RAM 时，才需要它，这对某些 NFV 工作负载非常重要。

警告

如果 **hw:numa_cpus.N** 或 **hw:numa_mem.N** 的组合分别大于可用的 CPU 或内存量，则计算服务会引发异常。

实例内存加密

使用下表中的属性键启用实例内存的加密。

表 8.15. 内存加密的类别元数据

键	描述
hw:mem_encryption	设置为 **True**，为实例请求内存加密。如需更多信息，请参阅配置 AMD SEV Compute 节点，以便为实例提供内存加密。

CPU 实时策略

使用下表中的属性键来定义实例中处理器的实时策略。

注意

- 虽然大多数实例 vCPU 都可以使用实时策略运行，但必须至少将一个 vCPU 标记为非实时客户机进程，以用于非实时客户机处理和模拟器开销。
- 要使用此额外规格，您必须启用固定 CPU。

表 8.16. CPU 实时策略的类别元数据

键	描述
hw:cpu_realtime	设置为 **yes**，以创建为实例 vCPU 分配实时策略的类别。 默认： **no**

hw:cpu_realtime_mask	指定要为其分配实时策略的 vCPU。您必须使用符号符号(^)添加掩码值。以下示例表示除 vCPU 0 和 1 以外的所有 vCPU 都有一个实时策略：
	
	`$ openstack flavor set <flavor> \`
	`--property hw:cpu_realtime="yes" \`
	`--property hw:cpu_realtime_mask=^0-1`
	
	  **注意**
	
	如果在镜像上设置 **hw_cpu_realtime_mask** 属性，则优先于类别上设置的 **hw:cpu_realtime_mask** 属性。

模拟器线程策略

您可以为实例分配 pCPU，以用于仿真程序线程。模拟器线程是与实例不直接相关的仿真程序进程。实时工作负载需要专用仿真程序线程 pCPU。要使用仿真程序线程策略，您必须通过设置以下属性来启用固定 CPU：

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

使用下表中的属性键来定义实例的仿真程序线程策略。

表 8.17. 模拟器线程策略的类别元数据

键	描述

键	描述
hw:emulator_threads_policy	指定用于实例的仿真程序线程策略。设置为以下有效值之一： • **共享** ：**NovaComputeCpuSharedSet** heat 参数中定义的 pCPU 的仿真程序线程浮点数。如果没有配置 **NovaComputeCpuSharedSet**，则与实例关联的固定 CPU 中的仿真程序线程浮点数。 • **隔离** ：为仿真程序线程为每个实例保留额外的专用 pCPU。请谨慎使用此策略，因为它是禁止大量资源。 • unset ：（默认）仿真程序线程策略没有被启用，与实例关联的固定 CPU 中的仿真程序线程浮点数。

实例内存页面大小

使用下表中的属性键创建具有显式内存页面大小的实例。

表 8.18. 内存页面大小的类别元数据

键	描述
hw:mem_page_size	指定用于支持实例的巨页大小。使用此选项会创建一个 1 个 NUMA 节点的隐式 NUMA 拓扑，除非由 **hw:numa_nodes** 指定。设置为以下有效值之一： • **large** ：选择大于主机上支持的最小页大小的页大小（在 x86_64 上可以是 2 MB 或 1 GB）。 • **Small** ：选择主机上支持的最小页面大小。在 x86_64 系统中，这是 4 kB （普通页）。 • **any** ：选择最大可用的巨页大小，具体由 libvirt 驱动程序决定。 • **<pagesize>**：如果工作负载具有特定要求，明确设置页的大小。对页大小（以 KB 为单位）或任何标准后缀使用整数值。例如：**4KB**、**2MB**、**2048**、**1GB**。 • unset: （默认）大页不用于备份实例，且不会生成隐式 NUMA 拓扑。

PCI 透传

使用下表中的属性键将物理 PCI 设备（如图形卡或网络设备）附加到实例。有关使用 PCI 透传的更多信息，请参阅配置 PCI 透传。

表 8.19. PCI 透传的类别元数据

键	描述

键	描述
pci_passthrough:alias	使用以下格式指定要分配给实例的 PCI 设备： <alias>:<count> • 将 **<** alias> 替换为与特定 PCI 设备类对应的别名。 • 将 **<count** > 替换为分配给实例的类型 &lt **;alias&** gt; 的 PCI 设备数。

hypervisor 签名

使用下表中的属性键从实例隐藏 hypervisor 签名。

表 8.20. 用于隐藏 hypervisor 签名的类别元数据

键	描述
hide_hypervisor_id	设置为 **True**，从实例隐藏虚拟机监控程序签名，以允许所有驱动程序加载并在实例上工作。

实例资源特征

每个资源供应商都有一组特征。特征是资源提供程序的限定方面，如存储类型或 Intel CPU 指令集扩展名。一个实例可以指定其所需的特征。

您可以指定的特征在 **os-traits** 库中定义。特征示例包括：

- **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**

有关如何使用 **os-traits** 库的详情，请参考 https://docs.openstack.org/os-traits/latest/user/index.html。

使用下表中的属性键来定义实例的资源特征。

表 8.21. 资源特征的类别元数据

键	描述

键	描述
trait:<trait_name>	指定 Compute 节点特征。将特征设置为以下有效值之一： • **必需** ：选择托管实例的 Compute 节点必须具有特征。 • **Forbid** id：选择 Compute 节点来托管实例不能具有特征。 例如： $ openstack flavor set --property trait:HW_CPU_X86_AVX512BW=required avx512-flavor

实例裸机资源类

使用下表中的属性键为实例请求裸机资源类。

表 8.22. 裸机资源类的类别元数据

键	描述

键	描述
resources:<resource_class_name>	使用此属性指定标准裸机资源类来覆盖值，或者指定实例所需的自定义裸机资源类。 您可以覆盖的标准资源类有**MEMORY_MB** 和 **DISK_GB**。为**防止计算调度程序使用裸机类别属性来调度实例，请将标准资源类的值设为 0。** **自定义资源类的名称必须以 CUSTOM_ 开头。要确定与 Bare Metal 服务节点的资源类对应的自定义资源类的名称，请将资源类转换为大写，请将所有 punctuation 替换为下划线，并使用 CUSTOM_ 前缀。** **例如，要将实例调度到具有 --resource-class baremetal.SMALL 的节点上，请创建以下类型：** **$ 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**

第 9 章 在实例中添加元数据

Compute (nova)服务使用元数据在启动时将配置信息传递给实例。实例可以使用配置驱动器或元数据服务访问元数据。

配置驱动器

配置驱动器是可在引导时附加到实例的特殊驱动器。配置驱动器以只读驱动器形式呈现给实例。实例可以挂载此驱动器并从中读取文件，以获取通常通过元数据服务提供的信息。

元数据服务

Compute 服务将元数据服务提供为 REST API，可用于检索特定于实例的数据。实例通过 **169.254.169.254** 或 **fe80::a9fe:a9fe** 访问此服务。

9.1. 实例元数据类型

云用户、云管理员和计算服务可以将元数据传递给实例：

云用户提供的数据

云用户可以指定在启动实例时要使用的附加数据，如实例在引导时运行的 shell 脚本。云用户可以使用用户数据功能将数据传递给实例，并在创建或更新实例时根据需要传递键值对。

云管理员提供数据

RHOSP 管理员使用 vendordata 功能将数据传递给实例。Compute 服务提供 vendordata 模块 **StaticJSON** 和 **DynamicJSON**，以允许管理员将元数据传递给实例：

- **StaticJSON** ：（默认）适用于所有实例的元数据。
- **DynamicJSON** ：将用于每个实例不同的元数据。此模块向外部 REST 服务发出请求，以确定要添加到实例的元数据。

vendordata 配置位于实例中以下只读文件之一：

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

计算服务提供的数据

Compute 服务使用元数据服务的内部实施将信息传递给实例，如实例请求的主机名，以及实例所在可用区。默认情况下发生这种情况，且不需要云用户或管理员配置。

9.2. 在所有实例中添加配置驱动器

作为管理员，您可以将计算服务配置为始终为实例创建配置驱动器，并使用特定于部署的元数据填充配置驱动器。例如，您可以使用配置驱动器的原因：

- 要在部署不使用 DHCP 为实例分配 IP 地址时传递网络配置。您可以通过配置驱动器传递实例的 IP 地址配置，实例可以在为实例配置网络设置前挂载和访问。
- 要将数据传递给对启动实例的用户不知道的实例，例如，用于通过 Active Directory 后注册实例的加密令牌。
- 要创建本地缓存的磁盘读取，以管理实例请求的负载，这可以减少对元数据服务器定期访问元数据服务器的影响，以检查和构建事实。

任何能够挂载 ISO 9660 或 VFAT 文件系统的实例操作系统都可以使用配置驱动器。

流程

1. 打开您的计算环境文件。
2. 要在启动实例时始终附加配置驱动器，请将以下参数设置为 **True** ：

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

3. 可选： 要将配置驱动器的格式从 **iso9660** 的默认值改为 **vfat**，请在您的配置中添加 **config_drive_format** 参数：

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

4. 保存对 Compute 环境文件的更新。
5. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

验证

1. 创建一个实例：

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

2. 登录该实例。
3. 挂载配置驱动器：
   - 如果实例操作系统使用 **udev** ：

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

   - 如果实例操作系统没有使用 **udev**，则首先需要识别与配置驱动器对应的块设备：

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

4. 根据您的元数据，检查挂载的配置驱动器目录中 **mnt/config/openstack/{version}/** 中的文件。

9.3. 向实例添加动态元数据

您可以配置部署以创建特定于实例的元数据，并通过 JSON 文件向该实例提供元数据。

提示

您可以使用 undercloud 上的动态元数据将 director 与 Red Hat Identity Management (IdM)服务器集成。当 overcloud 上启用了 SSL/TLS 时，IdM 服务器可用作证书颁发机构并管理 overcloud 证书。如需更多信息，请参阅将 undercloud 添加到 IdM。

流程

1. 打开您的计算环境文件。
2. 在 vendordata 供应商模块中添加 **DynamicJSON** ：

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

3. 指定要联系的 REST 服务来生成元数据。您可以根据需要指定任意数量的目标 REST 服务，例如：

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

Compute 服务生成 JSON 文件 **vendordata2.json**，使其包含从配置的目标服务检索的元数据，并将其存储在配置驱动器目录中。

注意

不要多次将同一名称用于目标服务。

4. 保存对 Compute 环境文件的更新。
5. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

第 10 章 为实例配置 CPU 功能标记

您可以在不更改主机 Compute 节点上的设置并重新引导 Compute 节点的情况下，为实例启用或禁用 CPU 功能标记。通过配置应用到实例的标准 CPU 功能标记集合，这有助于在 Compute 节点之间实现实时迁移兼容性。您还有助于管理实例的性能和安全性，方法是禁用对带有特定 CPU 模型的实例的安全性或性能的标记，或者启用提供安全问题缓解的标记。

10.1. 先决条件

- 主机 Compute 节点的硬件和软件必须支持 CPU 模型和功能标记：
  - 要检查主机支持的硬件，在 Compute 节点上输入以下命令：

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

  - 要检查主机上支持的 CPU 型号，请在 Compute 节点上输入以下命令：

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

    将 **<arch** > 替换为构架的名称，如 **x86_64**。

10.2. 为实例配置 CPU 功能标记

配置计算服务，将 CPU 功能标记应用到具有特定 vCPU 模型的实例。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

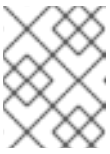
   ```
   [stack@director ~]$ source ~/stackrc
   ```

3. 打开您的计算环境文件。
4. 配置实例 CPU 模式：

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

   将 **<cpu_mode** > 替换为 Compute 节点上的每个实例的 CPU 模式。设置为以下有效值之一：

   - **host-model** ：（默认）使用主机 Compute 节点的 CPU 模型。使用此 CPU 模式为实例自动添加关键 CPU 标记，以提供安全漏洞的缓解方案。
   - **自定义** ：使用 来配置每个实例应使用的特定 CPU 型号。

     注意

     您还可以将 CPU 模式设置为 **host-passthrough**，以使用与该 Compute 节点上托管的实例相同的 CPU 模型和功能标记。

5. 可选：如果您将 **NovaLibvirtCPUMode** 设置为 **自定义**，请配置您要自定义的实例 CPU 型号：

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

将 **<cpu_model** > 替换为主机支持的 CPU 型号的逗号分隔列表。按顺序列出 CPU 型号，将更常见和较少的高级 CPU 型号放在列表中，最后是更丰富的 CPU 型号，例如 **SandyBridge,IvyBridge,Haswell,Broadwell**。有关模型名称列表，请参阅 **/usr/share/libvirt/cpu_map.xml**，或者在主机 Compute 节点上输入以下命令：

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

将 **<arch** > 替换为 Compute 节点的构架的名称，如 **x86_64**。

6. 为带有指定 CPU 模型的实例配置 CPU 功能标记：

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

将 **<cpu_feature_flags** > 替换为以逗号分隔的功能标记列表，以启用或禁用。使用 "+" 为每个标记添加前缀以启用标志，或"-"来禁用它。如果没有指定前缀，则会启用标志。有关给定 CPU 模型的可用功能标记列表，请参阅 **/usr/share/libvirt/cpu_map the.xml**。

以下示例为 **IvyBridge** 和 **Cascadelake-Server** 模型启用 CPU 功能标记 **pcid** 和 **sbd**，并禁用功能标记 **mtrr**。

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

7. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

第 11 章 配置手动节点重新引导以定义 KERNELARGS

当 overcloud 部署包含第一次设置 **KernelArgs** 时，overcloud 节点会自动重启。如果您要在生产环境中的部署中添加 **KernelArgs**，则重新引导节点可能是现有工作负载的问题。您可以在更新部署时禁用节点自动重新引导，而是在每个 overcloud 部署后手动执行节点重启。

注意

如果您禁用自动重启，然后在部署中添加新的 Compute 节点，则在初始置备过程中不会重启新节点。这可能导致部署错误，因为只在重启后应用 **KernelArgs** 的配置。

11.1. 配置手动节点重新引导以定义 KERNELARGS

您可以在首次配置 **KernelArgs** 时禁用节点自动重新引导，而是手动重新引导节点。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

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

3. 在自定义环境文件中启用 **KernelArgsDeferReboot** 角色参数，例如 **kernelargs_manual_reboot.yaml** ：

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

4. 使用其他环境文件将自定义环境文件添加到堆栈中，并部署 overcloud：

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

5. 检索 Compute 节点列表，以识别您要重新引导的节点的主机名：

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

6. 在您要重新引导的 Compute 节点上禁用 Compute 服务，以防止计算调度程序将新实例分配给节点：

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

将 **<**node> 替换为您要禁用 Compute 服务的节点的主机名。

7. 检索托管在您要迁移的 Compute 节点上的实例列表：

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

8. 将实例迁移到另一个 Compute 节点中。有关 迁移实例的详情，请参考在 Compute 节点之间迁移虚拟机实例。

9. 登录您要重新引导的节点。

10. 重新引导节点：

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

11. 稍等片刻，直到节点启动。

12. 重新启用 Compute 节点：

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

13. 确认是否已启用 Compute 节点：

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

第 12 章 配置 AMD SEV COMPUTE 节点，为实例提供内存加密

作为云管理员，您可以让云用户能够创建在启用了内存加密的 SEV 支持的 Compute 节点上运行的实例。

此功能可从第二代 AMD EPYC™ 7002 系列("Rome")中使用。

要让您的云用户创建使用内存加密的实例，您必须执行以下任务：

1. 指定用于内存加密的 AMD SEV Compute 节点。
2. 为内存加密配置 Compute 节点。
3. 部署 overcloud。
4. 创建类别或镜像以使用内存加密启动实例。

提示

如果 AMD SEV 硬件有限，您还可以配置主机聚合来优化 AMD SEV Compute 节点上的调度。要仅将请求内存加密的实例调度到 AMD SEV Compute 节点上，请创建一个具有 AMD SEV 硬件的 Compute 节点的主机聚合，并将计算调度程序配置为仅将请求内存加密的实例放在主机聚合中。如需更多信息，请参阅 Creating and managing host aggregates 和 Filtering by isolating host aggregates 。

12.1. 安全加密虚拟化(SEV)

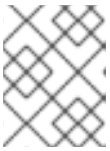
由 AMD 提供的安全加密虚拟化(SEV)保护正在运行的虚拟机实例正在使用的 DRAM 中的数据。SEV 使用唯一密钥加密每个实例的内存。

当使用非易失性内存技术(NVDIMM)时，SEV 会提高安全性，因为 NVDIMM 芯片可从具有数据的系统物理移除，类似于硬盘。如果没有加密，任何存储的信息（如敏感数据、密码或机密密钥）都可能会被破坏。

如需更多信息，请参阅 *AMD 安全加密虚拟化(SEV)* 文档。

具有内存加密的实例的限制

- 您无法实时迁移，或使用内存加密暂停和恢复实例。
- 您不能使用 PCI 透传直接访问具有内存加密的实例上的设备。
- 您不能使用 **virtio-blk** 作为之前早于 kernel-4.18.0-115.el8 (RHEL-8.1.0)内核的内存加密实例的引导磁盘。



注意

您可以使用 **virtio-scsi** 或 **SATA** 作为引导磁盘，或者 **virtio-blk** 用于非引导磁盘。

- 在加密实例中运行的操作系统必须提供 SEV 支持。如需更多信息，请参阅 在 RHEL 8 中启用 AMD 安全加密虚拟化 的 红帽知识库解决方案。
- 支持 SEV 的机器在其内存控制器中具有有限的插槽来存储加密密钥。每个带有加密内存的实例都会消耗其中一个插槽。因此，可同时运行的内存加密的实例数量仅限于内存控制器中的插槽数量。例如，在1st Gen AMD EPYC™ 7001 系列("Naples")中，这个限制为 16，在第二代 Gen AMD EPYC™ 7002 系列("Rome")时，这个限制为 255。

- 具有内存加密的实例固定 RAM 中的页面。Compute 服务无法交换这些页面，因此您无法过量使用托管内存加密的 Compute 节点上的内存。
- 您不能将内存加密用于具有多个 NUMA 节点的实例。

12.2. 为内存加密设计 AMD SEV COMPUTE 节点

要为使用内存加密的实例指定 AMD SEV Compute 节点，您必须创建一个新角色文件来配置 AMD SEV 角色，并配置一个新的 overcloud 类别和 AMD SEV 资源类，以用于标记 Compute 节点以进行内存加密。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

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

3. 生成包含 **ComputeAMDSEV** 角色的新角色数据文件，以及 overcloud 所需的任何其他角色。以下示例生成角色数据文件 **roles_data_amd_sev.yaml**，其中包括角色 **Controller** 和 **ComputeAMDSEV** ：

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

4. 打开 **roles_data_amd_sev.yaml** 并编辑或添加以下参数和部分：

section/Parameter	当前值	新值
角色注释	**Role: Compute**	**Role: ComputeAMDSEV**
角色名称	**名称 ：Compute**	**name: ComputeAMDSEV**
description	**基本 Compute 节点角色**	**AMD SEV Compute 节点角色**
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputeamdsev-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-amd-sev.yaml**

5. 将 overcloud 的 AMD SEV Compute 节点添加到节点定义模板、**node.json** 或 **node.yaml** 中，以注册它们。有关更多信息，请参阅 *Director 安装和使用* 指南中的 为 overcloud 注册节点。
6. 检查节点硬件：

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

如需更多信息，请参阅 *Director 安装和使用指南中的* 创建裸机节点硬件清单。

7. 为 AMD SEV Compute 节点创建 **compute-amd-sev** overcloud 类别：

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

- 将 **<ram_size_mb>** 替换为裸机节点的 RAM，以 MB 为单位。
- 将 **<disk_size_gb>** 替换为裸机节点中的磁盘大小（以 GB 为单位）。
- 将 **<no_vcpus>** 替换为裸机节点中的 CPU 数量。

注意

这些属性不可用于调度实例。但是，计算调度程序使用磁盘大小来确定根分区大小。

8. 检索节点列表来识别它们的 UUID：

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

9. 使用自定义 AMD SEV 资源类标记您要为内存加密指定的每个裸机节点：

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

将 **<node>** 替换为裸机节点的 ID。

10. 将 **compute-amd-sev** 类别与自定义 AMD SEV 资源类关联：

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

要确定与 Bare Metal 服务节点的资源类型对应的自定义资源类的名称，请将资源类转换为大写，将每个 punctuation 标记替换为下划线，并使用 **CUSTOM_** 前缀。

注意

类别只能请求一个裸机资源类实例。

11. 设置以下类别属性，以防止计算调度程序使用裸机类别属性来调度实例：

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

12. 可选：如果 **ComputeAMDSEV** 角色的网络拓扑与 **Compute** 角色的网络拓扑不同，则创建自定义网络接口模板。有关更多信息，请参阅*高级 Overcloud 自定义指南*中的自定义网络接口模板。如果 **ComputeAMDSEV** 角色的网络拓扑与 **Compute** 角色相同，您可以使用 **compute.yaml** 中定义的默认网络拓扑。

13. 在 **network-environment.yaml** 文件中注册 **ComputeAMDSEV** 角色的 **Net::SoftwareConfig** ：

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

将 < **amd_sev_net_top** > 替换为包含 **ComputeAMDSEV** 角色的网络拓扑的文件名称，如 **compute.yaml** 以使用默认网络拓扑。

14. 在 **node-info.yaml** 文件中添加以下参数，以指定 AMD SEV Compute 节点的数量，以及用于指定 AMD SEV 的 Compute 节点使用的 flavor：

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

15. 要验证角色是否已创建，请输入以下命令：

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

输出示例：

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

```
    "Provisioning State": "active",
    "Resource Class": "controller",
    "UUID": "474c2fc8-b884-4377-b6d7-781082a3a9c0"
  }
]
```

12.3. 为内存加密配置 AMD SEV COMPUTE 节点

要让您的云用户创建使用内存加密的实例，您必须配置具有 AMD SEV 硬件的 Compute 节点。

先决条件

- 您的部署必须包含在支持 SEV 的 AMD 硬件中运行的 Compute 节点，如 AMD EPYC CPU。您可以使用以下命令来确定您的部署是否支持 SEV：

```
$ lscpu | grep sev
```

流程

1. 打开您的计算环境文件。
2. 可选：在 Compute 环境文件中添加以下配置，以指定 AMD SEV Compute 节点可以同时托管的内存加密实例的最大数量：

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

注意

libvirt/num_memory_encrypted_guests 参数的默认值为 **none**。如果您没有设置自定义值，AMD SEV Compute 节点不会对节点可以同时托管的内存加密实例数量实施限制。相反，硬件决定了 AMD SEV Compute 节点可以同时托管的内存加密实例的最大数量，这可能会导致一些内存加密实例无法启动。

3. 可选： 要指定所有 x86_64 镜像默认使用 q35 机器类型，请在 Compute 环境文件中添加以下配置：

```
parameter_defaults:
  ComputeAMDSEVParameters:
    NovaHWMachineType: x86_64=q35
```

如果指定了此参数值，则不需要在每个 AMD SEV 实例镜像中将 **hw_machine_type** 属性设置为 **q35**。

4. 为确保 AMD SEV Compute 节点保留足够的内存以便主机级服务正常工作，为每个潜在的 AMD SEV 实例添加 16MB：

```
parameter_defaults:
  ComputeAMDSEVParameters:
    ...
```

```
    NovaReservedHostMemory: <libvirt/num_memory_encrypted_guests * 16>
```

5. 为 AMD SEV Compute 节点配置内核参数：

```
parameter_defaults:
  ComputeAMDSEVParameters:
    ...
    KernelArgs: "hugepagesz=1GB hugepages=32 default_hugepagesz=1GB
mem_encrypt=on kvm_amd.sev=1"
```

注意

首次将 **KernelArgs** 参数添加到角色的配置中时，overcloud 节点会自动重启。如果需要，您可以禁用节点自动重新引导，而是在每次 overcloud 部署后手动执行节点重启。如需更多信息，请参阅配置手动节点重新引导以定义 **KernelArgs**。

6. 保存对 Compute 环境文件的更新。

7. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

12.4. 为内存加密创建镜像

当 overcloud 包含 AMD SEV Compute 节点时，您可以创建一个 AMD SEV 实例镜像，供您的云用户用于启动具有内存加密的实例。

流程

1. 为内存加密创建新镜像：

```
(overcloud)$ openstack image create ...  \
 --property hw_firmware_type=uefi amd-sev-image
```

注意

如果您使用现有镜像，镜像必须将 **hw_firmware_type** 属性设置为 **uefi**。

2. 可选：将属性 **hw_mem_encryption=True** 添加到镜像中，以便在镜像中启用 AMD SEV 内存加密：

```
(overcloud)$ openstack image set  \
 --property hw_mem_encryption=True amd-sev-image
```

提示

您可以在类别上启用内存加密。如需更多信息，请参阅为内存加密创建类别。

3. 可选：如果尚未在 Compute 节点配置中设置，请将机器类型设置为 **q35** ：

```
(overcloud)$ openstack image set \
 --property hw_machine_type=q35 amd-sev-image
```

4. 可选：要在支持 SEV 的主机聚合上调度内存加密实例，请在镜像额外规格中添加以下特征：

```
(overcloud)$ openstack image set \
 --property trait:HW_CPU_X86_AMD_SEV=required amd-sev-image
```

提示

您还可以在类别上指定此特征。如需更多信息，请参阅为内存加密创建类别。

12.5. 为内存加密创建类别

当 overcloud 包含 AMD SEV Compute 节点时，您可以创建一个或多个 AMD SEV 类别，您的云用户可用于启动具有内存加密的实例。

注意

只有在镜像上未设置 **hw_mem_encryption** 属性时，才需要 AMD SEV 类别。

流程

1. 为内存加密创建类别：

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

2. 要将内存加密实例调度到支持 SEV 的主机聚合上，请在类别额外规格中添加以下特征：

```
(overcloud)$ openstack flavor set \
 --property trait:HW_CPU_X86_AMD_SEV=required m1.small-amd-sev
```

12.6. 使用内存加密启动实例

要验证您可以在启用了内存加密的 AMD SEV Compute 节点上启动实例，请使用内存加密类别或镜像来创建实例。

流程

1. 使用 AMD SEV 类别或镜像创建实例。以下示例使用在为内存加密创建类别和 为内存加密 创建镜像时创建的类别来创建实例： ???

```
(overcloud)$ openstack server create --flavor m1.small-amd-sev \
 --image amd-sev-image amd-sev-instance
```

2. 以云用户身份登录实例。

3. 要验证实例是否使用内存加密，请从实例输入以下命令：

```
$ dmesg | grep -i sev
AMD Secure Encrypted Virtualization (SEV) active
```

第 13 章 配置 NVDIMM COMPUTE 节点来为实例提供持久内存

非易失性双内存模块(NVDIMM)是为 DRAM 提供持久内存(PMEM)的技术。在断电后，标准计算机内存将会丢失其数据。NVDIMM 即使断电后也会维护其数据。使用 PMEM 的实例可以提供应用程序在电源周期之间保留应用程序的大量内存数量。这对需要大量内存的高性能计算(DSL)非常有用。

作为云管理员，您可以通过在具有 NVDIMM 硬件的 Compute 节点上创建和配置 PMEM 命名空间，将 PMEM 命名空间作为虚拟 PMEM (vPMEM)使用。然后，当云用户需要保留实例内容时，可以创建请求 vPMEM 的实例。

要让您的云用户创建使用 PMEM 的实例，您必须完成以下步骤：

1. 为 PMEM 指定 Compute 节点。
2. 为具有 NVDIMM 硬件的 PMEM 配置 Compute 节点。
3. 部署 overcloud。
4. 创建 PMEM 类别以启动具有 vPMEM 的实例。

提示

如果 NVDIMM 硬件有限，您还可以配置主机聚合来优化 PMEM Compute 节点上的调度。要仅在 PMEM Compute 节点上调度请求 vPMEM 的实例，请创建一个具有 NVDIMM 硬件的 Compute 节点的主机聚合，并将计算调度程序配置为仅将 PMEM 实例放在主机聚合中。如需更多信息，请参阅 Creating and managing host aggregates 和 Filtering by isolating host aggregates 。

先决条件

- 您的 Compute 节点具有持久内存硬件，如 Intel® Optane™ DC Persistent Memory。
- 您已在 PMEM 硬件设备中配置了后端 NVDIMM 区域来创建 PMEM 命名空间。您可以使用 Intel 提供的 ipmctl 工具来配置 PMEM 硬件。

使用 PMEM 设备时的限制

- 您无法冷迁移、实时迁移、调整大小或暂停和恢复使用 vPMEM 的实例。
- 只有运行 RHEL8 的实例可以使用 vPMEM。
- 在重建 vPMEM 实例时，持久性内存命名空间会被删除来恢复实例的初始状态。
- 使用新类别重新定义实例大小时，原始虚拟持久内存的内容不会复制到新的虚拟持久内存中。
- 不支持虚拟持久内存热插。
- 在为 vPMEM 实例创建快照时，不会包含虚拟持久镜像。

13.1. 为 PMEM 设计 COMPUTE 节点

要为 PMEM 工作负载指定 Compute 节点，您必须创建一个新角色文件来配置 PMEM 角色，并为 PMEM 配置一个新的 overcloud 类别和资源类，以用于标记 NVDIMM Compute 节点。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

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

3. 生成名为 **roles_data_pmem.yaml** 的新角色数据文件，其中包含 **Controller**, **Compute**, 和 **ComputePMEM** 角色：

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

4. 打开 **roles_data_pmem.yaml** 并编辑或添加以下参数和部分：

section/Parameter	当前值	新值
角色注释	**Role: Compute**	**Role: ComputePMEM**
角色名称	**名称 ：Compute**	**名称 ：ComputePMEM**
description	**基本 Compute 节点角色**	**PMEM Compute 节点角色**
HostnameFormatDefault	**%stackname%-novacompute-%index%**	**%stackname%-novacomputepmem-%index%**
deprecated_nic_config_name	**compute.yaml**	**compute-pmem.yaml**

5. 将 overcloud 的 NVDIMM Compute 节点添加到节点定义模板、**node.json** 或 **node.yaml** 中，以注册它们。有关更多信息，请参阅 *Director 安装和使用* 指南中的 为 overcloud 注册节点。
6. 检查节点硬件：

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

如需更多信息，请参阅 *Director 安装和使用指南中的* 创建裸机节点硬件清单。

7. 为 PMEM Compute 节点创建 **compute-pmem** overcloud 类别：

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

- 将 **<ram_size_mb>** 替换为裸机节点的 RAM，以 MB 为单位。
- 将 **<disk_size_gb>** 替换为裸机节点中的磁盘大小（以 GB 为单位）。
- 将 **<no_vcpus>** 替换为裸机节点中的 CPU 数量。

注意

这些属性不可用于调度实例。但是，计算调度程序使用磁盘大小来确定根分区大小。

8. 使用自定义 PMEM 资源类标记您要为 PMEM 工作负载指定的每个裸机节点：

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

将 **<node>** 替换为裸机节点的 ID。

9. 将 **compute-pmem** 类别与自定义 PMEM 资源类关联：

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_PMEM=1 \
 compute-pmem
```

要确定与 Bare Metal 服务节点的资源类对应的自定义资源类的名称，请将资源类转换为大写，请将所有 punctuation 替换为下划线，并使用 **CUSTOM_** 前缀。

注意

类别只能请求一个裸机资源类实例。

10. 设置以下类别属性，以防止计算调度程序使用裸机类别属性来调度实例：

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

11. 在 **node-info.yaml** 文件中添加以下参数，以指定 PMEM Compute 节点的数量，以及用于 PMEM 设计的 Compute 节点的类别：

```
parameter_defaults:
  OvercloudComputePMEMFlavor: compute-pmem
  ComputePMEMCount: 3 #set to the no of NVDIMM devices you have
```

12. 要验证角色是否已创建，请输入以下命令：

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

13.2. 配置 PMEM COMPUTE 节点

要让您的云用户创建使用 vPMEM 的实例，您必须配置具有 NVDIMM 硬件的 Compute 节点。

流程

1. 创建一个新的计算环境文件，用于配置 NVDIMM Compute 节点，如 **env_pmem.yaml**。
2. 要将 NVDIMM 区域分区为 PMEM 命名空间，请将 **NovaPMEMNamespaces** 角色特定参数添加到计算环境文件中的 PMEM 角色，并使用以下格式设置值：

```
<size>:<namespace_name>[,<size>:<namespace_name>]
```

使用以下后缀来表示大小：

- "k" or "K" for KiB
- "m" or "M" for MiB
- "g" 或 "G" 用于 GiB
- "t" 或 "T" 代表 TiB
  例如，以下配置创建了四个命名空间，三个大小为 6 GiB，大小为 100 GiB ：

```
parameter_defaults:
  ComputePMEMParameters:
    NovaPMEMNamespaces: "6G:ns0,6G:ns1,6G:ns2,100G:ns3"
```

3. 要将 PMEM 命名空间映射到可在类别中使用的标签，请将 **NovaPMEMMappings** 角色特定参数添加到 Compute 环境文件中的 PMEM 角色，并使用以下格式设置值：

```
<label>:<namespace_name>[|<namespace_name>][,<label>:<namespace_name>[|
<namespace_name>]].
```

例如，以下配置将三个 6 GiB 命名空间映射到标签 "6GB"，并将 100 GiB 命名空间映射到标签 "LARGE" ：

```
parameter_defaults:
  ComputePMEMParameters:
    NovaPMEMNamespaces: "6G:ns0,6G:ns1,6G:ns2,100G:ns3"
    NovaPMEMMappings: "6GB:ns0|ns1|ns2,LARGE:ns3"
```

4. 保存对 Compute 环境文件的更新。
5. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

6. 创建并配置您的云用户可以使用 vPMEM 启动具有 vPMEM 的实例的类别。以下示例创建了一个请求小 PMEM 设备 6GB 的类别，如第 3 步映射：

```
(overcloud)$ openstack flavor create --vcpus 1 --ram 512 --disk 2  \
 --property hw:pmem='6GB' small_pmem_flavor
```

验证

1. 使用其中一个 PMEM 类别创建一个实例：

```
(overcloud)$ openstack flavor list
(overcloud)$ openstack server create --flavor small_pmem_flavor \
 --image rhel8 pmem_instance
```

2. 以云用户身份登录实例。如需更多信息，请参阅 连接到实例。

3. 列出附加到实例的所有磁盘设备：

```
$ sudo fdisk -l /dev/pmem0
```

如果列出的其中一个设备是 NVDIMM，则实例有 vPMEM。

第 14 章 为实例配置虚拟 GPU

要在实例上支持基于 GPU 的渲染，您可以根据可用的物理 GPU 设备和虚拟机监控程序类型定义和管理虚拟 GPU (vGPU)资源。您可以使用此配置更有效地在所有物理 GPU 设备间划分渲染的工作负载，并更好地控制调度启用了 vGPU 的实例。

要在 Compute (nova)服务中启用 vGPU，请创建您的云用户可以使用 vGPU 设备创建 Red Hat Enterprise Linux (RHEL)实例的类别。然后，每个实例都可以支持使用与物理 GPU 设备对应的虚拟 GPU 设备的 GPU 工作负载。

Compute 服务跟踪您为每个主机上定义的每个 GPU 配置集可用的 vGPU 设备数量。Compute 服务根据类别将实例调度到这些主机，附加设备，并持续监控使用情况。删除实例时，计算服务会将 vGPU 设备重新添加到可用池中。

重要

红帽在 RHOSP 中使用 NVIDIA vGPU，而无需支持例外。但是，红帽不为 NVIDIA vGPU 驱动程序提供技术支持。NVIDIA vGPU 驱动程序由 NVIDIA 提供并支持。您需要 NVIDIA 认证的支持服务订阅来获取 NVIDIA vGPU 软件的 NVIDIA Enterprise 支持。对于由于无法使用 NVIDIA vGPU 导致在支持的组件中无法重现问题的问题，会应用以下支持政策：

- 当红帽认为问题涉及第三方组件时，会应用正常的 支持范围 和 Red Hat SLA。
- 当红帽认为涉及第三方组件时，客户将遵循红帽 第三方支持和验证策略。如需更多信息，请参阅 NVIDIA 中的知识库文章。

14.1. 支持的配置和限制

支持的 GPU 卡

有关支持的 NVIDIA GPU 卡列表，请参阅 NVIDIA 网站上的 虚拟 GPU 软件支持的产品。

使用 vGPU 设备时的限制

- 每个 Compute 节点上只能启用一个 vGPU 类型。
- 每个实例只能使用一个 vGPU 资源。
- 不支持在主机间实时迁移 vGPU 实例。
- 不支持撤离 vGPU 实例。
- 如果您需要重新引导托管 vGPU 实例的 Compute 节点，则 vGPU 不会自动重新分配给重新创建的实例。您必须在重启 Compute 节点前冷迁移实例，或者在重新引导后手动将每个 vGPU 分配给正确的实例。要手动分配每个 vGPU，您必须在重启前从 Compute 节点上运行的每个 vGPU 实例的实例 XML 检索 **mdev** UUID。您可以使用以下命令为每个实例发现 **mdev** UUID：

  ```
  # virsh dumpxml <instance_name> | grep mdev
  ```

  将 **<instance_name** > 替换为 libvirt 实例名称 **OS-EXT-SRV-ATTR:instance_name**，在 **/servers** 请求中返回给 Compute API。

- 由于 libvirt 限制，不支持对支持 vGPU 的实例上挂起操作。相反，您可以对实例进行快照或升级。

- 默认情况下，Compute 主机上的 vGPU 类型不会向 API 用户公开。若要授予访问权限，请将主机添加到主机聚合中。如需更多信息，请参阅创建和管理主机聚合。
- 如果使用 NVIDIA 加速器硬件，您必须符合 NVIDIA 许可证要求。例如，Nvidia vGPU GRID 需要许可服务器。有关 NVIDIA 许可要求的更多信息，请参阅 NVIDIA 网站上的 NVIDIA 许可证服务器发行注记。

14.2. 在 COMPUTE 节点上配置 VGPU

要让您的云用户创建使用虚拟 GPU (vGPU)的实例，您必须配置具有物理 GPU 的 Compute 节点：

1. 为 vGPU 指定 Compute 节点。
2. 为 vGPU 配置 Compute 节点。
3. 部署 overcloud。
4. 创建 vGPU 类别以启动具有 vGPU 的实例。

提示

如果 GPU 硬件有限，您还可以配置主机聚合来优化 vGPU Compute 节点上的调度。要仅在 vGPU Compute 节点上调度请求 vGPU 的实例，请创建一个 vGPU Compute 节点的主机聚合，并将计算调度程序配置为仅将 vGPU 实例放在主机聚合中。如需更多信息，请参阅 Creating and managing host aggregates 和 Filtering by isolating host aggregates 。

注意

要使用 NVIDIA GRID vGPU，您必须符合 NVIDIA GRID 许可证要求，且您必须具有自托管许可证服务器的 URL。如需更多信息，请参阅 NVIDIA 许可证服务器发行注记 网页。

14.2.1. 先决条件

- 您已从 NVIDIA 网站下载了与 GPU 设备对应的 NVIDIA GRID 主机驱动程序 RPM 软件包。要确定您需要的驱动程序，请参阅 NVIDIA 驱动程序下载门户。您必须是一个注册的 NVIDIA 客户，才能从门户下载驱动程序。
- 您已构建了一个自定义 overcloud 镜像，该镜像安装了 NVIDIA GRID 主机驱动程序。

14.2.2. 为 vGPU 设计 Compute 节点

要为 vGPU 工作负载指定 Compute 节点，您必须创建一个新角色文件来配置 vGPU 角色，并配置一个新的 overcloud 类别和资源类来标记启用了 GPU 的 Compute 节点。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

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

3. 生成一个名为 **roles_data_gpu.yaml** 的新角色数据文件，其中包含 **Controller**, **Compute**, 和 **ComputeGpu** 角色：

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

4. 打开 **roles_data_gpu.yaml** 并编辑或添加以下参数和部分：

section/Parameter	当前值	新值
角色注释	**Role: Compute**	**Role: ComputeGpu**
角色名称	**名称 ：Compute**	**name: ComputeGpu**
description	**基本 Compute 节点角色**	**GPU Compute 节点角色**
ImageDefault	不适用	**overcloud-full-gpu**
HostnameFormatDefault	**-compute-**	**-computegpu-**
deprecated_nic_config_name	**compute.yaml**	**compute-gpu.yaml**

5. 通过将其添加到节点定义模板 **node.json** 或 **node.yaml**，为 overcloud 注册启用了 GPU 的 Compute 节点。有关更多信息，请参阅 *Director 安装和使用* 指南中的 为 overcloud 注册节点。

6. 检查节点硬件：

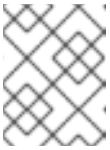
```
(undercloud)$ openstack overcloud node introspect --all-manageable \
 --provide
```

如需更多信息，请参阅 *Director 安装和使用指南中的* 创建裸机节点硬件清单。

7. 为 vGPU Compute 节点创建 **compute-vgpu-nvidia** overcloud 类别：

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

- 将 **<ram_size_mb>** 替换为裸机节点的 RAM，以 MB 为单位。
- 将 **<disk_size_gb>** 替换为裸机节点中的磁盘大小（以 GB 为单位）。
- 将 **<no_vcpus>** 替换为裸机节点中的 CPU 数量。



注意

这些属性不可用于调度实例。但是，计算调度程序使用磁盘大小来确定根分区大小。

8. 使用自定义 GPU 资源类标记您要为 GPU 工作负载指定的每个裸机节点：

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

将 **<node** > 替换为裸机节点的 ID。

9. 将 **compute-vgpu-nvidia** 类别与自定义 GPU 资源类关联：

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_GPU=1 \
  compute-vgpu-nvidia
```

要确定与 Bare Metal 服务节点的资源类对应的自定义资源类的名称，请将资源类转换为大写，请将所有 punctuation 替换为下划线，并使用 **CUSTOM_** 前缀。

注意

类别只能请求一个裸机资源类实例。

10. 设置以下类别属性，以防止计算调度程序使用裸机类别属性来调度实例：

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

11. 要验证角色是否已创建，请输入以下命令：

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

14.2.3. 为 vGPU 配置 Compute 节点并部署 overcloud

您需要检索并分配与环境中物理 GPU 设备对应的 vGPU 类型，并准备环境文件来为 vGPU 配置 Compute 节点。

流程

1. 在临时 Compute 节点上安装 Red Hat Enterprise Linux 和 NVIDIA GRID 驱动程序并启动节点。
2. 在 Compute 节点上，找到您要启用的物理 GPU 设备的 vGPU 类型。对于 libvirt，虚拟 GPU 是介质设备或 **mdev** 类型设备。要发现支持的 **mdev** 设备，请输入以下命令：

```
[root@overcloud-computegpu-0 ~]# ls
/sys/class/mdev_bus/0000\:06\:00.0/mdev_supported_types/
nvidia-11  nvidia-12  nvidia-13  nvidia-14  nvidia-15  nvidia-16  nvidia-17  nvidia-18  nvidia-19
nvidia-20  nvidia-21  nvidia-210  nvidia-22

[root@overcloud-computegpu-0 ~]# cat
/sys/class/mdev_bus/0000\:06\:00.0/mdev_supported_types/nvidia-18/description
num_heads=4, frl_config=60, framebuffer=2048M, max_resolution=4096x2160,
max_instance=4
```

3. 在 **network-environment.yaml** 文件中注册 **ComputeGpu** 角色的 **Net::SoftwareConfig** ：

```
resource_registry:
```

```
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::ComputeGpu::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute-gpu.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
```

4. 在 **node-info.yaml** 文件中添加以下参数，以指定 GPU Compute 节点的数量，以及用于 GPU 设计的 Compute 节点的类别：

```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeFlavor: compute
  OvercloudComputeGpuFlavor: compute-vgpu-nvidia
  ControllerCount: 1
  ComputeCount: 0
  ComputeGpuCount: 1
```

5. 创建 **gpu.yaml** 文件以指定 GPU 设备的 vGPU 类型：

```
parameter_defaults:
  ComputeGpuExtraConfig:
    nova::compute::vgpu::enabled_vgpu_types:
      - nvidia-18
```

注意

每个物理 GPU 只支持一个虚拟 GPU 类型。如果您在此属性中指定多个 vGPU 类型，则只使用第一个类型。

6. 保存对 Compute 环境文件的更新。

7. 使用其他环境文件将新角色和环境文件添加到堆栈中，并部署 overcloud：

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

14.3. 创建自定义 GPU 实例镜像

要让您的云用户创建使用虚拟 GPU (vGPU)的实例，您可以创建一个支持 vGPU 的镜像来启动实例。使用以下步骤，使用 NVIDIA GRID 客户机驱动程序和许可证文件创建自定义支持 vGPU 的实例镜像。

先决条件

- 您已配置并部署了启用了 GPU 的 Compute 节点的 overcloud。

流程

1. 以 **stack** 用户的身份登录 undercloud。

2. 查找 **overcloudrc** 凭证文件：

```
$ source ~/overcloudrc
```

3. 使用 vGPU 实例所需的硬件和软件配置集创建一个实例：

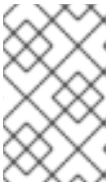
```
(overcloud)$ openstack server create --flavor <flavor> \
 --image <image> temp_vgpu_instance
```

- 将 &**lt;flavor** > 替换为具有 vGPU 实例所需的硬件配置集的类别的名称或 ID。有关创建 vGPU 类别的详情，请参阅为实例创建 vGPU 类别。
- 将 &**lt;image** > 替换为具有 vGPU 实例所需软件配置集的镜像的名称或 ID。有关下载 RHEL 云镜像的详情，请参考 镜像服务。

4. 以 cloud-user 用户身份登录到实例。

5. 按照 NVIDIA 指导在实例上创建 **网格.conf** NVIDIA GRID 许可证文件： 使用配置文件在 Linux 上 Licensing NVIDIA vGPU。

6. 在实例上安装 GPU 驱动程序。有关安装 NVIDIA 驱动程序的更多信息，请参阅在 Linux 上安装 NVIDIA vGPU 软件图形驱动程序。



注意

使用 **hw_video_model** 镜像属性来定义 GPU 驱动程序类型。如果要禁用 vGPU 实例的模拟 GPU，您可以选择 **none**。有关支持的驱动程序的更多信息，请参阅镜像元数据。

7. 创建实例的镜像快照：

```
(overcloud)$ openstack server image create \
 --name vgpu_image temp_vgpu_instance
```

8. 可选：删除实例。

14.4. 为实例创建 VGPU 类别

要让您的云用户能够为 GPU 工作负载创建实例，您可以创建一个 GPU 类别来启动 vGPU 实例，并将 vGPU 资源分配给该类别。

先决条件

- 您已使用 GPU 设计的 Compute 节点配置和部署 overcloud。

流程

1. 创建 NVIDIA GPU 类别，例如：

```
(overcloud)$ openstack flavor create --vcpus 6 \
 --ram 8192 --disk 100 m1.small-gpu
+----------------------------+-----------------------------------+
| Field                      | Value                             |
```

```
+----------------------------+-----------------------------------+
| OS-FLV-DISABLED:disabled   | False                             |
| OS-FLV-EXT-DATA:ephemeral  | 0                                 |
| disk                       | 100                               |
| id                         | a27b14dd-c42d-4084-9b6a-225555876f68 |
| name                       | m1.small-gpu                      |
| os-flavor-access:is_public | True                              |
| properties                 |                                   |
| ram                        | 8192                              |
| rxtx_factor                | 1.0                               |
| swap                       |                                   |
| vcpus                      | 6                                 |
+----------------------------+-----------------------------------+
```

2. 为您创建的类别分配一个 vGPU 资源。您只能为每个实例分配一个 vGPU。

```
(overcloud)$ openstack flavor set m1.small-gpu \
 --property "resources:VGPU=1"

(overcloud)$ openstack flavor show m1.small-gpu
+----------------------------+-----------------------------------+
| Field                      | Value                             |
+----------------------------+-----------------------------------+
| OS-FLV-DISABLED:disabled   | False                             |
| OS-FLV-EXT-DATA:ephemeral  | 0                                 |
| access_project_ids         | None                              |
| disk                       | 100                               |
| id                         | a27b14dd-c42d-4084-9b6a-225555876f68 |
| name                       | m1.small-gpu                      |
| os-flavor-access:is_public | True                              |
| properties                 | resources:VGPU='1'                |
| ram                        | 8192                              |
| rxtx_factor                | 1.0                               |
| swap                       |                                   |
| vcpus                      | 6                                 |
+----------------------------+-----------------------------------+
```

14.5. 启动 VGPU 实例

您可以为 GPU 工作负载创建启用 GPU 实例。

流程

1. 使用 GPU 类别和镜像创建实例，例如：

```
(overcloud)$ openstack server create --flavor m1.small-gpu \
 --image vgpu_image --security-group web --nic net-id=internal0 \
 --key-name lambda vgpu-instance
```

2. 以 cloud-user 用户身份登录到实例。

3. 要验证 GPU 是否可从实例访问，请从实例输入以下命令：

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

14.6. 为 GPU 设备启用 PCI 透传

您可以使用 PCI 透传将物理 PCI 设备（如图形卡）附加到实例。如果您将 PCI 透传用于设备，则实例会对该设备进行独占访问来执行任务，且设备不适用于主机。

先决条件

- **pciutils** 软件包安装在具有 PCI 卡的物理服务器中。
- GPU 设备的驱动程序必须安装在设备要传递给的实例上。因此，您需要创建一个自定义实例镜像，该镜像安装了所需的 GPU 驱动程序。有关如何使用安装 GPU 驱动程序创建自定义实例镜像的更多信息，请参阅 创建自定义 GPU 实例镜像。

流程

1. 要确定每种 passthrough 设备类型的供应商 ID 和产品 ID，请在具有 PCI 卡的物理服务器中输入以下命令：

```
# lspci -nn | grep -i <gpu_name>
```

例如，要决定 NVIDIA GPU 的厂商和产品 ID，请输入以下命令：

```
# lspci -nn | grep -i nvidia
3b:00.0 3D controller [0302]: NVIDIA Corporation TU104GL [Tesla T4] [10de:1eb8] (rev a1)
d8:00.0 3D controller [0302]: NVIDIA Corporation TU104GL [Tesla T4] [10de:1db4] (rev a1)
```

2. 要确定每个 PCI 设备是否有单根 I/O 虚拟化(SR-IOV)功能，请在具有 PCI 卡的物理服务器中输入以下命令：

```
# lspci -v -s 3b:00.0
3b:00.0 3D controller: NVIDIA Corporation TU104GL [Tesla T4] (rev a1)
        ...
        Capabilities: [bcc] Single Root I/O Virtualization (SR-IOV)
        ...
```

3. 要在 overcloud 上为 PCI 透传配置 Controller 节点，请创建一个环境文件，如 **pci_passthru_controller.yaml**。
4. 将 **PciPassthroughFilter** 添加到 **pci_passthru_controller.yaml** 中的 **NovaSchedulerDefaultFilters** 参数中：

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

5. 要为 Controller 节点上的设备指定 PCI 别名，请在 **pci_passthru_controller.yaml** 中添加以下配置：
   - 如果 PCI 设备具有 SR-IOV 功能：

```
ControllerExtraConfig:
  nova::pci::aliases:
    - name: "t4"
```

```
    product_id: "1eb8"
    vendor_id: "10de"
    device_type: "type-PF"
  - name: "v100"
    product_id: "1db4"
    vendor_id: "10de"
    device_type: "type-PF"
```

- 如果 PCI 设备没有 SR-IOV 功能：

```
ControllerExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
```

有关配置 **device_type** 字段的更多信息，请参阅 PCI passthrough 设备类型字段。

注意

如果 **nova-api** 服务在 Controller 以外的角色中运行，则将 **ControllerExtraConfig** 替换为用户角色，格式为 < **Role>ExtraConfig**。

6. 要在 overcloud 上为 PCI 透传配置 Compute 节点，请创建一个环境文件，如 **pci_passthru_compute.yaml**。

7. 要为 Compute 节点上的设备指定可用的 PCI，请将以下内容添加到 **pci_passthru_compute.yaml** 中：

```
parameter_defaults:
  NovaPCIPassthrough:
    - vendor_id: "10de"
      product_id: "1eb8"
```

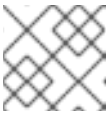
8. 您必须在 Compute 节点上为实例迁移和调整大小操作创建 PCI 别名副本。要为 Compute 节点上的设备指定 PCI 别名，请将以下内容添加到 **pci_passthru_compute.yaml** 中：

- 如果 PCI 设备具有 SR-IOV 功能：

```
ComputeExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
      device_type: "type-PF"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
      device_type: "type-PF"
```

- 如果 PCI 设备没有 SR-IOV 功能：

```
ComputeExtraConfig:
  nova::pci::aliases:
    - name: "t4"
      product_id: "1eb8"
      vendor_id: "10de"
    - name: "v100"
      product_id: "1db4"
      vendor_id: "10de"
```



注意

Compute 节点别名必须与 Controller 节点上的别名相同。

9. 要在 Compute 节点的服务器 BIOS 中启用 IOMMU 来支持 PCI 透传，请将 **KernelArgs** 参数添加到 **pci_passthru_compute.yaml** 中：

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

注意

首次将 **KernelArgs** 参数添加到角色的配置中时，overcloud 节点会自动重启。如果需要，您可以禁用节点自动重新引导，而是在每次 overcloud 部署后手动执行节点重启。如需更多信息，请参阅配置手动节点重新引导以定义 **KernelArgs**。

10. 使用其他环境文件将自定义环境文件添加到堆栈中，并部署 overcloud：

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

11. 配置类别以请求 PCI 设备。以下示例请求两个设备，每个设备供应商 ID 为 **10de**，产品 ID 为 **13f2** ：

```
# openstack flavor set m1.large \
  --property "pci_passthrough:alias"="t4:2"
```

验证

1. 使用 PCI 透传设备创建实例：

```
# openstack server create --flavor m1.large \
  --image <custom_gpu> --wait test-pci
```

将 < **custom_gpu** > 替换为安装所需 GPU 驱动程序的自定义实例镜像的名称。

2. 以云用户身份登录实例。

3. 要验证 GPU 是否可从实例访问，请从实例输入以下命令：

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

4. 要检查 NVIDIA System Management Interface 状态，从实例输入以下命令：

```
$ nvidia-smi
```

输出示例：

```
---------------------------------------------------------------------------
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================
====|
|   0  Tesla T4            Off  | 00000000:01:00.0 Off |                    0 |
| N/A   43C    P0    20W /  70W |      0MiB / 15109MiB |      0%      Default |
-------------------------------+----------------------+----------------------


---------------------------------------------------------------------------
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================
==|
|  No running processes found                                                 |
---------------------------------------------------------------------------
```

第 15 章 配置实时计算

作为云管理员，您可能需要 Compute 节点上的实例遵循低延迟策略并执行实时处理。实时 Compute 节点包含一个具有实时的内核、特定虚拟化模块和优化的部署参数，以方便实时处理要求并最小化延迟。

启用 Real-time Compute 的进程包括：

- 配置 Compute 节点的 BIOS 设置
- 使用实时内核和 Real-Time KVM (RT-KVM)内核模块构建实时镜像
- 将 **ComputeRealTime** 角色分配给 Compute 节点

对于实时计算部署工作复制的一个用例，请参阅 *Network Functions Virtualization Planning and Configuration Guide* 中的Example: Configuring OVS-DPDK with ODL and VXLAN tunnelling 部分。

注意

只有在 Red Hat Enterprise Linux 7.5 或更高版本支持实时 Compute 节点。

15.1. 为实时准备 COMPUTE 节点

在 overcloud 中部署 Real-time Compute 之前，您必须启用 Red Hat Enterprise Linux Real-Time KVM (RT-KVM)，配置 BIOS 以支持实时 overcloud 镜像。

先决条件

- 您必须将红帽认证的服务器用于 RT-KVM Compute 节点。详情请查看 Red Hat Enterprise Linux for Real Time 7 认证服务器。
- 您需要单独订阅 *Red Hat OpenStack Platform for Real Time*，才能访问 **rhel-8-for-x86_64-nfv-rpms** 存储库。有关为 undercloud 管理软件仓库和订阅的详情，请参阅 *Director 安装和使用指南*中的注册 undercloud 和附加订阅。

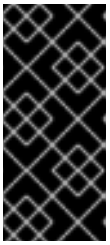
流程

1. 要构建实时 overcloud 镜像，您必须为 RT-KVM 启用 **rhel-8-for-x86_64-nfv-rpms** 存储库。要检查将从存储库中安装哪些软件包，请输入以下命令：

```
$ dnf repo-pkgs rhel-8-for-x86_64-nfv-rpms list
Loaded plugins: product-id, search-disabled-repos, subscription-manager
Available Packages
kernel-rt.x86_64                4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-rpms
kernel-rt-debug.x86_64          4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-
rpms
kernel-rt-debug-devel.x86_64    4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-
rpms
kernel-rt-debug-kvm.x86_64      4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-
rpms
kernel-rt-devel.x86_64          4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-
rpms
kernel-rt-doc.noarch            4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-rpms
kernel-rt-kvm.x86_64            4.18.0-80.7.1.rt9.153.el8_0              rhel-8-for-x86_64-nfv-
rpms
[ output omitted…]
```

2. 要为 Real-time Compute 节点构建 overcloud 镜像，请在 undercloud 上安装 **libguestfs-tools** 软件包以获取 **virt-customize** 工具：

```
(undercloud)$ sudo dnf install libguestfs-tools
```



重要

如果在 undercloud 上安装 **libguestfs-tools** 软件包，请禁用 **iscsid.socket** 以避免 undercloud 上的 **tripleo_iscsid** 服务的端口冲突：

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

3. 提取镜像：

```
(undercloud)$ tar -xf /usr/share/rhosp-director-images/overcloud-full.tar
(undercloud)$ tar -xf /usr/share/rhosp-director-images/ironic-python-agent.tar
```

4. 复制默认镜像：

```
(undercloud)$ cp overcloud-full.qcow2 overcloud-realtime-compute.qcow2
```

5. 注册镜像并配置所需的订阅：

```
(undercloud)$ virt-customize -a overcloud-realtime-compute.qcow2 --run-command
'subscription-manager register --username=<username> --password=<password>'
[  0.0] Examining the guest ...
[ 10.0] Setting a random seed
[ 10.0] Running: subscription-manager register --username=<username> --password=
<password>
[ 24.0] Finishing off
```

使用您的红帽客户帐户替换 **username** 和 **password** 的值。

有关构建实时 overcloud 镜像的常规信息，请参阅使用 virt-customize 修改 Red Hat Enterprise Linux OpenStack Platform Overcloud 镜像 的知识库文章。

6. 查找 *Red Hat OpenStack Platform for Real Time* 订阅的 SKU。SKU 可能位于已注册到 Red Hat Subscription Manager 的系统中，并使用相同的帐户和凭证：

```
$ sudo subscription-manager list
```

7. 将 *Red Hat OpenStack Platform for Real Time* 订阅附加到镜像：

```
(undercloud)$  virt-customize -a overcloud-realtime-compute.qcow2 --run-command
'subscription-manager attach --pool [subscription-pool]'
```

8. 创建在镜像中配置 **rt** 的脚本：

```
(undercloud)$ cat rt.sh
  #!/bin/bash
```

```
set -eux

subscription-manager repos --enable=[REPO_ID]
dnf -v -y --setopt=protected_packages= erase kernel.$(uname -m)
dnf -v -y install kernel-rt kernel-rt-kvm tuned-profiles-nfv-host

# END OF SCRIPT
```

9. 运行脚本来配置实时镜像：

```
(undercloud)$ virt-customize -a overcloud-realtime-compute.qcow2 -v --run rt.sh 2>&1 | tee virt-customize.log
```

10. 重新标记 SELinux：

```
(undercloud)$ virt-customize -a overcloud-realtime-compute.qcow2 --selinux-relabel
```

11. 提取 **vmlinuz** 和 **initrd**。例如：

```
(undercloud)$ mkdir image
(undercloud)$ guestmount -a overcloud-realtime-compute.qcow2 -i --ro image
(undercloud)$ cp image/boot/vmlinuz-4.18.0-80.7.1.rt9.153.el8_0.x86_64 ./overcloud-realtime-compute.vmlinuz
(undercloud)$ cp image/boot/initramfs-4.18.0-80.7.1.rt9.153.el8_0.x86_64.img ./overcloud-realtime-compute.initrd
(undercloud)$ guestunmount image
```

注意

vmlinuz 和 **initramfs** 文件名中的软件版本与内核版本不同。

12. 上传镜像：

```
(undercloud)$ openstack overcloud image upload \
 --update-existing --os-image-name
 overcloud-realtime-compute.qcow2
```

现在，您有一个实时镜像，可用于选择 Compute 节点上的 **ComputeRealTime** 可组合角色。

13. 要减少 Real-time Compute 节点上的延迟，您必须修改 Compute 节点上的 BIOS 设置。您应该在 Compute 节点 BIOS 设置中禁用以下组件的所有选项：

- 电源管理
- Hyper-Threading
- CPU 睡眠状态
- 逻辑处理器

有关这些设置的描述及其禁用它们的影响，请参阅设置 BIOS 参数。有关如何更改 BIOS 设置的详情，请查看您的硬件制造商文档。

15.2. 部署 REAL-TIME COMPUTE 角色

Red Hat OpenStack Platform (RHOSP) director 为 **ComputeRealTime** 角色提供模板，您可以使用它来部署实时 Compute 节点。您必须执行额外的步骤来指定 Compute 节点实时。

流程

1. 基于 **/usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-example.yaml** 文件，创建一个 **compute-real-time.yaml** 环境文件，该文件为 **ComputeRealTime** 角色设置参数。

```
cp /usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-example.yaml /home/stack/templates/compute-real-time.yaml
```

   该文件必须包含以下参数的值：

   - **IsolCpusList** 和 **NovaComputeCpuDedicatedSet** ：隔离 CPU 内核和虚拟 CPU 列表，以便为实时工作负载保留。这个值取决于您的实时 Compute 节点的 CPU 硬件。
   - **NovaComputeCpuSharedSet** ：要为仿真程序线程保留的主机 CPU 列表。
   - **KernelArgs** ：用来传递给 Real-time Compute 节点的内核的参数。例如，您可以使用 **default_hugepagesz=1G hugepagesz=1G hugepages=<number_of_1G_pages_to_reserve> hugepagesz=2M hugepages=<number_of_2M_pages>** 定义具有多个大小的客户机的内存要求。在这个示例中，默认大小为 1GB，但您也可以保留 2M 巨页。
   - **NovaComputeDisableIrqBalance** ：确保为 Real-time Compute 节点将此参数设置为 **true**，因为 **tuned** 服务为实时部署管理 IRQ 平衡，而不是 **irqbalance** 服务。

2. 将 **ComputeRealTime** 角色添加到您的角色数据文件并重新生成该文件。例如：

```
$ openstack overcloud roles generate -o /home/stack/templates/rt_roles_data.yaml Controller Compute ComputeRealTime
```

   此命令生成 **ComputeRealTime** 角色，其内容类似以下示例，并将 **ImageDefault** 选项设置为 **overcloud-realtime-compute**。

```
- name: ComputeRealTime
  description: |
    Compute role that is optimized for real-time behaviour. When using this role
    it is mandatory that an overcloud-realtime-compute image is available and
    the role specific parameters IsolCpusList, NovaComputeCpuDedicatedSet and
    NovaComputeCpuSharedSet are set accordingly to the hardware of the real-time compute nodes.
  CountDefault: 1
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Tenant:
      subnet: tenant_subnet
    Storage:
      subnet: storage_subnet
  HostnameFormatDefault: '%stackname%-computerealtime-%index%'
  ImageDefault: overcloud-realtime-compute
  RoleParametersDefault:
    TunedProfileName: "realtime-virtual-host"
```

```
    KernelArgs: ""      # these must be set in an environment file
    IsolCpusList: ""    # or similar according to the hardware
    NovaComputeCpuDedicatedSet: ""  # of real-time nodes
    NovaComputeCpuSharedSet: ""     #
    NovaLibvirtMemStatsPeriodSeconds: 0
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::BootParams
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::ComputeCeilometerAgent
    - OS::TripleO::Services::ComputeNeutronCorePlugin
    - OS::TripleO::Services::ComputeNeutronL3Agent
    - OS::TripleO::Services::ComputeNeutronMetadataAgent
    - OS::TripleO::Services::ComputeNeutronOvsAgent
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::IpaClient
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MetricsQdr
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronBgpVpnBagpipe
    - OS::TripleO::Services::NeutronLinuxbridgeAgent
    - OS::TripleO::Services::NeutronVppAgent
    - OS::TripleO::Services::NovaCompute
    - OS::TripleO::Services::NovaLibvirt
    - OS::TripleO::Services::NovaLibvirtGuests
    - OS::TripleO::Services::NovaMigrationTarget
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::OpenDaylightOvs
    - OS::TripleO::Services::Podman
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::SkydiveAgent
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timesync
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Vpp
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::OVNMetadataAgent
```

有关自定义角色和 **roles-data.yaml** 的常规信息，请参阅 角色。

3. 创建 **compute-realtime** 类别，以标记您要为实时工作负载指定的节点。例如：

```
$ source ~/stackrc
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 compute-realtime
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property "capabilities:profile"="compute-realtime"
compute-realtime
```

4. 使用 **compute-realtime** 配置集标记您要为实时工作负载指定的每个节点。

```
$ openstack baremetal node set --property capabilities='profile:compute-
realtime,boot_option:local' <node_uuid>
```

5. 通过创建包含以下内容的环境文件，将 **ComputeRealTime** 角色映射到 **compute-realtime** 类别：

```
parameter_defaults:
  OvercloudComputeRealTimeFlavor: compute-realtime
```

6. 使用其他环境文件将环境文件和新角色文件添加到堆栈中，并部署 overcloud：

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -r /home/stack/templates/rt~/my_roles_data.yaml \
 -e home/stack/templates/compute-real-time.yaml
```

15.3. 部署和测试场景示例

以下示例流程使用简单的单节点部署来测试环境变量和其他支持配置是否已正确设置。实际性能结果可能会因您在云中部署的节点和实例数量而异。

流程

1. 使用以下参数创建 **compute-real-time.yaml** 文件：

```
parameter_defaults:
  ComputeRealTimeParameters:
    IsolCpusList: "1"
    NovaComputeCpuDedicatedSet: "1"
    NovaComputeCpuSharedSet: "0"
    KernelArgs: "default_hugepagesz=1G hugepagesz=1G hugepages=16"
    NovaComputeDisableIrqBalance: true
```

2. 使用 **ComputeRealTime** 角色创建一个新的 **rt_roles_data.yaml** 文件：

```
$ openstack overcloud roles generate \
 -o ~/rt_roles_data.yaml Controller ComputeRealTime
```

3. 使用其他环境文件将 **compute-real-time.yaml** 和 **rt_roles_data.yaml** 添加到堆栈中，并部署 overcloud：

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

```
-e /home/stack/templates/compute-real-time.yaml
```

此命令部署一个 Controller 节点和一个 Real-time Compute 节点。

4. 登录到 Real-time Compute 节点并检查以下参数：

```
[root@overcloud-computerealtime-0 ~]# uname -a
Linux overcloud-computerealtime-0 4.18.0-80.7.1.rt9.153.el8_0.x86_64 #1 SMP PREEMPT
RT Wed Dec 13 13:37:53 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
[root@overcloud-computerealtime-0 ~]# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.18.0-80.7.1.rt9.153.el8_0.x86_64 root=UUID=45ae42d0-
58e7-44fe-b5b1-993fe97b760f ro console=tty0 crashkernel=auto console=ttyS0,115200
default_hugepagesz=1G hugepagesz=1G hugepages=16
[root@overcloud-computerealtime-0 ~]# tuned-adm active
Current active profile: realtime-virtual-host
[root@overcloud-computerealtime-0 ~]# grep ^isolated_cores /etc/tuned/realtime-virtual-host-
variables.conf
isolated_cores=1
[root@overcloud-computerealtime-0 ~]# cat /usr/lib/tuned/realtime-virtual-
host/lapic_timer_adv_ns
4000 # The returned value must not be 0
[root@overcloud-computerealtime-0 ~]# cat
/sys/module/kvm/parameters/lapic_timer_advance_ns
4000 # The returned value must not be 0
# To validate hugepages at a host level:
[root@overcloud-computerealtime-0 ~]# cat /proc/meminfo | grep -E
HugePages_Total|Hugepagesize
HugePages_Total:      64
Hugepagesize:    1048576 kB
# To validate hugepages on a per NUMA level (below example is a two NUMA compute
host):
[root@overcloud-computerealtime-0 ~]# cat
/sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
32
[root@overcloud-computerealtime-0 ~]# cat
/sys/devices/system/node/node1/hugepages/hugepages-1048576kB/nr_hugepages
32
[root@overcloud-computerealtime-0 ~]# crudini --get /var/lib/config-data/puppet-
generated/nova_libvirt/etc/nova/nova.conf compute cpu_dedicated_set
1
[root@overcloud-computerealtime-0 ~]# crudini --get /var/lib/config-data/puppet-
generated/nova_libvirt/etc/nova/nova.conf compute cpu_shared_set
0
[root@overcloud-computerealtime-0 ~]# systemctl status irqbalance
● irqbalance.service - irqbalance daemon
   Loaded: loaded (/usr/lib/systemd/system/irqbalance.service; enabled; vendor preset:
enabled)
   Active: inactive (dead) since Tue 2021-03-30 13:36:31 UTC; 2s ago
```

15.4. 启动和调优实时实例

部署和配置 Real-time Compute 节点后，您可以在这些节点上启动实时实例。您可以使用 CPU 固定、NUMA 拓扑过滤器和巨页进一步配置这些实时实例。

先决条件

- overcloud 上存在 **compute-realtime** 类别，如部署 Real-time Compute 角色 中所述。

流程

1. 启动实时实例：

```
# openstack server create  --image <rhel> \
 --flavor r1.small --nic net-id=<dpdk_net> test-rt
```

2. 可选：验证实例是否使用分配的仿真程序线程：

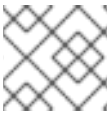
```
# virsh dumpxml <instance_id> | grep vcpu -A1
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='3'/>
  <vcpupin vcpu='2' cpuset='5'/>
  <vcpupin vcpu='3' cpuset='7'/>
  <emulatorpin cpuset='0-1'/>
  <vcpusched vcpus='2-3' scheduler='fifo'
  priority='1'/>
</cputune>
```

固定 CPU 并设置仿真程序线程策略

为确保每个 Real-time Compute 节点上有足够的 CPU 以用于实时工作负载，您需要将一个虚拟 CPU (vCPU)固定到主机上的物理 CPU (pCPU)。然后，该 vCPU 的仿真程序线程保持专用于该 pCPU。

将您的类别配置为使用专用的 CPU 策略。为此，请将 **hw:cpu_policy** 参数设置为 **dedicated** on 类别。例如：

```
# openstack flavor set --property hw:cpu_policy=dedicated 99
```



注意

确保您的资源配额有足够的 pCPU，供 Real-time Compute 节点使用。

优化网络配置

根据部署的需求，您可能需要在 **network-environment.yaml** 文件中设置参数，以便为某些实时工作负载调整网络。

若要查看为 OVS-DPDK 优化的示例配置，请参阅 *网络功能虚拟化计划和配置指南*中的配置 OVS-DPDK 参数部分。

配置巨页

建议将默认巨页大小设置为 1GB。否则，TLP 刷新可能会在 vCPU 执行中创建 jitter。有关使用巨页的常规信息，请参阅正在运行的 DPDK 应用程序 网页。

禁用性能监控单元(PMU)模拟

实例可以通过指定镜像或带有 vPMU 的类别来提供 PMU 指标。提供 PMU 指标引入了延迟。

注意

当 **NovaLibvirtCPUMode** 设置为 **host-passthrough** 时，vPMU 默认为 enabled。

如果您不需要 PMU 指标，请在用于创建实例的镜像或类别中将 PMU 属性设置为"False"来禁用 vPMU 以减少延迟：

- Image: **hw_pmu=False**
- 类型： **hw:pmu=False**

第 16 章 管理实例

作为云管理员，您可以监控和管理云上运行的实例。

16.1. 保护到实例的 VNC 控制台的连接

您可以通过将允许的 TLS 密码和最小协议版本配置为强制进入到 VNC 代理服务的客户端连接来保护到实例的 VNC 控制台的连接。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

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

3. 打开您的计算环境文件。
4. 配置用于 VNC 控制台连接实例的最小协议版本：

```
parameter_defaults:
  ...
  NovaVNCProxySSLMinimumVersion: <version>
```

将 **<version** > 替换为允许的 SSL/TLS 协议版本。设置为以下有效值之一：

- **默认** ：使用底层系统 OpenSSL 默认值。
- **tlsv1_1** ：如果您有不支持更新的版本的客户端，请使用。

注意

RHEL 8 中已弃用 TLS 1.0 和 TLS 1.1，在 RHEL 9 中不支持。

- **tlsv1_2** ：如果要配置用于 VNC 控制台连接的 SSL/TLS 密码，请使用。

5. 如果将允许的 SSL/TLS 协议版本设置为 **tlsv1_2**，请将 SSL/TLS 密码配置为用于 VNC 控制台与实例的连接：

```
parameter_defaults:
  NovaVNCProxySSLCiphers: <ciphers>
```

将 **<ciphers** > 替换为要允许的密码套件的冒号分隔列表。从 **openssl** 检索可用密码的列表。

6. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

16.2. 数据库清理

Compute 服务包含一个管理工具 **nova-manage**，可用于执行部署、升级、清理和维护相关的任务，如应用数据库模式、在升级过程中执行在线数据迁移，以及管理和清理数据库。

director 使用 cron 在 overcloud 上自动执行以下数据库管理任务：

- 通过将已删除的行从 production 表中移到影子表格，从而存档已删除的实例记录。
- 在归档完成后，从影子表格中清除已删除的行。

16.2.1. 配置数据库管理

cron 作业使用默认设置来执行数据库管理任务。默认情况下，数据库存档 cron 作业每天在 00:01 运行，数据库清除 cron 作业每天在 05:00 运行，两者都在 0 到 3600 秒之间运行。您可以使用 heat 参数根据需要修改这些设置。

流程

1. 打开您的计算环境文件。
2. 添加控制您要添加或修改的 cron 作业的 heat 参数。例如，要在归档后立即清除影子表格，请将以下参数设置为 "True" ：

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

   有关管理数据库 cron 作业的 heat 参数的完整列表，请参阅 计算服务自动化数据库管理的配置选项。

3. 保存对 Compute 环境文件的更新。
4. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

16.2.2. Compute 服务自动化数据库管理的配置选项

使用以下 heat 参数启用和修改管理数据库的自动化 cron 作业。

表 16.1. Compute (nova)服务 cron 参数

参数	描述
NovaCronArchiveDeleteAllCells	将此参数设置为 "True" 以对所有单元归档已删除的实例记录。 Default: **True**

参数	描述
NovaCronArchiveDeleteRowsAge	使用此参数根据其年龄以天为单位归档已删除的实例记录。 设置为 **0，** 以在影子表格中立即存档数据。 默认： **90**
NovaCronArchiveDeleteRowsDestination	使用此参数配置文件，以记录已删除的实例记录。 默认： **/var/log/nova/nova-rowsflush.log**
NovaCronArchiveDeleteRowsHour	使用此参数配置小时，在其中运行 cron 命令将已删除的实例记录移到另一表中。 默认： **0**
NovaCronArchiveDeleteRowsMaxDelay	在将已删除的实例记录移动到另一表之前，使用这个参数配置最大延迟（以秒为单位）。 默认： **3600**
NovaCronArchiveDeleteRowsMaxRows	使用此参数配置可移动到另一表格的最大已删除实例记录数。 默认： **1000**
NovaCronArchiveDeleteRowsMinute	使用此参数配置小时的分钟，该小时运行 cron 命令将已删除的实例记录移到另一表中。 默认： **1**
NovaCronArchiveDeleteRowsMonthday	使用此参数配置月份的某一天，以运行 cron 命令，以将已删除的实例记录移到另一表中。 默认： **Chat**（每天）
NovaCronArchiveDeleteRowsMonth	使用此参数配置在其中运行 cron 命令，以将已删除的实例记录移到另一表中。 默认： **Chat**（每月）
NovaCronArchiveDeleteRowsPurge	将此参数设置为 "True"，以在计划的归档后立即清除影子表格。 默认 ： **False**
NovaCronArchiveDeleteRowsUntilComplete	将此参数设置为 "True" 以继续将已删除的实例记录移动到另一表中，直到所有记录都移动为止。 Default: **True**

参数	描述
NovaCronArchiveDeleteRowsUser	使用此参数配置拥有归档已删除实例记录的 crontab 的用户，以及可访问 crontab 使用的日志文件。 默认 ： **nova**
NovaCronArchiveDeleteRowsWeekday	使用此参数配置每周几天，以运行 cron 命令，以将已删除的实例记录移到另一表中。 默认： **Chat**（每天）
NovaCronPurgeShadowTablesAge	使用此参数根据其年龄（以天为单位）清除影子表格。 设置为 **0,** 以清除比现在旧的影子表格。 默认 **：14**
NovaCronPurgeShadowTablesAllCells	将此参数设置为 "True"，以清除所有单元中的影子表格。 Default: **True**
NovaCronPurgeShadowTablesDestination	使用此参数配置用于记录清除影子表格的文件。 默认： **/var/log/nova/nova-rowspurge.log**
NovaCronPurgeShadowTablesHour	使用此参数配置小时，在其中运行 cron 命令来清除影子表格。 默认： **5**
NovaCronPurgeShadowTablesMaxDelay	在清除影子表前，使用此参数配置最大延迟（以秒为单位）。 默认： **3600**
NovaCronPurgeShadowTablesMinute	使用此参数配置小时的分钟，该小时运行 cron 命令来清除影子表格。 默认： **0**
NovaCronPurgeShadowTablesMonth	使用此参数配置在其中运行 cron 命令以清除影子表格的月。 默认： **Chat**（每月）
NovaCronPurgeShadowTablesMonthday	使用此参数配置月份的哪个天，以运行 cron 命令来清除影子表格。 默认： **Chat**（每天）

参数	描述
NovaCronPurgeShadowTablesUser	使用此参数配置拥有 crontab 的用户，该用户清除影子表格，并有权访问 crontab 使用的日志文件。 默认 ： **nova**
NovaCronPurgeShadowTablesVerbose	使用此参数在日志文件中为清除影子表格启用详细日志记录。 默认 ： **False**
NovaCronPurgeShadowTablesWeekday	使用此参数配置每周的某一天，以运行 cron 命令来清除影子表格。 默认： **Chat**（每天）

16.3. 在 COMPUTE 节点之间迁移虚拟机实例

有时，您需要将实例从一个 Compute 节点迁移到另一个 Compute 节点，以执行维护、重新平衡工作负载或替换故障或出现故障的节点。

Compute 节点维护

如果您需要临时将 Compute 节点从服务外来，以执行硬件维护或修复、内核升级和软件更新，您可以将 Compute 节点上运行的实例迁移到另一个 Compute 节点。

Compute 节点失败

如果 Compute 节点要出现故障，并且需要服务或替换它，您可以将故障 Compute 节点中的实例迁移到健康的 Compute 节点。

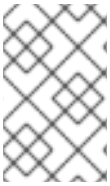
Compute 节点失败

如果 Compute 节点已经失败，您可以撤离实例。您可以使用名称、UUID、网络地址以及实例在 Compute 节点失败前的任何其他分配资源，从另一个 Compute 节点上的原始镜像重建实例。

工作负载重新平衡

您可以将一个或多个实例迁移到另一个 Compute 节点，以重新平衡工作负载。例如，您可以在 Compute 节点上整合实例以节省电源，将实例迁移到物理地连接到其他联网资源的 Compute 节点，以减少延迟或跨 Compute 节点分发实例以避免热点并增加弹性。

director 配置所有 Compute 节点以提供安全迁移。所有 Compute 节点还需要一个共享的 SSH 密钥，以便每个主机的用户在迁移过程中能够访问其他 Compute 节点。director 使用 **OS::TripleO::Services::NovaCompute** 可组合服务创建该密钥。此可组合服务是所有 Compute 角色上默认包含的主要服务之一。有关更多信息，请参阅*高级 Overcloud 自定义*指南中的可组合服务和自定义角色。



注意

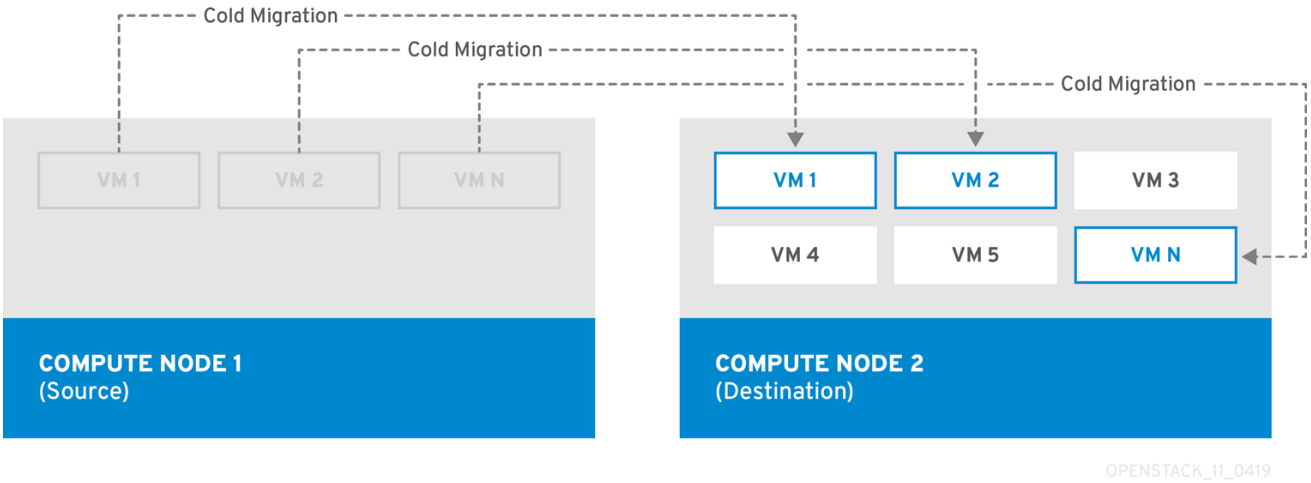
如果您有可正常工作的 Compute 节点，并且您希望复制实例用于备份目的，或者将实例复制到不同的环境中，请按照 *Director 安装和使用指南中的* 将虚拟机导入到 overcloud 中的步骤。

16.3.1. 迁移类型

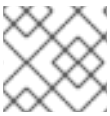
Red Hat OpenStack Platform (RHOSP)支持以下迁移类型。

冷迁移

冷迁移或非实时迁移涉及关闭正在运行的实例，然后再将其从源 Compute 节点迁移到目标 Compute 节点。



冷迁移涉及实例的一些停机时间。迁移的实例维护对同一卷和 IP 地址的访问。

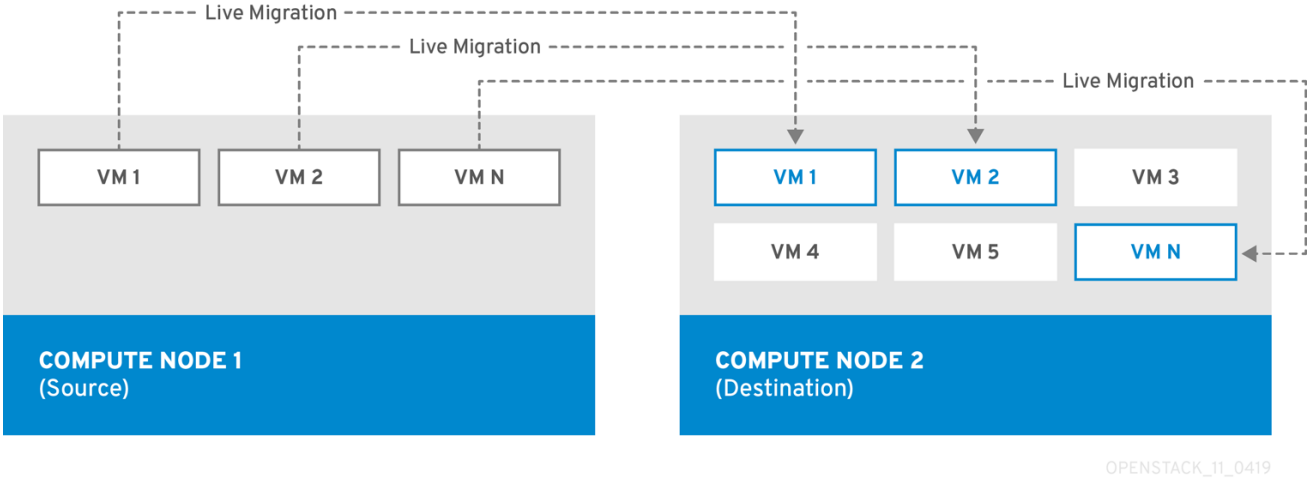


注意

冷迁移要求源和目标 Compute 节点都在运行。

实时迁移

实时迁移涉及将实例从源 Compute 节点移到目标 Compute 节点，而不将其关闭，同时保持状态一致性。



实时迁移实例涉及很少或不明显的停机时间。但是，实时迁移会影响迁移操作的持续时间的性能。因此，在迁移时，应从关键路径获取实例。

重要

实时迁移会影响正在移动的工作负载的性能。红帽不支持在实时迁移过程中增加数据包丢失、网络延迟、内存延迟或降低网络范围、内存带宽、存储 IO 或 CPU 性能。

注意

实时迁移要求源和目标 Compute 节点都在运行。

在某些情况下，实例无法使用实时迁移。如需更多信息，请参阅 *迁移限制*。

撤离

如果需要迁移实例，因为源 Compute 节点已经失败，您可以撤离实例。

16.3.2. 迁移限制

迁移限制通常会出现块迁移、配置磁盘或者一个或多个实例访问 Compute 节点上的物理硬件时。

CPU 限制

源和目标 Compute 节点必须具有相同的 CPU 架构。例如，红帽不支持将实例从 **x86_64** CPU 迁移到 **ppc64le** CPU。

不支持在不同 CPU 型号之间迁移。在某些情况下，源和目标 Compute 节点的 CPU 必须完全匹配，如使用 CPU 主机透传的实例。在所有情况下，目标节点的 CPU 功能必须是源节点上 CPU 功能的超集。

内存限制

目标 Compute 节点必须有足够的可用 RAM。内存超额订阅可能会导致迁移失败。

块迁移限制

迁移使用 Compute 节点上本地存储的磁盘的实例比迁移使用共享存储的卷支持的实例（如 Red Hat Ceph Storage）要长得多。这是因为 OpenStack Compute (nova)默认通过 control plane 网络在 Compute 节点间迁移本地磁盘块。相反，使用共享存储的卷实例（如 Red Hat Ceph Storage）不必迁移卷，因为每个 Compute 节点已经能够访问共享存储。

注意

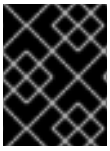
由迁移大量 RAM 的本地磁盘或实例导致的 control plane 网络中的网络拥塞可能会影响使用 control plane 网络的其他系统的性能，如 RabbitMQ。

只读驱动器迁移限制

只有在驱动器同时具有读写功能时，才支持迁移驱动器。例如，OpenStack Compute (nova)无法迁移 CD-ROM 驱动器或只读配置驱动器。但是，OpenStack Compute (nova)可以同时迁移具有读取和写入功能的驱动器，包括带有驱动器格式的配置驱动器，如 **vfat**。

实时迁移限制

在某些情况下，实时迁移实例涉及额外的限制。



重要

实时迁移会影响正在移动的工作负载的性能。红帽不支持在实时迁移期间提高数据包丢失、网络延迟、内存延迟或网络带宽、内存带宽、存储 IO 或 CPU 性能。

迁移过程中没有新的操作

要在源和目标节点上实现实例副本之间的状态一致性，RHOSP 必须防止实时迁移期间的新操作。否则，如果写入内存的速度比实时迁移复制内存状态快，实时迁移可能需要很长时间或可能永远不会结束。

使用 NUMA 的 CPU 固定

Compute 配置中的 **NovaSchedulerDefaultFilters** 参数必须包含值 **AggregateInstanceExtraSpecsFilter** 和 **NUMATopologyFilter**。

多单元云

在多单元云中，可以将实例实时迁移到同一单元中的不同主机，但不能跨单元格迁移。

浮动实例

在实时迁移浮动实例时，如果目标 Compute 节点上的 **NovaComputeCpuSharedSet** 配置与源 Compute 节点上的 **NovaComputeCpuSharedSet** 的配置不同，则实例不会分配给为目标 Compute 节点上的共享（未固定）实例配置的 CPU。因此，如果您需要实时迁移浮动实例，您必须为专用（固定）和共享（未固定）实例使用相同的 CPU 映射来配置所有 Compute 节点，或者将主机聚合用于共享实例。

目标 Compute 节点容量

目标 Compute 节点必须有足够的容量来托管要迁移的实例。

SR-IOV 实时迁移

具有基于 SR-IOV 的网络接口的实例可以实时迁移。带有直接模式 SR-IOV 网络接口实时迁移实例会导致网络停机。这是因为，在迁移过程中需要分离和重新附加直接模式接口。

ML2/OVN 部署上的数据包丢失

ML2/OVN 不支持在没有数据包丢失的情况下进行实时迁移。这是因为 OVN 无法处理多个端口绑定，因此不知道何时迁移端口。

要最小化实时迁移过程中的软件包丢失，请将 ML2/OVN 部署配置为在迁移完成后声明目标主机上的实例：

```
parameter_defaults:
  ComputeExtraConfig:
    nova::config::nova_config:
      workarounds/enable_qemu_monitor_announce_self:
        value: 'True'
```

ML2/OVS 部署上的实时迁移

在实时迁移过程中，在目标主机中取消暂停虚拟机时，元数据服务可能无法使用，因为元数据服务器代理尚未生成。这种不可用是简明的。该服务很快就会可用，实时迁移可以成功。

阻止实时迁移的限制

您无法实时迁移使用以下功能的实例。

PCI 透传

QEMU/KVM 虚拟机监控程序支持将 Compute 节点上的 PCI 设备附加到实例。使用 PCI 透传为实例授予 PCI 设备独占访问，就像它们物理连接到实例的操作系统一样。但是，由于 PCI 透传涉及直接访问物理设备，QEMU/KVM 不支持使用 PCI 透传实时迁移实例。

端口资源请求

您无法实时迁移使用资源请求的端口的实例，如保证最小带宽 QoS 策略。使用以下命令检查端口是否有资源请求：

```
$ openstack port show <port_name/port_id>
```

16.3.3. 准备迁移

在迁移一个或多个实例前，您需要确定 Compute 节点名称和要迁移的实例的 ID。

流程

1. 识别源 Compute 节点主机名和目标 Compute 节点主机名：

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

2. 列出源 Compute 节点上的实例，并找到您要迁移的实例或实例的 ID：

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

将 **<source>** 替换为源 Compute 节点的名称或 ID。

3. 可选： 如果您要从源 Compute 节点迁移实例以便在节点上执行维护，则必须禁用该节点，以防止调度程序在维护期间将新实例分配给节点：

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

将 **<source** > 替换为源 Compute 节点的主机名。

现在，您已准备好执行迁移。按照 冷迁移实例或 实时迁移实例的详细步骤操作。

16.3.4. 冷迁移实例

冷迁移实例涉及停止实例并将其移动到另一个 Compute 节点。冷迁移有助于实时迁移无法进行迁移，如迁移使用 PCI 透传的实例。调度程序自动选择目标 Compute 节点。如需更多信息，请参阅 迁移限制。

流程

1. 要冷迁移实例，请输入以下命令关闭并移动实例：

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

- 将 **<instance** > 替换为要迁移的实例的名称或 ID。
- 如果迁移本地存储的卷，则指定 **--block-migration** 标记。

2. 等待迁移完成。等待实例迁移完成后，您可以检查迁移状态。如需更多信息，请参阅 检查迁移状态。

3. 检查实例的状态：

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

"VERIFY_RESIZE" 状态表示您需要确认或恢复迁移：

- 如果迁移按预期工作，请确认它：

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

将 **<instance** > 替换为要迁移的实例的名称或 ID。"ACTIVE" 状态表示实例已准备就绪。

- 如果迁移无法正常工作，请恢复它：

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

将 **<instance>** 替换为实例的名称或 ID。

4. 重启实例：

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

将 **<instance>** 替换为实例的名称或 ID。

5. 可选：如果您禁用了源 Compute 节点进行维护，您必须重新启用该节点，以便可以为其分配新实例：

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

将 **<source>** 替换为源 Compute 节点的主机名。

16.3.5. 实时迁移实例

实时迁移将实例从源 Compute 节点移到目标 Compute 节点，且停机时间最少。实时迁移可能并不适用于所有实例。如需更多信息，请参阅 迁移限制。

流程

1. 要实时迁移实例，请指定实例和目标 Compute 节点：

```
(overcloud)$ openstack server migrate <instance> --live-migration [--host <dest>] --wait
```

- 将 **<instance>** 替换为实例的名称或 ID。
- 将 **<dest>** 替换为目标 Compute 节点的名称或 ID。

注意

openstack server migrate 命令涵盖迁移使用共享存储的实例，这是默认设置。指定 **--block-migration** 标志来迁移本地存储的卷：

```
(overcloud)$ openstack server migrate <instance> --live-migration [--host <dest>] --wait --block-migration
```

2. 确认实例正在迁移：

```
(overcloud)$ openstack server show <instance>

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

3. 等待迁移完成。等待实例迁移完成后，您可以检查迁移状态。如需更多信息，请参阅 检查迁移状态。

4. 检查实例的状态，以确认迁移是否成功：

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

将 **<dest>** 替换为目标 Compute 节点的名称或 ID。

5. 可选：如果您禁用了源 Compute 节点进行维护，您必须重新启用该节点，以便可以为其分配新实例：

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

将 **<source>** 替换为源 Compute 节点的主机名。

16.3.6. 检查迁移状态

迁移涉及完成迁移前的几个状态转换。在正常运行的迁移期间，迁移状态通常会有如下变换：

1. **Queued ：** 计算服务已接受迁移实例的请求，迁移是待处理的。
2. **Preparing:** Compute 服务正在准备迁移实例。
3. **Running:** Compute 服务正在迁移实例。
4. **Post-migrating ：** 计算服务已在目标 Compute 节点上构建实例，并在源 Compute 节点上释放资源。
5. **completed ：** 计算服务已完成迁移实例，并完成从源 Compute 节点上释放资源。

流程

1. 检索实例的迁移 ID 列表：

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

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

将 **<instance>** 替换为实例的名称或 ID。

2. 显示迁移的状态：

```
$ nova server-migration-show <instance> <migration_id>
```

- 将 **<instance>** 替换为实例的名称或 ID。
- 将 **<migration_id>** 替换为迁移 ID。
  运行 **nova server-migration-show** 命令会返回以下示例输出：

```
+------------------------+--------------------------------------+
```

```
| Property              | Value                                |
+-----------------------+--------------------------------------+
| created_at            | 2017-03-08T02:53:06.000000           |
| dest_compute          | controller                           |
| dest_host             | -                                    |
| dest_node             | -                                    |
| disk_processed_bytes  | 0                                    |
| disk_remaining_bytes  | 0                                    |
| disk_total_bytes      | 0                                    |
| id                    | 2                                    |
| memory_processed_bytes | 65502513                            |
| memory_remaining_bytes | 786427904                           |
| memory_total_bytes    | 1091379200                           |
| server_uuid           | d1df1b5a-70c4-4fed-98b7-423362f2c47c |
| source_compute        | compute2                             |
| source_node           | -                                    |
| status                | running                              |
| updated_at            | 2017-03-08T02:53:47.000000           |
+-----------------------+--------------------------------------+
```

提示

OpenStack Compute 服务根据要复制的剩余内存字节数来测量迁移的进度。如果这个数字没有随着时间的推移减少，则迁移可能无法完成，计算服务可能会中止它。

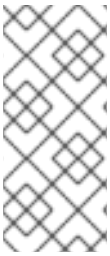
有时，实例迁移可能需要很长时间或遇到错误。如需更多信息，请参阅 迁移故障排除。

16.3.7. 清空实例

如果要将实例从死或关闭的 Compute 节点移动到同一环境中的新主机，您可以撤离它。

撤离过程会破坏原始实例，并使用原始镜像、实例名称、UUID、网络地址以及原始实例为其分配的任何其他资源在另一个 Compute 节点上重建它。

如果实例使用共享存储，则在撤离过程中不会重建实例根磁盘，因为磁盘可以被目标 Compute 节点访问。如果实例没有使用共享存储，那么目标 Compute 节点上也会重建实例根磁盘。



注意

- 您只能在 Compute 节点被隔离时执行撤离，API 报告 Compute 节点的状态为 "down" 或 "forced-down"。如果 Compute 节点没有报告为 "down" 或 "forced-down"，则 **evacuate** 命令会失败。
- 要执行撤离，您必须是云管理员。

16.3.7.1. 清空一个实例

您可以一次撤离实例。

流程

1. 确认实例没有运行：

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

- 将 **<node>** 替换为托管实例的 Compute 节点的名称或 UUID。

2. 确认主机 Compute 节点已被隔离或关闭：

```
(overcloud)[stack@director ~]$ openstack baremetal node show <node>
```

- 将 **<node>** 替换为托管要撤离的 Compute 节点的名称或 UUID。要执行撤离，Compute 节点必须处于 **down** 或 **forced-down** 状态。

3. 禁用 Compute 节点：

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

- 将 **<node>** 替换为要从中撤离实例的 Compute 节点的名称。
- 将 **<disable_host_reason>** 替换为您禁用 Compute 节点的详情。

4. 撤离实例：

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

- 可选：将 **<pass>** 替换为访问撤离实例所需的管理密码。如果没有指定密码，则生成随机密码并在撤离完成后输出。

注意

只有在临时实例磁盘存储在本地虚拟机监控程序磁盘上时，才会更改密码。如果实例托管在共享存储上，或者实例附加了 Block Storage 卷，且没有显示错误消息来告知您密码没有改变，则不会更改密码。

- 将 **<instance>** 替换为要撤离的实例的名称或 ID。
- 可选：将 **<dest>** 替换为要撤离实例的 Compute 节点的名称。如果没有指定目标 Compute 节点，则计算调度程序会为您选择一个。您可以使用以下命令查找可能的 Compute 节点：

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

5. 可选：在恢复时启用 Compute 节点：

```
(overcloud)[stack@director ~]$ openstack compute service set \
 <node> nova-compute --enable
```

- 将 **<node>** 替换为要启用的 Compute 节点的名称。

16.3.7.2. 清空主机上的所有实例

您可以撤离指定 Compute 节点上的所有实例。

流程

1. 确认要撤离的实例没有运行：

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

- 将 **<node** > 替换为托管要撤离的 Compute 节点的名称或 UUID。

2. 确认主机 Compute 节点已被隔离或关闭：

```
(overcloud)[stack@director ~]$ openstack baremetal node show <node>
```

- 将 **<node** > 替换为托管要撤离的 Compute 节点的名称或 UUID。要执行撤离，Compute 节点必须处于 **down** 或 **forced-down** 状态。

3. 禁用 Compute 节点：

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

- 将 **<node** > 替换为要从中撤离实例的 Compute 节点的名称。
- 将 **<disable_host_reason** > 替换为您禁用 Compute 节点的详情。

4. 撤离指定 Compute 节点上的所有实例：

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

- 可选：将 **<dest** > 替换为目标 Compute 节点的名称，以撤离实例。如果没有指定目的地，计算调度程序会为您选择一个。您可以使用以下命令查找可能的 Compute 节点：

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

- 将 **<node** > 替换为要从中撤离实例的 Compute 节点的名称。

5. 可选：在恢复时启用 Compute 节点：

```
(overcloud)[stack@director ~]$ openstack compute service set \
 <node> nova-compute --enable
```

- 将 **<node** > 替换为要启用的 Compute 节点的名称。

16.3.8. 迁移故障排除

实例迁移过程中可能会出现以下问题：

- 迁移过程遇到错误。
- 迁移过程永远不结束。
- 迁移后实例的性能会降低。

16.3.8.1. 迁移过程中出错

以下问题可使迁移操作进入错误状态：

- 使用不同版本的 Red Hat OpenStack Platform (RHOSP)运行集群。
- 指定无法找到的实例 ID。

- 您尝试迁移的实例 处于错误状态。
- Compute 服务正在关闭。
- 发生争用情形。
- 实时迁移进入失败状态。

当实时迁移进入失败状态时，通常会随之进入错误状态。以下常见问题可能导致失败状态：

- 目标 Compute 主机不可用。
- 发生调度程序异常。
- 由于计算资源不足，重新构建过程失败。
- 服务器组检查失败。
- 源 Compute 节点上的实例在完成迁移到目标 Compute 节点之前被删除。

16.3.8.2. 永不结束的实时迁移

实时迁移可能无法完成，这会使迁移 **持续运行**。实时迁移永不完成时，对源 Compute 节点上运行的实例的客户端请求创建的速度比计算服务将其复制到目标 Compute 节点更快。

使用以下方法之一解决这种情况：

- 中止实时迁移。
- 强制实时迁移完成。

中止实时迁移

如果实例状态变化比迁移步骤将其复制到目标节点更快，并且您不想临时暂停实例操作，您可以中止实时迁移。

流程

1. 检索实例的迁移列表：

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

   将 **<instance>** 替换为实例的名称或 ID。

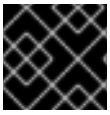
2. 中止实时迁移：

```
$ nova live-migration-abort <instance> <migration_id>
```

   - 将 **<instance>** 替换为实例的名称或 ID。
   - 将 **<migration_id>** 替换为迁移 ID。

强制实时迁移完成

如果实例状态变化比迁移步骤将其复制到目标节点更快，并且您想要临时暂停实例操作来强制迁移完成，您可以强制完成实时迁移步骤。



重要

强制完成实时迁移可能导致明显的停机时间。

流程

1. 检索实例的迁移列表：

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

将 **<instance>** 替换为实例的名称或 ID。

2. 强制实时迁移完成：

```
$ nova live-migration-force-complete <instance> <migration_id>
```

- 将 **<instance>** 替换为实例的名称或 ID。
- 将 **<migration_id>** 替换为迁移 ID。

16.3.8.3. 迁移后实例性能下降

对于使用 NUMA 拓扑的实例，源和目标 Compute 节点必须具有相同的 NUMA 拓扑和配置。目标 Compute 节点的 NUMA 拓扑必须有足够的可用资源。如果源和目标 Compute 节点之间的 NUMA 配置不相同，则当实例性能降低时，实时迁移可能会成功。例如，如果源 Compute 节点将 NIC 1 映射到 NUMA 节点 0，但目标 Compute 节点会将 NIC 1 映射到 NUMA 节点 5，之后实例可能会将流量从总线路由到带有 NUMA 节点 5 的第二个 CPU，以将流量路由到 NIC 1。这可能导致预期的行为，但性能会降低。同样，如果源 Compute 节点上的 NUMA 节点 0 有足够的可用 CPU 和 RAM，但目的地 Compute 节点上的 NUMA 节点 0 已经具有使用了一些资源的实例，则实例可能会正确运行，但性能将下降。如需更多信息，请参阅 迁移限制。