Red Hat OpenStack Platform 17.0

为实例创建配置 Compute 服务

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

Last Updated: 2025-05-10

Red Hat OpenStack Platform 17.0 为实例创建配置 Compute 服务

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

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

本指南为云管理员使用 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

第 1 章 计算服务(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。计算服务抽象其上运行的底层硬件，而不是公开底层主机平台的具体信息。例如，计算服务公开多个虚拟 CPU (vCPU)，而不是公开主机上运行的 CPU 的类型和拓扑，并允许过度使用这些 vCPU。

计算服务使用 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，以处理与 KVM 的所有交互，并允许创建虚拟机实例。要创建和置备实例，计算服务与以下 RHOSP 服务交互：

- 进行身份验证的身份(keystone)服务。
- 资源清单跟踪和选择放置服务。
- 磁盘和实例镜像的镜像服务(glance)。
- 用于调配实例在启动时连接到的虚拟或物理网络的联网(neutron)服务。

计算服务由守护进程进程和服务组成，名为 **nova channel**。以下是核心计算服务：

计算服务(nova-compute)

此服务通过对 KVM 或 QEMU hypervisor API 使用 libvirt 创建、管理和终止实例，并使用实例状态更新数据库。

计算编排器(nova-conductor)

此服务调解计算服务和数据库之间的交互，使 Compute 节点无法直接访问数据库。不要在运行 **nova-compute** 服务的节点上部署该服务。

计算调度程序(nova-scheduler)

此服务从队列获取实例请求，并确定在哪个 Compute 节点上托管该实例。

Compute 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，并为构建但不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部分信息。它由 API 数据库、编排器和计算服务使用。

消息队列

此消息传递服务供所有服务在单元格和全局服务中相互通信。

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的配置中设置 secret 关键字，以允许服务进行通信。计算元数据服务可全局运行，作为计算 API 的一部分，或者在每个单元中运行。

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 Compute 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。

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 Compute (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。根据以下优先级顺序，使用以下配置方法自定义计算服务配置：

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:
  NovaNfsEnabled: true
  NovaNfsOptions: "context=system_u:object_r:nfs_t:s0"
  NovaNfsShare: "192.0.2.254:/export/nova"
  NovaNfsVersion: "4.2"
  NovaSchedulerEnabledFilters:
    - AggregateInstanceExtraSpecsFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
```

2. **Puppet 参数** --如 **/etc/puppet/modules/nova/manifests configured**:

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

注意

仅在等同的 heat 参数不存在时才使用此方法。

3. 当不存在 heat 或 Puppet 参数时，**手动 hieradata overrides**- 用于自定义参数。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，但不是 heat 参数，请使用 Puppet 参数，而不使用手动覆盖方法。仅在没有等同的 heat 或 Puppet 参数时，才使用手动覆盖方法。

提示

按照 您要修改的参数中的指导来确定要修改的参数，以确定是否有 heat 或 Puppet 参数可用于自定义特定的配置。

有关如何配置 overcloud 服务的更多信息，请参阅 *Director 安装和使用* 指南中的 Heat 参数。

2.1. 为过度分配配置内存

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

注意

如果您的 **NovaRAMAllocationRatio** 参数设置为 < **1**，请按照 RHEL 对 swap 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 计算交换大小。

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. 将 **enable_swap.yaml** 环境文件添加到带有其他环境文件的堆栈中，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

要确定主机进程保留的 RAM 总量，您需要为每个主机进程分配足够的内存：

- 例如，在主机上运行的资源（如 OSD 消耗 3 GB 内存）。
- 托管实例所需的仿真程序开销。
- 每个实例的虚拟机监控程序。

在计算了内存的额外需求后，使用以下公式来帮助确定每个节点上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 hypervisor **开销** 替换开销。
- 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** 替换为此类资源的 RAM 量。

2.3. 计算 SWAP 大小

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

- overcommit_ratio = **NovaRAMAllocationRatio** - 1
- 最小 swap 大小(MB)= **(total_RAM * overcommit_ratio)+ RHEL_min_swap**
- recommended (maximum) swap size (MB)= **total_RAM *(overcommit_ratio + percentage_of_RAM_to_use_for_swap)**

percentage_of_RAM_to_use_for_swap 变量创建用于 QEMU 开销的缓冲，以及操作系统或主机服务消耗的任何其他资源。

例如，对于 swap 使用 25% 的可用 RAM，RAM 总量为 64GB，**NovaRAMAllocationRatio** 设置为 **1**：

- 推荐的（最大） swap 大小 = 64000 MB *(0 + 0.25)= 16000 MB

有关如何计算 **NovaReservedHostMemory** 值的详情，请参考在 Compute 节点上计算保留主机内存。

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以目标特殊工作负载（包括 NFV 和高性能计算(HPC)）来配置实例调度和放置，以获得最佳性能。

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 固定到物理 CPU。
- **仿真程序线程** ：将与实例关联的仿真程序线程转换到物理 CPU。
- **巨页** ：为普通内存(4k 页)和巨页(2 MB 或 1 GB 页)调整实例内存分配策略。

注意

如果没有 NUMA 拓扑，则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，而没有其他实例 vCPU 进程可以使用。在启用了 CPU 固定的 Compute 节点上运行的实例有一个 NUMA 拓扑。实例 NUMA 拓扑的每个 NUMA 节点映射到主机 Compute 节点上的 NUMA 节点。

您可以配置计算调度程序，以调度具有相同 Compute 节点上具有共享(floating) CPU 和具有共享(floating) CPU 的实例。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. 为需要 CPU 固定的实例创建类别。
5. 为启动使用共享或浮动 CPU 的实例创建类别。

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，并使用 CPU 固定资源类配置裸机节点，以标记 Compute 节点以进行 CPU 固定。

注意

以下流程适用于尚未调配的新 overcloud 节点。要将资源类分配给已置备的现有 overcloud 节点，您必须使用 scale down 过程取消置备节点，然后使用扩展过程使用新的资源类分配来重新置备节点。有关更多信息，请参阅 扩展 overcloud 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_cpu_pinning.yaml** 的新角色数据文件，其中包含 **Controller**、**Compute** 和 **ComputeCPUPinning** 角色，以及 overcloud 所需的任何其他角色：

```
(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
角色评论	**Role: Compute**	**角色：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. 将 overcloud 的 CPU 固定 Compute 节点添加到节点定义模板 **node.json** 或 **node.yaml** 中，为 overcloud 注册 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. 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。

8. 将 **ComputeCPUPinning** 角色添加到节点定义文件 **overcloud-baremetal-deploy.yaml** 中，并定义您要分配给节点的任何预先节点放置、资源类、网络拓扑或其他属性：

```
- name: Controller
  count: 3
- name: Compute
  count: 3
- name: ComputeCPUPinning
  count: 1
  defaults:
    resource_class: baremetal.CPU-PINNING
    network_config:
      template: /home/stack/templates/nic-config/myRoleTopology.j2 1
```

1 您可以重复使用现有的网络拓扑，或为角色创建新的自定义网络接口模板。如需更多信息，请参阅 *Director 安装和使用* 指南中的 自定义网络接口模板。如果您不使用 **network_config** 属性定义网络定义，则使用默认网络定义。

有关您可以在节点定义文件中配置节点属性的属性的更多信息，请参阅 裸机节点置备属性。如需节点定义文件示例，请参阅 节点定义文件示例。

9. 运行置备命令为您的角色置备新节点：

```
(undercloud)$ openstack overcloud node provision \
--stack <stack> \
[--network-config \]
--output /home/stack/templates/overcloud-baremetal-deployed.yaml \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- 将 **&** lt;stack> 替换为置备裸机节点的堆栈的名称。如果未指定，则默认为 **overcloud**。
- 包含 **--network-config** 可选参数，为 **cli-overcloud-node-network-config.yaml** Ansible playbook 提供网络定义。如果您不使用 **network_config** 属性定义网络定义，则使用默认网络定义。

10. 在单独的终端中监控调配进度。当置备成功后，节点状态会从 **available** 改为 **active** ：

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

11. 如果您没有使用 **--network-config** 选项运行 **provisioning 命令，请在 network-environment.yaml 文件中配置 <Role>NetworkConfigTemplate** 参数以指向 NIC 模板文件：

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputeCPUPinningNetworkConfigTemplate: /home/stack/templates/nic-
configs/<cpu_pinning_net_top>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

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。

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 节点中为主机进程保留一些 CPU 内核，以提高效率。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	Core 1	Core 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，将 Compute 节点配置为为固定实例、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 **NovaSchedulerEnabledFilters** 参数中（如果尚不存在）：

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - 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，请将以下配置添加到 **cpu_pinning.yaml** 中：

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

4. 要为共享实例保留物理 CPU 内核，请将以下配置添加到 **cpu_pinning.yaml** 中：

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

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

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

将 **<ram** > 替换为要保留 RAM 量（以 MB 为单位）。

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
```

使用列表或范围（用逗号分隔的 CPU 索引）指定 **IsolCpusList** 参数的值。

7. 使用其他环境文件将新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/overcloud-baremetal-deployed.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，您可以创建一个专用 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 **dedicated** ：

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

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

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

注意

- 如果主机没有 SMT 架构，或者有可用线程同级的 CPU 内核，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程序。
- 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. 要验证该类别创建了具有专用 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. 要验证新实例的放置是否正确，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，您可以使用共享 CPU 策略创建类别来启动实例。

先决条件

- Compute 节点配置为为共享 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. 要验证该类别是否创建了使用共享 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. 要验证新实例的放置是否正确，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. 在具有并发多线程的 Compute 节点上配置 CPU 固定(SMT)

如果 Compute 节点支持并发多线程(SMT)，则组线程在专用或共享集中同时同级。线程同级一些通用硬件，这意味着在一个线程上运行的进程可能会影响其他线程的性能。

例如，主机在带有 SMT: 0、1、2 和 3 的双核 CPU 中识别四个逻辑 CPU 内核。在这四个线程中，有两对线程同级对：

- 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 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 内核是线程同级设备：

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

以下输出表明逻辑 CPU 内核 0 和逻辑 CPU 内核 2 是同一核上的线程：

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

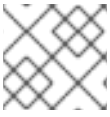
3.1.7. 其他资源

- 在 *网络功能虚拟化规划和配置指南* 中发现您的 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，仿真程序线程在与实例相同的 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单独的 CPU 上运行仿真程序线程与实例使用。



注意

为避免数据包丢失，您永远不会在 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 Compute 环境文件。
3. 要为需要 CPU 固定的实例保留物理 CPU 内核，请在 Compute 环境文件中配置 **NovaComputeCpuDedicatedSet** 参数。例如，以下配置使用 32 核 CPU 在 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 Compute 环境文件中配置 **NovaComputeCpuSharedSet** 参数。例如，以下配置使用 32 核 CPU 在 Compute 节点上设置共享 CPU：

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

注意

计算调度程序还将共享集合中的 CPU 用于在共享或浮动 CPU 上运行的实例。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 **NovaSchedulerEnabledFilters** 参数（如果尚不存在）。
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，它在专用 CPU 上运行实例的仿真程序线程，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 Emulator 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 Compute 环境文件。
2. 配置巨页内存量，以便在每个 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 **size** 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 **count** 值替换为每个 NUMA 节点的 OVS 使用的巨页数量。例如，对于 Open vSwitch 使用的 4096 个套接字内存，请将其设置为 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 期间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

- 不需要将 CPU 功能标记配置为只允许实例请求 2 MB 巨页。
- 只有主机支持 1G 巨页分配时，才能为实例分配 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**，则不需要将 **pdpe1gb** 设为 **NovaLibvirtCPUModelExtraFlags**。**pdpe1gb** 标志仅包含在 10.0.0.1_G4 和 together_G5 CPU 模型中，它不包括在 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 标记。如需更多信息，请参阅 RHOS Mitigation for MDS ("Microarchitectural Data Sampling") Security Flaws。

5. 要避免在应用 Meltdown 保护后性能会降低的问题，请配置 CPU 功能标志 **NovaLibvirtCPUModelExtraFlags** 来包含 **+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 功能标记对 OpenStack 客户机的 *Meltdown* CVE 修复的性能影响。

6. 将 **NUMATopologyFilter** 添加到 **NovaSchedulerEnabledFilters** 参数（如果尚不存在）。
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，您可以使用 **hw:mem_page_size** 额外 spec 键创建用于启动实例的类别。

先决条件

- Compute 节点配置为巨页。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，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，对于 x86_64 系统可能为 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 （常规页面）。
- **任意** - 选择最大可用巨页大小，由 libvirt 驱动程序决定。
- <pageSize>: （字符串）如果工作负载具有特定要求，则设置显式页面大小。使用整数值（以 KB 为单位）或任何标准后缀。例如：4KB、2MB、2048、1GB。

3. 要验证该类别创建了带有巨页的实例，请使用您的新类别来启动实例：

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

计算调度程序识别具有足够可用所需大小的巨页的主机，以支持实例的内存。如果调度程序无法找到具有足够页面的主机和 NUMA 节点，则请求将失败，并显示 **NoValidHost** 错误。

3.3.2. 第一次引导过程中挂载多个巨页文件夹

您可以将 Compute 服务(nova)配置为处理多个页面大小，作为第一个引导过程的一部分。首次启动节点时，第一次引导过程会将 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. 创建第一个引导模板文件 **hugepages.yaml**，该文件运行一个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 **config** 脚本执行以下任务：

a. 通过指定与 **'co?mp'** 匹配的主机名，过滤主机以在 上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 daemon-reload**，使其包含新创建的单元文件。

f. 为 2M 和 1G 页大小启用每个挂载。您可以根据需要更新此循环使其包含额外的 pagesize。

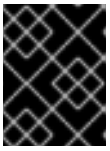
2. 可选： **/dev** 文件夹自动绑定到 **nova_compute** 和 **nova_libvirt** 容器。如果您已将不同的目的地用于巨页挂载，则需要将挂载传递给 **nova_compute** 和 **nova_libvirt** 容器：

```
parameter_defaults
  NovaComputeOptVolumes:
```

```
    - /opt/dev:/opt/dev
  NovaLibvirtOptVolumes:
    - /opt/dev:/opt/dev
```

3. 在 **~/templates/firstboot.yaml** 环境文件中，将 heat 模板注册为 **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，为实例使用文件支持的内存

您可以通过在 libvirt 内存后备目录中分配文件作为实例内存，使用文件支持的内存来扩展计算节点内存容量。您可以配置可用于实例内存的主机磁盘数量，以及实例内存文件的磁盘上的位置。

计算服务报告为放置服务配置的文件支持内存的容量，作为总系统内存容量。这允许 Compute 节点托管更多实例，而不是通常适合系统内存。

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

限制

- 您无法在启用了文件支持内存的 Compute 节点间实时迁移实例，以及没有启用文件支持内存的 Compute 节点。
- 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置到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。当文件支持的内存被使用时，保留内存对应于不为文件支持的内存设置磁盘空间。文件支持的内存将报告给放置服务作为总系统内存，RAM 用作缓存内存。

先决条件

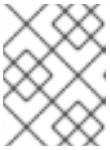
- **NovaRAMAllocationRatio** 必须设置为节点上的"1.0"，并且节点的任何主机聚合都添加到其中。
- **NovaReservedHostMemory** 必须设置为 "0"。

流程

1. 打开您的 Compute 环境文件。
2. 通过将以下参数添加到计算环境文件，将主机磁盘空间（以 MiB 为单位）提供给实例 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，则内存后备目录默认为 **/var/lib/libvirt/qemu/ram/**。



注意

您必须在位于 或默认目录位置 **/var/lib/libvirt/qemu/ram/** 的某一目录中找到您的后备存储。

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. 将更新保存到计算环境文件。
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. 在替代的后备设备上创建文件系统。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，输入以下命令在默认的 libvirt 内存后备目录中挂载 **/dev/sdb** ：

```
# 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 COMPUTE 服务存储

您可以从基础镜像创建实例，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（即实例的后端）也基于基础镜像。

您可以配置 Compute 服务，将临时磁盘磁盘数据存储在主机 Compute 节点上，或者远程存储在 NFS 共享或 Ceph 集群中。或者，您也可以配置计算服务，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，您也可以将 Compute 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来配置计算服务在计算节点上实施和管理镜像缓存的方式。

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，计算服务使用此周期来自动删除未使用的缓存的镜像。 设置为 **0，** 默认指标间隔为 60 秒（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字可能会导致预缓存性能较慢，并可能导致镜像服务上的 DDoS。 ● 将此参数设置为低数字可减少镜像服务的负载，但可能会导致运行时完成，因为预缓存是作为更连续的操作执行的。 默认： **1**

配置方法	参数	描述
puppet	**nova::compute::image_cache::remove_unused_base_images**	设置为 **True**，以在使用 **manager_interval** 配置的时间间隔自动从缓存中移除未使用的基础镜像。如果使用 **NovaImageCacheTTL** 指定的时间，则镜像定义为未使用。 Default: **True**
puppet	**nova::compute::image_cache::remove_unused_resized_minimum_age_seconds**	指定必须删除未使用调整大小的基础镜像的最小年龄（以秒为单位）。未使用的调整了大小不长的基础镜像，不会被删除。设置为 **undef** 以禁用。 默认： **3600**
puppet	**nova::compute::image_cache::subdirectory_name**	指定存储缓存镜像的文件夹名称，相对于 **$instances_path**。 默认： **_base**
heat	**NovaImageCacheTTL**	指定当 Compute 节点上的任何实例不再使用时，Compute 服务应该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这个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来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)不支持实例磁盘的 LVM 镜像类型。因此，**[libvirt]/volume_clear** 配置选项（在实例被删除时擦除临时磁盘）不被支持，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** 对于新的大型磁盘，**ext4** 格式提供比 ext3 的初始化速度要快得多。 默认：**ext4**
puppet	**nova::compute::force_raw_images**	设置为 **True**，将非原始缓存基础镜像转换为 raw 格式。raw 镜像格式使用的空间比其他镜像格式多，如 qcow2。非原始镜像格式使用更多 CPU 进行压缩。当设置为 **False** 时，计算服务在压缩过程中从基础镜像中删除所有压缩，以避免 CPU 瓶颈。如果您的系统带有较慢的 I/O 或低可用空间，以降低输入带宽，则设置为 **False**。 Default: **True**
puppet	**nova::compute::use_cow_images**	设置为 **True**，为实例磁盘以 qcow2 格式使用 CoW (Copy on Write)镜像。使用 CoW 时，根据后备存储和主机缓存，每个实例可以在自己的副本上运行更好的并发性。 设置为 **False** 以使用 raw 格式。原始格式在磁盘镜像的常用部分中使用更多空间。 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： • **无** 存储在实例启动时置备。 • **空间** - 计算服务通过在实例磁盘镜像上运行 **fallocate (1)**，在实例启动时完全分配存储。这可减少 CPU 开销和文件碎片，提高 I/O 性能，并帮助保证所需的磁盘空间。 默认：**none**

配置方法	参数	描述
hieradata override	**DEFAULT/resize_fs_using_block_device**	设置为 **True**，以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，因为它启用直接挂载因为安全原因可能禁用的镜像。 Default: **False**
hieradata override	**[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 不支持实例磁盘的 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**，则使用 **_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 **use_cow_images** 设为 **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** Default: **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 **/var/lib/nova/instances** 中。您可以为计算服务配置 NFS 存储后端，将这些实例文件存储在共享的 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 支持备注。

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，请将以下配置添加到 **nfs_instance_disk_backend.yaml** 中：

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

将 **<nfs_share** > 替换为要为实例文件存储挂载的 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，则同时使用双引号和单引号，例如 **'"[fdd0::1]:/export/nova'** "。

5. 可选：当启用 NFS 后端存储时，NFS 存储的默认挂载 SELinux 上下文为 **'context=system_u:object_r:nfs_t:s0'**。在 NFS 实例文件存储挂载点的挂载选项中添加以下参数：

```
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** 手册页：

```
$ man 8 mount.
```

6. 将更新保存到环境文件。
7. 使用其他环境文件将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. 直接从 RED HAT CEPH RADOS BLOCK DEVICE (RBD)配置镜像下载

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 Red Hat Ceph RADOS 块设备(RBD)。
- 计算服务将基于文件的本地临时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 Compute 环境文件。
3. 要直接从 RBD 后端下载镜像，请在 Compute 环境文件中添加以下配置：

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

4. 可选：如果镜像服务被配置为使用多个 Red Hat Ceph Storage 后端，请在 Compute 环境文件中添加以下配置，以识别 RBD 后端来下载镜像：

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

   将 **<rbd_backend_id** > 替换为在 **GlanceMultistoreConfig** 配置中用于指定后端的 ID，如 **rbd2_store**。

5. 将以下配置添加到 Compute 环境文件中，以指定镜像服务 RBD 后端，计算服务等待连接到镜像服务 RBD 后端（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. 要验证计算服务直接从 RBD 下载镜像，请创建一个实例，然后检查实例调试日志中的条目"Attempting to export RBD 镜像："。

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，实例会保留对执行任务的设备的专用访问权限，且设备对主机不可用。

重要

将 PCI 透传与路由供应商网络搭配使用

计算服务不支持跨越多个提供商网络的单一网络。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，如果您使用路由的供应商网络，则必须在所有 Compute 节点间使用相同的 **physical_network** 名称。

如果您将路由供应商网络与 VLAN 或扁平网络一起使用，则必须在所有片段中使用相同的 **physical_network** 名称。然后，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：

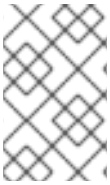
1. 为 PCI 透传指定 Compute 节点。
2. 为具有所需 PCI 设备的 PCI 透传配置 Compute 节点。
3. 部署 overcloud。
4. 创建用于启动附加 PCI 设备的实例的类别。

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，并使用 PCI 透传资源类配置裸机节点，以标记 PCI 透传的 Compute 节点。



注意

以下流程适用于尚未调配的新 overcloud 节点。要将资源类分配给已置备的现有 overcloud 节点，您必须使用 scale down 过程取消置备节点，然后使用扩展过程使用新的资源类分配来重新置备节点。有关更多信息，请参阅 扩展 overcloud 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 **ComputePCI** 角色，以及 overcloud 所需的任何其他角色：

```
(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
角色评论	**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. 将 overcloud 的 PCI 透传 Compute 节点添加到节点定义模板 **node.json** 或 **node.yaml** 中，为 overcloud 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. 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。

8. 将 **ComputePCI** 角色添加到节点定义文件 **overcloud-baremetal-deploy.yaml** 中，并定义您要分配给节点的任何预先节点放置、资源类、网络拓扑或其他属性：

```
- name: Controller
  count: 3
- name: Compute
  count: 3
- name: ComputePCI
  count: 1
  defaults:
    resource_class: baremetal.PCI-PASSTHROUGH
    network_config:
      template: /home/stack/templates/nic-config/myRoleTopology.j2 1
```

1 您可以重复使用现有的网络拓扑，或为角色创建新的自定义网络接口模板。如需更多信息，请参阅 *Director 安装和使用* 指南中的 自定义网络接口模板。如果您不使用 **network_config** 属性定义网络定义，则使用默认网络定义。

有关您可以在节点定义文件中配置节点属性的属性的更多信息，请参阅 裸机节点置备属性。如需节点定义文件示例，请参阅 节点定义文件示例。

9. 运行置备命令为您的角色置备新节点：

```
(undercloud)$ openstack overcloud node provision \
--stack <stack> \
[--network-config \]
--output /home/stack/templates/overcloud-baremetal-deployed.yaml \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- 将 **&** lt;stack> 替换为置备裸机节点的堆栈的名称。如果未指定，则默认为 **overcloud**。
- 包含 **--network-config** 可选参数，为 **cli-overcloud-node-network-config.yaml** Ansible playbook 提供网络定义。如果您不使用 **network_config** 属性定义网络定义，则使用默认网络定义。

10. 在单独的终端中监控调配进度。当置备成功后，节点状态会从 **available** 改为 **active** ：

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

11. 如果您没有使用 **--network-config** 选项运行 **provisioning 命令，请在 network-environment.yaml 文件中配置 <Role>NetworkConfigTemplate** 参数以指向 NIC 模板文件：

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputePCINetworkConfigTemplate: /home/stack/templates/nic-
configs/<pci_passthrough_net_top>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

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

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. 创建一个环境文件，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 **NovaSchedulerEnabledFilters** 参数中：

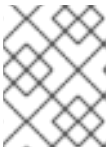
```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
```

```
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - NUMATopologyFilter
```

3. 要为 Controller 节点上的设备指定 PCI 别名，请将以下配置添加到 **pci_passthrough_controller.yaml** 中：

```
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 device type 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**。

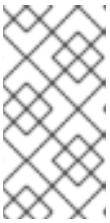
6. 要为 Compute 节点上的设备指定可用的 PCI，请使用 **vendor_id** 和 **product_id** 选项将所有匹配的 PCI 设备添加到可用于透传到实例的 PCI 设备池中。例如，要将所有 Intel® 以太网控制器 X710 设备添加到可用于透传到实例的 PCI 设备池中，请将以下配置添加到 **pci_passthrough_compute.yaml** 中：

```
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**的指南。

7. 您必须在 Compute 节点上为实例迁移和调整大小操作创建 PCI 别名的副本。要为 PCI passthrough 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，如果您在 **pci_passthrough_controller.yaml** 中将 **numa_affinity** 添加到 **nova::pci::aliases** 中，还必须将其添加到 **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 以支持 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，请将 **KernelArgs** 设置为 **"amd_iommu=on iommu=pt** "。

注意

当您第一次将 **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启，而是在每次 overcloud 部署后手动执行节点重新引导。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] \
  -r /home/stack/templates/roles_data_pci_passthrough.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/pci_passthrough_controller.yaml \
  -e /home/stack/templates/pci_passthrough_compute.yaml \
  -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
  -e /home/stack/templates/node-info.yaml
```

10. 创建并配置您的云用户可以使用的类别来请求 PCI 设备。以下示例请求两个设备，每个设备的供应商 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类别或镜像中：

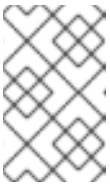
- 要使用类别覆盖默认 NUMA 关联性策略，请添加 **hw:pci_numa_affinity_policy** 属性键：

```
(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 Flavor metadata。

- 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

如果在镜像和类别上设置 NUMA 关联性策略，则属性值必须匹配。类别设置优先于镜像和默认设置。因此，只有在类别上没有设置属性时，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 PCI 设备是否可从实例访问，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。下表列出了您可以将 **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。指定这个设备类型来透传支持 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。反之，使用 **vendor_id** 和 **product_id**，因为它们更为稳定，**或使用** NIC 的地址。
- 要传递特定的物理功能(PF)，您可以使用 **address** 参数，因为 PCI 地址对每个设备是唯一的。或者，您可以使用 **product_id** 参数来传递一个 PF，但如果您有多个相同类型的 PF 时，需要指定 PF 的**地址**。
- 要传递所有虚拟功能(VF)，请只指定您要用于 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**。
- 要只传递 PF 的 VF，而不是 PF 本身，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 **dict** 映射来设置 **address** 参数的值。

字符串格式

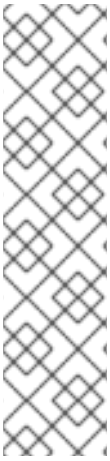
如果使用字符串指定地址，您可以包含通配符 HEKETI，如下例所示：

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

字典格式

如果您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
- 总线 - 0xFF
- slot - 0x1F
- 功能 - 0x7

计算服务支持带有 16 位地址域的 PCI 设备。计算服务忽略具有 32 位地址域的 PCI 设备。

第 6 章 配置 VDPA COMPUTE 节点以启用使用 VDPA 端口的实例

重要

该功能在此发行版本中作为*技术预览*提供，因此不享有红帽的全面支持。它只应用于测试，不应部署在生产环境中。有关技术预览功能的更多信息，请参阅覆盖范围详细信息。

VIRTIO 数据路径加速 (VDPA) 通过 VIRTIO 提供有线速数据传输。VDPA 设备在 SR-IOV 虚拟功能 (VF) 上提供 VIRTIO 抽象，它允许在实例上不使用特定于供应商的驱动程序。

注意

当您将 NIC 用作 VDPA 接口时，它必须专用于 VDPA 接口。您不能将 NIC 用于其他连接，因为您必须使用 **switchdev** 模式配置 NIC 的物理功能(PF)，并使用硬件卸载的 OVS 管理 PF。

要让您的云用户创建使用 VDPA 端口的实例，请完成以下任务：

1. 可选：为 VDPA 设计 Compute 节点。
2. 为 VDPA 配置具有所需 VDPA 驱动程序的 Compute 节点。
3. 部署 overcloud。

提示

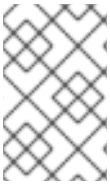
如果 VDPA 硬件有限，您还可以配置主机聚合来优化 VDPA Compute 节点上的调度。要仅在 VDPA Compute 节点上调度请求 VDPA 的实例，请创建一个具有 VDPA 硬件的 Compute 节点的主机聚合，并将计算调度程序配置为仅将 VDPA 实例放在主机聚合上。如需更多信息，请参阅通过隔离主机聚合和创建和管理主机聚合来过滤。

先决条件

- 您的 Compute 节点具有所需的 VDPA 设备和驱动程序。
- 您的 Compute 节点有 Mellanox NIC。
- 您的 overcloud 被配置为 OVS 硬件卸载。如需更多信息，请参阅配置 OVS 硬件卸载。
- 您的 overcloud 配置为使用 ML2/OVN。

6.1. 为 VDPA 设计 COMPUTE 节点

要为请求 VIRTIO 数据路径加速 (VDPA) 接口的实例指定计算节点，请创建一个新的角色文件来配置 VDPA 角色，并使用 VDPA 资源类配置裸机节点，以标记 VDPA 的计算节点。



注意

以下流程适用于尚未调配的新 overcloud 节点。要将资源类分配给已调配的现有 overcloud 节点，请缩减 overcloud 以取消置备节点，然后扩展 overcloud，以使用新的资源类分配来重新置备节点。有关更多信息，请参阅 扩展 overcloud 节点。

流程

1. 以 **stack** 用户身份登录 undercloud 主机。
2. 查找 **stackrc** undercloud 凭证文件：

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

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

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

4. 为 VDPA 角色更新 **roles_data_vdpa.yaml** 文件：

```
###############################################################################
####
# Role: ComputeVdpa                                                           #
###############################################################################
####
- name: ComputeVdpa
  description: |
    VDPA Compute Node role
  CountDefault: 1
  # Create external Neutron bridge
  tags:
    - compute
    - external_bridge
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Tenant:
      subnet: tenant_subnet
    Storage:
      subnet: storage_subnet
  HostnameFormatDefault: '%stackname%-computevdpa-%index%'
  deprecated_nic_config_name: compute-vdpa.yaml
```

5. 将 overcloud 的 VDPA Compute 节点添加到节点定义模板中，注册它们：**node.json** 或 **node.yaml**。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. 使用自定义 VDPA 资源类标记您要为 VDPA 指定的每个裸机节点：

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

将 **<node>** 替换为裸机节点的名称或 UUID。

8. 将 **ComputeVdpa** 角色添加到节点定义文件 **overcloud-baremetal-deploy.yaml** 中，并定义您要分配给节点的任何预先节点放置、资源类、网络拓扑或其他属性：

```
- name: Controller
  count: 3
- name: Compute
  count: 3
- name: ComputeVdpa
  count: 1
    defaults:
      resource_class: baremetal.VDPA
      network_config:
        template: /home/stack/templates/nic-config/<role_topology_file>
```

   - 将 **<role_topology_file>** 替换为用于 ComputeVdpa 角色的拓扑文件的名称，如 **myRoleTopology.j2**。您可以重复使用现有的网络拓扑，或为角色创建新的自定义网络接口模板。如需更多信息，请参阅 *Director 安装和使用* 指南中的 自定义网络接口模板。要使用默认网络定义设置，请不要在角色定义中包含 **network_config**。

   有关您可以在节点定义文件中配置节点属性的属性的更多信息，请参阅裸机节点置备属性。如需节点定义文件示例，请参阅 节点定义文件示例。

9. 为您的角色置备新节点：

```
(undercloud)$ openstack overcloud node provision \
[--stack <stack>] \
[--network-config \]
--output <deployment_file> \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

   - 可选：将 **<stack>** 替换为置备裸机节点的堆栈的名称。默认为 **overcloud**。
   - 可选：包含 **--network-config** 可选参数，为 **cli-overcloud-node-network-config.yaml** Ansible playbook 提供网络定义。如果您使用 **network_config** 属性在节点定义文件中未定义网络定义，则使用默认网络定义。
   - 将 **<deployment_file>** 替换为用于部署命令生成的 heat 环境文件的名称，如 **/home/stack/templates/overcloud-baremetal-deployed.yaml**。

10. 在单独的终端中监控调配进度。当置备成功后，节点状态会从 **available** 改为 **active** ：

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

11. 如果您在没有 **--network-config** 选项运行 provisioning 命令，请在 **network-environment.yaml** 文件中配置 **<Role>NetworkConfigTemplate** 参数以指向 NIC 模板文件：

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputeVdpaNetworkConfigTemplate: /home/stack/templates/nic-
configs/<vdpa_net_top>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

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

6.2. 配置 VDPA COMPUTE 节点

要让您的云用户创建使用 VIRTIO 数据路径加速 (VDPA) 端口的实例，请配置具有 VDPA 设备的 Compute 节点。

流程

1. 创建一个新的 Compute 环境文件，用于配置 VDPA Compute 节点，如 **vdpa_compute.yaml**。
2. 将 **PciPassthroughFilter** 和 **NUMATopologyFilter** 添加到 **vdpa_compute.yaml** 中的 **NovaSchedulerDefaultFilters** 参数中：

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

3. 将 **NovaPCIPassthrough** 参数添加到 **vdpa_compute.yaml**，以指定 Compute 节点上的 VDPA 设备的可用 PCI。例如，要将 NVIDIA® ConnectX®-6 Dx 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 **vdpa_compute.yaml** 中：

```
parameter_defaults:
  ...
  ComputeVdpaParameters:
    NovaPCIPassthrough:
    - vendor_id: "15b3"
      product_id: "101d"
      address: "06:00.0"
      physical_network: "tenant"
    - vendor_id: "15b3"
      product_id: "101d"
      address: "06:00.1"
      physical_network: "tenant"
```

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

4. 通过将 **KernelArgs** 参数添加到 **vdpa_compute.yaml**，在每个 Compute 节点 BIOS 中启用 input-output 内存管理单元 (IOMMU)。例如，使用以下 **KernalArgs** 设置来启用 Intel 公司 IOMMU：

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

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

注意

第一次将 **KernelArgs** 参数添加到角色的配置中时，overcloud 节点会在 overcloud 部署期间自动重新引导。如果需要，您可以禁用节点自动重启，而是在每次 overcloud 部署后手动执行节点重新引导。如需更多信息，请参阅配置手动节点重新引导以定义 **KernelArgs**。

5. 打开网络环境文件，并添加以下配置来定义物理网络：

```
parameter_defaults:
  ...
  NeutronBridgeMappings:
  - <bridge_map_1>
  - <bridge_map_n>
  NeutronTunnelTypes: '<tunnel_types>'
  NeutronNetworkType: '<network_types>'
  NeutronNetworkVLANRanges:
  - <network_vlan_range_1>
  - <network_vlan_range_n>
```

- 将 **<bridge_map_1>** 以及所有网桥映射（直到 **<bridge_map_n>**）替换为您要用于 VDPA 网桥的逻辑网桥映射。例如，**tenant:br-tenant**.
- 将 **<tunnel_types>** 替换为项目网络的隧道类型的逗号分隔列表。例如，**geneve**。
- 将 **<network_types>** 替换为 Networking 服务 (neutron) 的项目网络类型的逗号分隔列表。在所有可用网络用尽前，系统会使用您指定的第一种类型，然后会使用下一个类型。例如，**geneve,vlan**。
- 将 **<network_vlan_range_1>**，所有物理网络和 VLAN 范围（直到 **<network_vlan_range_n>**） 替换为您要支持的 ML2 和 OVN VLAN 映射范围。例如，**datacentre:1:1000,tenant:100:299**。

6. 打开您的网络接口模板，并添加以下配置，将 VDPA 支持的网络接口指定为 OVN 网桥的成员：

```
- type: ovn_bridge
  name: br-tenant
  members:
    - type: sriov_pf
      name: enp6s0f0
      numvfs: 8
      use_dhcp: false
      vdpa: true
      link_mode: switchdev
    - type: sriov_pf
      name: enp6s0f1
      numvfs: 8
      use_dhcp: false
      vdpa: true
      link_mode: switchdev
```

7. 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] \
```

```
-r /home/stack/templates/roles_data_vdpa.yaml \
-e /home/stack/templates/network-environment.yaml \
-e /home/stack/templates/vdpa_compute.yaml \
-e /home/stack/templates/overcloud-baremetal-deployed.yaml \
-e /home/stack/templates/node-info.yaml
```

验证

1. 使用 VDPA 设备创建实例。如需更多信息，请参阅 *创建和管理实例*指南中的创建使用 VDPA 接口的实例。
2. 以云用户身份登录实例。如需更多信息，请参阅 *创建和管理实例*指南中的连接到实例。
3. 验证 VDPA 设备是否可从实例访问：

```
$ openstack port show vdpa-port
```

第 7 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

主机聚合是指根据硬件或性能特征等属性将计算节点分组到逻辑单元。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，然后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匹配到主机聚合元数据。计算调度程序可以在启用所需的过滤器时使用此元数据来调度实例。您在主机聚合中指定的元数据将该主机的使用限制为在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单独的硬件类别。

在创建主机聚合时，您可以指定区名称。向云用户显示此名称作为他们可以选择的可用性区域。

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。
您只能将每个 Compute 节点分配给一个可用区。您可以配置一个默认可用区，当云用户没有指定区域时，将调度实例。您可以指示云用户使用具有特定功能的可用区。

7.1. 启用主机聚合上的调度

要在具有特定属性的主机聚合上调度实例，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

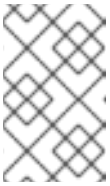
1. 打开您的 Compute 环境文件。
2. 在 **NovaSchedulerEnabledFilters** 参数中添加以下值（如果它们尚不存在）：
   - **AggregateInstanceExtraSpecsFilter**: 添加此值来根据与类别额外规格匹配的主机聚合元数据过滤 Compute 节点。

     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。

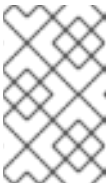
   - **AggregateImagePropertiesIsolation** ：添加此值来根据与镜像元数据属性匹配的主机聚合元数据过滤 Compute 节点。



注意

要使用镜像元数据属性过滤主机聚合元数据，主机聚合元数据键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 配置参数。

- **AvailabilityZoneFilter**: 添加此值以在启动实例时根据可用性区域过滤。



注意

您可以使用放置服务处理可用区请求，而不是使用 **AvailabilityZoneFilter** Compute 调度程序服务过滤器。如需更多信息，请参阅使用 放置服务根据可用性区域过滤。

3. 将更新保存到计算环境文件。
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
```

7.2. 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，键可以是任意字符串，如 **ssd=true**。如果使用 **AggregateImagePropertiesIsolation** 过滤器，则键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更多信息，请参阅镜像 配置参数。
- 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** > 替换为主机聚合的名称，以将 Compute 节点添加到其中。
- 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. 在与主机聚合上的键值对匹配的类别或镜像上设置一个或多个键值对。
   - 要在类别上设置键值对，请使用 scope **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
```

7.3. 创建可用区

作为云管理员，您可以创建一个可用性区域，云用户在创建实例时可以选择。

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. 要使现有主机聚合成为可用区，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。

7.4. 删除主机聚合

要删除主机聚合，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** > 替换为主机聚合的名称，以便从中删除 Compute 节点。
- 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，请输入以下命令来删除主机聚合：

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

7.5. 创建项目隔离主机聚合

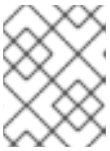
您可以创建一个仅适用于特定项目的主机聚合。只有分配给主机聚合的项目才能在主机聚合中启动实例。

注意

项目隔离使用放置服务过滤各个项目的主机聚合。这个过程取代 **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 Compute 环境文件。
2. 要在项目隔离主机聚合上调度项目实例，请在 Compute 环境文件中将 **NovaSchedulerLimitTenantsToPlacementAggregate** 参数设置为 **True**。
3. 可选：要确保只有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，没有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. 将更新保存到计算环境文件。
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。
- 使用您需要分配给主机聚合的每个项目 ID 替换 **<project_id0>**, **<project_id1>**, 以及直到 **<project_idn>** 的所有项目 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

您可以通过从 **filter_tenant_id** 元数据键省略后缀，创建仅适用于单个项目的主机聚合：

```
(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。

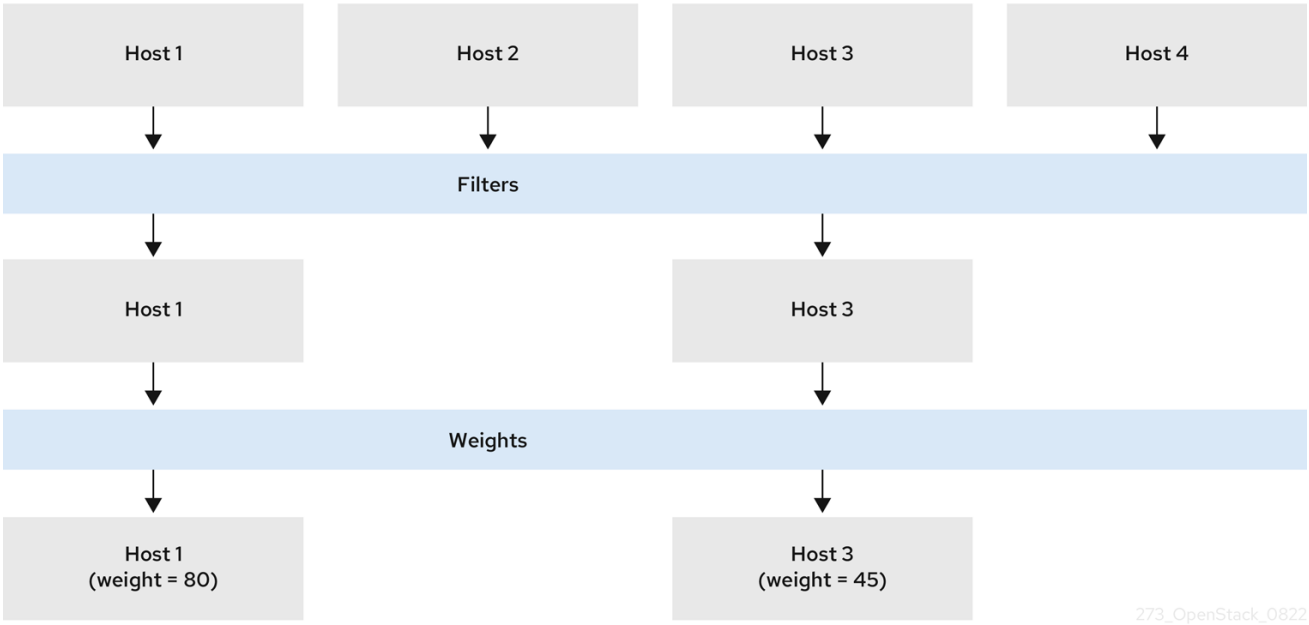
第 8 章 配置实例调度和放置

计算调度程序服务决定将实例放置到哪个 Compute 节点或主机聚合上。当计算(nova)服务收到启动或移动实例的请求时，它使用请求、类别和镜像中提供的规格来查找合适的主机。例如，类别可以指定实例需要具有的特征，如存储磁盘的类型，或者 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. **放置服务 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，放置服务会自动排除禁用的 Compute 节点。
2. **过滤器** ：由计算调度程序服务使用来确定在其上启动实例的初始计算节点集合。
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。最高的权重具有最高的优先级。

在以下示意图中，主机 1 和 3 在过滤后有资格。主机 1 具有最高的权重，因此具有调度最高的优先级。



8.1. 使用放置服务进行过滤

计算服务(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情况，如 Compute 节点、共享存储池或 IP 分配池，以及它们的可用数量资源，如可用的 vCPU。任何需要管理资源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定性资源到资源提供程序的映射，如资源提供程序具有的存储磁盘特征的类型。

放置服务根据放置服务资源提供程序清单和特征将 prefilters 应用到一组候选 Compute 节点。您可以基于以下条件创建 prefilters：

- 支持的镜像类型
- 遍历
- 项目或租户
- 可用区

8.1.1. 根据请求的镜像类型支持过滤

您可以排除不支持启动实例的镜像的磁盘格式的 Compute 节点。当您的环境使用 Red Hat Ceph Storage 作为临时后端时，这很有用，它不支持 QCOW2 镜像。启用此功能可确保调度程序不会向由 Red Hat Ceph Storage 支持的 Compute 节点发送使用 QCOW2 镜像启动实例的请求。

流程

1. 打开您的 Compute 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. 将更新保存到计算环境文件。
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
```

8.1.2. 根据资源供应商特征过滤

每个资源提供程序都有一组特征。特征是资源提供程序的定性方面，例如存储磁盘类型或 Intel CPU 指令集扩展。

Compute 节点报告放置服务的功能作为特征。实例可以指定它所需的这些特征，或者资源提供程序不能有的特征。计算调度程序可以使用这些特征来识别合适的 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。

8.1.2.1. 创建需要或禁止资源供应商特征的镜像

您可以创建实例镜像，可供您的云用户在具有特定特征的主机上启动实例。

流程

1. 创建新镜像：

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

2. 确定需要主机或主机聚合的特征。您可以选择现有的特征或创建新特征：
   - 要使用现有的特征，请列出现有的特征来检索特征名称：

```
(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，并且仅包含字母 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 /')
```

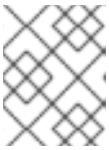
4. 检查所需主机或主机聚合的现有资源提供程序特征：

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

5. 如果所需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名称。您可以根据需要 **多次使用--trait** 选项来添加其他特征。



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供应商特征列表，然后再次设置它们，以防止被删除。

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，请将以下特征添加到镜像额外规格中：

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

7. 要过滤掉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，要防止将实例调度到支持多重附加卷的主机或主机聚合上，请将以下特征添加到镜像额外规格中：

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

8.1.2.2. 创建需要或禁止资源提供程序特征的类别

您可以创建类别，可供您的云用户用于在具有特定特征的主机上启动实例。

流程

1. 创建类别：

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

2. 确定需要主机或主机聚合的特征。您可以选择现有的特征或创建新特征：

   - 要使用现有的特征，请列出现有的特征来检索特征名称：

     ```
     (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，并且仅包含字母 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 /')
   ```

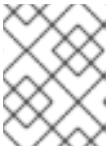
4. 检查所需主机或主机聚合的现有资源提供程序特征：

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

5. 如果所需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名称。您可以根据需要 **多次使用--trait** 选项来添加其他特征。

   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供应商特征列表，然后再次设置它们，以防止被删除。

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，请将以下特征添加到类别额外规格中：

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

7. 要过滤掉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，要防止将实例调度到支持多重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
   ```

8.1.3. 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那些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有资格在该主机聚合中的 Compute 节点上调度。

流程

1. 打开您的 Compute 环境文件。
2. 要隔离主机聚合，以仅托管其类别和镜像特征与聚合元数据匹配的实例，请在 Compute 环境文件中将 **NovaSchedulerEnableIsolatedAggregateFiltering** 参数设置为 **True**。
3. 将更新保存到计算环境文件。
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. 确定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特征：
   - 要使用现有的特征，请列出现有的特征来检索特征名称：

     ```
     (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，并且仅包含字母 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提供程序特征：

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

8. 如果所需特征还没有添加到资源提供程序中，请将现有的特征和所需的特征添加到主机聚合中每个 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名称。您可以根据需要 **多次使用--trait** 选项来添加其他特征。

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供应商特征列表，然后再次设置它们，以防止被删除。

9. 对主机聚合中的每个 Compute 节点重复步骤 6 - 8。
10. 将 trait 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：

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

8.1.4. 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，放置聚合必须存在以匹配可用区主机聚合的成员资格和 UUID。

流程

1. 打开您的 Compute 环境文件。
2. 要使用放置服务根据可用区过滤，请在 Compute 环境文件中将 **NovaSchedulerQueryPlacementForAvailabilityZone** 参数设置为 **True**。
3. 从 **NovaSchedulerEnabledFilters** 参数中删除 **AvailabilityZoneFilter** 过滤器。
4. 将更新保存到计算环境文件。
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

- 有关创建将主机聚合用作可用区的更多信息，请参阅创建可用性区域。

8.2. 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计算节点集合。

流程

1. 打开您的 Compute 环境文件。

2. 将您希望调度程序用于 **NovaSchedulerEnabledFilters** 参数的过滤器，例如：

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AggregateInstanceExtraSpecsFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
```

3. 指定用于计算每个 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：配置应用于每个 weigher 的倍数。例如，要指定 Compute 节点的可用 RAM 比其他默认 weighers 的权重高，并且计算调度程序优先选择在那些可用 RAM 的节点中具有更多可用 RAM 的 Compute 节点，请使用以下配置：

```
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，要优先选择比具有更多可用 RAM 的那些可用 RAM 的 Compute 节点，请将 **ram_weight_multiplier** 设置为 **-2.0**。

5. 将更新保存到计算环境文件。

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

- 有关可用计算调度程序服务过滤器的列表，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。

8.3. 计算调度程序过滤器

您可以在 Compute 环境文件中配置 **NovaSchedulerEnabledFilters** 参数，以指定在选择托管实例时必须应用的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 8.1. 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，则属于该主机聚合的 Compute 节点是从该镜像启动实例的候选节点。调度程序只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 配置参数。
AggregateInstanceExtraSpecsFilter	使用此过滤器匹配实例的类别额外规格中定义的具有主机聚合元数据的命名空间属性。 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。 如果任何主机聚合元数据与类别额外规格的元数据匹配，则属于该主机聚合的 Compute 节点是从该镜像启动实例的候选节点。
AggregateIoOpsFilter	使用此过滤器，通过带有 per-aggregate **filter_scheduler/max_io_ops_per_host** 值的 I/O 操作来过滤主机。如果没有找到 per-aggregate 值，则值会返回全局设置。如果主机位于多个聚合中，且找到了多个值，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。

Filter	描述
AggregateNumInstancesFilter	使用此过滤器来限制聚合中每个 Compute 节点可以托管的实例数量。您可以使用 **filter_scheduler/max_instances_per_host** 参数，配置每个实例的最大实例数量。如果没有找到 per-aggregate 值，则值会返回全局设置。如果 Compute 节点处于多个聚合，调度程序将使用最低的 **max_instances_per_host** 值。
AggregateTypeAffinityFilter	如果没有设置类别元数据键，或者使用此过滤器传递主机，或者类别聚合元数据值包含所请求类别的名称。类别元数据条目的值是可包含单个类别名称或以逗号分隔的类别名称列表的字符串，如 **m1.nano** 或 **m1.nano,m1.small**。
AllHostsFilter	使用此过滤器考虑所有可用的 Compute 节点以进行实例调度。 **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	使用此过滤器将实例的类别额外规格中定义的命名空间属性与 Compute 节点功能匹配。您必须使用 **capabilities:** namespace 为类别额外规格添加前缀。 使用 **ComputeCapabilitiesFilter** 过滤器的更有效的替代方法是在您的类别中使用 CPU 特征，这些特征被报告给放置服务。特征为 CPU 功能提供一致的命名。如需更多信息，请参阅使用资源提供程序特征过滤。
ComputeFilter	使用此过滤器传递可运行并启用的所有 Compute 节点。此过滤器应始终存在。
DifferentHostFilter	使用此过滤器，从一组特定实例在不同的 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，实例 UUID 作为值： **$ 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**

Filter	描述
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** - Corresponds 到主机的构架，如 x86、ARM 和 Power。 • **img_hv_type** - Corresponds 到 hypervisor 类型，如 KVM、QEMU、Xen 和 LXC。 • **img_hv_requested_version** - Corresponds 到 hypervisor 版本，即 Compute 服务报告。 • **hw_vm_mode** - Corresponds 到 hyperviser 类型，如 hvm、xen、uml 或 exe。 可以支持实例中包含的指定镜像属性的计算节点传递到调度程序。如需有关镜像属性的更多信息，请参阅 镜像配置参数。

IsolatedHostsFilter	使用此过滤器来仅在隔离的 Compute 节点上调度带有隔离镜像的实例。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	使用此过滤器过滤具有并发 I/O 操作的主机，该操作超过配置的 **filter_scheduler/max_io_ops_per_host**，它指定了允许在主机上运行的最大 I/O 密集型实例数。

Filter	描述
MetricsFilter	使用此过滤器将调度限制为报告使用 **metrics/weight_setting** 配置的指标的 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，您可以使用 **update_resources_interval** 配置选项增加指标数据刷新的频率。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，在支持 NUMA 的 Compute 节点上调度带有 NUMA 拓扑的实例。使用类别 **extra_specs** 和 image 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 单元的超订阅限制。
NumInstancesFilter	使用此过滤器过滤运行超过 **max_instances_per_host** 选项的实例的 Compute 节点。
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带有 PCI 设备（通常昂贵且有限）的节点，请使用此过滤器。
SameHostFilter	使用此过滤器，启用与一组特定实例在同一 Compute 节点上调度实例。要在启动实例时指定这些实例，请使用 the- **hint** 参数，其 key_**host** 与键，实例 UUID 为值： $ openstack server create --image cedef40a-ed67-4d10-800e-17455edce175 \ --flavor 1 --hint same_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 \ --hint same_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1

Filter	描述

ServerGroupAffinityFilter	使用此过滤器，在同一 Compute 节点上调度关联性服务器组中的实例。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 group 作为键，组 UUID 作为值，使用 **--hint** 参数：
	`$ openstack server create --image <image> \`
	`--flavor <flavor> \`
	`--hint group=<group_uuid> <instance_name>`

ServerGroupAntiAffinityFilter	使用此过滤器将属于不同 Compute 节点上的反关联性服务器组的实例调度。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 group 作为键，组 UUID 作为值，使用 **--hint** 参数：
	`$ openstack server create --image <image> \`
	`--flavor <flavor> \`
	`--hint group=<group_uuid> <instance_name>`

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，请使用 **--hint** 参数在启动实例时传递键 **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>`

8.4. 计算调度程序权重

每个 Compute 节点都有一个权重，调度程序可以使用它来优先调度实例。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 到 1.0 之间的值。
2. 调度程序将规范化权重乘以 weigher multiplier。

计算调度程序通过使用在 candidate Compute 节点上资源可用性的 lower 和 upper 值来计算每种资源类型的权重规范化：

- 资源最低可用性(minval)的节点被分配为 '0'。

- 资源可用性最高的节点（最大）将被分配 '1'。
- 在 minval - maxval 范围内资源可用性的节点分配了一个规范化的权重，使用以下公式计算：

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

如果所有 Compute 节点都有相同的资源可用性，则它们都规范化为 0。

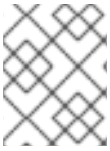
例如，调度程序计算 10 个 Compute 节点间可用 vCPU 的规范化权重，每个节点都有不同数量的可用 vCPU，如下所示：

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)) + ...
```

下表描述了权重的可用配置选项。



注意

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

表 8.2. 计算调度程序权重

配置选项	类型	描述

配置选项	类型	描述
filter_scheduler/weight_classes	字符串	使用此参数配置以下哪些属性来计算每个 Compute 节点的权重： • **nova.scheduler.weights.ram.RAMWeigher** - 邻居计算节点上可用的 RAM。 • **nova.scheduler.weights.cpu.CPUWeigher** - 邻居计算节点上可用的 CPU。 • **nova.scheduler.weights.disk.DiskWeigher** - 权衡 Compute 节点上的可用磁盘。 • **nova.scheduler.weights.metrics.MetricsWeigher** - 邻居 Compute 节点的指标。 • **nova.scheduler.weights.affinity.ServerGroupSoftAffinityWeigher** - 将 Compute 节点的几率增加到给定实例组中的其他节点。 • **nova.scheduler.weights.affinity.ServerGroupSoftAntiAffinityWeigher** - 将 Compute 节点的几率增加到给定实例组中的其他节点。 • **nova.scheduler.weights.compute.BuildFailureWeigher** - Weighs Compute 节点按最近失败的引导尝试次数。 • **nova.scheduler.weights.io_ops.IoOpsWeigher** - 根据工作复制为 Compute 节点加权重。 • **nova.scheduler.weights.pci.PCIWeigher** - Weighs Compute 节点通过其 PCI 可用性。 • **nova.scheduler.weights.cross_cell.CrossCellWeigher** - Weighs Compute 节点基于它们所在的单元，在移动实例时优先选择源单元格中的计算节点。 • **nova.scheduler.weights.all_weighers** - (Default)使用以上所有 weighers。
filter_scheduler/ram_weight_multiplier	浮点	使用此参数根据可用 RAM 指定用于 weigh 主机的倍数。 设置为正值，以首选具有更多可用 RAM 的主机，这会将实例分散到多个主机上。 设置为负值，以首选首选具有较少可用 RAM 的主机，这会尽可能地填满（堆栈）主机，然后再调度到较用的主机。 绝对值（无论是正还是负）控制 RAM weigher 相对于其他 Weighers 的强度量。 Default: 1.0 - 调度程序均匀地将实例分散到所有主机中。

配置选项	类型	描述
filter_scheduler/disk_weight_multiplier	浮点	使用此参数指定根据可用磁盘空间的 weigh 主机所使用的倍数。 设置为正值，以首选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，以首选首选具有较少可用磁盘空间的主机，这会尽可能地填满（堆栈）主机，然后再调度到较用的主机。 绝对值（无论是正还是负）控制磁盘 weigher 相对于其他 Weighers 的强度量。 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	使用此参数根据可用的 vCPU 指定用于 weigh 主机的倍数。 设置为正值，以首选具有更多可用 vCPU 的主机，这会将实例分散到多个主机上。 设置为负值，以首选首选具有较少可用 vCPU 的主机，这会尽可能地填满（堆栈）主机，然后再调度到较用的主机。 绝对值（无论是正还是负数）控制与其它 Weighers 相对的 vCPU 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	使用此参数指定用于基于主机工作负载的主机的倍数。 设置为负值，以首选具有较轻工作负载的主机，这会将工作负载分发到更多主机上。 设置为正值，以首选具有繁重工作负载的主机，后者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 I/O 操作相对于其他 Weighers 的强度量。 default: -1.0 - 调度程序将工作负载分发到更多主机上。

配置选项	类型	描述
filter_scheduler/build_failure_weight_multiplier	浮点	根据最近的构建失败，使用此参数指定用于 weigh 主机的数量。 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，以通过最近失败次数禁用计算主机。 **默认： 1000000.0**
filter_scheduler/cross_cell_move_weight_multiplier	浮点	使用此参数指定在跨单元移动期间用于权衡主机的倍数。此选项决定了在主机上放置了多少权重，它在移动实例时在同一源单元内。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，以首选实例当前运行的同一单元中的主机。设置为负值，以首选位于实例当前运行的不同单元中的主机。 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 设备数和实例请求的 PCI 设备数量指定用于 weigh 主机的数量。如果实例请求 PCI 设备，则 Compute 节点分配的 PCI 设备越高，分配给 Compute 节点的权重越高。 例如，如果有三个主机可用，一个具有单个 PCI 设备，一个具有多个 PCI 设备，另一个没有 PCI 设备，则计算调度程序会根据实例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，则调度程序应首选第一个主机，如果实例不请求 PCI 设备，则调度程序应首选第三个主机。 配置这个选项，以防止非 PCI 实例在带有 PCI 设备的主机上占用资源。 默认：1.0

配置选项	类型	描述
filter_scheduler/host_subset_size	整数	使用此参数指定过滤的主机子集的大小，以选择主机。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。调度程序忽略小于 1 的任何值，并使用 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选择同一主机，从而造成潜在的竞争条件。通过从最适合请求的 N 主机中随机选择主机，会减少冲突的可能性。但是，您设置这个值越高，所选主机可能是给定请求的最佳选择。 默认： 1
filter_scheduler/soft_affinity_weight_multiplier	正浮动点	使用此参数指定用于 weigh 主机进行组 soft-affinity 的倍数。  **注意** 创建使用此策略的组时，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 weigh 主机用于组 soft-anti-affinity 的倍数。  **注意** 创建使用此策略的组时，需要指定 microversion： $ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity <group_name> 默认：1.0

配置选项	类型	描述
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**，这会将实例分散到可能的主机上。 设置为大于 1.0 的数字，以增加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，以排列主机具有较低指标和主机中的堆栈实例的优先级。 默认：1.0
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 idle time ● **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** - Generic 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配置的 **metrics/weight_setting** 不可用的指标： ● **需要** true- Metrics。如果指标不可用，则引发异常。为避免异常，请在 **NovaSchedulerEnabledFilters** 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。

配置选项	类型	描述
metrics/weight_of_unavailable	浮点	如果任何 **metrics/weight_setting** 指标不可用时，使用此参数指定要使用的权重，并且 **metrics/required=False**。 默认： -10000.0

第 9 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内存。

您可以通过将类别 public 提供给所有项目或特定项目或域来指定哪些类别可以使用。

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。

您还可以通过与主机聚合上设置的 **extra_specs** 元数据匹配，使用类别元数据键来查找合适的主机聚合来托管该实例。要将实例调度到主机聚合上，您必须通过使用 aggregate_instance_ **extra_specs：命名空间作为 extra_specs** 键前缀来限制类别元数据。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一组默认公共类别。

表 9.1. 默认类别

名称	VCPU	RAM	Root 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会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将覆盖它们指定的镜像的属性。

9.1. 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的需求。
- 添加元数据，以强制实例使用特定的 I/O 速率或匹配主机聚合。

流程

1. 创建指定可供实例使用的基本资源的类别：

```
(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_vcpus>** 替换为为此类别创建的实例的 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，这表示它可供所有项目使用。

注意

创建后，您无法将公共类别私有。

- 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，请使用键值对设置必要属性：

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

- 将 **<**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。有关可用元数据键的列表，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>** 替换为您的类别的名称。
  例如，使用以下类别启动的实例有两个 CPU 套接字，每个 CPU 有两个 CPU：

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

9.2. 类别参数

openstack flavor create 命令具有一个位置参数 **<**flavor_name>，用于指定新类别的名称。

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 9.2. 可选类别参数

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

可选参数	描述
--disk	（必需）用于 root (/)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** 创建具有 **磁盘** 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，这意味着没有创建二级临时磁盘。临时磁盘提供链接到实例生命周期的计算机本地磁盘存储。临时磁盘不包含在任何快照中。此磁盘将被销毁，所有数据都会在实例被删除时丢失。 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，请不要在类别中指定 **交换**。 默认： 0 GB
--vcpus	（必需）实例的虚拟 CPU 数。 默认： 1
--public	该类别可供所有项目使用。默认情况下，类别是公共的，可供所有项目使用。
--private	该类别仅适用于使用-- **project 选项指定** 的项目。如果您创建专用类别，但没有项目添加到其中，则该类别仅可供云管理员使用。
--property	metadata, 或 "extra specs"，使用以下格式的键值对指定： **--property <key=value>** 重复这个选项来设置多个属性。
--project	指定可以使用 private 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。 重复这个选项，以允许访问多个项目。
--project-domain	指定可以使用 private 类别的项目域。您必须将此参数与-- **private** 选项一起使用。 重复这个选项，以允许访问多个项目域。

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

9.3. 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，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 使用量限制。

注意

用于限制实例 CPU 资源使用的额外 spec 是特定于主机的可调属性，这些属性直接传递给 libvirt，然后将限制传递给主机操作系统。因此，支持的实例 CPU 资源限值配置取决于底层主机操作系统。

有关如何在 RHOSP 部署中为 Compute 节点配置实例 CPU 资源使用情况的更多信息，请参阅 RHEL 9 文档中的了解 cgroups 以及 Libvirt 文档中的 CPU 调优。

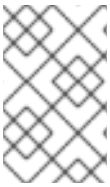
表 9.3. 资源使用量的类别元数据

键	描述
quota:cpu_shares	指定域的 CPU 时间的比例加权共享。默认为提供的默认值。计算调度程序相对于同一域中其他实例上的此属性设置的值。例如，使用 **quota:cpu_shares=2048** 配置的实例被分配为使用 **quota:cpu_shares=1024** 配置的实例加倍。
quota:cpu_period	指定以微秒为单位强制 **cpu_quota** 的时间周期。在 **cpu_period** 中，每个 vCPU 无法消耗超过运行时的 **cpu_quota**。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来调优实例磁盘性能。



注意

计算服务将以下服务质量设置应用到计算服务调配的存储，如临时存储。要调整 Block Storage (cinder)卷的性能，还必须为卷类型配置和关联服务质量(QoS)规格。如需更多信息，请参阅存储指南中的块存储服务(cinder)服务质量规格。

表 9.4. 用于磁盘调优的类别元数据

键	描述
quota:disk_read_bytes_sec	指定实例可使用的最大磁盘读取，以每秒为单位。
quota:disk_read_iops_sec	指定 IOPS 中可供实例使用的最大磁盘读取。
quota:disk_write_bytes_sec	指定实例可使用的最大磁盘写入，以每秒为单位。
quota:disk_write_iops_sec	指定 IOPS 中可供实例使用的最大磁盘写入。
quota:disk_total_bytes_sec	指定可供实例使用的最大 I/O 操作，以字节为单位。
quota:disk_total_iops_sec	指定可供实例使用的最大 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来配置实例网络流量的带宽限制。

注意

配额 :vif** swig 属性已弃用。反之，您应该使用 Networking (neutron)服务服务质量(QoS)策略。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 9.5. 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（单位为 kbps）。
quota:vif_inbound_burst	（已弃用）指定以 KB 为单位的峰值速度的最大传入流量量。
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。

键	描述
quota:vif_outbound_average	（已弃用）指定从实例传出的流量所需的平均位率（单位为 kbps）。
quota:vif_outbound_burst	（已弃用）指定可在峰值速度(KB)中突发的最大传出流量。
quota:vif_outbound_peak	（已弃用）指定实例可以发送传出流量（以 kbps 为单位）的最大速率。

硬件视频 RAM

使用下表中的 property key 来配置用于视频设备的实例 RAM 的限制。

表 9.6. 视频设备的类别元数据

键	描述
hw_video:ram_max_mb	指定用于视频设备的最大 RAM，以 MB 为单位。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 property 键，在实例上启用虚拟硬件 watchdog 设备。

表 9.7. watchdog 行为的类别元数据

键	描述

键	描述
hw:watchdog_action	指定启用虚拟硬件 watchdog 设备并设置其行为。如果实例挂起或失败，则 watchdog 设备执行配置的操作。watchdog 使用 i6300esb 设备，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： • **disabled** ：（默认）未附加该设备。 • **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，但如果实例挂起或失败时不执行任何操作。  **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。

随机数字生成器(RNG)

使用下表中的属性键在实例上启用 RNG 设备。

表 9.8. 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**
hw_rng:rate_bytes	指定实例可以从主机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 property 键，为实例启用 vPMU。

表 9.9. vPMU 的 flavor 元数据

键	描述

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 来更准确地分析和监控实例性能。对于实时工作负载，vPMU 的模拟可能会带来额外的延迟，这些延迟可能并不正常。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定义实例中处理器的拓扑。

表 9.10. 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	指定实例的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	指定用户可以使用镜像属性为每个插槽选择的最大内核数。
hw:cpu_max_threads	指定每个内核可使用镜像属性为实例选择的最大线程数。

串行端口

使用下表中的 property key 来配置每个实例的串行端口数量。

表 9.11. 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)是带有一个核心和一个线程的套接字。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 9.12. 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 之间的实例 vCPU 浮点值。 • **专用** ：将实例 vCPU 固定到一组主机 pCPU。这会创建一个实例 CPU 拓扑，它与实例固定到的 CPU 拓扑匹配。这个选项意味着 1.0 的过量使用比率。

hw:cpu_thread_policy	指定在 **hw:cpu_policy=dedicated** 时使用的 CPU 线程策略。设置为以下有效值之一：
	• **prefer**:（默认）主机可能或可能没有 SMT 架构。如果存在 SMT 架构，计算调度程序会优先选择线程同级数据。
	• **隔离** ：主机不能有 SMT 架构，或者必须模拟非SMT 构架。此策略通过请求不报告 **HW_CPU_HYPERTHREADING** 特征的主机来确保计算调度程序将实例放置到没有 SMT 的主机上。您还可以使用以下属性显式请求这个特征：
	`--property trait:HW_CPU_HYPERTHREADING=forbidden`
	如果主机没有 SMT 架构，则 Compute 服务会如预期将每个 vCPU 放置到不同的内核中。如果主机有 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，调度会失败。
	◦ **假** ：计算服务将每个 vCPU 放置到不同的物理内核中。计算服务不会将其他实例的 vCPU 放置到同一核心上。因此，保证在每个使用的内核上同级一个线程都不可用。
	• **require**: 主机必须具有 SMT 架构。此策略确保计算调度程序通过请求报告 **HW_CPU_HYPERTHREADING** 特征的主机，将实例放置到带有 SMT 的主机上。您还可以使用以下属性显式请求这个特征：
	`--property trait:HW_CPU_HYPERTHREADING=required`
	计算服务在线程同级上分配每个 vCPU。如果主机没有 SMT 架构，则不使用它。如果主机有 SMT 架构，但没有足够内核，且有可用线程同级内核，则调度会失败。

实例 PCI NUMA 关联性策略

使用下表中的 property key 创建类别，以指定 PCI 透传设备和 SR-IOV 接口的 NUMA 关联性策略。

表 9.13. PCI NUMA 关联性策略的类别元数据

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** ：计算服务创建一个实例，该实例仅当实例的其中一个 NUMA 节点与 PCI 设备关联时才会请求 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。如果不可能，计算服务将实例调度到没有与 PCI 设备关联性的 NUMA 节点上。
	• **传统** ：（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 节点的关联。
	  ○ PCI 设备不提供有关其 NUMA 关联性的信息。

实例 NUMA 拓扑

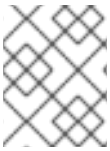
您可以使用属性创建类别，以定义实例 vCPU 线程的主机 NUMA 放置，以及从主机 NUMA 节点分配实例 vCPU 和内存。

为实例定义 NUMA 拓扑可提高实例操作系统的性能，其内存和 vCPU 分配会大于计算主机中的 NUMA 节点的大小。

计算调度程序使用这些属性来确定适合实例的主机。例如，云用户使用以下类型启动一个实例：

```
$ 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，其能力运行 8 个 CPU 和 4GB RAM，则计算调度程序不会将其视为有效的匹配。



注意

由类别定义的 NUMA 拓扑不能被镜像定义的 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。

使用下表中的属性键来定义实例 NUMA 拓扑。

表 9.14. NUMA 拓扑的类别元数据

键	描述
hw:numa_nodes	指定将实例 vCPU 线程的执行限制为的主机 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	映射到实例 NUMA 节点 N 的实例 vCPU 列表。如果没有指定此密钥，则 vCPU 会平均划分到可用的 NUMA 节点。 N 从 0 开始。请谨慎使用 3.0.N 值，只有在至少有两个 NUMA 节点时才使用。 只有在设置了 **hw:numa_nodes** 并且只有实例的 NUMA 节点有非对称分配 CPU 和 RAM （对于某些 NFV 工作负载很重要）时才需要此属性有效。
hw:numa_mem.N	映射到实例 NUMA 节点 N 的实例内存数量。如果没有指定这个键，则会在可用的 NUMA 节点之间平均划分内存。 N 从 0 开始。请谨慎使用 3.0.N 值，只有在至少有两个 NUMA 节点时才使用。 只有在设置了 **hw:numa_nodes** 并且只有实例的 NUMA 节点有非对称分配 CPU 和 RAM （对于某些 NFV 工作负载很重要）时才需要此属性有效。

警告

*如果 **hw:numa_cpus.N** 或 **hw:numa_mem.N** 的组合值分别大于 CPU 或内存的数量，则计算服务会引发异常。*

CPU 实时策略

使用下表中的属性键定义实例中处理器的实时策略。

注意

- *虽然大多数实例 vCPU 可使用实时策略运行，但您必须将至少一个 vCPU 标记为非实时客户机进程和仿真程序开销进程。*
- *要使用这个额外规格，您必须启用固定的 CPU。*

表 9.15. CPU 实时策略的类别元数据

键	描述
hw:cpu_realtime	设置为 **yes**，以创建将实时策略分配给实例 vCPU 的类别。 默认： **no**

hw:cpu_realtime_mask	指定不为其分配实时策略的 vCPU。您必须使用符号(^)添加掩码值。以下示例显示除 vCPU 0 和 1 外的所有 vCPU 都有实时策略：
	
	`$ openstack flavor set <flavor> \`
	`--property hw:cpu_realtime="yes" \`
	`--property hw:cpu_realtime_mask=^0-1`
	
	 
	
	注意
	
	如果镜像上设置了 **hw_cpu_realtime_mask** 属性，则它优先于类别上设置的 **hw:cpu_realtime_mask** 属性。

仿真程序线程策略

您可以为实例分配一个用于仿真程序线程的 pCPU。仿真程序线程是不与实例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的 CPU：

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

使用下表中的 property 键来定义实例的仿真程序线程策略。

表 9.16. 仿真程序线程策略的类别元数据

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** ：仿真程序线程浮点值在 **NovaComputeCpuSharedSet** heat 参数中定义的 pCPUs。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，因为它会禁止资源密集型。 - unset ：（默认）仿真程序线程策略没有启用，在与实例关联的固定 CPU 之间仿真程序线程浮点值。

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 9.17. 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大页大小。使用此选项会创建 1 NUMA 节点的隐式 NUMA 拓扑，除非被 **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 （常规页面）。 • **任何** ：选择最大可用巨页大小，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。使用整数值（以 KB 为单位）或任何标准后缀。例如： **4KB**、**2MB**、**2048**、**1GB**。 • unset ：（默认）大页不用于支持实例，且不会生成隐式 NUMA 拓扑。

PCI 透传

使用下表中的 property key 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 9.18. PCI 透传的类别元数据

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 &lt **;alias&** gt; 类型的 PCI 设备数。

hypervisor 签名

使用下表中的 property key 从实例隐藏管理程序签名。

表 9.19. 用于隐藏虚拟机监控程序签名的类别元数据

键	描述
hide_hypervisor_id	设置为 **True** 以隐藏实例的虚拟机监控程序签名，以允许所有驱动程序在实例上加载和工作。

UEFI 安全引导

使用下表中的 property key 创建使用 UEFI 安全引导保护的实例。

注意

带有 UEFI 安全引导的实例必须支持 UEFI 和 GUID 分区表(GPT)标准，并包括一个 EFI 系统分区。

表 9.20. UEFI 安全引导的类别元数据

键	描述
os:secure_boot	设置为 **required**，以便为使用此类别启动的实例启用安全引导。默认禁用此选项。

实例资源特征

每个资源提供程序都有一组特征。特征是资源提供程序的定性方面，例如存储磁盘类型或 Intel CPU 指令集扩展。实例可以指定它需要的特征。

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 property 键来定义实例的资源特征。

表 9.21. 资源特征的类别元数据

键	描述
trait:<trait_name>	指定 Compute 节点特征。将 trait 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。 • **禁止** ：选择托管实例的 Compute 节点不能具有特征。 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 property key 为实例请求裸机资源类。

表 9.22. 裸机资源类的类别元数据

键	描述

resources:<resource_class_name>	使用此属性指定标准裸机资源类来覆盖实例所需的自定义裸机资源类。
	
	您可以覆盖的标准资源类是 **VCPU**、**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 服务节点的资源类型对应的自定义资源类的名称，请将资源类转换为大写，请将所有标点替换为下划线，并将 prefix 替换为 CUSTOM_。
	
	例如，要在具有 **-resource-class baremetal.SMALL 的节点上调度实例**，请创建以下类别：
	
	`$ 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 10 章 向实例添加元数据

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来访问元数据。

配置驱动器

配置驱动器是您可以在实例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的形式呈现给实例。实例可以挂载此驱动器并读取其中的文件，以获取通常可通过元数据服务提供的信息。

元数据服务

计算服务提供元数据服务作为 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。

10.1. 实例元数据的类型

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

云用户可以指定要在启动实例时使用的额外数据，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以必要属性传递键值对。

云管理员提供的数据

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）用于所有实例相同的元数据。
- **DynamicJSON**：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 配置位于实例上的以下只读文件中之一：

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

计算服务提供数据

计算服务使用其元数据服务的内部实现将信息传递给实例，如实例请求的主机名，以及实例所在可用区。默认情况下，这不会由云用户或管理员进行配置。

10.2. 在所有实例中添加配置驱动器

作为管理员，您可以将 Compute 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，出于以下原因，您可以使用配置驱动器：

- 当您的部署不使用 DHCP 将 IP 地址分配给实例时，以传递网络配置。您可以通过配置驱动器传递实例的 IP 地址配置，在为实例配置网络设置前，实例可以挂载和访问。
- 要将数据传递给在启动实例时不知道的实例，例如，用于在引导后将实例注册到 Active Directory 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，这可减少定期访问元数据服务器的实例影响，以检查和构建事实。

任何可以挂载 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 Compute 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** 改为 **vfat**，请将 **config_drive_format** 参数添加到您的配置中：

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

4. 将更新保存到计算环境文件。
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. 创建实例：

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

2. 登录实例。
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**，则需要首先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。

10.3. 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 文件为该实例提供元数据。

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。IdM 服务器可用作证书颁发机构，并在 overcloud 上启用了 SSL/TLS 时管理 overcloud 证书。如需更多信息，请参阅 安全和强化指南 中的使用 Ansible 实施 TLS-e 。

流程

1. 打开您的 Compute 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"
```

计算服务生成 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 config 驱动器目录中。

注意

不要多次对目标服务使用相同的名称。

4. 将更新保存到计算环境文件。
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 11 章 为实例配置 CPU 功能标记

您可以启用或禁用实例的 CPU 功能标记，而无需更改主机 Compute 节点上的设置并重新引导 Compute 节点。通过配置应用到实例的标准 CPU 功能标记，您有助于跨 Compute 节点实现实时迁移兼容性。您还可以通过禁用对具有特定 CPU 模型的实例安全性或性能造成负面影响的标记来管理实例的性能和安全性，或者启用从安全问题提供缓解或缓解性能问题的标志。

11.1. 先决条件

- 主机 Compute 节点的硬件和软件支持 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，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**。

11.2. 为实例配置 CPU 功能标记

配置计算服务，以将 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 Compute 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，以提供针对安全漏洞的缓解方案。
   - **Custom**：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 节点上托管的实例的 Compute 节点相同的 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，将更常见和不太高级 CPU 型号放在列表中，最后是功能丰富的 CPU 模型，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 节点的架构的名称，如 **x86_64**。

6. 为具有指定 CPU 型号的实例配置 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 "+" 前缀每个标记以启用标志，或 "-" 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 configured.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 12 章 配置手动节点重新引导以定义 KERNELARGS

当 overcloud 部署包括第一次设置 **KernelArgs** 时，overcloud 节点会自动重启。如果您要将 **KernelArgs** 添加到生产环境中的部署中，重新引导节点可能会出现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启，而是在每次 overcloud 部署后手动执行节点重新引导。

注意

如果您禁用自动重新引导，然后将新的 Compute 节点添加到部署中，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，因为只有重启后应用 **KernelArgs** 的配置。

12.1. 配置手动节点重新引导以定义 KERNELARGS

当您首次配置 **KernelArgs** 时，您可以禁用节点自动重启，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** role 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：

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

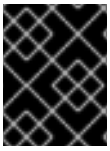
11. 稍等片刻，直到节点启动。
12. 重新启用 Compute 节点：

```
(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 13 章 为实例配置虚拟 GPU



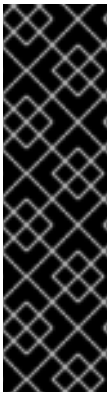
重要

该功能在此发行版本中作为技术预览提供，因此不享有红帽的全面支持。它只应用于测试，不应部署在生产环境中。有关技术预览功能的更多信息，请参阅覆盖范围详细信息。

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 hypervisor 类型定义和管理虚拟 GPU (vGPU) 资源。您可以使用此配置更加有效地划分所有物理 GPU 设备之间的渲染工作负载，并更好地控制支持 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，每个实例可以使用与物理 GPU 设备对应的虚拟 GPU 设备支持 GPU 工作负载。

Compute 服务跟踪可用于每个主机上定义的 GPU 配置集的 vGPU 设备数量。计算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的池中。



重要

红帽启用了在 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 认证支持服务订阅来获取 NVIDIA vGPU 软件的支持。对于使用无法在支持的组件中重现问题的 NVIDIA vGPU 的问题，会应用以下支持政策：

- 当红帽不怀疑问题涉及第三方组件时，会应用正常的 支持范围 和红帽 SLA。
- 当红帽怀疑问题涉及第三方组件时，客户将遵循红帽 第三方支持和证书政策。如需更多信息，请参阅知识库文章 NVIDIA 支持。

13.1. 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。
- 不支持在主机之间实时迁移 vGPU 实例。
- 不支持删除 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。在重新引导 Compute 节点前，您必须冷迁移实例，或者在重启后手动将每个 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 实例从实例 XML 检索 **mdev** UUID。您可以使用以下命令发现每个实例的 **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** 请求中返回到 Compute API。

- 由于 libvirt 限制，不支持对支持 vGPU 的实例暂停操作。相反，您可以 snapshot 或 shelve 实例。
- 默认情况下，计算主机上的 vGPU 类型不会公开给 API 用户。要授予访问权限，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，您必须遵守 NVIDIA 许可要求。例如，Nvidia vGPU GRID 需要许可服务器。有关 NVIDIA 许可证要求的更多信息，请参阅 NVIDIA 网站上的 NVIDIA License Server 发行注记。

13.2. 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. 为启动具有 vGPU 的实例创建 vGPU 类别。

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。要仅调度在 vGPU Compute 节点上请求 vGPU 的实例，请创建 vGPU Compute 节点的主机聚合，并将 Compute 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

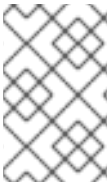
要使用 NVIDIA GRID vGPU，您必须遵守 NVIDIA GRID 许可要求，且您必须有自托管许可证服务器的 URL。如需更多信息，请参阅 NVIDIA License Server 发行注记 网页。

13.2.1. 先决条件

- 您已从 NVIDIA 网站下载了与您的 GPU 设备对应的 NVIDIA GRID 主机驱动程序 RPM 软件包。要确定您需要哪个驱动程序，请参阅 NVIDIA Driver Downloads Portal。您必须是一个注册的 NVIDIA 客户才能从门户下载驱动程序。
- 您已构建了一个安装了 NVIDIA GRID 主机驱动程序的自定义 overcloud 镜像。

13.2.2. 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，并使用 GPU 资源类配置裸机节点，以标记启用了 GPU 的 Compute 节点。



注意

以下流程适用于尚未调配的新 overcloud 节点。要将资源类分配给已置备的现有 overcloud 节点，您必须使用 scale down 过程取消置备节点，然后使用扩展过程使用新的资源类分配来重新置备节点。有关更多信息，请参阅 扩展 overcloud 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，以及 overcloud 所需的任何其他角色：

```
(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
角色评论	**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**
HostnameFormatDefault	**-compute-**	**-computegpu-**
deprecated_nic_config_name	**compute.yaml**	**compute-gpu.yaml**

5. 将 overcloud 的启用了 GPU 的 Compute 节点添加到节点定义模板(**node.json** 或 **node.yaml**) 中，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. 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 baremetal 节点的 ID。

8. 将 **ComputeGpu** 角色添加到节点定义文件 **overcloud-baremetal-deploy.yaml** 中，并定义您要分配给节点的任何预先节点放置、资源类、网络拓扑或其他属性：

```
- name: Controller
  count: 3
- name: Compute
  count: 3
- name: ComputeGpu
  count: 1
  defaults:
    resource_class: baremetal.GPU
    network_config:
      template: /home/stack/templates/nic-config/myRoleTopology.j2 1
```

1 您可以重复使用现有的网络拓扑，或为角色创建新的自定义网络接口模板。如需更多信息，请参阅 Director 安装和使用 指南中的 自定义网络接口模板。如果您不使用 **network_config** 属性定义网络定义，则使用默认网络定义。

有关您可以在节点定义文件中配置节点属性的属性的更多信息，请参阅 裸机节点置备属性。如需节点定义文件示例，请参阅 节点定义文件示例。

9. 运行置备命令为您的角色置备新节点：

```
(undercloud)$ openstack overcloud node provision \
--stack <stack> \
[--network-config \]
--output /home/stack/templates/overcloud-baremetal-deployed.yaml \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- 将 **&** lt;stack> 替换为置备裸机节点的堆栈的名称。如果未指定，则默认为 **overcloud**。
- 包含 **--network-config** 可选参数，为 **cli-overcloud-node-network-config.yaml** Ansible playbook 提供网络定义。如果您不使用 **network_config** 属性定义网络定义，则使用默认网络定义。

10. 在单独的终端中监控调配进度。当置备成功后，节点状态会从 **available** 改为 **active**：

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

11. 如果您没有使用 **--network-config** 选项运行 **provisioning** 命令，请在 **network-environment.yaml** 文件中配置 **<Role>NetworkConfigTemplate** 参数以指向 NIC 模板文件：

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputeGpuNetworkConfigTemplate: /home/stack/templates/nic-configs/<gpu_net_top>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

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

13.2.3. 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，或者 **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. 创建一个 **gpu.yaml** 文件来指定 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。

4. 将更新保存到计算环境文件。
5. 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/overcloud-baremetal-deployed.yaml \
  -e /home/stack/templates/node-info.yaml
```

13.3. 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，您可以为启动实例创建自定义支持 vGPU 的镜像。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. 使用您的 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 **<flavor>** 替换为包含 vGPU 实例所需的硬件配置集的类别名称或 ID。有关创建 vGPU 类别的详情，请参考为实例创建 vGPU 类别。
   - 将 **<image>** 替换为包含 vGPU 实例所需的软件配置集的镜像名称或 ID。有关下载 RHEL 云镜像的详情，请参阅管理镜像。

4. 以 cloud-user 用户身份登录到实例。
5. 按照 NVIDIA 指南在实例上创建 **gridd.conf** NVIDIA GRID 许可证文件：通过使用配置文件 在 Linux 上许可 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** 镜像属性定义 GPU 驱动程序类型。如果要为 vGPU 实例禁用模拟 GPU，您可以选择 **none**。有关支持的驱动程序的更多信息，请参阅 镜像配置参数。

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。

13.4. 为实例创建 VGPU 类型

要让您的云用户为 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 资源分配给该类型。

先决条件

- 您已使用 GPU 设计的 Compute 节点配置并部署了 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                                    |
+----------------------------+--------------------------------------+
```

13.5. 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>
```

13.6. 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，实例会保留对执行任务的设备的专用访问权限，且设备对主机不可用。

先决条件

- **pciutils** 软件包安装在具有 PCI 卡的物理服务器上。
- GPU 设备的驱动程序必须安装在设备被传递给的实例上。因此，您需要已创建了安装了所需 GPU 驱动程序的自定义实例镜像。有关如何安装 GPU 驱动程序的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 设备类型的厂商 ID 和产品 ID，请在具有 PCI 卡的物理服务器上输入以下命令：

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

例如，要确定 NVIDIA GPU 的供应商和产品 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 卡的物理服务器上输入以下命令：

```
# 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，例如 **pci_passthru_controller.yaml**。

4. 将 **PciPassthroughFilter** 添加到 **pci_passthru_controller.yaml** 中的 **NovaSchedulerEnabledFilters** 参数中：

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
```

```
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - NUMATopologyFilter
```

5. 要为 Controller 节点上的设备指定 PCI 别名，请将以下配置添加到 **pci_passthru_controller.yaml** 中：

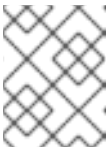
   - 如果 PCI 设备具有 SR-IOV 功能：

```
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 device type 字段。

   注意

   如果 **nova-api** 服务以 Controller 以外的角色运行，然后将 **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"
```

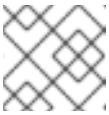
8. 您必须在 Compute 节点上为实例迁移和调整大小操作创建 PCI 别名的副本。要为 Compute 节点上的设备指定 PCI 别名，请将以下内容添加到 **pci_passthru_compute.yaml** 中：

   - 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 以支持 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

当您第一次将 **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启，而是在每次 overcloud 部署后手动执行节点重新引导。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，每个设备的供应商 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>*** *替换为安装了所需 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 14 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。

14.1. 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到 VNC 代理服务，为实例保护到 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 Compute 环境文件。
4. 配置用于到实例的 VNC 控制台连接的最低协议版本：

```
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** ：如果您有不支持较新版本的客户端，则使用。

注意

TLS 1.0 和 TLS 1.1 在 RHEL 8 中已弃用，在 RHEL 9 中不支持。

- **tlsv1_2** ：如果要配置 SSL/TLS 密码，以用于到实例的 VNC 控制台连接，则使用 tlsv1_2。
- **tlsv1_3** ：如果要将标准密码库用于 TLSv1.3，则使用。**NovaVNCProxySSLCiphers** 参数的配置将被忽略。

5. 如果将允许的 SSL/TLS 协议版本设置为 **tlsv1_2**，则配置 SSL/TLS 密码，以用于到实例的 VNC 控制台连接：

```
parameter_defaults:
  NovaVNCProxySSLCiphers: <ciphers>
```

将 & **lt;ciphers** > 替换为允许的密码套件列表。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
```

14.2. 数据库清理

计算服务包含一个管理工具 **nova-manage**，可用于执行部署、升级、清理和维护相关任务，如应用数据库架构、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 上自动化以下数据库管理任务：

- 通过将已删除的行从 production 表中移到影子表格中来存档已删除的实例记录。
- 归档完成后，从影子表格清除已删除的行。

14.2.1. 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，两者的 jitter 为 0 到 3600 秒。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 Compute 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
```

   有关用于管理数据库 cron 作业的完整 heat 参数列表，请参阅 计算服务自动化数据库管理的配置选项。

3. 将更新保存到计算环境文件。
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
```

14.2.2. 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 cron 作业。

表 14.1. Compute (nova)服务 cron 参数

参数	描述
NovaCronArchiveDeleteAllCells	将此参数设置为 "True"，以归档所有单元中删除的实例记录。 Default: **True**

参数	描述
NovaCronArchiveDeleteRowsAge	使用此参数根据其年龄归档已删除的实例记录（以天为单位）。 设置为 **0，** 以在影子表格中存档超过今天的数据。 默认： **90**
NovaCronArchiveDeleteRowsDestination	使用此参数配置文件，以记录已删除的实例记录。 默认值： **/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个表中。 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个表的已删除实例记录的最大数量。 默认： **1000**
NovaCronArchiveDeleteRowsMinute	使用此参数配置超过小时的分钟，以运行 cron 命令将已删除的实例记录移到另一个表。 默认： **1**
NovaCronArchiveDeleteRowsMonthday	使用此参数配置月中的哪个日期，以运行 cron 命令，将已删除的实例记录移到另一个表中。 默认 **：*** （每天）
NovaCronArchiveDeleteRowsMonth	使用此参数配置在哪个月中运行 cron 命令，将已删除的实例记录移到另一个表中。 默认 **：*** （每月）
NovaCronArchiveDeleteRowsPurge	将此参数设置为 "True"，以在调度归档后立即清除影子表。 Default: **False**
NovaCronArchiveDeleteRowsUntilComplete	将此参数设置为 "True"，以继续将已删除的实例记录移到另一个表，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存档已删除实例记录的 crontab 的用户，以及有权访问 crontab 使用的日志文件。 默认： **nova**
NovaCronArchiveDeleteRowsWeekday	使用此参数配置星期几，以运行 cron 命令，将已删除的实例记录移到另一个表中。 默认 **：*** （每天）
NovaCronPurgeShadowTablesAge	使用此参数根据其年龄清除影子表。 设置为 **0** 以清除今天旧的影子表格。 默认 **：14**
NovaCronPurgeShadowTablesAllCells	将此参数设置为 "True"，以清除所有单元中的影子表。 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	使用此参数配置过去几小时的分钟，在其中运行 cron 命令来清除影子表格。 默认： **0**
NovaCronPurgeShadowTablesMonth	使用此参数配置在哪个月中运行 cron 命令，以清除影子表格。 默认 **：*** （每月）
NovaCronPurgeShadowTablesMonthday	使用此参数配置月中的哪个日期，以运行 cron 命令来清除影子表格。 默认 **：*** （每天）

参数	描述
NovaCronPurgeShadowTablesUser	使用此参数配置拥有 crontab 的用户，以清除影子表格并且有权访问 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。 Default: **False**
NovaCronPurgeShadowTablesWeekday	使用此参数配置星期几，以运行 cron 命令来清除影子表格。 默认 **：*** （每天）

14.3. 在 COMPUTE 节点间迁移虚拟机实例

有时，您需要将实例从一个 Compute 节点迁移到 overcloud 中的另一个 Compute 节点，以执行维护、重新平衡工作负载或替换失败节点。

Compute 节点维护

如果您需要临时将 Compute 节点退出服务，例如，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 节点应该失败，且您需要服务或替换它，您可以将失败的 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。您可以使用与 Compute 节点失败前相同的名称、UUID、网络地址和其他分配的资源，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，将实例迁移到与其他联网资源更接近的 Compute 节点，以减少延迟，或在 Compute 节点上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。默认情况下，可组合服务是所有 Compute 角色中包含的主要服务之一。有关更多信息，请参阅 Director 安装和使用 指南中的 可组合服务和自定义角色。*

注意

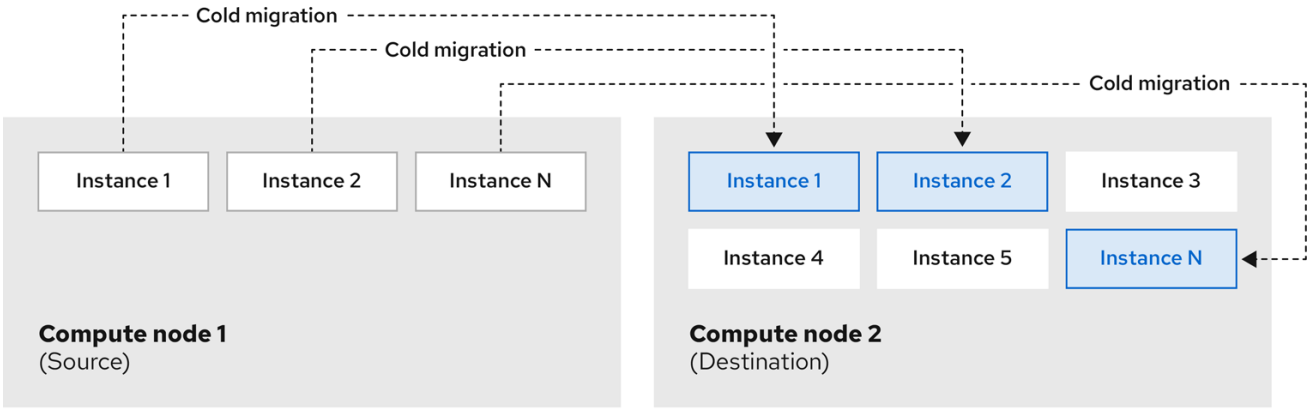
如果您有一个正常工作的 Compute 节点，并且您要为备份目的制作实例副本，或者将实例复制到其他环境中，请按照 Director 安装和使用指南中的 将虚拟机导入到 overcloud 中的步骤进行操作。

14.3.1. 迁移类型

Red Hat OpenStack Platform (RHOSP)支持以下类型的迁移。

冷迁移

冷迁移或非实时迁移涉及在从源 Compute 节点迁移到目标 Compute 节点之前关闭正在运行的实例。



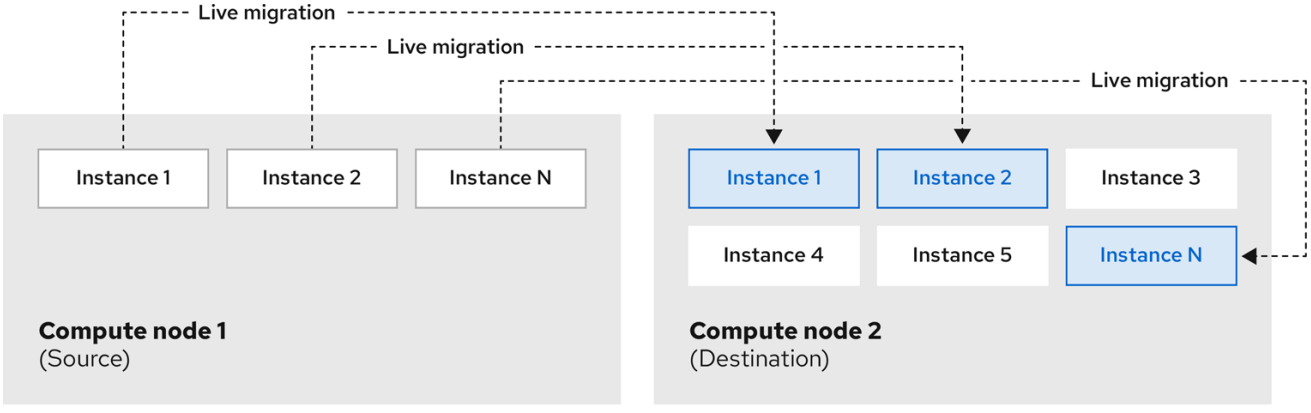
冷迁移涉及实例的一些停机时间。迁移的实例维护对同一卷和 IP 地址的访问。

注意

冷迁移要求源和目标 Compute 节点都正在运行。

实时迁移

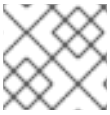
实时迁移涉及将实例从源 Compute 节点移到目标 Compute 节点，而不将其关闭，同时保持状态一致性。



实时迁移实例需要很少或没有明显的停机时间。但是，实时迁移会在迁移操作期间影响性能。因此，在迁移时，实例应该从关键路径中获取。

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、内存延迟或网络带宽、存储 IO 或 CPU 频率降低。



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。

14.3.2. 迁移限制

迁移限制通常随块迁移、配置磁盘或者一个或多个实例访问 Compute 节点上的物理硬件而发生。

CPU 约束

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 **ppc64le** CPU 迁移到 **x86_64** CPU。

不支持在不同 CPU 型号间迁移。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所花费的时间比迁移使用共享存储的卷支持的实例要长得多，如 Red Hat Ceph Storage。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，使用共享存储的卷支持的实例（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

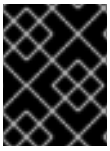
由迁移消耗大量 RAM 的本地磁盘或实例导致的 control plane 网络中的网络拥塞可能会影响使用 control plane 网络的其他系统的性能，如 RabbitMQ。

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) 可以迁移具有读写功能的驱动器，包括带有驱动器格式（如 **vfat** ）的配置驱动器。

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。红帽不提供在实时迁移过程中提高数据包丢失、网络延迟、内存延迟或降低网络带宽、内存带宽、存储 IO 或 CPU 性能的支持。

在迁移过程中没有新操作

要在源和目标节点上的实例副本之间实现状态一致性，RHOSP 必须在实时迁移过程中防止新操作。否则，如果写入内存的速度比实时迁移复制内存状态更快，实时迁移可能需要很长时间或可能永远不会结束。

使用 NUMA 固定 CPU

计算配置中的 **NovaSchedulerEnabledFilters** 参数必须包含 **AggregateInstanceExtraSpecsFilter** 和 **NUMATopologyFilter** 的值。

多单元云

在多单元云中，您可以将实例实时迁移到同一单元中的不同主机上，但不能在单元格中实时迁移。

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** 配置不同，则实例不会分配给目标 Compute 节点上配置为共享(unpinned)实例的 CPU。因此，如果您需要实时迁移浮动实例，您必须为专用（固定）实例配置具有相同 CPU 映射的所有 Compute 节点，或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您要迁移的实例。

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。使用直接模式 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 部署中的数据包丢失

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。
要在实时迁移过程中尽量减少数据包丢失，请将 ML2/OVN 部署配置为在迁移完成后在目标主机上宣布实例：

```
parameter_defaults:
  ComputeExtraConfig:
    nova::workarounds::enable_qemu_monitor_announce_self: true
```

ML2/OVS 部署上的实时迁移

要在 ML2/OVS 部署中实时迁移实例时数据包丢失，请将 ML2/OVS 部署配置为启用网络服务(neutron)实时迁移事件，并在迁移完成后声明目标主机上的实例：

```
parameter_defaults:
  NetworkExtraConfig:
    neutron::server::notifications::nova::live_migration_events: true
  ComputeExtraConfig:
    nova::workarounds::enable_qemu_monitor_announce_self: true
```

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 透传为实例提供对 PCI 设备的独占访问权限，其显示和行为就像它们被物理附加到实例的操作系统一样。但是，因为 PCI 透传涉及直接访问物理设备，QEMU/KVM 不支持使用 PCI 透传进行实例实时迁移。

端口资源请求

您无法实时迁移使用具有资源请求的端口的实例，如保证的最小带宽 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>
```

14.3.3. 准备迁移

在迁移一个或多个实例之前，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：如果要将实例从源 Compute 节点迁移到节点上执行维护，您必须禁用该节点，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。

14.3.4. 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促进的迁移方案，如迁移使用 PCI 直通的实例。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：如果您为维护禁用了源 Compute 节点，您必须重新启用该节点，以便可以把新实例分配给该节点：

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

将 **<source>** 替换为源 Compute 节点的主机名。

14.3.5. 实时迁移实例

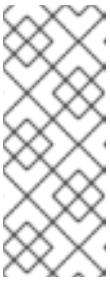
实时迁移将实例从源 Compute 节点移动到目标 Compute 节点，停机时间最少。实时迁移可能并不适用于所有实例。如需更多信息，请参阅 迁移限制。

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 命令涵盖使用共享存储迁移实例，这是默认设置。指定用于 **迁移本地存储的卷的--block-migration** 标志：

```
(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：如果您为维护禁用了源 Compute 节点，您必须重新启用该节点，以便可以把新实例分配给该节点：

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

将 **<source>** 替换为源 Compute 节点的主机名。

14.3.6. 检查迁移状态

迁移涉及迁移完成前的几个状态转换。在正常运行的迁移期间，迁移状态通常会有如下变换：

1. **Queued**：计算服务接受迁移实例的请求，并且迁移处于待处理状态。
2. **Preparing:** Compute 服务正在准备迁移实例。
3. **Running**：计算服务正在迁移实例。
4. **post-igrating**：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 : Compute 服务已完成迁移实例，并在源 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>** 替换为迁移的 ID。
  运行 **nova server-migration-show** 命令返回以下示例输出：

```
+------------------------+--------------------------------------+
| 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 计算服务会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而减少，迁移可能无法完成，计算服务可能会中止。

有时，实例迁移需要很长时间或遇到错误。如需更多信息，请参阅故障排除迁移。

14.3.7. 清空实例

如果要将实例从死机或关闭的 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，则不会在撤离过程中重新构建实例根磁盘，因为该磁盘仍然可以被目标 Compute 节点访问。如果实例不使用共享存储，则目标 Compute 节点上也会重新构建实例根磁盘。

注意

- 您只能在 Compute 节点被隔离时执行撤离，并且 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。

14.3.7.1. 清空一个实例

您可以一次撤离一个实例。

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>** 替换为托管要撤离实例的 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：

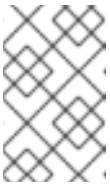
```
(overcloud)[stack@director ~]$ openstack compute service set \
 <node> nova-compute --disable --disable-reason <disable_host_reason>
```

- 将 **<node>** 替换为要撤离实例的 Compute 节点的名称。
- 将 **<disable_host_reason>** 替换为您为什么禁用 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 evacuated 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时，才会更改密码。如果实例托管在共享存储上，或者附加了 Block Storage 卷，则不会更改密码，且不会显示错误消息来告知您密码没有被改变。

- 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，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。

14.3.7.2. 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>** 替换为托管要撤离实例的 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 Compute 节点的名称，以便从中撤离实例。
   - 将 **<disable_host_reason>** 替换为您为什么禁用 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 Compute 节点的名称，以便从中撤离实例。

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。

14.3.8. 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下降。

14.3.8.1. 在迁移过程中出错

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 节点上的实例在迁移到目标 Compute 节点完成前被删除。

14.3.8.2. 永不结束的实时迁移

实时迁移可能无法完成，这会使迁移处于 **持续运行** 状态。实时迁移从未完成的一个常见原因是对源 Compute 节点上运行的实例的客户端请求创建更改的速度快于计算服务可以将其复制到目标 Compute 节点的速度。

使用以下方法之一应对这种情况：

- 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，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。

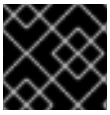
2. 中止实时迁移：

```
$ nova live-migration-abort <instance> <migration_id>
```

- 将 **<instance>** 替换为实例的名称或 ID。
- 将 **<migration_id>** 替换为迁移的 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，并且希望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过程。



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>** 替换为迁移的 ID。

14.3.8.3. 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 拓扑必须具有足够的资源可用。如果源和目标 Compute 节点之间的 NUMA 配置不相同，则实时迁移在实例性能降低时可能会成功。例如，如果源 Compute 节点将 NIC 1 映射到 NUMA 节点 0，但目标 Compute 节点在迁移后将 NIC 1 映射到 NUMA 节点 5，那么在迁移实例后，可以将网络流量从总线中的第一个 CPU 路由到带有 NUMA 节点 5 的第二个 CPU，以将流量路由到 NIC 1。这可能导致预期行为，但会降低性能。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。