



# Red Hat OpenStack Platform 17.1

## 为实例创建配置 Compute 服务

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



# Red Hat OpenStack Platform 17.1 为实例创建配置 Compute 服务

---

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

OpenStack Team  
rhos-docs@redhat.com

## 法律通告

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## 摘要

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

# 目录

使开源包含更多 .....	4
对红帽文档提供反馈 .....	5
第 1 章 计算服务(NOVA)功能 .....	6
第 2 章 配置计算服务(NOVA) .....	8
第 3 章 创建启动实例的类别 .....	10
3.1. 创建类别 .....	10
3.2. 类别参数 .....	11
3.3. 类型元数据 .....	13
第 4 章 在 COMPUTE 节点上配置 CPU .....	27
4.1. 在 COMPUTE 节点上配置 CPU 固定 .....	27
4.2. 配置仿真程序线程 .....	35
4.3. 为实例配置 CPU 功能标记 .....	36
第 5 章 在 COMPUTE 节点上配置内存 .....	39
5.1. 为超额分配配置内存 .....	39
5.2. 计算 COMPUTE 节点上的保留主机内存 .....	39
5.3. 计算 SWAP 大小 .....	40
5.4. 在 COMPUTE 节点上配置巨页 .....	40
5.5. 配置 COMPUTE 节点，将文件支持的内存用于实例 .....	45
5.6. 配置 AMD SEV COMPUTE 节点，为实例提供内存加密 .....	46
第 6 章 配置计算服务存储 .....	54
6.1. 镜像缓存的配置选项 .....	54
6.2. 实例临时存储属性的配置选项 .....	55
6.3. 配置要附加到一个实例的存储设备的最大数量 .....	57
6.4. 配置共享实例存储 .....	59
6.5. 配置镜像直接从 RED HAT CEPH RADOS 块设备(RBD)下载。 .....	60
6.6. 其他资源 .....	61
第 7 章 配置实例调度和放置 .....	62
7.1. 使用放置服务进行 PREFILTERING .....	62
7.2. 为计算调度程序服务配置过滤器和权重 .....	67
7.3. 计算调度程序过滤器 .....	69
7.4. 计算调度程序权重 .....	73
7.5. 声明自定义特征和资源类 .....	79
7.6. 创建和管理主机聚合 .....	82
第 8 章 配置 PCI 透传 .....	88
8.1. 为 PCI 透传设计 COMPUTE 节点 .....	88
8.2. 配置 PCI 透传 COMPUTE 节点 .....	90
8.3. PCI PASSTHROUGH 设备类型字段 .....	93
8.4. 配置 NOVAPCIPASSTHROUGH的指南 .....	94
第 9 章 配置 VDPA COMPUTE 节点以启用使用 VDPA 端口的实例 .....	96
9.1. 为 VDPA 设计 COMPUTE 节点 .....	96
9.2. 配置 VDPA COMPUTE 节点 .....	99
第 10 章 为实例配置虚拟 GPU .....	102
10.1. 支持的配置和限制 .....	102

10.2. 在 COMPUTE 节点上配置 VGPU	103
10.3. 创建自定义 VGPU 资源供应商特征	107
10.4. 创建自定义 GPU 实例镜像	108
10.5. 为实例创建 VGPU 类别	109
10.6. 启动 VGPU 实例	109
10.7. 为 GPU 设备启用 PCI 透传	110
<b>第 11 章 在实例中添加元数据</b>	<b>114</b>
11.1. 实例元数据的类型	114
11.2. 在所有实例中添加配置驱动器	114
11.3. 在实例中添加动态元数据	115
<b>第 12 章 配置手动节点重新引导以定义 KERNELARGS</b>	<b>117</b>
12.1. 配置手动节点重新引导以定义 KERNELARGS	117
<b>第 13 章 配置实例安全性</b>	<b>119</b>
13.1. 保护到实例的 VNC 控制台的连接	119
13.2. 配置 COMPUTE 节点，为实例提供模拟受信任的平台模块(TPM)设备	120
<b>第 14 章 数据库清理</b>	<b>124</b>
14.1. 配置数据库管理	124
14.2. COMPUTE 服务自动化数据库管理的配置选项	124
<b>第 15 章 在 COMPUTE 节点间迁移虚拟机实例</b>	<b>128</b>
15.1. 迁移类型	128
15.2. 迁移限制	130
15.3. 准备迁移	131
15.4. 冷迁移实例	132
15.5. 实时迁移实例	133
15.6. 检查迁移状态	133
15.7. 清空实例	135
15.8. 迁移故障排除	137



## 使开源包含更多

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



---

## 对红帽文档提供反馈

我们感谢您对文档提供反馈信息。与我们分享您的成功秘诀。

### 在 JIRA 中提供文档反馈

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

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

## 第 1 章 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。Compute 服务提取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，计算服务不会公开主机上运行的 CPU 的类型和拓扑结构，而计算服务会公开多个虚拟 CPU (vCPU)，并允许过量使用这些 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，并启用虚拟机实例的创建。要创建并置备实例，计算服务与以下 RHOSP 服务交互：

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

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

### 计算服务(nova-compute)

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

### 计算编排器(nova-conductor)

此服务调节 Compute 服务和数据库之间的交互，这会使 Compute 节点无法直接访问数据库。不要在运行 **nova-compute** 服务的节点上部署此服务。

### 计算调度程序(nova-scheduler)

此服务从队列获取实例请求，并确定要在其上托管实例的 Compute 节点。

### Compute API (nova-api)

此服务向用户提供外部 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一个临时位置。在多单元部署中，此数据库还包含单元映射，用于为每个单元指定数据库连接。

### cell 数据库

此数据库包含有关实例的大多数信息。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用于与单元和全局服务相互通信。

### 计算元数据

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

您可以部署多个 Compute 节点。运行实例的虚拟机监控程序在每个 Compute 节点上运行。每个 Compute 节点至少需要两个网络接口。Compute 节点也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。此单元包含控制和管理虚拟机实例的所有计算服务和数据库，以及所有实例和实例元数据。对于较大的部署，您可以使用多单元部署 overcloud，以适应大量 Compute 节点。在安装新的 overcloud 或之后的任何时间，您可以在环境

中添加单元格。

## 第 2 章 配置计算服务(NOVA)

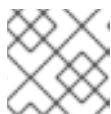
作为云管理员，您可以使用环境文件来自定义 Compute (nova)服务。Puppet 生成此配置并将其存储在 `/var/lib/config-data/puppet-generated/<nova_container>/etc/nova/nova.conf` 文件中。使用以下配置方法按以下优先级自定义计算服务配置：

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 Compute 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/manifestsAttr` 中定义的：

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



### 注意

只有当等同的 heat 参数不存在时才使用此方法。

3. **手动 hieradata 覆盖** - 在不存在 heat 或 Puppet 参数时用于自定义参数。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，则使用 Puppet 参数而不是手动覆盖方法。只有在没有等同的 heat 或 Puppet 参数时，才使用手动覆盖方法。

### 提示

按照 [您需要修改的参数中的指导操作](#)，以确定是否可用于自定义特定配置的 heat 或 Puppet 参数。

有关如何配置 overcloud 服务的更多信息，请参阅自定义 *Red Hat OpenStack Platform* 部署指南中的 [Heat 参数](#)。

## 第 3 章 创建启动实例的类别

实例类别是指定实例的虚拟硬件配置文件的资源模板。云用户在启动实例时必须指定类别。

类别可以指定 Compute 服务必须分配给实例的以下资源数量：

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

您可以通过将类别 `public` 放置到所有项目或特定项目或域，来指定能够使用类别。

类别可以使用元数据（也称为“额外规格”）来指定实例硬件支持和配额。类别元数据会影响实例放置、资源使用量限值 and 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中设置的 `extra_specs` 元数据匹配，使用类别元数据来查找适合主机聚合的主机聚合来托管实例的主机聚合。要在主机聚合上调度实例，您必须通过使用 `aggregate_instance_extra_specs` : 命名空间作为 `extra_specs` 键作为前缀来限定类别元数据。如需更多信息，请参阅 [创建和管理主机聚合](#)。

Red Hat OpenStack Platform (RHOSP)部署包括以下一组默认公共类别，您的云用户可以使用它们。

表 3.1. 默认类别

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



### 注意

使用类别属性设置的行为会覆盖使用镜像设置的行为。当云用户启动实例时，它们指定类别的属性覆盖其指定的镜像的属性。

### 3.1. 创建类别

您可以创建和管理特定功能或行为的专用类别，例如：

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

#### 流程

1. 创建一个类别，用于指定供实例使用的基本资源：

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

- 将 `<size_mb>` 替换为要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>` 替换为要为使用此类别创建的实例保留的 vCPU 数量。
- 可选：指定 `--private` 和 `--project` 选项，使该类别只能被特定项目或用户组访问。将 `<project_id>` 替换为可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

您不能在创建后使公共类别私有。

- 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的属性：

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

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

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

## 3.2. 类别参数

`openstack flavor create` 命令具有一个位置参数 `<flavor_name>` 来指定新类别的名称。

下表详细介绍了您可以在创建新类别时根据需要指定的可选参数。

表 3.2. 可选的类别参数

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

可选参数	描述
<b>--disk</b>	<p>(必需) 用于 root (/)分区 (以 GB 为单位) 的磁盘空间。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。</p> <div style="display: flex; align-items: center;">  <div> <p><b>注意</b></p> <p>创建具有 <b>--disk</b> 设置为 <b>0</b> 的类别, 要求实例从卷引导。</p> </div> </div> <p>默认: 0 GB</p>
<b>--ephemeral</b>	<p>用于临时磁盘的磁盘空间量 (以 GB 为单位)。默认值为 0 GB, 这意味着不会创建辅助临时磁盘。临时磁盘提供与实例生命周期关联的机器本地磁盘存储。临时磁盘不包含在任何快照中。此磁盘将被销毁, 当实例被删除时, 所有数据都会丢失。</p> <p>默认: 0 GB</p>
<b>--swap</b>	<p>以 MB 为单位的 swap 磁盘大小。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 <b>swap</b>。</p> <p>默认: 0 GB</p>
<b>--vcpus</b>	<p>(必需) 实例的虚拟 CPU 数量。</p> <p>默认: 1</p>
<b>--public</b>	<p>类别可用于所有项目。默认情况下, 类别是公共的, 可用于所有项目。</p>
<b>--private</b>	<p>该类别仅适用于使用 <b>--project</b> 选项指定的项目。如果您创建专用类别, 但没有添加任何项目, 则该类别仅可供云管理员使用。</p>
<b>--property</b>	<p>元数据或 "extra specs", 使用以下格式的键值对指定:</p> <p><b>--property &lt;key=value&gt;</b></p> <p>重复这个选项来设置多个属性。</p>
<b>--project</b>	<p>指定可以使用私有类别的项目。您必须将此参数与 <b>--private</b> 选项一起使用。如果没有指定任何项目, 则类别仅对 <b>admin</b> 用户可见。</p> <p>重复此选项以允许访问多个项目。</p>



可选参数	描述
<b>--project-domain</b>	指定可以使用私有类别的项目域。您必须将此参数与 <b>--private</b> 选项一起使用。  重复此选项，以允许访问多个项目域。
<b>--description</b>	类别的描述。长度限制为 65535 个字符。您只能使用可打印的字符。

### 3.3. 类型元数据

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

#### 实例资源使用量

使用下表中的属性键配置实例的 CPU、内存和磁盘 I/O 使用情况的限制。



#### 注意

限制实例 CPU 资源使用的额外规格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 资源限制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](#) 文档，以及 Libvirt 文档中的 [CPU 调优](#)。

表 3.3. 资源使用情况的类别元数据

键	描述
<b>quota:cpu_shares</b>	指定域的 CPU 时间成比例权重共享。默认为 OS 提供的默认值。计算调度程序相对于同一域中的其他实例上设置此属性的计算调度程序。例如，使用 <b>quota:cpu_shares=2048</b> 配置的实例会加倍 CPU 时间作为使用 <b>quota:cpu_shares=1024</b> 配置的实例。
<b>quota:cpu_period</b>	以微秒为单位指定强制实施 <b>cpu_quota</b> 的时间周期。在 <b>cpu_period</b> 中，每个 vCPU 无法消耗超过 <b>cpu_quota</b> 运行时。设置为范围 1000 - 1000000 中的值。设置为 <b>0</b> 以禁用。

键	描述
<b>quota:cpu_quota</b>	<p>以微秒为单位，指定每个 <b>cpu_period</b> 中 vCPU 允许的最大带宽：</p> <ul style="list-style-type: none"> <li>● 设置为范围 1000 - 18446744073709551 中的值。</li> <li>● 设置为 <b>0</b> 以禁用。</li> <li>● 设置为负值以允许无限带宽。</li> </ul> <p>您可以使用 <b>cpu_quota</b> 和 <b>cpu_period</b> 来确保所有 vCPU 以相同的速度运行。例如，您可以使用以下类别来启动最多可消耗的物理 CPU 能力 50% 的实例：</p> <pre>\$ openstack flavor set cpu_limits_flavor \   --property quota:cpu_quota=10000 \   --property quota:cpu_period=20000</pre>

## 实例磁盘调整

使用下表中的属性键调整实例磁盘性能。



### 注意

Compute 服务将以下服务质量设置应用到调配了 Compute 服务的存储，如临时存储。要调整块存储(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 [配置持久性存储](#) 指南中的 [块存储服务\(cinder\)服务质量规格](#)。

表 3.4. 磁盘调整的类别元数据

键	描述
<b>quota:disk_read_bytes_sec</b>	指定实例可以使用的最大磁盘读取，以字节为单位。
<b>quota:disk_read_iops_sec</b>	在 IOPS 中指定实例可以使用的最大磁盘读取。
<b>quota:disk_write_bytes_sec</b>	指定实例可以使用的最大磁盘写入，以字节为单位。
<b>quota:disk_write_iops_sec</b>	指定 IOPS 中可用于实例的最大磁盘写入。
<b>quota:disk_total_bytes_sec</b>	指定实例可以使用的最大 I/O 操作，以每秒字节数为单位。
<b>quota:disk_total_iops_sec</b>	指定 IOPS 中可用于实例的最大 I/O 操作。

## 实例网络流量带宽

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



## 注意

**quota:vif** curator 属性已弃用。相反，您应该使用 Networking (neutron) 服务服务质量 (QoS) 策略。如需有关 QoS 策略的更多信息，请参阅 [配置 Red Hat OpenStack Platform 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 **NeutronOVSEthernetDriver** 设置为 **iptables\_hybrid** 时，才会支持 **quota:vif\_\*** 属性。

表 3.5. 带宽限值的类别元数据

键	描述
<b>quota:vif_inbound_average</b>	(已弃用) 指定传入实例的流量所需的平均位率，单位为 kbps。
<b>quota:vif_inbound_burst</b>	(已弃用) 指定可以 KB 为单位突发的最大传入流量量。
<b>quota:vif_inbound_peak</b>	(已弃用) 指定实例可以接收传入流量的最大率，单位为 kbps。
<b>quota:vif_outbound_average</b>	(已弃用) 指定来自实例的流量所需的平均位率（单位为 kbps）。
<b>quota:vif_outbound_burst</b>	(已弃用) 指定可以 KB 为单位突发的最大传出流量量。
<b>quota:vif_outbound_peak</b>	(已弃用) 指定实例可以发送传出流量的最大速率（单位为 kbps）。

## 硬件视频 RAM

使用下表中的 property 键配置实例 RAM 的限值，以用于视频设备。

表 3.6. 视频设备的类别元数据


键	描述
<b>hw_video:ram_max_mb</b>	指定用于视频设备的最大 RAM，以 MB 为单位。使用 <b>hw_video_ram</b> 镜像属性。 <b>hw_video_ram</b> 必须小于或等于 <b>hw_video:ram_max_mb</b> 。

## 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 3.7. watchdog 行为的类别元数据

键	描述
---	----

键	描述
<b>hw:watchdog_action</b>	<p>指定以启用虚拟硬件 watchdog 设备并设置其行为。如果实例挂起或失败，则 watchdog 设备执行配置的操作。watchdog 使用 i6300esb 设备，它模拟 PCI Intel 6300ESB。如果没有指定 <b>hw:watchdog_action</b>，则禁用 watchdog。</p> <p>设置为以下有效值之一：</p> <ul style="list-style-type: none"> <li>● <b>disabled</b>：（默认）设备没有附加。</li> <li>● <b>重置</b>：强制实例重置。</li> <li>● <b>poweroff</b>: 强制实例关闭。</li> <li>● <b>暂停</b>：暂停实例。</li> <li>● <b>none</b>：启用 watchdog，但如果实例挂起或失败，则不执行任何操作。</li> </ul> <p> <b>注意</b></p> <p>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。</p>

## 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 3.8. RNG 的类别元数据

键	描述
<b>hw_rng:allowed</b>	<p>设置为 <b>False</b>，以禁用通过镜像属性添加到实例的 RNG 设备。</p> <p>Default: <b>True</b></p>
<b>hw_rng:rate_bytes</b>	指定实例在每个期间可以从主机的熵读取的最大字节数。
<b>hw_rng:rate_period</b>	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 3.9. vPMU 的类别元数据

键	描述
---	----

键	描述
<b>hw:pmu</b>	<p>设置为 <b>True</b>，为实例启用 vPMU。</p> <p><b>perf</b> 等工具使用实例上的 vPMU 来为配置集和监控实例性能提供更准确的信息。对于实时工作负载，vPMU 的模拟可能会带来额外的延迟，这可能会不可靠。如果它提供的遥测不需要，请设置 <b>hw:pmu=False</b>。</p>

### 虚拟受信任的平台模块(vTPM)设备

使用下表中的属性键为实例启用 vTPM 设备。

表 3.10. vTPM 的类别元数据

键	描述
<b>hw:tpm_version</b>	<p>设置为要使用的 TPM 版本。TPM 版本 <b>2.0</b> 是唯一受支持的版本。</p>
<b>hw:tpm_model</b>	<p>设置为要使用的 TPM 设备模型。如果没有配置 <b>hw:tpm_version</b>，则忽略。设置为以下有效值之一：</p> <ul style="list-style-type: none"> <li>● <b>TPM -tis</b>: (默认) TPM 接口规格。</li> <li>● <b>TPM-crb</b> : 命令响应缓冲.仅与 TPM 版本 2.0 兼容。</li> </ul>

### 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 3.11. CPU 拓扑的类别元数据

键	描述
<b>hw:cpu_sockets</b>	<p>指定实例的首选插槽数。</p> <p>默认：请求的 vCPU 数量</p>
<b>hw:cpu_cores</b>	<p>指定实例的每个插槽的首选内核数。</p> <p>默认：1</p>
<b>hw:cpu_threads</b>	<p>指定实例的每个内核的首选线程数量。</p> <p>默认：1</p>

键	描述
<b>hw:cpu_max_sockets</b>	指定用户可以使用镜像属性为其实例选择的最大插槽数。  示例： <b>hw:cpu_max_sockets=2</b>
<b>hw:cpu_max_cores</b>	指定每个插槽的最大内核数，用户可以使用镜像属性为其实例选择的最大内核数。
<b>hw:cpu_max_threads</b>	指定用户可使用镜像属性为实例选择的每个内核的最大线程数量。

## 串行端口

使用下表中的 property 键来配置每个实例的串行端口数量。

表 3.12. 串行端口的类别元数据

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

## CPU 固定策略

默认情况下，实例虚拟 CPU (vCPU) 是带有一个内核和一个线程的套接字。您可以使用属性来创建将实例 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 线程的行为，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 3.13. CPU 固定的类别元数据

键	描述
<b>hw:cpu_policy</b>	指定要使用的 CPU 策略。设置为以下有效值之一： <ul style="list-style-type: none"> <li>● <b>共享</b>：（默认）主机 pCPU 实例 vCPU 浮点值。</li> <li>● <b>专用</b>：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。这个选项意味着过量使用比率 1.0。</li> <li>● <b>混合</b>：实例 vCPU 使用专用（固定）主机 pCPU 和共享（未固定）主机 pCPU 的组合。</li> </ul>

键	描述
<p><b>hw:cpu_thread_policy</b></p>	<p>指定当 <b>hw:cpu_policy=dedicated</b> 时使用的 CPU 线程策略。设置为以下有效值之一：</p> <ul style="list-style-type: none"> <li>● <b>prefer</b>: (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线程。</li> <li>● <b>隔离</b>：主机不能具有 SMT 架构，或者必须模拟非 SMT 架构。此策略确保计算调度程序通过请求不报告 <b>HW_CPU_HYPERTHREADING</b> 特征的主机，将实例放置在没有 SMT 的主机上。也可以使用以下属性显式请求此特征： <pre data-bbox="790 622 1428 719"> --property trait:HW_CPU_HYPERTHREADING=forbidden </pre> <p>如果主机没有 SMT 架构，Compute 服务会将每个 vCPU 按预期放在不同的内核中。如果主机有 SMT 架构，则行 为由 <b>[workarounds]/disable_fallback_pcpu_query</b> 参数的配置决定：</p> <ul style="list-style-type: none"> <li>○ <b>true</b>：不使用 SMT 架构的主机，调度会失败。</li> <li>○ <b>false</b>：计算服务将每个 vCPU 放置到不同的物理内核中。Compute 服务不会将来自其他实例的 vCPU 放置到同一内核中。因此，每个使用的内核中有一个线程同级状态都保证不可用。</li> </ul> </li> <li>● <b>require</b>: 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 <b>HW_CPU_HYPERTHREADING</b> 特征的主机，将实例放置到带有 SMT 的主机。也可以使用以下属性显式请求此特征： <pre data-bbox="790 1294 1428 1391"> --property trait:HW_CPU_HYPERTHREADING=required </pre> <p>Compute 服务在线程同级时分配每个 vCPU。如果主机没有 SMT 架构，则不会使用它。如果主机具有 SMT 架构，但没有足够的空闲线程同级内核可用，则调度会失败。</p> </li> </ul>
<p><b>hw:cpu_dedicated_mask</b></p>	<p>指定专用（固定）或共享（未固定/浮动）的 CPU。</p> <ul style="list-style-type: none"> <li>● 要指定专用 CPU，请指定 CPU 号或 CPU 范围。例如，将属性设置为 <b>2-3</b>，以指定 CPU 2 和 3 专用，并且所有剩余的 CPU 都共享。</li> <li>● 要指定共享 CPU，请在 CPU 号或 CPU 范围前加上一个 caret (^)。例如，将属性设置为 <b>^0-1</b>，以指定 CPU 0 和 1 被共享，并且所有剩余的 CPU 都专用。</li> </ul>

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

表 3.14. PCI NUMA 关联性策略的类别元数据

键	描述
<code>hw:pci_numa_affinity_policy</code>	<p>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：</p> <ul style="list-style-type: none"> <li>● <b>必需</b>：计算服务会创建一个实例，只有在实例的至少一个 NUMA 节点与 PCI 设备关联时才请求 PCI 设备。这个选项提供最佳性能。</li> <li>● <b>preferred</b>：计算服务会尝试根据 NUMA 关联性选择 PCI 设备。如果无法做到这一点，则计算服务会将实例调度到没有与 PCI 设备关联性的 NUMA 节点上。</li> <li>● <b>传统</b>：（默认）计算服务创建请求 PCI 设备的实例： <ul style="list-style-type: none"> <li>○ PCI 设备与至少一个 NUMA 节点关联。</li> <li>○ PCI 设备不提供有关其 NUMA 事务的信息。</li> </ul> </li> <li>● <b>套接字</b>：计算服务会创建一个实例，只有在至少有一个实例 NUMA 节点与 PCI 设备相同的主机套接字中的 NUMA 节点关联时，才会请求 PCI 设备。例如，以下主机架构有两个套接字，每个套接字有两个 NUMA 节点，而 PCI 设备则连接到其中一个套接字中的一个节点。</li> </ul>  <p>计算服务可以将有两个 NUMA 节点和 <b>套接字</b> PCI NUMA 关联性策略的实例固定到主机节点的以下组合，因为它们都至少有一个实例 NUMA 节点固定到 PCI 设备的套接字：</p> <ul style="list-style-type: none"> <li>○ 节点 0 和节点 1</li> <li>○ 节点 0 和节点 2</li> <li>○ 节点 0 和节点 3</li> <li>○ 节点 1 和节点 2</li> <li>○ 节点 1 和节点 3</li> </ul> <p>实例无法固定到的主机节点的唯一组合是节点 2 和节点 3，因为这些节点都与 PCI 设备位于同一个套接字上。如果其他节点被其他实例使用，且只有节点 2 和 3 可用，则实例不会启动。</p>

## 实例 NUMA 拓扑

您可以使用属性来创建类别，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 分配大于计算主机中的 NUMA 节点大小。

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

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

计算调度程序搜索有两个 NUMA 节点的主机，一个有 3GB RAM，另一个运行 6 个 CPU，另一个具有 1GB RAM 和两个 CPU。如果主机只有一个 NUMA 节点，其能力运行 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 拓扑无法被镜像定义的 NUMA 拓扑覆盖。如果镜像 NUMA 拓扑与类别 NUMA 拓扑冲突，Compute 服务会引发 **ImageNUMATopologyForbidden** 错误。

### 小心

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

使用下表中的属性键来定义实例 NUMA 拓扑。

表 3.15. NUMA 拓扑的类别元数据

键	描述
<b>hw:numa_nodes</b>	指定主机 NUMA 节点数量，以限制实例 vCPU 线程的执行。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。
<b>hw:numa_cpus.N</b>	以逗号分隔的实例 vCPU 列表，映射到实例 NUMA 节点 N。如果没有指定此密钥，则 vCPU 在可用的 NUMA 节点之间平均划分。  N 从 0 开始。请谨慎使用 ./N 值，只有在您至少有两个 NUMA 节点时才使用。  只有在设置了 <b>hw:numa_nodes</b> 时，此属性才有效，只有在实例的 NUMA 节点具有非对称分配 CPU 和 RAM 时才需要，这对于某些 NFV 工作负载非常重要。

键	描述
<b>hw:numa_mem.N</b>	<p>映射到实例 NUMA 节点 N 的实例内存数量。如果没有指定此密钥，则内存存在可用的 NUMA 节点之间平均划分。</p> <p>N 从 0 开始。请谨慎使用 ./N 值，只有在您至少有两个 NUMA 节点时才使用。</p> <p>只有在设置了 <b>hw:numa_nodes</b> 时，此属性才有效，只有在实例的 NUMA 节点具有非对称分配 CPU 和 RAM 时才需要，这对于某些 NFV 工作负载非常重要。</p>

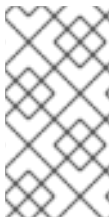


### 警告

如果 **hw:numa\_cpus.N** 或 **hw:numa\_mem.N** 的合并值分别大于可用 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 都可以使用实时策略运行，但您必须将至少一个 vCPU 标记为非实时客户端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 CPU。

表 3.16. CPU 实时策略的类别元数据

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

键	描述
<b>hw:cpu_realtime_mask</b>	<p>指定不为其分配实时策略的 vCPU。您必须使用逗号(^)前添加掩码值。以下示例显示，除 vCPU 0 和 1 以外的所有 vCPU 都有一个实时策略：</p> <pre>\$ openstack flavor set &lt;flavor&gt; \   --property hw:cpu_realtime="yes" \   --property hw:cpu_realtime_mask=^0-1</pre> <p> <b>注意</b></p> <p>如果在镜像上设置 <b>hw_cpu_realtime_mask</b> 属性，它将优先于类别上设置的 <b>hw:cpu_realtime_mask</b> 属性。</p>

### 仿真程序线程策略

您可以将 pCPU 分配给用于仿真程序线程的实例。仿真程序线程是与实例直接相关的仿真程序进程。实时工作负载需要专用的仿真程序线程 pCPU。要使用仿真程序线程策略，您必须通过设置以下属性来启用固定 CPU：

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

使用下表中的 property 键来定义实例的仿真程序线程策略。

表 3.17. 仿真程序线程策略的类别元数据

键	描述
<b>hw:emulator_threads_policy</b>	<p>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：</p> <ul style="list-style-type: none"> <li>● <b>共享</b>：在 <b>NovaComputeCpuSharedSet</b> heat 参数中定义的 pCPU 之间仿真程序线程浮点值。如果没有配置 <b>NovaComputeCpuSharedSet</b>，则与实例关联的固定 CPU 中的仿真程序线程 float。</li> <li>● <b>隔离</b>：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，因为它会特别考虑资源密集型。</li> <li>● <b>unset</b>：（默认）没有启用仿真程序线程策略，以及与实例关联的固定 CPU 中的仿真程序线程浮点。</li> </ul>

### 实例内存页面大小

使用下表中的属性键创建带有显式内存页面大小的实例。

表 3.18. 内存页面大小的类别元数据

键	描述
<b>hw:mem_page_size</b>	<p>指定用于支持实例的大页面大小。使用此选项会创建 1 NUMA 节点的隐式 NUMA 拓扑，除非由 <b>hw:numa_nodes</b> 指定。设置为以下有效值之一：</p> <ul style="list-style-type: none"> <li>● <b>large</b>：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）。</li> <li>● <b>Small</b>：选择主机上支持的最小页面大小。在 x86_64 系统上，这是 4 kB（常规页面）。</li> <li>● <b>any</b>：选择由 libvirt 驱动程序决定的最大可用巨页大小。</li> <li>● <b>&lt;pagesize&gt;</b>：如果工作负载具有特定要求，明确设置页的大小。将整数值用于页大小（以 KB 为单位）或任何标准后缀。例如：<b>4KB</b>、<b>2MB</b>、<b>2048</b>、<b>1GB</b>。</li> <li>● <b>unset</b>：（默认）大页面不用于支持实例，且不会生成隐式 NUMA 拓扑。</li> </ul>

## PCI passthrough

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

表 3.19. PCI 透传的类别元数据

键	描述
<b>pci_passthrough:alias</b>	<p>使用以下格式指定要分配给实例的 PCI 设备：</p> <pre>&lt;alias&gt;:&lt;count&gt;</pre> <ul style="list-style-type: none"> <li>● 将 <b>&lt;alias&gt;</b> 替换为与特定 PCI 设备类对应的别名。</li> <li>● 将 <b>&lt;count&gt;</b> 替换为 <b>分配给实例的</b> PCI 设备数量。</li> </ul>

## hypervisor 签名

使用下表中的 property 键从实例中隐藏 hypervisor 签名。

表 3.20. 用于隐藏管理程序签名的类别元数据

键	描述
<b>hide_hypervisor_id</b>	<p>设置为 <b>True</b> 以从实例中隐藏虚拟机监控程序签名，以允许所有驱动程序加载并在实例上工作。</p>

## UEFI 安全引导

使用下表中的 property 键创建一个通过 UEFI 安全引导保护的实例。



### 注意

具有 UEFI 安全引导的实例必须支持 UEFI 和 GUID 分区表(GPT)标准，并包含一个 EFI 系统分区。

表 3.21. UEFI 安全引导的类别元数据

键	描述
<code>os:secure_boot</code>	将 <code>secure_boot</code> 设置为 <b>required</b> ，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，如存储磁盘的类型或 Intel CPU 指令集扩展。实例可以指定其所需的特征。

您可以指定的特征在 `os-traits` 库中定义。特征示例包括：

- `COMPUTE_TRUSTED_CERTS`
- `COMPUTE_NET_ATTACH_INTERFACE_WITH_TAG`
- `COMPUTE_IMAGE_TYPE_RAW`
- `HW_CPU_X86_AVX`
- `HW_CPU_X86_AVX512VL`
- `HW_CPU_X86_AVX512CD`

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

使用下表中的 `property` 键来定义实例的资源特征。

表 3.22. 资源特征的类别元数据

键	描述
<code>trait:&lt;trait_name&gt;</code>	<p>指定 Compute 节点特征。将 <code>trait</code> 设置为以下有效值之一：</p> <ul style="list-style-type: none"> <li>• <b>必需</b>：选择托管实例的 Compute 节点必须具有特征。</li> <li>• <b>禁止</b>：选择用于托管实例的 Compute 节点不能有特征。</li> </ul> <p>例如：</p> <pre>\$ openstack flavor set --property trait:HW_CPU_X86_AVX512BW=required avx512- flavor</pre>

### 实例裸机资源类

使用下表中的 property 键为实例请求裸机资源类。

表 3.23. 裸机资源类的类别元数据

键	描述
<p><b>resources:&lt;resource_class_name&gt;</b></p>	<p>使用此属性指定标准裸机资源类来覆盖的值，或者指定实例所需的自定义裸机资源类。</p> <p>您可以覆盖的标准资源类是 <b>VCPU</b>、<b>MEMORY_MB</b> 和 <b>DISK_GB</b>。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 <b>0</b>。</p> <p>自定义资源类的名称必须以 <b>CUSTOM_</b> 开头。要确定与 Bare Metal 服务节点的资源类对应的自定义资源类的名称，请将资源类转换为大写，将所有标点替换为下划线，并将前缀替换为 <b>CUSTOM_</b>。</p> <p>例如，要在具有 <b>--resource-class baremetal.SMALL</b> 的节点上调度实例，请创建以下类别：</p> <pre data-bbox="687 857 1353 1093">\$ openstack flavor set \   --property   resources:CUSTOM_BAREMETAL_SMALL=1 \   --property resources:VCPU=0 --property   resources:MEMORY_MB=0 \   --property resources:DISK_GB=0 compute-small</pre>

## 第 4 章 在 COMPUTE 节点上配置 CPU

作为云管理员，您可以通过创建自定义类别以目标专用工作负载（包括 NFV 和高性能计算(HPC)）来配置实例的调度和放置，以获得最佳性能。

使用以下功能调整实例的最佳 CPU 性能：

- **CPU 固定**：将虚拟 CPU 固定到物理 CPU。
- **仿真程序线程**：与物理 CPU 实例关联的仿真程序线程。
- **CPU 功能标志**：配置应用于实例的标准 CPU 功能标记，以提高 Compute 节点的实时迁移兼容性。

### 4.1. 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，没有其他实例 vCPU 进程可以使用。在启用了 CPU 固定的 Compute 节点上运行的实例具有 NUMA 拓扑。实例 NUMA 拓扑的每个 NUMA 节点都映射到主机 Compute 节点上的 NUMA 节点。

您可以配置计算调度程序，以在同一 Compute 节点上使用专用（固定）CPU 和实例来调度实例。要在具有 NUMA 拓扑的 Compute 节点上配置 CPU 固定，您必须完成以下操作：

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



#### 注意

配置 CPU 固定会在实例上创建一个隐式 NUMA 拓扑，即使未请求 NUMA 拓扑。不要在同一主机上运行 NUMA 和非 NUMA 虚拟机(VM)。如需更多信息，[请参阅使用 NUMA 时限制](#)。

#### 4.1.1. 先决条件

- 您知道 Compute 节点的 NUMA 拓扑。
- 您已在 Compute 节点上配置了 **NovaReservedHugePages**。如需更多信息，[请参阅在 Compute 节点上配置巨页](#)。

#### 4.1.2. 为 CPU 固定设计 Compute 节点

要为带有固定 CPU 的实例指定 Compute 节点，您必须创建一个新的角色文件来配置 CPU 固定角色，并使用 CPU 固定资源类配置裸机节点，以使用标记 CPU 固定的 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调配的现有 overcloud 节点，您必须使用缩减流程取消置备节点，然后使用扩展步骤使用新资源类分配重新置备节点。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. Open **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
role 注释	<b>Role: Compute</b>	<b>Role: ComputeCPUPinning</b>
角色名称	<b>Name: Compute</b>	<b>名称 : ComputeCPUPinning</b>
<b>description</b>	<b>基本 Compute 节点角色</b>	<b>CPU 固定 Compute 节点角色</b>
<b>HostnameFormatDefault</b>	<b>%stackname%-novacompute-%index%</b>	<b>%stackname%-novacomputepinning-%index%</b>
<b>deprecated_nic_config_name</b>	<b>compute.yaml</b>	<b>compute-cpu-pinning.yaml</b>

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，以注册它们。有关更多信息，请参阅 [安装和管理 Red Hat OpenStack Platform](#) 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 [安装和管理 Red Hat OpenStack Platform](#) 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 ❶
```

- ❶ 您可以重复使用现有网络拓扑，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 [使用 director 安装和管理 Red Hat OpenStack Platform 指南中的自定义网络接口模板](#)。如果您没有使用 **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](#)。有关节点定义文件的示例，请参阅 [节点定义文件示例](#)。

9. 运行 provisioning 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。如果您没有使用 **network\_config** 属性定义网络定义，则使用默认网络定义。

10. 在一个单独的终端中监控置备进度。当置备成功时，节点状态将从 **available** 变为 **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。

### 4.1.3. 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。为主机进程为所有 NUMA 节点保留一些 CPU 内核，以获得效率。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 4.1. NUMA 拓扑示例

NUMA 节点 0		NUMA 节点 1	
Core 0	核心 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，以配置 Compute 节点，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** 添加到计算环境文件中的 **NovaSchedulerEnabledFilters** 参数中（如果尚不存在）：

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - NUMATopologyFilter
```

如需有关 **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
```

- 如果您不使用文件支持的内存，请为主机进程指定要保留的 RAM 量：

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

将 **<ram>** 替换为要保留的 RAM 量（以 MB 为单位）。

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

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

使用以逗号分开的 CPU 索引的列表或范围指定 **IsolCpusList** 参数的值。

- 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
```

#### 4.1.4. 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，您可以创建一个具有可启动实例的专用 CPU 策略的类别。

##### 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

- 获取 **overcloudrc** 文件：

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

- 为需要 CPU 固定的实例创建类别：

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

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

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

4. 如果您不使用文件支持的内存，请将类别的 **hw:mem\_page\_size** 属性设置为启用 NUMA 感知内存分配：

```
(overcloud)$ openstack flavor set \
  --property hw:mem_page_size=<page_size> pinned_cpus
```

- 将 **<page\_size>** 替换为以下有效值之一：
  - **大**：选择主机上支持的最大页面大小，其在 x86\_64 系统上可能为 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 系统上，这是 4 kB（常规页面）。
  - **any**：使用镜像中设置的 **hw\_mem\_page\_size** 选择页大小。如果镜像没有指定页面大小，请选择由 libvirt 驱动程序决定的最大可用页面大小。
  - **<page\_size>**：如果工作负载具有特定要求，则设置显式页面大小。将整数值用于页大小（以 KB 为单位）或任何标准后缀。例如：4KB、2MB、2048、1GB。



### 注意

要将 **hw:mem\_page\_size** 设置为 **small** 或 **any**，您必须已配置了内存页面数量，以便为不是实例的进程在每个 NUMA 节点上保留。如需更多信息，[请参阅在 Compute 节点上配置巨页](#)。

5. 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 架构或足够具有可用线程的 CPU 内核，则调度会失败。要防止这种情况，将 **hw:cpu\_thread\_policy** 设置为 **prefer** 而不是 **require**。**prefer** 策略是默认策略，可确保在可用时使用线程同级。
- 如果使用 **hw:cpu\_thread\_policy=isolate**，则必须禁用 SMT，或使用不支持 SMT 的平台。

## 验证

1. 要验证该类别创建了专用 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
```

## 4.1.5. 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 策略来创建启动实例。

### 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
```

4. 如果您不使用文件支持的内存，请将类别的 **hw:mem\_page\_size** 属性设置为启用 NUMA 感知内存分配：

```
(overcloud)$ openstack flavor set \
--property hw:mem_page_size=<page_size> pinned_cpus
```

- 将 **<page\_size>** 替换为以下有效值之一：
  - **大**：选择主机上支持的最大页面大小，其在 x86\_64 系统上可能为 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 系统上，这是 4 kB（常规页面）。
  - **any**：使用镜像中设置的 **hw\_mem\_page\_size** 选择页大小。如果镜像没有指定页面大小，请选择由 libvirt 驱动程序决定的最大可用页面大小。
  - **<pageSize>**：如果工作负载具有特定要求，则设置显式页面大小。将整数值用于页大小（以 KB 为单位）或任何标准后缀。例如：4KB、2MB、2048、1GB。



#### 注意

要将 **hw:mem\_page\_size** 设置为 **small** 或 **any**，您必须已配置了内存页面数量，以便为不是实例的进程在每个 NUMA 节点上保留。如需更多信息，[请参阅在 Compute 节点上配置巨页](#)。

### 4.1.6. 为实例创建混合 CPU 类别

要让您的云用户创建混合有专用和共享 CPU 的实例，您可以创建一个具有启动实例混合 CPU 策略的类别。

### 流程

1. 获取 **overcloudrc** 文件：

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

2. 为需要混合专用和共享 CPU 的实例创建类别：

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

3. 指定哪些 CPU 必须专用或共享：

```
(overcloud)$ openstack flavor set \
--property hw:cpu_dedicated_mask=<CPU_number> \
mixed_CPUs_flavor
```

- 将 **<CPU\_number>** 替换为必须专用或共享的 CPU：
  - 要指定专用 CPU，请指定 CPU 号或 CPU 范围。例如，将属性设置为 **2-3**，以指定 CPU 2 和 3 专用，并且所有剩余的 CPU 都共享。
  - 要指定共享 CPU，请在 CPU 号或 CPU 范围前加上一个 caret (^)。例如，将属性设置为 **^0-1**，以指定 CPU 0 和 1 被共享，并且所有剩余的 CPU 都专用。

4. 如果您不使用文件支持的内存，请将类别的 **hw:mem\_page\_size** 属性设置为启用 NUMA 感知内存分配：

```
(overcloud)$ openstack flavor set \
--property hw:mem_page_size=<page_size> pinned_cpus
```

- 将 **<page\_size>** 替换为以下有效值之一：
  - **大**：选择主机上支持的最大页面大小，其在 x86\_64 系统上可能为 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 系统上，这是 4 kB（常规页面）。
  - **any**：使用镜像中设置的 **hw\_mem\_page\_size** 选择页大小。如果镜像没有指定页面大小，请选择由 libvirt 驱动程序决定的最大可用页面大小。
  - **<pageSize>**：如果工作负载具有特定要求，则设置显式页面大小。将整数值用于页大小（以 KB 为单位）或任何标准后缀。例如：4KB、2MB、2048、1GB。



### 注意

要将 **hw:mem\_page\_size** 设置为 **small** 或 **any**，您必须已配置了内存页面数量，以便为不是实例的进程在每个 NUMA 节点上保留。如需更多信息，[请参阅在 Compute 节点上配置巨页](#)。

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

如果 Compute 节点支持并发多线程(SMT)，在专用或共享集中组线程同级在一起。线程同级了一些常见的硬件，这意味着可以在一个线程上运行的进程，从而影响其他线程同级的性能。

例如，主机标识了双核 CPU 中的四个逻辑 CPU 内核，带有 SMT : 0、1、2 和 3。在这四个中，有两对线程同级：

- 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，不要将逻辑 CPU 内核 0 和 1 分配为专用，2 和 3 作为共享。相反，将 0 和 2 分配为专用，并将 1 和 3 分配为共享。

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

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

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

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

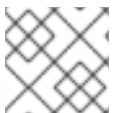
#### 4.1.8. 其他资源

- [发现 NUMA 节点拓扑](#)

## 4.2. 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，以用于实例使用的不同 CPU。



#### 注意

- 为避免数据包丢失，不得先抢占 NFV 部署中的 vCPU。

#### 先决条件

- 必须启用 CPU 固定。

#### 流程

1. 以 `stack` 用户的身份登录 `undercloud`。
2. 打开 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，以下配置在具有 32 核 CPU 的 Compute 节点上设置专用 CPU：

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



如需更多信息，请参阅在 [Compute 节点上配置 CPU 固定](#)。

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

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



### 注意

计算调度程序还将共享集中的 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 **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](#)。

## 4.3. 为实例配置 CPU 功能标记

您可以为实例启用或禁用 CPU 功能标记，而无需更改主机 Compute 节点上的设置并重启 Compute 节点。通过配置应用到实例的标准 CPU 功能标记，您可以帮助实现跨 Compute 节点的实时迁移兼容性。您还有助于通过禁用对具有特定 CPU 模型的实例的安全性或性能造成负面影响的标记，或者启用提供安全问题或性能问题的缓解的标记，从而帮助管理实例的性能和安全性。

### 4.3.1. 先决条件

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

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

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

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



将 `<arch>` 替换为架构的名称，如 `x86_64`。

### 4.3.2. 为实例配置 CPU 功能标记

配置计算服务，以将 CPU 功能标志应用到具有特定 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 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 模式自动向实例添加关键 CPU 标记，以提供对安全漏洞的缓解方案。
- **自定义**：使用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 模型和功能标志。

5. 可选：如果您将 **NovaLibvirtCPUMode** 设置为 **custom**，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 模型的逗号分隔列表。按顺序列出 CPU 型号，首先将更常见和不太高级 CPU 型号放在列表中，而功能丰富的 CPU 模型最后是 **SandyBridge,IvyBridge,Haswell**。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 节点架构的名称，如 `x86_64`。

6. 为使用指定 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。前缀每个带有 "+" 的标记来启用标志，或者 "-" 来禁用它。如果没有指定前缀，则会启用标志。有关给定 CPU 模型的可用功能标记列表，请参阅 `/usr/share/libvirt/cpu_mapVRF.xml`。

以下示例为 **IvyBridge** 和 **Cascadelake-Server** 模型启用 CPU 功能标记 **pcid** 和 **ssbd**，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 5 章 在 COMPUTE 节点上配置内存

作为云管理员，您可以通过创建自定义类别以目标专用工作负载（包括 NFV 和高性能计算(HPC)）来配置实例的调度和放置，以获得最佳性能。

使用以下功能调整实例以优化内存性能：

- **Overallocation:** 将虚拟 RAM 连接至物理 RAM 分配比率。
- **swap**：取消分配的 swap 大小，以处理内存过量使用。
- **巨页**：对于普通内存(4k 页)和巨页(2 MB 或 1 GB 页面)实例内存分配策略。
- **文件支持的内存**：使用扩展您的 Compute 节点内存容量。
- **SEV**：使用您的云用户创建使用内存加密的实例。

### 5.1. 为超额分配配置内存

使用内存过量使用(**NovaRAMAllocationRatio**  $\geq$  1.0)时，您需要部署具有足够交换空间的 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 建议 swap 大小进行操作。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 swap 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. 使用其他环境文件将 **enable\_swap.yaml** 环境文件添加到堆栈中，并部署 overcloud：

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

### 5.2. 计算 COMPUTE 节点上的保留主机内存

要确定要为主机进程保留的 RAM 总量，您需要为以下每个进程分配足够的内存：

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

计算内存的额外要求后，使用以下公式来帮助确定要为每个节点上的主机进程保留的内存量：

$$\text{NovaReservedHostMemory} = \text{total\_RAM} - (\text{vm\_no} * (\text{avg\_instance\_size} + \text{overhead})) + (\text{resource1} * \text{resource\_ram}) + (\text{resourcen} * \text{resource\_ram})$$

- 将 **vm\_no** 替换为实例数量。
- 将 **avg\_instance\_size** 替换为每个实例可以使用的平均内存量。
- 使用每个实例所需的虚拟机监控程序 **开销** 替换开销。
- 将 **resource1** 以及最多 **<resourcen >** 的所有资源替换为节点上资源类型的数量。
- 将 **resource\_ram** 替换为此类型所需的每个资源的 RAM 量。

### 5.3. 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来计算节点所需的 swap 大小：

- $\text{overcommit\_ratio} = \text{NovaRAMAllocationRatio} - 1$
- 最小 swap 大小(MB) =  $(\text{total\_RAM} * \text{overcommit\_ratio}) + \text{RHEL\_min\_swap}$
- 推荐的(maximum) swap size (MB) =  $\text{total\_RAM} * (\text{overcommit\_ratio} + \text{percentage\_of\_RAM\_to\_use\_for\_swap})$

**percentage\_of\_RAM\_to\_use\_for\_swap** 变量创建一个缓冲区来考虑 QEMU 开销以及操作系统或主机服务消耗的任何其他资源。

例如，要将可用 RAM 的 25% 用于交换，内存为 64GB，**NovaRAMAllocationRatio** 设置为 1：

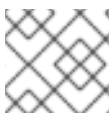
- 推荐的（最大）交换大小 =  $64000 \text{ MB} * (0 + 0.25) = 16000 \text{ MB}$

有关如何计算 **NovaReservedHostMemory** 值的详情，请参考在 [Compute 节点上计算保留的主机内存](#)。

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 swap 空间](#)。

### 5.4. 在 COMPUTE 节点上配置巨页

作为云管理员，您可以配置 Compute 节点，以启用实例来请求巨页。



#### 注意

配置巨页会在实例上创建一个隐式 NUMA 拓扑，即使未请求 NUMA 拓扑。

## 流程

1. 打开 Compute 环境文件。
2. 将巨页内存量配置为在每个 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 节点使用 OVS 的巨页数量。例如，对于 Open vSwitch 使用的 4096 个套接字内存，将其设置为 2。
3. 在 Compute 节点上配置巨页：

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



### 注意

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

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

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



## 注意

- CPU 功能标志不需要配置为允许实例仅请求 2 MB 巨页。
- 当主机支持 1G 巨页分配时，您只能将 1G 巨页分配给实例。
- 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 **NovaLibvirtCPUModel**，则不需要将 **pdpe1gb** 设置为 **NovaLibvirtCPUModelExtraFlags**。**pdpe1gb** 标志仅包含在 Opteron\_G4 和 Opteron\_G5 CPU 模型中，它不包含在 QEMU 支持的任何 Intel CPU 模型中。
- 要缓解 CPU 硬件问题，如 Microarchitectural Data Sampling (MDS)，您可能需要配置其他 CPU 标记。如需更多信息，请参阅 [RHOS Mitigation for MDS \("Microarchitectural Data Sampling"\)安全 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:
    NovaLibvirtCPUModel: 'custom'
    NovaLibvirtCPUModels: 'Haswell-noTSX'
    NovaLibvirtCPUModelExtraFlags: 'vmx, pdpe1gb, +pcid'
```

## 提示

如需更多信息，请参阅使用 "PCID" CPU 功能标记为 OpenStack 客户机提供 [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
```

### 5.4.1. 为实例创建巨页类型

要让您的云用户创建使用巨页的实例，您可以创建一个带有 **hw:mem\_page\_size** 额外 spec 键的类别，以用于启动实例。

#### 先决条件

- Compute 节点被配置为巨页。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，请将类别的 **hw:mem\_page\_size** 属性设置为所需的大小：

```
$ openstack flavor set huge_pages --property hw:mem_page_size=<page_size>
```

- 将 **<page\_size>** 替换为以下有效值之一：
  - **大**：选择主机上支持的最大页面大小，其在 x86\_64 系统上可能为 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 系统上，这是 4 kB（常规页面）。
  - **any**：使用镜像中设置的 **hw\_mem\_page\_size** 选择页大小。如果镜像没有指定页面大小，请选择由 libvirt 驱动程序决定的最大可用页面大小。
  - **<pageSize>**：如果工作负载具有特定要求，则设置显式页面大小。将整数值用于页大小（以 KB 为单位）或任何标准后缀。例如：4KB、2MB、2048、1GB。

3. 要验证类别会创建一个带有巨页的实例，请使用您的新类别启动实例：

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

计算调度程序标识了具有足够可用巨页的主机，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。

### 5.4.2. 第一次引导过程中挂载多个巨页文件夹

您可以配置 Compute 服务(nova)，作为第一次引导过程的一部分处理多个页面大小。第一次引导过程会在首次启动节点时将 heat 模板配置添加到所有节点。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**，该脚本将运行脚本来为巨页文件夹创建挂载。您可以使用 **OS::TripleO::MultipartMime** 资源类型来发送配置脚本：

```
heat_template_version: <version>

description: >
  Huge pages configuration

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

  hugepages_config:
    type: OS::Heat::SoftwareConfig
    properties:
      config: |
```

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

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

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

outputs:

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

此模板中的 **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，为上的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，以启用使用页面大小创建新挂载。
  - c. 确保首先创建文件夹。
  - d. 为每个 **页大小** 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 **pagesize** 启用每个挂载。您可以根据需要更新此循环以包含额外的 **pagesize**。
2. 可选：**/dev** 文件夹会自动绑定到 **nova\_compute** 和 **nova\_libvirt** 容器。如果您已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
```

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

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



### 重要

您只能将 **NodeUserData** 资源注册到每个资源的一个 heat 模板。后续用法会覆盖要使用的 heat 模板。

- 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 \
...
```

## 5.5. 配置 COMPUTE 节点，将文件支持的内存用于实例

您可以通过将 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 Compute 节点内存容量。您可以配置可用于实例内存的主机磁盘量，以及实例内存文件磁盘上的位置。

计算服务会将为文件支持内存配置的容量报告到放置服务，作为系统内存总量。这使得 Compute 节点可以托管比通常适合系统内存更多的实例。

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

### 限制

- 您无法在启用了文件支持内存的 Compute 节点之间实时迁移实例，以及未启用文件支持的 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的内存的 Compute 节点上启动。使用主机聚合来确保使用巨页的实例不会被放在启用了文件支持的内存的 Compute 节点上。
- 文件支持的内存与内存过量使用不兼容。
- 您不能使用 **NovaReservedHostMemory** 为主机进程保留内存。当使用文件支持的内存时，保留内存对应于不为文件支持的内存设置磁盘空间。文件支持的内存作为总系统内存报告给放置服务，RAM 用作缓存内存。

### 先决条件

- 节点上的 **NovaRAMAllocationRatio** 必须设置为 "1.0"，并将该节点添加到任何主机聚合。
- NovaReservedHostMemory** 必须设为 "0"。

## 流程

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

```
parameter_defaults:
  NovaLibvirtFileBackedMemory: 102400
```

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



### 注意

您必须在默认目录位置 **/var/lib/libvirt/qemu/ram/** 的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. 保存对 Compute 环境文件的更新。
5. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

## 5.5.1. 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备用磁盘中。

## 流程

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

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

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

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



### 注意

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

## 5.6. 配置 AMD SEV COMPUTE 节点，为实例提供内存加密

作为云管理员，您可以为云用户提供创建在支持 SEV 的 Compute 节点上运行的实例，并启用了内存加密。

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

要让您的云用户创建使用内存加密的实例，您必须执行以下任务：

1. 为内存加密指定 AMD SEV Compute 节点。
2. 配置 Compute 节点以进行内存加密。
3. 部署 overcloud。
4. 创建类别或镜像以使用内存加密启动实例。

## 提示

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

### 5.6.1. 安全加密的虚拟化(SEV)

由 AMD 提供的安全加密虚拟化(SEV)保护运行虚拟机实例使用的 DRAM 中的数据。SEV 使用唯一密钥加密每个实例的内存。

当使用非易失性内存技术(NVDIMM)时 SEV 会增加安全性，因为 NVDIMM 芯片可以物理地从具有数据的系统中删除，这与硬盘类似。如果没有加密，任何存储的信息（如敏感数据、密码或密钥）都可能被破坏。

如需更多信息，请参阅 [AMD 安全加密虚拟化\(SEV\)](#) 文档。

#### 带有内存加密的实例的限制

- 您无法实时迁移，或使用内存加密挂起和恢复实例。
- 您不能使用 PCI 透传直接访问带有内存加密的实例的设备。
- 您不能使用 **virtio-blk** 作为带有比 kernel-4.18.0-115.el8 (RHEL-8.1.0)更早的 Red Hat Enterprise Linux (RHEL)内核进行内存加密的实例的引导磁盘。



#### 注意

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

- 在加密实例中运行的操作系统必须提供 SEV 支持。如需更多信息，请参阅红帽知识库解决方案在 [RHEL 8 中启用 AMD 安全加密虚拟化](#)。
- 支持 SEV 的机器在其内存控制器中有有限数量的插槽来存储加密密钥。每个带有加密内存的实例都会消耗其中一个插槽。因此，可以同时运行的内存加密的实例数量限制为内存控制器中的插槽数量。例如，在 1st Gen AMD EPYC™ 7001 系列("Naples")中，这个限制为 16，在第二代 Gen AMD EPYC™ 7002 系列("Rome")时，这个限制为 255。
- 带有内存加密固定页面的实例。Compute 服务无法交换这些页面，因此您无法在托管内存加密的实例的 Compute 节点上过量使用内存。
- 您不能将内存加密用于具有多个 NUMA 节点的实例。

## 5.6.2. 为内存加密设计 AMD SEV Compute 节点

要为使用内存加密的实例指定 AMD SEV Compute 节点，您必须创建一个新角色文件来配置 AMD SEV 角色，并使用 AMD SEV 资源类配置裸机节点，以标记用于内存加密的 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调配的现有 overcloud 节点，您必须使用缩减流程取消置备节点，然后使用扩展步骤使用新资源类分配重新置备节点。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. 生成包含 **ComputeAMDSEV** 角色的新角色数据文件，以及 overcloud 所需的任何其他角色。以下示例生成角色数据文件 **roles\_data\_amd\_sev.yaml**，其中包括角色 **Controller** 和 **ComputeAMDSEV**：

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

4. Open **roles\_data\_amd\_sev.yaml** 并编辑或添加以下参数和部分：

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

5. 将 overcloud 的 AMD SEV 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 [安装和管理 Red Hat OpenStack Platform 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 [安装和管理 Red Hat OpenStack Platform](#) 指南中的创建 [裸机节点硬件清单](#)。

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

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

将 **<node>** 替换为裸机节点的名称或 ID。

- 将 **ComputeAMDSEV** 角色添加到您的节点定义文件 **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: ComputeAMDSEV
  count: 1
  defaults:
    resource_class: baremetal.AMD-SEV
    network_config:
      template: /home/stack/templates/nic-config/myRoleTopology.j2 1
```

- 1** 您可以重复使用现有网络拓扑，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 [使用 director 安装和管理 Red Hat OpenStack Platform](#) 指南中的 [自定义网络接口模板](#)。如果您没有使用 **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](#)。有关节点定义文件的示例，请参阅 [节点定义文件示例](#)。

- 运行 provisioning 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。如果您没有使用 **network\_config** 属性定义网络定义，则使用默认网络定义。

- 在一个单独的终端中监控置备进度。当置备成功时，节点状态将从 **available** 变为 **active**：

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

- 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
  ComputeAMDSEVNetworkConfigTemplate: /home/stack/templates/nic-
  configs/<amd_sev_net_top>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

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

### 5.6.3. 为内存加密配置 AMD SEV Compute 节点

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



#### 注意

从 RHOSP OSP17.0 开始，Q35 是默认的机器类型。Q35 机器类型使用 PCIe 端口。您可以通过配置 heat 参数 **NovaLibvirtNumPciePorts** 来管理 PCIe 端口设备的数量。可附加到 PCIe 端口的设备数量比之前版本上运行的实例要少。如果要使用更多设备，则必须使用 **hw\_disk\_bus=scsi** 或 **hw\_scsi\_model=virtio-scsi** 镜像属性。如需更多信息，请参阅 [虚拟硬件的元数据属性](#)。

#### 先决条件

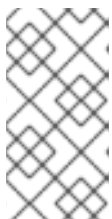
- 您的部署必须包含在支持 SEV 的 AMD 硬件中运行的 Compute 节点，如 AMD EPYC CPU。您可以使用以下命令来确定部署是否支持 SEV：

```
$ lscpu | grep sev
```

#### 流程

1. 打开 Compute 环境文件。
2. 可选：在 Compute 环境文件中添加以下配置，以指定 AMD SEV Compute 节点可同时托管的最大内存加密实例数：

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



#### 注意

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

3. 可选：要指定所有 x86\_64 镜像默认使用 q35 机器类型，请在 Compute 环境文件中添加以下配置：

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

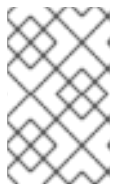
如果指定了此参数值，则不需要在每个 AMD SEV 实例镜像上设置 `hw_machine_type` 属性设置为 `q35`。

4. 为确保 AMD SEV Compute 节点为主机级服务保留足够的内存才能正常工作，请为每个潜在的 AMD SEV 实例添加 16MB：

```
parameter_defaults:
  ComputeAMDSEVParameters:
    ...
    NovaReservedHostMemory: <libvirt/num_memory_encrypted_guests * 16>
```

5. 为 AMD SEV Compute 节点配置内核参数：

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



### 注意

当您首先将 **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，您可以禁用自动重新引导节点，并在每个 overcloud 部署后手动重启节点。如需更多信息，请参阅[配置手动节点重新引导以定义 KernelArgs](#)。

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

```
(undercloud)$ openstack overcloud deploy --templates \
-e [your environment files] \
-r /home/stack/templates/roles_data_amd_sev.yaml \
-e /home/stack/templates/network-environment.yaml \
-e /home/stack/templates/<compute_environment_file>.yaml \
-e /home/stack/templates/overcloud-baremetal-deployed.yaml \
-e /home/stack/templates/node-info.yaml
```

#### 5.6.4. 为内存加密创建镜像

当 overcloud 包含 AMD SEV Compute 节点时，您可以创建一个 AMD SEV 实例镜像，您的云用户可以使用该镜像来启动具有内存加密的实例。



### 注意

从 RHOSP OSP17.0 开始，Q35 是默认的机器类型。Q35 机器类型使用 PCIe 端口。您可以通过配置 heat 参数 **NovaLibvirtNumPciePorts** 来管理 PCIe 端口设备的数量。可附加到 PCIe 端口的设备数量比之前版本上运行的实例要少。如果要使用更多设备，则必须使用 **hw\_disk\_bus=scsi** 或 **hw\_scsi\_model=virtio-scsi** 镜像属性。如需更多信息，请参阅[虚拟硬件的元数据属性](#)。



## 流程

1. 为内存加密创建新镜像：

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



### 注意

如果使用现有镜像，镜像必须将 **hw\_firmware\_type** 属性设置为 **uefi**。

2. 可选：在镜像中添加 property **hw\_mem\_encryption=True**，以便在镜像上启用 AMD SEV 内存加密：

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

### 提示

您可以在类别上启用内存加密。如需更多信息，[请参阅为内存加密创建类别](#)。

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

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

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

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

### 提示

您还可以在类别中指定此特征。如需更多信息，[请参阅为内存加密创建类别](#)。

## 5.6.5. 为内存加密创建类别

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



### 注意

只有在镜像中没有设置 **hw\_mem\_encryption** 属性时，才需要 AMD SEV 类别。

## 流程

1. 为内存加密创建类别：

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



2. 要在支持 SEV 的主机聚合上调度内存加密实例，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_AMD_SEV=required m1.small-amd-sev
```

### 5.6.6. 使用内存加密启动实例

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

#### 流程

1. 使用 AMD SEV 类别或镜像创建实例。以下示例使用在为内存加密创建类别时创建的类别，以及在创建用于内存加密的镜像的镜像来创建实例：

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

2. 以云用户身份登录实例。
3. 要验证实例是否使用内存加密，请在实例中输入以下命令：

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

## 第 6 章 配置计算服务存储

您可以从基础镜像创建实例，该镜像从 Image (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，以便在主机 Compute 节点上本地存储临时实例磁盘数据，或者在 NFS 共享或 Ceph 集群中远程存储。另外，您还可以配置 Compute 服务，将实例磁盘数据存储于块存储(Cinder)服务提供的持久性存储中。

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

### 6.1. 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 Compute 节点上如何实施和管理镜像缓存。

表 6.1. Compute (nova)服务镜像缓存参数

配置方法	参数	描述
puppet	<code>nova::compute::image_cache::manager_interval</code>	<p>指定在镜像缓存管理器运行之间等待的秒数，用于管理 Compute 节点上的基础镜像缓存。当 <code>nova::compute::image_cache::remove_unused_base_images</code> 设置为 <code>True</code> 时，计算服务使用此周期来自动移除未使用的缓存镜像。</p> <p>设置为 <code>0</code>，以默认指标间隔为 60 秒（不推荐）运行。设置为 <code>-1</code> 以禁用镜像缓存管理器。</p> <p>默认：<code>2400</code></p>
puppet	<code>nova::compute::image_cache::precache_concurrency</code>	<p>指定可以并行预缓存镜像的最大 Compute 节点数量。</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p><b>注意</b></p> <ul style="list-style-type: none"> <li>将此参数设置为高数值可能会导致缓存性能较慢，并可能导致镜像服务的 DDoS。</li> <li>将此参数设置为低数值可减少镜像服务的负载，但可能会导致较长的运行时完成，因为预缓存被执行成更连续的操作。</li> </ul> </div> </div> <p>默认：<code>1</code></p>

配置方法	参数	描述
puppet	<b>nova::compute::image_cache::remove_unused_base_images</b>	<p>设置为 <b>True</b>，以根据使用 <b>manager_interval</b> 配置的间隔自动从缓存中移除未使用的基础镜像。如果镜像在 <b>NovalImageCacheTTL</b> 指定期间没有访问，则镜像被定义为未使用。</p> <p>Default: <b>True</b></p>
puppet	<b>nova::compute::image_cache::remove_unused_resized_minimum_age_seconds</b>	<p>指定未使用的大小基础镜像的最短期限（以秒为单位）。比此年轻而未使用的基础镜像调整了大小，则不会被删除。设置为 <b>undef</b> 以禁用。</p> <p>默认： <b>3600</b></p>
puppet	<b>nova::compute::image_cache::subdirectory_name</b>	<p>指定存储缓存镜像的文件夹名称，相对于 <b>\$instances_path</b>。</p> <p>默认： <b>_base</b></p>
Heat	<b>NovalImageCacheTTL</b>	<p>指定 Compute 服务在不再供 Compute 节点上的任何实例使用时，Compute 服务应继续缓存镜像的时间长度（以秒为单位）。Compute 服务从缓存目录中删除超过这个配置生命周期的 Compute 节点上缓存的镜像，直到再次需要它们。</p> <p>默认： 86400 (24 小时)</p>

## 6.2. 实例临时存储属性的配置选项

使用下表中详述的参数配置实例使用的临时存储的性能和安全性。




### 注意

Red Hat OpenStack Platform (RHOSP) 不支持实例磁盘的 LVM 镜像类型。因此，**[libvirt]/volume\_clear** 配置选项（当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 时应用。

表 6.2. 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	<b>nova::compute::default_ephemeral_format</b>	<p>指定用于新临时卷的默认格式。设置为以下有效值之一：</p> <ul style="list-style-type: none"> <li>● <b>ext2</b></li> <li>● <b>ext3</b></li> <li>● <b>ext4</b></li> </ul> <p><b>ext4</b> 格式提供比 <b>ext3</b> 对于新的大磁盘的初始化速度要快得多。</p> <p>默认：<b>ext4</b></p>
puppet	<b>nova::compute::force_raw_images</b>	<p>设置为 <b>True</b>，将非原始缓存的基础镜像转换为 <b>raw</b> 格式。<b>raw</b> 镜像格式使用比其他镜像格式更多的空间，如 <b>qcow2</b>。非原始镜像格式使用更多 CPU 进行压缩。当设置为 <b>False</b> 时，计算服务会在压缩过程中从基础镜像中删除任何压缩，以避免 CPU 瓶颈。如果您有一个有较慢的 I/O 或低可用空间的系统来减少输入带宽，则设置为 <b>False</b>。</p> <p>Default: <b>True</b></p>
puppet	<b>nova::compute::use_cow_images</b>	<p>设置为 <b>True</b>，为实例磁盘使用 <b>qcow2</b> 格式的 CoW（在 Write 上复制）镜像。使用 CoW 时，根据后备存储和主机缓存，通过让每个实例在自己的副本上运行时，可能会获得更好的并发性。</p> <p>设置为 <b>False</b>，以使用 <b>raw</b> 格式。<b>raw</b> 格式对磁盘镜像的通用部分使用更多空间。</p> <p>Default: <b>True</b></p>
puppet	<b>nova::compute::libvirt::preallocate_images</b>	<p>指定实例磁盘的预分配模式。设置为以下有效值之一：</p> <ul style="list-style-type: none"> <li>● <b>none</b> - 实例启动时没有置备存储。</li> <li>● <b>space</b> - Compute 服务首先在实例磁盘镜像上运行 <b>fallocate (1)</b> 来完全分配存储。这可减少 CPU 开销和文件碎片，提高了 I/O 性能，并有助于保证所需的磁盘空间。</li> </ul> <p>Default: <b>none</b></p>

配置方法	参数	描述
hieradata 覆盖	<b>DEFAULT/resize_fs_using_block_device</b>	<p>设置为 <b>True</b>，以通过块设备访问镜像来启用直接调整基础镜像的大小。这只适用于旧版本 cloud-init 的镜像无法调整自身大小。</p> <p>默认情况下不启用此参数，因为它启用了直接挂载镜像，因为安全原因可能会禁用这些镜像。</p> <p>Default: <b>False</b></p>
hieradata 覆盖	<b>[libvirt]/images_type</b>	<p>指定用于实例磁盘的镜像类型。设置为以下有效值之一：</p> <ul style="list-style-type: none"> <li>● <b>raw</b></li> <li>● <b>qcow2</b></li> <li>● <b>flat</b></li> <li>● <b>rbd</b></li> <li>● <b>default</b></li> </ul> <p> <b>注意</b></p> <p>RHOSP 不支持实例磁盘的 LVM 镜像类型。</p> <p>当设置了一个不是 <b>default</b> 的有效值时，镜像列表会取代 <b>use_cow_images</b> 的配置。如果指定了 <b>default</b>，则 <b>use_cow_images</b> 的配置决定了镜像类型：</p> <ul style="list-style-type: none"> <li>● 如果将 <b>use_cow_images</b> 设置为 <b>True</b>（默认），则镜像类型是 <b>qcow2</b>。</li> <li>● 如果 <b>use_cow_images</b> 设为 <b>False</b>，则镜像类型是 <b>Flat</b>。</li> </ul> <p>默认值由 <b>NovaEnableRbdBackend</b> 的配置决定：</p> <ul style="list-style-type: none"> <li>● <b>NovaEnableRbdBackend: False</b> 默认值：<b>default</b></li> <li>● <b>NovaEnableRbdBackend: True</b> Default: <b>rbd</b></li> </ul>

### 6.3. 配置要附加到一个实例的存储设备的最大数量

默认情况下，您可以将无限数量的存储设备附加到单个实例。将大量磁盘设备附加到实例可能会降低实例的性能。您可以根据您的环境可以支持的边界，调整实例的最大设备数量。实例支持的存储磁盘数量取决于磁盘使用的总线。例如，IDE 磁盘总线限制为 4 个附加的设备。您可以将最多 500 个磁盘设备附加到

类型为 Q35 的实例。



### 注意

从 RHOSP OSP17.0 开始，Q35 是默认的机器类型。Q35 机器类型使用 PCIe 端口。您可以通过配置 heat 参数 **NovaLibvirtNumPciePorts** 来管理 PCIe 端口设备的数量。可附加到 PCIe 端口的设备数量比之前版本上运行的实例要少。如果要使用更多设备，则必须使用 **hw\_disk\_bus=scsi** 或 **hw\_scsi\_model=virtio-scsi** 镜像属性。如需更多信息，请参阅 [虚拟硬件的元数据属性](#)。



### 警告

- 如果最大数量低于已附加到实例的设备数量，在具有活跃实例的 Compute 节点上更改 **NovaMaxDiskDevicesToAttach** 参数的值可能会导致重建失败。例如，如果实例 A 附加了 26 个设备，并且将 **NovaMaxDiskDevicesToAttach** 更改为 20，则重建实例 A 的请求将失败。
- 在冷迁移过程中，仅在您要迁移的实例的源上强制配置的最大存储设备数。移动前不会检查目的地。这意味着，如果 Compute 节点 A 具有 26 个附加的磁盘设备，且 Compute 节点 B 配置最多 20 个附加的磁盘设备，则从 Compute 节点 A 到 Compute 节点 B 的附加了 26 个设备的实例冷迁移。但是，在 Compute 节点 B 中重建实例的后续请求会失败，因为 26 个设备已连接超过配置的最大值 20。



### 注意

在 shelved 卸载实例上不会强制配置的最大存储设备数，因为它们没有 Compute 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：

```
$ source ~/stackrc
```

3. 创建新的环境文件，或打开现有的环境文件。
4. 通过在环境文件中添加以下配置，对可附加到单个实例的存储设备的最大数量配置限制：

```
parameter_defaults:
...
NovaMaxDiskDevicesToAttach: <max_device_limit>
...
```

- 将 **< max\_device\_limit >** 替换为可附加到实例的存储设备的最大数量。
5. 将更新保存到环境文件中。
  6. 使用其他环境文件将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/<environment_file>.yaml
```

## 6.4. 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 Compute 服务配置 NFS 存储后端，将这些实例文件存储在共享 NFS 存储上。

### 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 仅支持说明](#)。

### 流程

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

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

3. 创建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，请使用 double 和 single-quotes，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'`。添加以下参数来修改 NFS 实例文件存储挂载点的挂载选项：

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

使用您要用于 NFS 实例文件存储的挂载选项列表替换 `<additional_nfs_mount_options>`。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中。
7. 使用其他环境文件将新的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
```

## 6.5. 配置镜像直接从 RED HAT CEPH RADOS 块设备(RBD)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 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，请在计算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 >** 替换为用于在 **GlanceMultistoreConfig** 配置中指定后端的 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，以及计算服务等待连接到镜像服务 RBD 后端的最大时长，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，然后检查条目 "Attempting to export RBD image:"。

## 6.6. 其他资源

- [配置计算服务\(nova\)](#)

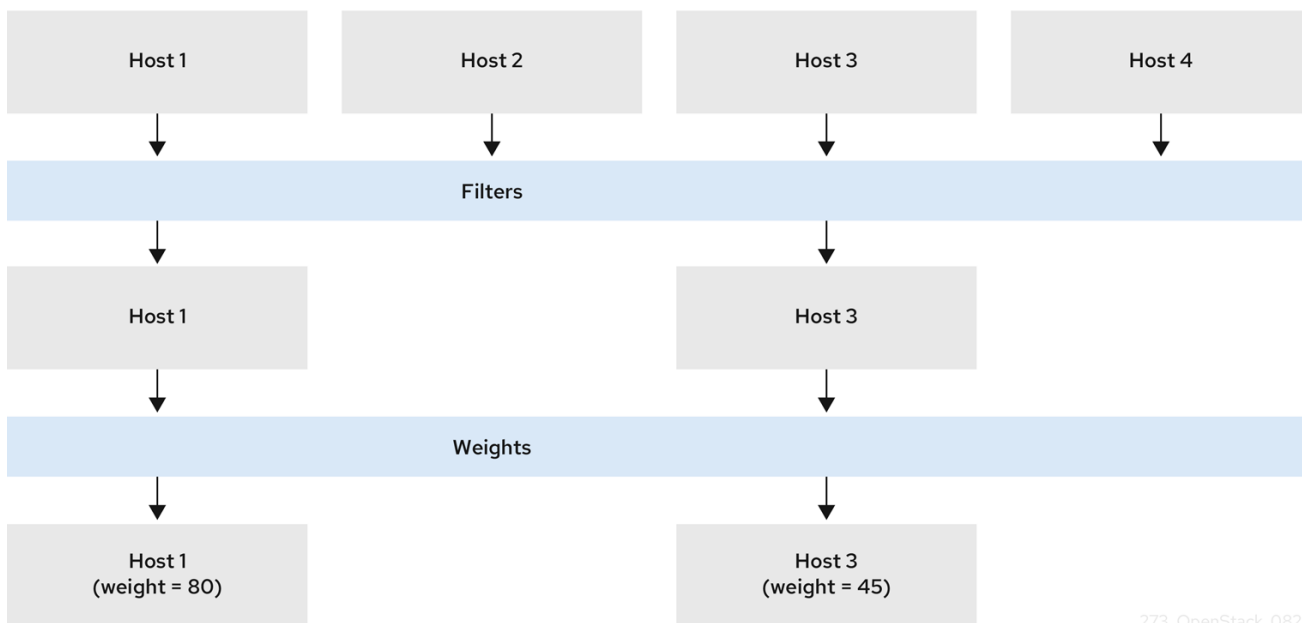
## 第 7 章 配置实例调度和放置

计算调度程序服务决定在哪个 Compute 节点或主机聚合上放置实例。当计算(nova)服务收到启动或移动实例时，它会使用请求、类别和镜像中提供的规格来查找合适的主机。例如，类别可以指定实例所需的特征，如存储磁盘的类型或 Intel CPU 指令集扩展。

计算调度程序服务按照以下顺序使用以下组件的配置来确定在哪个 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用来决定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。权重最高的优先级。

在下图中，在过滤后主机 1 和 3 有资格。主机 1 具有最高权重，因此具有调度的最高优先级。



### 7.1. 使用放置服务进行 PREFILTERING

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，如 Compute 节点、共享存储池或 IP 分配池，以及它们可用的数量资源，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。

放置服务还会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类型。

放置服务根据放置服务提供商清单和特征将 prefilters 应用到一组 candidate 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

### 7.1.1. 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，这非常有用，它不支持 QCOW2 镜像。启用此功能可确保调度程序不会向 Red Hat Ceph Storage 支持的 Compute 节点发送使用 QCOW2 镜像来启动实例的请求。

#### 流程

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

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

### 7.1.2. 按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，如存储磁盘的类型或 Intel CPU 指令集扩展。

Compute 节点将其功能报告到放置服务作为特征。实例可以指定其需要的特征，或者资源供应商不能具有哪些特征。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。

#### 其他资源

- [第 7.5 节 “声明自定义特征和资源类”](#)

#### 7.1.2.1. 创建需要或禁止资源供应商特征的镜像

您可以创建云用户可以使用的实例镜像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. 确定您需要主机或主机聚合具有的特征。您可以选择现有的特征或创建新特征：

- 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，且仅包含字母 A 到 Z，数字 0 到 9，下划线的 "\_" 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. 检查您需要主机或主机聚合的特征的现有资源供应商特征：

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

5. 如果您需要的特征还没有添加到资源供应商中，请将现有的特征和所需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** 选项添加额外的特征。



### 注意

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

6. 要将实例调度到具有所需特征的主机或主机聚合上，请将特征添加到镜像额外规格中。例如，要在支持 AVX-512 的主机或主机聚合上调度实例，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，要防止实例调度到支持 multi-attach 卷的主机或主机聚合，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
```

#### 7.1.2.2. 创建需要或禁止资源提供程序特征的 flavor

您可以创建云用户可以使用的类别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. 确定您需要主机或主机聚合具有的特征。您可以选择现有的特征或创建新特征：

- 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，且仅包含字母 A 到 Z，数字 0 到 9，下划线的 "\_" 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 /')
```

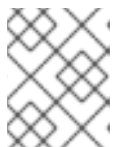
## 4. 检查您需要主机或主机聚合的特征的现有资源供应商特征：

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

## 5. 如果您需要的特征还没有添加到资源供应商中，请将现有的特征和所需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** 选项添加额外的特征。

**注意**

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

## 6. 要将实例调度到具有所需特征的主机或主机聚合上，请将特征添加到类别额外规格中。例如，要在支持 AVX-512 的主机或主机聚合上调度实例，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。例如，要防止实例调度到支持多重附加卷的主机或主机聚合，请将以下特征添加到类别额外规格中：

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

### 7.1.3. 通过隔离主机聚合过滤

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

#### 流程

1. 打开 Compute 环境文件。
2. 要将主机聚合隔离为仅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. 保存对 Compute 环境文件的更新。
4. 使用其他环境文件将计算环境文件添加到堆栈中，并部署 overcloud：

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

5. 识别您要隔离主机聚合的特征。您可以选择现有的特征或创建新特征：

- 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，且仅包含字母 A 到 Z，数字 0 到 9，下划线的 "\_" 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特征：

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

8. 如果您需要的特征还没有添加到资源供应商中，请将现有的特征和所需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** 选项添加额外的特征。



### 注意

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

9. 对主机聚合中的每个 Compute 节点重复步骤 6 - 8。

10. 将特征的 metadata 属性添加到主机聚合中：

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

11. 将特征添加到类别或镜像中：

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

## 7.1.4. 使用放置服务根据可用区过滤

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

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

### 其他资源

- 有关创建主机聚合以用作可用区的更多信息，请参阅[创建可用区](#)。

## 7.2. 为计算调度程序服务配置过滤器和权重

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

## 流程

1. 打开 Compute 环境文件。
2. 将您希望调度程序使用的过滤器添加到 **NovaSchedulerEnabledFilters** 参数，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：将倍数配置为应用到每个 weigher。例如，要指定 Compute 节点的可用 RAM 比其他默认权重高，并且计算调度程序首选具有比可用 RAM 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，要在那些具有更多可用 RAM 的节点上首选具有较少可用 RAM 的 Compute 节点，请将 **ram\_weight\_multiplier** 设置为 **-2.0**。

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

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

## 其他资源

- 有关可用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](#)。




### 7.3. 计算调度程序过滤器

您可以在 Compute 环境文件中配置 **NovaSchedulerEnabledFilters** 参数，以指定在选择托管一个实例的适当 Compute 节点时必须满足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** : 计算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 节点尚未在指定组中托管实例。
- **ServerGroupAffinityFilter** : Compute 节点已在指定组中托管实例。

您可以添加和删除过滤器。下表描述了所有可用的过滤器。

表 7.1. 计算调度程序过滤器

Filter	描述
<b>AggregateImagePropertiesIsolation</b>	使用此过滤器与带有主机聚合元数据的实例的镜像元数据匹配。如果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者。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 <a href="#">镜像 配置参数</a> 。
<b>AggregateInstanceExtraSpecsFilter</b>	<p>使用此过滤器匹配带有主机聚合元数据的实例类型额外规格中定义的命名空间属性。</p> <p>您需要限制类型 <b>extra_specs</b> 键的范围，使用 <b>aggregate_instance_extra_specs</b>:命名空间作为前缀。</p> <p>如果有任何主机聚合元数据与类别额外规格的元数据匹配，则属于该主机聚合的 Compute 节点是从该镜像启动实例的候选者。</p>
<b>AggregateIopsFilter</b>	使用此过滤器，根据 I/O 操作，使用 per-aggregate <b>filter_scheduler/max_io_ops_per_host</b> 值过滤主机。如果没有找到 per-aggregate 值，则该值会返回全局设置。如果主机位于多个聚合中，并且找到多个值，调度程序将使用最小值。
<b>AggregateMultiTenancyIsolation</b>	<p>使用此过滤器将项目隔离主机聚合中的 Compute 节点可用性限制为一组指定的项目。只有使用 <b>filter_tenant_id</b> 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 <a href="#">创建项目隔离主机聚合</a>。</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p><b>注意</b></p> <p>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 <b>NovaSchedulerPlacementAggregateRequiredForTenants</b> 参数。</p> </div> </div>

Filter	描述
<b>AggregateNumInstancesFilter</b>	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 <b>filter_scheduler/max_instances_per_host</b> 参数配置每个 aggregate 的最大实例数量。如果没有找到 per-aggregate 值，则该值会返回全局设置。如果 Compute 节点位于多个聚合中，调度程序将使用最低的 <b>max_instances_per_host</b> 值。
<b>AggregateTypeAffinityFilter</b>	如果没有设置任何类别元数据密钥，则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能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 <b>m1.nano</b> 或 <b>m1.nano,m1.small</b> 。
<b>AllHostsFilter</b>	使用此过滤器考虑所有可用的 Compute 节点以进行实例调度。  <div style="display: flex; align-items: center;">  <div> <p><b>注意</b></p> <p>使用此过滤器不会禁用其他过滤器。</p> </div> </div>
<b>AvailabilityZoneFilter</b>	使用此过滤器在实例指定的可用区的 Compute 节点上启动实例。
<b>ComputeCapabilitiesFilter</b>	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 <b>capabilities: namespace</b> 为类别额外规格添加前缀。  使用 <b>ComputeCapabilitiesFilter</b> 过滤器的一个更为有效的方法是，在您的类别中使用 CPU 特征，这些特征将报告给放置服务。特征为 CPU 功能提供一致的命名。如需更多信息， <a href="#">请参阅使用资源供应商特征过滤</a> 。
<b>ComputeFilter</b>	使用此过滤器传递操作并启用的所有 Compute 节点。此过滤器应始终存在。
<b>DifferentHostFilter</b>	使用此过滤器，允许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 <b>different_host</b> 的 <b>--hint</b> 参数，并将实例 UUID 用作值：  <pre>\$ openstack server create --image cedef40a-ed67-4d10-800e-17455edce175 \ --flavor 1 --hint different_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 \ --hint different_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1</pre>

Filter	描述
<b>ImagePropertiesFilter</b>	<p>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：</p> <ul style="list-style-type: none"> <li>● <b>hw_architecture</b> - Corresponds 到主机架构，如 x86、ARM 和 Power。</li> <li>● <b>img_hv_type</b> - Corresponds 到 hypervisor 类型，如 KVM、QEMU、Xen 和 LXC。</li> <li>● <b>img_hv_requested_version</b> - Corresponds 到计算服务报告的 hypervisor 版本。</li> <li>● <b>hw_vm_mode</b> - Corresponds 到 hypervisor 类型，如 hvm、xen、uml 或 exe。</li> </ul> <p>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。有关镜像属性的更多信息，请参阅 <a href="#">镜像配置参数</a>。</p>
<b>IsolatedHostsFilter</b>	<p>使用此过滤器，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 <b>filter_scheduler/restrict_isolated_hosts_to_isolated_images</b> 来防止在隔离的 Compute 节点上构建实例。</p> <p>要指定隔离的镜像和主机集合，请使用 <b>filter_scheduler/isolated_hosts</b> 和 <b>filter_scheduler/isolated_images</b> 配置选项，例如：</p> <pre>parameter_defaults:   ComputeExtraConfig:     nova::config::nova_config:       filter_scheduler/isolated_hosts:         value: server1, server2       filter_scheduler/isolated_images:         value: 342b492c-128f-4a42-8d3a-c5088cf27d13, ebd267a6- ca86-4d6c-9a0e-bd132d6b7d09</pre>
<b>IoOpsFilter</b>	<p>使用此过滤器过滤具有超过配置的 <b>filter_scheduler/max_io_ops_per_host</b> 的并发 I/O 操作的主机，它指定了允许在主机上运行的最大 I/O 密集型实例数。</p>
<b>MetricsFilter</b>	<p>使用此过滤器，将调度限制为使用 <b>metrics/weight_setting</b> 报告配置的指标的 Compute 节点。</p> <p>要使用此过滤器，请在 Compute 环境文件中添加以下配置：</p> <pre>parameter_defaults:   ComputeExtraConfig:     nova::config::nova_config:       DEFAULT/compute_monitors:         value: 'cpu.virt_driver'</pre> <p>默认情况下，计算调度程序服务每 60 秒更新指标。</p>

Filter	描述
<b>NUMATopologyFilter</b>	<p>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 <b>extra_specs</b> 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 单元的超额订阅限制。</p>
<b>NumInstancesFilter</b>	<p>使用此过滤器过滤运行超过 <b>max_instances_per_host</b> 选项指定的实例的 Compute 节点。</p>
<b>PciPassthroughFilter</b>	<p>使用此过滤器，使用类别 <b>extra_specs</b> 将具有实例请求的 Compute 节点上调度实例。</p> <p>如果您要为节点保留 PCI 设备（通常为昂贵且有限）用于请求它们的实例，请使用此过滤器。</p>
<b>SameHostFilter</b>	<p>使用此过滤器，在与一组特定实例相同的 Compute 节点上调度实例。要在启动实例时指定这些实例，请使用 <b>--hint</b> 参数和键 <b>相同的_host</b>，实例 UUID 作为值：</p> <pre>\$ openstack server create --image cedef40a-ed67-4d10-800e-17455edce175 \   --flavor 1 --hint same_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 \   --hint same_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1</pre>
<b>ServerGroupAffinityFilter</b>	<p>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：</p> <pre>\$ openstack server group create --policy affinity &lt;group_name&gt;</pre> <p>要在此组中启动实例，请使用 <b>--hint</b> 参数并将 <b>group</b> 用作键，组 UUID 作为值：</p> <pre>\$ openstack server create --image &lt;image&gt; \   --flavor &lt;flavor&gt; \   --hint group=&lt;group_uuid&gt; &lt;instance_name&gt;</pre>

Filter	描述
<b>ServerGroupAntiAffinity Filter</b>	<p>使用此过滤器来调度属于不同 Compute 节点上的反关联性服务器组的实例。运行以下命令来创建服务器组：</p> <pre>\$ openstack server group create --policy anti-affinity &lt;group_name&gt;</pre> <p>要在此组中启动实例，请使用 <b>--hint</b> 参数并将 <b>group</b> 用作键，组 UUID 作为值：</p> <pre>\$ openstack server create --image &lt;image&gt; \   --flavor &lt;flavor&gt; \   --hint group=&lt;group_uuid&gt; &lt;instance_name&gt;</pre>
<b>SimpleCIDRAffinityFilter</b>	<p>使用此过滤器，在具有特定 IP 子网范围的 Compute 节点上调度实例。要指定所需的范围，请使用 <b>--hint</b> 参数在启动实例时传递键 <b>build_near_host_ip</b> 和 <b>cidr</b>：</p> <pre>\$ openstack server create --image &lt;image&gt; \   --flavor &lt;flavor&gt; \   --hint build_near_host_ip=&lt;ip_address&gt; \   --hint cidr=&lt;subnet_mask&gt; &lt;instance_name&gt;</pre>

## 7.4. 计算调度程序权重

每个 Compute 节点都有一个权重，调度程序可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 节点中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候选 Compute 节点内资源可用性的下半值来计算每种资源类型的权重规范化：

- 资源可用性最低的节点(`minval`)被分配为 '0'。
- 资源可用性最高的节点(`maxval`)被分配为 '1'。
- `minval - maxval` 范围内具有资源可用性的节点分配使用以下公式计算的规范化权重：

$$\frac{(\text{node\_resource\_availability} - \text{minval})}{(\text{maxval} - \text{minval})}$$

如果所有 Compute 节点都有相同的资源可用性，则它们都会规范化为 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：

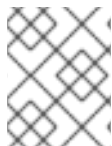
Compute 节点	1	2	3	4	5	6	7	8	9	10

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：

$$(w1\_multiplier * norm(w1)) + (w2\_multiplier * norm(w2)) + \dots$$

下表描述了可用于权重的配置选项。



### 注意

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

表 7.2. 计算调度程序权重

配置选项	类型	描述
------	----	----

配置选项	类型	描述
<b>filter_scheduler/weight_classes</b>	字符串	<p>使用此参数配置以下哪些属性用于计算每个 Compute 节点的权重：</p> <ul style="list-style-type: none"> <li>● <b>nova.scheduler.weights.ram.RAMWeigher</b> - 权衡 Compute 节点上的可用 RAM。</li> <li>● <b>nova.scheduler.weights.cpu.CPUWeigher</b> - 权衡 Compute 节点上的可用 CPU。</li> <li>● <b>nova.scheduler.weights.disk.DiskWeigher</b> - 权衡 Compute 节点上的可用磁盘。</li> <li>● <b>nova.scheduler.weights.metrics.MetricsWeigher</b> - 权衡 Compute 节点的指标。</li> <li>● <b>nova.scheduler.weights.affinity.ServerGroupSoftAffinityWeigher</b> - 将 Compute 节点的概率增加到给定实例组中的其他节点。</li> <li>● <b>nova.scheduler.weights.affinity.ServerGroupSoftAntiAffinityWeigher</b> - 将 Compute 节点的概率增加到给定实例组中的其他节点。</li> <li>● <b>nova.scheduler.weights.compute.BuildFailureWeigher</b> - Weighs Compute 节点的数量，最近失败的引导尝试次数。</li> <li>● <b>nova.scheduler.weights.io_ops.IoOpsWeigher</b> - 根据工作复制为 Compute 节点加权重。</li> <li>● <b>nova.scheduler.weights.pci.PCIWeigher</b> - 按其 PCI 可用性计算 Compute 节点。</li> <li>● <b>nova.scheduler.weights.cross_cell.CrossCellWeigher</b> - 基于它们所在的单元，在移动实例时优先使用源单元格中的 Compute 节点。</li> <li>● <b>nova.scheduler.weights.all_weighers</b> - (Default)使用上述所有 weighers。</li> </ul>
<b>filter_scheduler/ram_weight_multiplier</b>	浮点	<p>使用此参数指定基于可用 RAM 的 eigh 主机的倍数。</p> <p>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。</p> <p>设置为负值，以首选使用较少可用 RAM 的主机，这会尽可能填充（堆栈）主机，然后再调度到较早使用的主机。</p> <p>绝对值（无论是正还是负值）控制 RAM weigher 相对于其他 Weighers 的强度量。</p> <p>Default: 1.0 - 调度程序均匀地将实例分散到所有主机中。</p>

配置选项	类型	描述
<b>filter_scheduler/disk_weight_multiplier</b>	浮点	<p>使用此参数根据可用磁盘空间指定用于 weigh 主机的倍数。</p> <p>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。</p> <p>设置为负值，以首选使用较少的可用磁盘空间的主机，这会尽可能填充(stacks)主机，然后再调度到较早使用的主机。</p> <p>绝对值（无论是正还是负）控制磁盘我们相对于其他 Weighers 的强度程度。</p> <p>Default: 1.0 - 调度程序均匀地将实例分散到所有主机中。</p>
<b>filter_scheduler/cpu_weight_multiplier</b>	浮点	<p>使用此参数根据可用的 vCPU 指定用于 weigh 主机的倍数。</p> <p>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。</p> <p>设置为负值，以首选具有较少可用 vCPU 的主机，这将尽可能填充（堆栈）主机，然后再调度到较早使用的主机。</p> <p>绝对值（无论是正还是负）控制 vCPU weigher 相对于其他 Weighers 的强度。</p> <p>Default: 1.0 - 调度程序均匀地将实例分散到所有主机中。</p>
<b>filter_scheduler/io_ops_weight_multiplier</b>	浮点	<p>使用此参数指定基于主机负载的 eigh 主机的倍数。</p> <p>设置为负值，以首选将工作负载分布到更多主机的主机。</p> <p>设置为正数值，以首选具有更高工作负载的主机，这会将实例调度到已经忙碌的主机上。</p> <p>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。</p> <p>Default: -1.0 - 调度程序在更多主机间分发工作负载。</p>
<b>filter_scheduler/build_failure_weight_multiplier</b>	浮点	<p>使用此参数指定根据最近的构建失败，用于 weigh 主机的倍数。</p> <p>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。</p> <p>设置为 <b>0</b>，以禁用受最近故障数的计算主机。</p> <p>默认：1000000.0</p>



配置选项	类型	描述
<b>filter_scheduler/cross_cell_move_weight_multiplier</b>	浮点	<p>使用此参数指定跨移动期间用于 weigh 主机的倍数。此选项决定了在移动实例时放入相同源单元的主机上放置了多少权重。默认情况下，调度程序在迁移实例时优先选择同一源单元中的主机。</p> <p>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格中的主机。</p> <p>默认：1000000.0</p>
<b>filter_scheduler/pci_weight_multiplier</b>	正浮动点	<p>使用此参数指定根据主机上的 PCI 设备数量以及实例请求的 PCI 设备数量，以权衡主机的倍数。如果实例请求 PCI 设备，则 Compute 节点越多的 PCI 设备越高，权重越高。</p> <p>例如，如果存在三个可用的主机，一个主机只有一个 PCI 设备，一个具有多个 PCI 设备，一个没有 PCI 设备，则计算调度程序会根据实例的需求来优先选择这些主机。如果实例请求一个 PCI 设备，调度程序应优先选择第一个主机，如果实例请求一个 PCI 设备，则第二个主机如果实例请求了 PCI 设备，则第二个主机是第三个主机。</p> <p>配置这个选项，以防止非 PCI 实例使用 PCI 设备在主机上占用资源。</p> <p>Default: 1.0</p>
<b>filter_scheduler/host_subset_size</b>	整数	<p>使用此参数指定要选择主机的过滤主机子集的大小。您必须将这个选项设置为至少 1。1 代表选择由权重函数返回的第一个主机。调度程序忽略小于 1 的任何值，而是使用 1。</p> <p>设置为大于 1 的值，以防止多个调度程序处理选择同一主机的类似请求，从而造成潜在的竞争条件。从最适合请求的 N 主机中随机选择主机，从而降低冲突的可能性。但是，您设置这个值越高，所选主机对于给定请求可能的最佳选择。</p> <p>默认：1</p>
<b>filter_scheduler/soft_affinity_weight_multiplier</b>	正浮动点	<p>使用此参数指定用于 weigh 主机进行组软关联性的倍数。</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p><b>注意</b></p> <p>在创建具有此策略的组时，您需要指定 microversion：</p> <pre>\$ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity &lt;group_name&gt;</pre> </div> </div> <p>Default: 1.0</p>

配置选项	类型	描述
<b>filter_scheduler/soft_anti_affinity_weight_multiplier</b>	正浮动点	<p>使用此参数指定用于 weigh 主机组 soft-anti-affinity 的倍数。</p>  <p><b>注意</b></p> <p>在创建具有此策略的组时，您需要指定 microversion :</p> <pre>\$ openstack --os-compute-api-version 2.15 server group create --policy soft-affinity &lt;group_name&gt;</pre> <p>Default: 1.0</p>
<b>metrics/weight_multiplier</b>	浮点	<p>使用此参数指定用于权重指标的倍数。默认情况下，<b>weight_multiplier=1.0</b>，这会将实例分散到可能的主机上。</p> <p>设置为大于 1.0 的数字，以提高指标对总体权重的影响。</p> <p>设置为 0.0 和 1.0 之间的数字，以减少指标对整体权重的影响。</p> <p>设置为 0.0 以忽略指标值，并返回 <b>weight_of_unavailable</b> 选项的值。</p> <p>设置为负数，以优先选择具有较低指标的主机以及主机中的堆栈实例。</p> <p>Default: 1.0</p>

配置选项	类型	描述
<b>metrics/weight_setting</b>	以逗号分隔的 <b>metric=ratio</b> 对列表	<p>使用此参数指定用于权重的指标，以及用于计算每个指标的权重的比率。有效的指标名称：</p> <ul style="list-style-type: none"> <li>● <b>cpu.frequency</b> - CPU 频率</li> <li>● <b>cpu.user.time</b> - CPU 用户模式时间</li> <li>● <b>cpu.kernel.time</b> - CPU 内核时间</li> <li>● <b>cpu.idle.time</b> - CPU 空闲时间</li> <li>● <b>cpu.iowait.time</b> - CPU I/O 等待时间</li> <li>● <b>cpu.user.percent</b> - CPU 用户模式百分比</li> <li>● <b>cpu.kernel.percent</b> - CPU 内核百分比</li> <li>● <b>cpu.idle.percent</b> - CPU 空闲百分比</li> <li>● <b>cpu.iowait.percent</b> - CPU I/O 等待百分比</li> <li>● <b>cpu.percent</b> - Generic CPU 使用</li> </ul> <p>示例：<b>weight_setting=cpu.user.time=1.0</b></p>
<b>metrics/必需</b>	布尔值	<p>使用此参数指定如何处理不可用的已配置的 <b>metrics/weight_setting</b> 指标：</p> <ul style="list-style-type: none"> <li>● <b>true</b> - 需要指标。如果指标不可用，则会引发异常。要避免异常，请使用 <b>NovaSchedulerEnabledFilters</b> 中的 <b>MetricsFilter</b> 过滤器。</li> <li>● <b>False</b> - 不可用的指标在 weighing 进程中被视为一个负因素。使用 <b>weight_of_unavailable</b> 配置选项设置返回的值。</li> </ul>
<b>metrics/weight_of_unavailable</b>	浮点	<p>如果有任何 <b>metrics/weight_setting</b> 指标不可用，则使用此参数指定要使用的权重，以及 <b>metrics/required=False</b>。</p> <p>默认：-10000.0</p>

## 7.5. 声明自定义特征和资源类

作为管理员，您可以通过在 YAML 文件 **provider.yaml** 中定义自定义资源，来声明 Red Hat OpenStack Platform (RHOSP) overcloud 节点上有哪些自定义物理功能和可消耗的资源。

您可以通过定义自定义特征来声明物理主机功能的可用性，如 **CUSTOM\_DIESEL\_BACKUP\_POWER**、**CUSTOM\_FIPS\_COMPLIANT** 和 **CUSTOM\_HPC\_OPTIMIZED**。您还可以通过定义资源类（如 **CUSTOM\_DISK\_IOPS**）和 **CUSTOM\_POWER\_WATTS** 来声明可消耗资源的可用性。



## 注意

您可以使用类别元数据来请求自定义资源和自定义特征。如需更多信息，请参阅 [实例裸机资源类和实例资源特征](#)。

## 流程

1. 在 `/home/stack/templates/` 中创建一个名为 `provider.yaml` 的文件。
2. 要配置资源供应商，请在 `provider.yaml` 文件中添加以下配置：

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

- 将 `<node_uuid>` 替换为节点的 UUID，例如 `'5213b75d-9260-42a6-b236-f39b0fd10561'`。或者，您可以使用 `name` 属性来识别资源 `provider: name: 'EXAMPLE_RESOURCE_PROVIDER'`。
3. 要为资源供应商配置可用的自定义资源类，请在 `provider.yaml` 文件中添加以下配置：

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

- 将 `CUSTOM_EXAMPLE_RESOURCE_CLASS` 替换为资源类的名称。自定义资源类必须以 `CUSTOM_` 开头，且仅包含字母 A 到 Z，数字 0 到 9，下划线的 "\_" 字符。
- 将 `<total_available>` 替换为此资源提供程序的可用 `CUSTOM_EXAMPLE_RESOURCE_CLASS` 的数量。
- 将 `<reserved>` 替换为此资源提供程序的可用 `CUSTOM_EXAMPLE_RESOURCE_CLASS` 的数量。
- 将 `<min_unit>` 替换为单个实例可以消耗的资源的最小单元。
- 将 `<max_unit>` 替换为单个实例可以使用的最大资源单元。
- 将 `<step_size>` 替换为此资源提供程序的可用 `CUSTOM_EXAMPLE_RESOURCE_CLASS` 的数量。
- 将 `<allocation_ratio>` 替换为设置分配比率的值。如果将 `allocation_ratio` 设置为 1.0，则不允许过度分配。但是，如果 `allocation_ratio` 大于 1.0，则可用资源总数超过物理现有资源。

4. 要为资源供应商配置可用的特征，请在 `provider.yaml` 文件中添加以下配置：

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

- 将 `CUSTOM_EXAMPLE_TRAIT` 替换为特征的名称。自定义特征必须以前缀 `CUSTOM_` 开头，且仅包含字母 A 到 Z，数字 0 到 9，下划线的 "\_" 字符。

### provider.yaml 文件示例

以下示例声明了一个自定义资源类，另一个用于资源提供程序的自定义特征。

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

- ① 这个虚拟机监控程序有 22 个顶层缓存单元(LLC)。
- ② 为主机保留两个 LLC 单元。
- ③ ④ `min_unit` 和 `max_unit` 值定义单个虚拟机可以使用的资源单元数量。
- ⑤ 步骤大小定义消耗递增。
- ⑥ 分配比率配置资源的过度分配。

5. 保存并关闭 **provider.yaml** 文件。
6. 使用其他环境文件将 **provider.yaml** 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/provider.yaml
```

## 7.6. 创建和管理主机聚合

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

### 主机聚合

主机聚合是根据硬件或性能特征等属性将 Compute 节点分组到逻辑单元中。您可以将 Compute 节点分配到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。当启用所需的过滤器时，计算调度程序可以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、强制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，您可以指定区域名称。此名称呈现给云用户，作为他们可以选择的可用区。

### 可用区

可用域是主机聚合的云用户视图。云用户无法在可用区中看到 Compute 节点，或者查看可用区的元数据。云用户只能看到可用区的名称。

您可以将每个 Compute 节点分配到一个可用区。您可以配置一个默认可用区，在云用户没有指定区时调度实例。您可以指示云用户使用具有特定功能的可用区。

### 7.6.1. 启用在主机聚合上的调度

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

#### 流程

1. 打开 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，以通过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, 以通过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**: 在启动实例时添加此值来根据可用区进行过滤。



### 注意

您可以使用放置服务来处理可用性区域请求, 而不使用 **AvailabilityZoneFilter** 计算调度程序服务过滤器。如需更多信息, 请参阅使用 [放置服务过滤可用区](#)。

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

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

## 7.6.2. 创建主机聚合

作为云管理员, 您可以根据需要创建任意数量的主机聚合。

### 流程

1. 要创建主机聚合, 请输入以下命令 :

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

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

2. 在主机聚合中添加元数据 :

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

- 将 **<key=value>** 替换为元数据键值对。如果您使用 **AggregateInstanceExtraSpecsFilter** 过滤器, 则键可以是任意字符串, 如 **ssd=true**。如果使用 **AggregateImagePropertiesIsolation** 过滤器, 则密钥必须与有效的镜像元数据属性匹配。有关有效镜像元数据属性的更多信息, 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>** 替换为要向其添加 Compute 节点的主机聚合的名称。
- 将 **<host\_name>** 替换为要添加到主机聚合中的 Compute 节点的名称。

#### 4. 为主机聚合创建类别或镜像：

- 创建类别：

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

- 创建镜像：

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

#### 5. 在类别或镜像上设置与主机聚合上的键值对匹配的一个或多个键值对。

- 要在类别上设置键值对，请使用范围 **aggregate\_instance\_extra\_specs**：

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

- 要在镜像上设置键值对，请使用有效的镜像元数据属性作为键：

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

### 7.6.3. 创建可用区

作为云管理员，您可以创建一个可用性区域，在云用户创建实例时可以选择该可用区。

#### 流程

1. 要创建可用区，您可以创建一个新的可用区主机聚合，或者将现有主机聚合一个可用区：
  - 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：

```
(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。

#### 7.6.4. 删除主机聚合

要删除主机聚合，您首先从主机聚合中删除所有 Compute 节点。

##### 流程

1. 要查看分配给主机聚合的所有 Compute 节点的列表，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，请为每个 Compute 节点输入以下命令：

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

- 将 **<aggregate\_name>** 替换为要从中删除 Compute 节点的主机聚合的名称。
- 将 **<host\_name>** 替换为要从主机聚合中删除的 Compute 节点的名称。

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

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

#### 7.6.5. 创建项目隔离主机聚合

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



## 注意

项目隔离使用放置服务来过滤各个项目的主机聚合。此过程取代了 **AggregateMultiTenancyIsolation** 过滤器的功能。因此，您不需要使用 **AggregateMultiTenancyIsolation** 过滤器。

## 流程

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

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

6. 创建主机聚合。
7. 检索项目 ID 列表：

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

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

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

- 将 **<ID0>**、**<ID1>**，以及直到 **<IDn>** 的所有 ID 替换为您要创建的每个项目过滤器的唯一值。
- 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** metadata 键中省略后缀来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 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，实例会保留对设备执行任务的独占访问，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

Compute 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，如果您使用路由供应商网络，则必须在所有 Compute 节点上使用相同的 **physical\_network** 名称。

如果您将路由供应商网络与 VLAN 或扁平网络搭配使用，则必须对所有片段使用相同的 **physical\_network** 名称。然后，您可以为网络创建多个片段，并将片段映射到适当的子网。

要让您的云用户创建附加了 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. 创建一个类别来启动实例，并连接了 PCI 设备。

### 先决条件

- Compute 节点具有所需的 PCI 设备。

## 8.1. 为 PCI 透传设计 COMPUTE 节点

要为附加了物理 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 Compute 节点以进行 PCI 透传。



### 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，您必须使用缩减流程取消置备节点，然后使用扩展步骤使用新资源类分配重新置备节点。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
role 注释	<b>Role: Compute</b>	<b>Role: ComputePCI</b>
角色名称	<b>Name: Compute</b>	<b>Name: ComputePCI</b>
<b>description</b>	基本 <b>Compute</b> 节点角色	<b>PCI Passthrough Compute</b> 节点角色
<b>HostnameFormatDefault</b>	<b>%stackname%-novacompute-%index%</b>	<b>%stackname%-novacomputepci-%index%</b>
<b>deprecated_nic_config_name</b>	<b>compute.yaml</b>	<b>compute-pci-passthrough.yaml</b>

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。有关更多信息，请参阅 [安装和管理 Red Hat OpenStack Platform](#) 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 [安装和管理 Red Hat OpenStack Platform](#) 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-PASSTROUGH <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-PASSTROUGH
    network_config:
      template: /home/stack/templates/nic-config/myRoleTopology.j2 1
```

- 1 您可以重复使用现有网络拓扑，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[使用 director 安装和管理 Red Hat OpenStack Platform 指南中的自定义网络接口模板](#)。如果您没有使用 `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](#)。有关节点定义文件的示例，请参阅[节点定义文件示例](#)。

9. 运行 provisioning 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 `<stack>` 替换为置备裸机节点的堆栈名称。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。如果您没有使用 `network_config` 属性定义网络定义，则使用默认网络定义。
10. 在一个单独的终端中监控置备进度。当置备成功时，节点状态将从 **available** 变为 **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。

## 8.2. 配置 PCI 透传 COMPUTE 节点

要让您的云用户创建附加了 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：

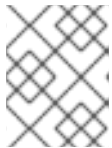
```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
```

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

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

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

有关配置 `device_type` 字段的更多信息，请参阅 [PCI passthrough 设备类型字段](#)。



### 注意

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

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

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

5. 要在 overcloud 上为 PCI 透传配置 Compute 节点，请创建一个环境文件，如 `pci_passthrough_compute.yaml`。
6. 要为 Compute 节点上的设备指定可用的 PCI，请使用 `vendor_id` 和 `product_id` 选项将所有匹配的 PCI 设备添加到可用于透传到实例的 PCI 设备池中。例如，要将所有 Intel® Ethernet Controller X710 设备添加到可用于透传到实例的 PCI 设备池中，请将以下配置添加到 `pci_passthrough_compute.yaml`：

```
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。

- 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 透传 Compute 节点上的设备指定 PCI 别名，请将以下内容添加到 `pci_passthrough_compute.yaml` 中：

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



### 注意

Compute 节点别名必须与 Controller 节点上的别名相同。因此，如果您在 `pci_passthrough_controller.yaml` 中添加了 `numa_affinity` 到 `nova::pci::aliases`，那么您还必须将其添加到 `pci_passthrough_compute.yaml` 中的 `nova::pci::aliases` 中。

- 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 **KernelArgs** 设置来启用 Intel IOMMU：

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

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



### 注意

当您首先将 **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，您可以禁用自动重新引导节点，并在每个 overcloud 部署后手动重启节点。如需更多信息，请参阅 [配置手动节点重新引导以定义 KernelArgs](#)。

- 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
```

- 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，每个设备的厂商 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 flavor 或镜像中：

- 要使用类别覆盖默认 NUMA 关联性策略，请添加 **hw:pci\_numa\_affinity\_policy** 属性键：

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

有关 **hw:pci\_numa\_affinity\_policy** 的有效值的更多信息，请参阅 [类别元数据](#)。

- 要使用镜像覆盖默认 NUMA 关联性策略，请添加 **hw\_pci\_numa\_affinity\_policy** 属性键：

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



### 注意

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

## 验证

1. 使用 PCI 透传设备创建实例：

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

2. 以云用户身份登录实例。如需更多信息，请参阅 [连接到实例](#)。
3. 要验证 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>
```

## 8.3. PCI PASSTHROUGH 设备类型字段

计算服务将 PCI 设备归类为三种类型之一，具体取决于设备报告的功能。以下列表您可以将 **device\_type** 字段设置为的有效值：

### type-PF

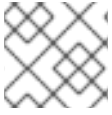
该设备支持 SR-IOV，它是父设备或 root 设备。指定这个设备类型来透传一个支持 SR-IOV 的设备。

### 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。

## 8.4. 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 而不是 PF 本身的 VF，您可以使用 **address** 参数指定 PF 和 **product\_id** 的 PCI 地址来指定 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 `push`，如下例所示：

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

#### 字典格式

如果使用字典格式指定地址，您可以包含正则表达式语法，如下例所示：

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



## 注意

Compute 服务将 **address** 字段配置限制为以下最大值：

- 域 - 0xFFFF
- 总线 - 0xFF
- slot - 0x1F
- function - 0x7

计算服务支持具有 16 位地址域的 PCI 设备。计算服务会忽略 32 位地址域的 PCI 设备。

## 第 9 章 配置 VDPA COMPUTE 节点以启用使用 VDPA 端口的实例

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。

## 9.1. 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 [安装和管理 Red Hat OpenStack Platform 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 [安装和管理 Red Hat OpenStack Platform 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 **vdpa\_net\_top.j2**。您可以重复使用现有网络拓扑，或为角色创建新的自定义网络接口模板。如需更多信息，请参阅使用 *director 安装和管理 Red Hat OpenStack Platform* 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](#)。有关节点定义文件的示例，请参阅 [节点定义文件示例](#)。

9. 打开您的网络接口模板 **vdpa\_net\_top.j2**，并添加以下配置，将 VDPA 支持的网络接口指定为 OVS 网桥的成员：

```

- type: ovs_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

```

10. 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**。

11. 在一个单独的终端中监控置备进度。当置备成功时，节点状态将从 **available** 变为 **active**：

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

12. 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/<role_topology_file>
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

将 **<role\_topology\_file>** 替换为包含 **ComputeVdpa** 角色的网络拓扑的文件的名称，如 **vdpa\_net\_top.j2**。设置为 **compute.j2**，以使用默认的网络拓扑。

## 9.2. 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 **NovaSchedulerEnabledFilters** 参数中：

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
  ['AvailabilityZoneFilter','ComputeFilter','ComputeCapabilitiesFilter','ImagePropertiesFilter','ServerGroupAntiAffinityFilter','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 **KernelArgs** 设置来启用 Intel 公司 IOMMU：

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

要启用 AMD IOMMU，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，您可以禁用自动重新引导节点，并在每个 overcloud 部署后手动重启节点。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. 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 指南中的使用 VDPA 接口创建实例](#)。
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 10 章 为实例配置虚拟 GPU

要支持实例上基于 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有效地在所有物理 GPU 设备之间划分渲染工作负载，并更好地控制启用了 vGPU 的实例。

要在 Compute (nova) 服务中启用 vGPU，请创建类别，您的云用户可以使用 vGPU 设备创建 Red Hat Enterprise Linux (RHEL) 实例。然后，每个实例都可以支持 GPU 工作负载，其虚拟 GPU 设备对应于物理 GPU 设备。

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

### 重要

红帽启用了在 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得 NVIDIA Enterprise Support for NVIDIA vGPU 软件的支持。对于使用 NVIDIA vGPU 的问题，当您无法在支持的组件中重现问题，请应用以下支持策略：

- 当红帽不怀疑涉及第三方组件时，会应用正常的 [支持范围](#) 和 [Red Hat SLA](#)。
- 当红帽怀疑涉及第三方组件的问题时，客户将被定向到 NVIDIA，与红帽 [第三方支持和认证政策](#) 一致。如需更多信息，请参阅知识库文章 [从 NVIDIA 获得支持](#)。

### 10.1. 支持的配置和限制

#### 支持的 GPU 卡

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

#### 使用 vGPU 设备时的限制

- 每个实例只能使用一个 vGPU 资源。
- 不支持在主机之间实时迁移 vGPU 实例。
- 不支持 vGPU 实例撤离。
- 如果您需要重新引导托管 vGPU 实例的 Compute 节点，则 vGPU 不会自动重新分配给重新创建的实例。您必须在重启 Compute 节点前冷迁移实例，或者在重启后手动将每个 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，不支持在支持 vGPU 的实例上暂停操作。相反，您可以对实例进行快照或清空。

- 默认情况下，Compute 主机上的 vGPU 类型不会向 API 用户公开。要将 Compute 主机上的 vGPU 类型公开给 API 用户，您必须配置资源供应商特征并创建需要特征的类别。如需更多信息，请参阅 [创建自定义 vGPU 资源供应商特征](#)。或者，如果您只有一个 vGPU 类型，您可以通过将主机添加到主机聚合来授予访问权限。如需更多信息，请参阅 [创建和管理主机聚合](#)。
- 如果使用 NVIDIA 加速器硬件，您必须符合 NVIDIA 许可要求。例如：NVIDIA vGPU GRID 需要许可服务器。有关 NVIDIA 许可要求的更多信息，请参阅 [NVIDIA 网站上的 NVIDIA License Server 发行注记](#)。

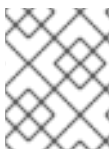
## 10.2. 在 COMPUTE 节点上配置 vGPU

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. 可选：为 vGPU 类型创建自定义特征。
5. 可选：创建自定义 GPU 实例镜像。
6. 创建一个 vGPU 类别来启动具有 vGPU 的实例。

### 提示

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



### 注意

要使用 NVIDIA GRID vGPU，您必须符合 NVIDIA GRID 许可要求，您必须具有自托管许可证服务器的 URL。如需更多信息，请参阅 [NVIDIA License Server 发行注记](#) 网页。

### 10.2.1. 先决条件

- 您已从 NVIDIA 网站下载了与 GPU 设备对应的 NVIDIA GRID 主机驱动程序 RPM 软件包。要确定您需要的驱动程序，请参阅 [NVIDIA 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。
- 您已构建了一个自定义 overcloud 镜像，该镜像安装了 NVIDIA GRID 主机驱动程序。

### 10.2.2. 为 vGPU 设计 Compute 节点

要为 vGPU 工作负载指定 Compute 节点，您必须创建一个新的角色文件来配置 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调配的现有 overcloud 节点，您必须使用缩减流程取消置备节点，然后使用扩展步骤使用新资源类分配重新置备节点。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
role 注释	<b>Role: Compute</b>	<b>Role: ComputeGpu</b>
角色名称	<b>Name: Compute</b>	<b>name: ComputeGpu</b>
<b>description</b>	<b>基本 Compute 节点角色</b>	<b>GPU Compute 节点角色</b>
<b>HostnameFormatDefault</b>	<b>-compute-</b>	<b>-computegpu-</b>
<b>deprecated_nic_config_name</b>	<b>compute.yaml</b>	<b>compute-gpu.yaml</b>

5. 通过将 overcloud 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 [安装和管理 Red Hat OpenStack Platform 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 [安装和管理 Red Hat OpenStack Platform 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。

- 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 ❶
```

- ❶ 您可以重复使用现有网络拓扑，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 *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 [自定义网络接口模板](#)。如果您没有使用 `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](#)。有关节点定义文件的示例，请参阅 [节点定义文件示例](#)。

- 运行 provisioning 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。如果您没有使用 `network_config` 属性定义网络定义，则使用默认网络定义。

- 在一个单独的终端中监控置备进度。当置备成功时，节点状态将从 **available** 变为 **active**：

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

- 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。

### 10.2.3. 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 驱动程序并启动该节点。
2. 虚拟 GPU 是介质设备或 **mdev** 类型设备。检索每个 Compute 节点上的每个 **mdev** 设备的 PCI 地址：

```
$ ls /sys/class/mdev_bus/
```

PCI 地址用作设备驱动程序目录名称，如 **0000:84:00.0**。

3. 查看每个 Compute 节点上每个可用 pGPU 设备支持的 **mdev** 类型，以发现可用的 vGPU 类型：

```
$ ls /sys/class/mdev_bus/<mdev_device>/mdev_supported_types
```

- 将 **<mdev\_device >** 替换为 **mdev** 设备的 PCI 地址，例如 **0000:84:00.0**。  
例如，以下 Compute 节点有 4 个 pGPU，每个 pGPU 都支持相同的 11 vGPU 类型：

```
[root@overcloud-computegpu-0 ~]# ls
/sys/class/mdev_bus/0000:84:00.0/mdev_supported_types:
nvidia-35 nvidia-36 nvidia-37 nvidia-38 nvidia-39 nvidia-40 nvidia-41 nvidia-42 nvidia-43
nvidia-44 nvidia-45
[root@overcloud-computegpu-0 ~]# ls
/sys/class/mdev_bus/0000:85:00.0/mdev_supported_types:
nvidia-35 nvidia-36 nvidia-37 nvidia-38 nvidia-39 nvidia-40 nvidia-41 nvidia-42 nvidia-43
nvidia-44 nvidia-45
[root@overcloud-computegpu-0 ~]# ls
/sys/class/mdev_bus/0000:86:00.0/mdev_supported_types:
nvidia-35 nvidia-36 nvidia-37 nvidia-38 nvidia-39 nvidia-40 nvidia-41 nvidia-42 nvidia-43
nvidia-44 nvidia-45
[root@overcloud-computegpu-0 ~]# ls
/sys/class/mdev_bus/0000:87:00.0/mdev_supported_types:
nvidia-35 nvidia-36 nvidia-37 nvidia-38 nvidia-39 nvidia-40 nvidia-41 nvidia-42 nvidia-43
nvidia-44 nvidia-45
```

4. 创建一个 **gpu.yaml** 文件来指定每个 GPU 设备支持的 vGPU 类型：

```
parameter_defaults:
  ComputeGpuExtraConfig:
    nova::compute::vgpu::enabled_vgpu_types:
      - nvidia-35
      - nvidia-36
```

5. 可选：要配置多个 vGPU 类型，请将支持的 vGPU 类型映射到 pGPU：

```
parameter_defaults:
  ComputeGpuExtraConfig:
    nova::compute::vgpu::enabled_vgpu_types:
      - nvidia-35
```

```
- nvidia-36
NovaVGPUTypesDeviceAddressesMapping: {'vgpu_<vgpu_type>': ['<pci_address>',
'<pci_address>'],'vgpu_<vgpu_type>': ['<pci_address>', '<pci_address>']}
```

- 将 **<vgpu\_type>** 替换为 vGPU 类型的名称，以便为 vGPU 组创建一个标签，如 **vgpu\_nvidia-35**。使用以逗号分隔的 **vgpu\_&lt ;vgpu\_type>** 定义列表来映射额外的 vGPU 类型。
- 将 **<pci\_address>** 替换为支持 vGPU 类型的 pGPU 设备的 PCI 地址，如 **0000:84:00.0**。使用以逗号分隔的 **< pci\_address>** 定义列表将 vGPU 组映射到额外的 pGPU。

Example:

```
NovaVGPUTypesDeviceAddressesMapping: {'vgpu_nvidia-35': ['0000:84:00.0',
'0000:85:00.0'],'vgpu_nvidia-36': ['0000:86:00.0']}
```

- 在 PCI 地址 **0000:84:00.0** 和 **0000:85:00.0** 中的 pGPU 支持 **nvidia-35** vGPU 类型。
- **nvidia-36** vGPU 类型只支持 PCI 地址 **0000:86:00.0** 中的 pGPU。

6. 保存对 Compute 环境文件的更新。

7. 使用其他环境文件将新的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
```

### 10.3. 创建自定义 VGPU 资源供应商特征

您可以为 RHOSP 环境支持的每个 vGPU 类型创建自定义资源供应商特征。然后，您可以创建云用户可以使用的类别在具有这些自定义特征的主机上启动实例。自定义特征在大写字母中定义，且必须以前缀 **CUSTOM\_** 开头。如需有关资源供应商特征的更多信息，请参阅 [按资源供应商特征过滤](#)。

#### 流程

1. 创建新特征：

```
(overcloud)$ openstack --os-placement-api-version 1.6 trait \
create CUSTOM_<TRAIT_NAME>
```

- 将 **<TRAIT\_NAME>** 替换为特征的名称。名称只能包含字母 A 到 Z，数字 0 到 9，下划线的 "\_" 字符。

2. 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 /')
```

3. 检查您需要主机或主机聚合的特征的现有资源供应商特征：

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

- 如果您需要的特征还没有添加到资源供应商中，请将现有的特征和所需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 CUSTOM_<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** 选项添加额外的特征。



### 注意

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

## 10.4. 创建自定义 GPU 实例镜像

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

### 先决条件

- 您已配置并部署了启用了 GPU 的 Compute 节点。

### 流程

- 以 **stack** 用户的身份登录 undercloud。
- 查找 **overcloudrc** 凭证文件：

```
$ source ~/overcloudrc
```

- 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>** 替换为具有 vGPU 实例所需的硬件配置集类别名称或 ID。有关创建 vGPU 类型的详情，请参考 [为实例创建 vGPU 类别](#)。
  - 将 **<image>** 替换为具有 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，请参考 [创建和管理镜像](#) 中的 [创建 RHEL KVM 或 RHOSP 兼容 镜像](#)。
- 以 cloud-user 用户身份登录到实例。
  - 在实例上创建 **gridd.conf** NVIDIA GRID 许可证文件，遵循 NVIDIA 指南：[使用配置文件在 Linux 上创建 NVIDIA vGPU](#)。
  - 在实例上安装 GPU 驱动程序。有关安装 NVIDIA 驱动程序的更多信息，请参阅在 [Linux 上安装 NVIDIA vGPU 软件图形驱动程序](#)。





### 注意

使用 `hw_video_model` 镜像属性来定义 GPU 驱动程序类型。如果要为 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。

## 10.5. 为实例创建 vGPU 类别

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 资源分配给该类别。

### 先决条件

- 您已配置并部署了带有 GPU 指定的 Compute 节点的 overcloud。

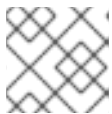
### 流程

1. 创建 NVIDIA GPU 类别，例如：

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

2. 为类别分配 vGPU 资源：

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



### 注意

您只能为每个实例分配一个 vGPU。

3. 可选：要为特定的 vGPU 类型自定义类别，请在类型中添加所需的特征：

```
(overcloud)$ openstack flavor set m1.small-gpu \
--property trait:CUSTOM_NVIDIA_11=required
```

有关如何为每个 vGPU 类型创建自定义资源供应商特征的详情，请参考 [创建自定义 vGPU 资源供应商特征](#)。

## 10.6. 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>
```

## 10.7. 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，实例会保留对设备执行任务的独占访问，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要传递给的实例上。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。有关如何创建安装了 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. 要确定每个 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 vendor 和 product 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 设备是否具有单根 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，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，请在 `pci_passthru_controller.yaml` 中添加以下配置：

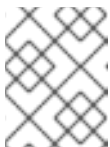
- 如果 PCI 设备具有 SR-IOV 功能：

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

- 如果 PCI 设备没有 SR-IOV 功能：

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

有关配置 `device_type` 字段的更多信息，请参阅 [PCI passthrough 设备类型字段](#)。



### 注意

如果 `nova-api` 服务以 Controller 以外的角色运行，请将 `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

当您首先将 `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，您可以禁用自动重新引导节点，并在每个 overcloud 部署后手动重启节点。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
```

- 配置类别以请求 PCI 设备。以下示例请求两个设备，每个设备的厂商 ID 为 **10de**，产品 ID 为 **13f2**：

```
# openstack flavor set m1.large \
  --property "pci_passthrough:alias"="t4:2"
```

## 验证

- 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。

- 以云用户身份登录实例。如需更多信息，请参阅 [连接到实例](#)。
- 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>
```

- 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 11 章 在实例中添加元数据

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

### 配置驱动器

配置驱动器是您可以在实例引导时附加到实例的特殊驱动器。将配置驱动器作为只读驱动器提供给实例。实例可以挂载此驱动器并从中读取文件，以获取通常通过元数据服务提供的信息。

### 元数据服务

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。

### 11.1. 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

云用户可以指定在启动实例时要使用的额外数据，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将键值对作为必要属性传递。

#### 云管理员提供数据

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

- **StaticJSON**：（默认）用于所有实例相同的元数据。
- **DynamicJSON**：对每个实例不同的元数据使用。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：

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

#### 计算服务提供的数据

计算服务使用其内部实施元数据服务将信息传递给实例，如实例请求的主机名，以及实例所在可用区。默认情况下会出现这种情况，且云用户或管理员不需要配置。

### 11.2. 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，您可以使用配置驱动器作为以下原因：

- 当部署不使用 DHCP 为实例分配 IP 地址时，要传递网络配置。您可以通过配置驱动器传递实例的 IP 地址配置，实例可以在为实例配置网络设置前挂载和访问。
- 将数据传递给启动实例的用户不知道的实例，例如，用于在启动后将实例注册到 Active Directory。
- 创建本地缓存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来签入和构建事实的实例的影响。

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

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

## 验证

1. 创建实例：

```
(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**，您需要首先识别与配置驱动器对应的块设备：

```
# blkid -t LABEL="config-2" -o device
/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。

## 11.3. 在实例中添加动态元数据

您可以配置部署以创建特定于实例的元数据，并通过 JSON 文件将元数据提供给该实例。

## 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 [强化 Red Hat OpenStack Platform 中的使用 Ansible 实施 TLS-e](#)。

## 流程

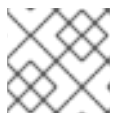
1. 打开 Compute 环境文件。
2. 将 **DynamicJSON** 添加到 vendordata 供应商模块中：

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

3. 指定要联系以生成元数据的 REST 服务。您可以根据需要指定多个目标 REST 服务，例如：

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

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



### 注意

不要对目标服务多次使用相同的名称。

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

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



## 第 12 章 配置手动节点重新引导以定义 KERNELARGS

当 overcloud 部署包含第一次设置 **KernelArgs** 时，overcloud 节点会自动重启。如果您要将 **KernelArgs** 添加到生产环境中的部署中，重新引导节点可能会成为现有工作负载的问题。您可以在更新部署时禁用自动重新引导节点，而是在每次 overcloud 部署后手动重启节点。



### 注意

如果您禁用自动重新引导，然后向部署添加新的 Compute 节点，则在初始置备过程中不会重启新节点。这可能导致部署错误，因为 **KernelArgs** 的配置仅在重新引导后被应用。

### 12.1. 配置手动节点重新引导以定义 KERNELARGS

您可在第一次配置 **KernelArgs** 时禁用自动重新引导节点，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. 禁用您要重新引导的 Compute 节点上的 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 **&lt;node>** 替换为您要禁用 Compute 服务的节点的主机名。

7. 检索托管在您要迁移的 Compute 节点上的实例列表：

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

8. 将实例迁移到另一个 Compute 节点中。有关 [迁移实例的详情](#)，请参考在 [Compute 节点之间迁移虚拟机实例](#)。
9. 登录您要重新引导的节点。
10. 重新引导节点：

```
[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 配置实例安全性

作为云管理员，您可以为云上运行的实例配置以下安全功能：

- **UEFI 安全引导**：您可以创建一个启用了属性键 `os:secure_boot` 的 UEFI 安全引导类别。云用户可以使用此类别创建通过 UEFI 安全引导保护的实例。如需更多信息，请参阅 [UEFI 安全引导](#)。
- **VNC 控制台安全**：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，来保护到实例的 VNC 控制台的连接。如需更多信息，请参阅 [保护到实例的 VNC 控制台的连接](#)。
- **模拟虚拟受信任的平台模块(vTPM)**：您可以让云用户能够创建具有模拟 vTPM 设备的实例。如需更多信息，请参阅 [配置 Compute 节点](#)，以便为实例提供模拟的受信任的平台模块(TPM)设备。
- **SEV**：使用您的云用户创建使用内存加密的实例。如需更多信息，请参阅 [配置 AMD SEV Compute 节点](#)，以便为实例提供内存加密。

### 13.1. 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，来保护到实例的 VNC 控制台的连接。

#### 流程

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

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

3. 打开 Compute 环境文件。
4. 配置到实例的 VNC 控制台连接的最低协议版本：

```
parameter_defaults:
...
NovaVNCProxySSLMinimumVersion: <version>
```

将 **<version>** 替换为最低允许的 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**：如果您的客户端不支持更新的版本，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 VNC 控制台连接到实例。
  - **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. 如果将最低允许的 SSL/TLS 协议版本设置为 **tlsv1\_2**，则将 SSL/TLS 密码配置为用于实例的 VNC 控制台连接：

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

将 **<ciphers>** 替换为要允许的密码套件的以冒号分隔的列表。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
```

## 13.2. 配置 COMPUTE 节点，为实例提供模拟受信任的平台模块(TPM)设备

作为云管理员，您可以为云用户提供创建具有模拟虚拟信任平台模块(vTPM)设备的实例。

要让您的云用户创建具有 vTPM 设备的实例，您必须执行以下任务：

1. 启用使用 vTPM 设备的实例支持并部署 overcloud。
2. 创建类别或镜像以使用 vTPM 设备启动实例。

### 先决条件

- Key Manager 服务(barbican)包含在您的 RHOSP 部署中，用于存储 vTPM 密钥。有关使用密钥管理器服务管理 secret 的详情，[请参考使用密钥管理器服务管理 secret](#)。

### 带有 vTPM 设备的实例限制

- 您无法实时迁移或撤离具有 vTPM 设备的实例。
- 您不能救援或她具有 vTPM 设备的实例。
- 实例必须具有 Q35 机器类型。

### 13.2.1. 启用使用 vTPM 设备的实例支持

要让您的云用户创建具有 vTPM 设备的实例，您必须配置 overcloud 来为实例启用 vTPM 设备。

#### 流程

1. 打开 Compute 环境文件。
2. 启用 vTPM 设备支持：

```
parameter_defaults:
  ComputeParameters:
  ...
  NovaEnableVTPM: True
```

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

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

### 13.2.2. 为 vTPM 设备创建镜像

当 overcloud 启用来创建具有 vTPM 设备的实例时，您可以创建一个 vTPM 设备实例镜像，您的云用户可以使用该镜像来启动具有 vTPM 设备的实例。



#### 注意

如果类别和镜像同时指定了 TPM 设备模型，并且两个值不匹配，则调度会失败。

#### 流程

1. 为 vTPM 设备创建新镜像：

```
(overcloud)$ openstack image create ... \
--property hw_tpm_version=2.0 vtpm-image
```



#### 注意

不支持 TPM 版本 **1.2**。

2. 可选：指定要使用的 TPM 模型：

```
(overcloud)$ openstack image set \
--property hw_tpm_model=<tpm_model> \
vtpm-image
```

- 将 **<tpm\_model>** 替换为要使用的 TPM 设备的模型。设置为以下有效值之一：
  - **TPM-tis**：（默认） TPM 接口规格。
  - **TPM-crb**：命令响应缓冲。



#### 注意

如果没有设置 **hw\_tpm\_version** 属性，Compute 服务会忽略 **hw\_tpm\_model** 属性的配置。

#### 验证

1. 使用 vTPM 镜像创建实例：

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

2. 以云用户身份登录实例。

- 要验证实例是否可以访问 vTPM 设备，请从实例输入以下命令：

```
$ dmesg | grep -i tpm
```

### 13.2.3. 为 vTPM 设备创建类别

当 overcloud 被启用来创建具有 vTPM 设备的实例时，您可以创建一个或多个 vTPM 设备类别，供您的云用户用来启动具有 vTPM 设备的实例。



#### 注意

只有 **hw\_tpm\_model** 和 **hw\_tpm\_version** 属性没有在镜像上设置时，才需要 vTPM 设备类别。如果类别和镜像同时指定了 TPM 设备模型，并且两个值不匹配，则调度会失败。

#### 流程

- 为 vTPM 设备创建类别：

```
(overcloud)$ openstack flavor create --vcpus 1 --ram 512 --disk 2 \  
--property hw:tpm_version=2.0 \  
vtpm-flavor
```



#### 注意

不支持 TPM 版本 **1.2**。

- 可选：指定要使用的 TPM 模型：

```
(overcloud)$ openstack flavor set \  
--property hw:tpm_model=<tpm_model> \  
vtpm-flavor
```

- 将 **<tpm\_model>** 替换为要使用的 TPM 设备的模型。设置为以下有效值之一：
  - TPM-tis**：（默认）TPM 接口规格。
  - TPM-crb**：命令响应缓冲. 仅与 TPM 版本 2.0 兼容。



#### 注意

如果没有设置 **hw:tpm\_model** 属性，Compute 服务会忽略 **hw:tpm\_model** 属性的配置。

#### 验证

- 使用 vTPM 类别创建一个实例：

```
(overcloud)$ openstack server create --flavor vtpm-flavor \  
--image rhel-image vtpm-instance
```

- 以云用户身份登录实例。

3. 要验证实例是否可以访问 vTPM 设备，请从实例输入以下命令：

```
$ dmesg | grep -i tpm
```

## 第 14 章 数据库清理

Compute 服务包含一个管理工具 **nova-manage**，您可以使用它来执行部署、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：

- 归档删除的实例记录，方法是将已删除的行从生产表格移到影子表格中。
- 归档完成后，从影子表中清除已删除的行。

### 14.1. 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 作业在每天 05:00 运行，两者在 0 到 3600 秒之间有一个 jitter。您可以使用 heat 参数根据需要修改这些设置。

#### 流程

1. 打开 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，请将以下参数设置为 "True"：

```
parameter_defaults:
...
NovaCronArchiveDeleteRowsPurge: True
```

有关管理数据库 cron 作业的 heat 参数的完整列表，请参阅 [Compute 服务自动化数据库管理的配置选项](#)。

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

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

### 14.2. COMPUTE 服务自动化数据库管理的配置选项

使用以下 heat 参数，以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
<b>NovaCronArchiveDeleteAllCells</b>	将此参数设置为 "True"，以从所有单元归档已删除的实例记录。  Default: <b>True</b>



参数	描述
<b>NovaCronArchiveDeleteRowsAge</b>	<p>使用此参数根据其天数归档已删除的实例记录。</p> <p>设置为 <b>0</b> 以归档影子表格中今天旧的数据。</p> <p>默认：<b>90</b></p>
<b>NovaCronArchiveDeleteRowsDestination</b>	<p>使用此参数配置文件，以记录已删除的实例记录。</p> <p>默认：<b>/var/log/nova/nova-rowsflush.log</b></p>
<b>NovaCronArchiveDeleteRowsHour</b>	<p>使用此参数配置小时，以运行 cron 命令，将已删除的实例记录移到另一个表中。</p> <p>默认：<b>0</b></p>
<b>NovaCronArchiveDeleteRowsMaxDelay</b>	<p>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）。</p> <p>默认：<b>3600</b></p>
<b>NovaCronArchiveDeleteRowsMaxRows</b>	<p>使用此参数配置可移动到另一个表的最大已删除实例记录数。</p> <p>默认：<b>1000</b></p>
<b>NovaCronArchiveDeleteRowsMinute</b>	<p>使用此参数配置一分钟，达到这个小时后，运行 cron 命令将已删除的实例记录移到另一个表中。</p> <p>默认：<b>1</b></p>
<b>NovaCronArchiveDeleteRowsMonthday</b>	<p>使用此参数配置月的哪个天，以运行 cron 命令，将已删除的实例记录移至另一表。</p> <p>默认：<b>*</b>（每天）</p>
<b>NovaCronArchiveDeleteRowsMonth</b>	<p>使用此参数配置哪个月，以运行 cron 命令，将已删除的实例记录移到另一个表中。</p> <p>默认：<b>*</b>（每月）</p>
<b>NovaCronArchiveDeleteRowsPurge</b>	<p>将此参数设置为"True"，以在调度归档后立即清除影子表。</p> <p>Default: <b>False</b></p>
<b>NovaCronArchiveDeleteRowsUntilComplete</b>	<p>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中，直到所有记录都移动。</p> <p>Default: <b>True</b></p>

参数	描述
<b>NovaCronArchiveDeleteRowsUser</b>	<p>使用此参数配置拥有 crontab 的用户，该用户存档已删除的实例记录，并且可以访问 crontab 使用的日志文件。</p> <p>Default: <b>nova</b></p>
<b>NovaCronArchiveDeleteRowsWeekday</b>	<p>使用此参数配置每周的哪个天，以运行 cron 命令将已删除的实例记录移到另一个表中。</p> <p>默认 : * (每天)</p>
<b>NovaCronPurgeShadowTablesAge</b>	<p>使用此参数根据它们的年龄（以天为单位）清除影子表格。</p> <p>设置为 <b>0</b> 以清除比今天旧的影子表格。</p> <p>默认 : <b>14</b></p>
<b>NovaCronPurgeShadowTablesAllCells</b>	<p>将此参数设置为"True"，以从所有单元清除影子表格。</p> <p>Default: <b>True</b></p>
<b>NovaCronPurgeShadowTablesDestination</b>	<p>使用此参数配置用于记录清除影子表格的文件。</p> <p>默认 : <b>/var/log/nova/nova-rowspurge.log</b></p>
<b>NovaCronPurgeShadowTablesHour</b>	<p>使用此参数配置小时，在其中运行 cron 命令来清除影子表。</p> <p>默认 : <b>5</b></p>
<b>NovaCronPurgeShadowTablesMaxDelay</b>	<p>在清除影子表格前，使用此参数配置最大延迟（以秒为单位）。</p> <p>默认 : <b>3600</b></p>
<b>NovaCronPurgeShadowTablesMinute</b>	<p>使用此参数配置一分钟，超过小时，以便运行 cron 命令来清除影子表。</p> <p>默认 : <b>0</b></p>
<b>NovaCronPurgeShadowTablesMonth</b>	<p>使用此参数配置哪个月，以运行 cron 命令清除影子表格。</p> <p>默认 : * (每月)</p>

参数	描述
<b>NovaCronPurgeShadowTablesMonthday</b>	<p>使用此参数配置月的哪个天，以运行 cron 命令清除影子表格。</p> <p>默认：<b>*</b>（每天）</p>
<b>NovaCronPurgeShadowTablesUser</b>	<p>使用此参数配置拥有 crontab 的用户，以清除影子表格并可访问 crontab 使用的日志文件。</p> <p>Default: <b>nova</b></p>
<b>NovaCronPurgeShadowTablesVerbose</b>	<p>使用此参数在日志文件中为清除的影子表格启用详细日志记录。</p> <p>Default: <b>False</b></p>
<b>NovaCronPurgeShadowTablesWeekday</b>	<p>使用此参数配置每周的哪个天，以运行 cron 命令清除影子表。</p> <p>默认：<b>*</b>（每天）</p>

## 第 15 章 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的节点。

### Compute 节点维护

如果您需要临时从服务中移除 Compute 节点（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 节点要失败，且您需要服务或替换它，您可以将实例从故障 Compute 节点迁移到一个健康的 Compute 节点。

### 失败的 Compute 节点

如果 Compute 节点已经失败，您可以撤离实例。您可以使用相同的名称、UUID、网络地址以及实例在 Compute 节点失败前具有的任何其他分配资源，从另一个 Compute 节点上原始镜像重建实例。

### 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，将实例迁移到更接近其他联网资源的 Compute 节点，以缩短延迟，或跨 Compute 节点分配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 SSH 密钥，以便每个主机的用户在迁移过程中可以访问其他 Compute 节点。director 使用

**OS::TripleO::Services::NovaCompute** 可组合服务创建此密钥。此可组合服务是所有 Compute 角色中包含的主要服务之一。如需更多信息，[请参阅自定义 Red Hat OpenStack Platform 部署指南中的可组合服务和自定义角色](#)。



### 注意

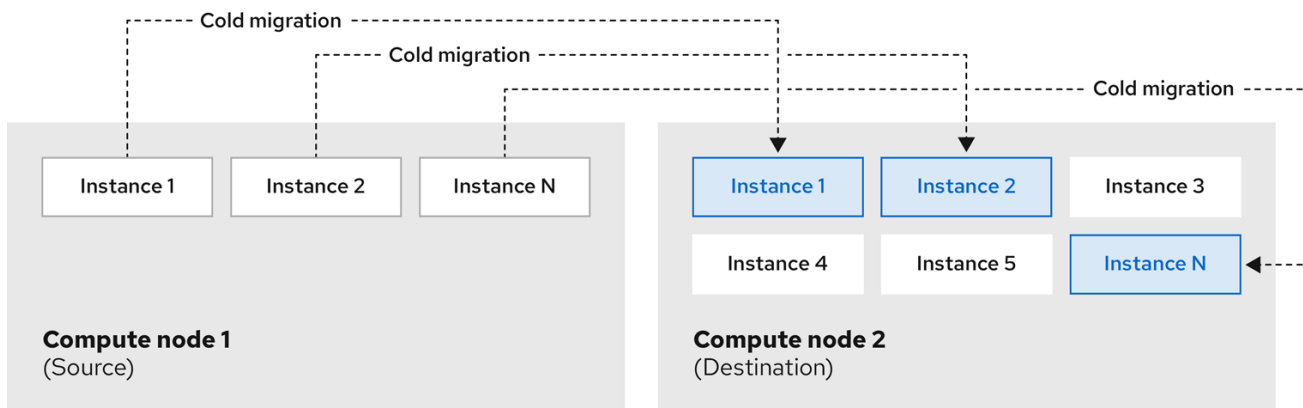
如果您有可正常工作的 Compute 节点，并且希望为备份目的制作实例副本，或者要复制实例到不同环境，请按照 [安装和管理 Red Hat OpenStack Platform 指南中的将虚拟机导入到 overcloud 中](#)。

## 15.1. 迁移类型

Red Hat OpenStack Platform (RHOSP)支持以下迁移类型。

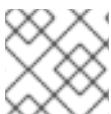
### 冷迁移

冷迁移或非实时迁移涉及在将正在运行的实例从源 Compute 节点迁移到目标 Compute 节点之前将其关闭。



273\_OpenStack\_0822

冷迁移涉及实例出现一些停机时间。迁移的实例保持对同一卷和 IP 地址的访问权限。

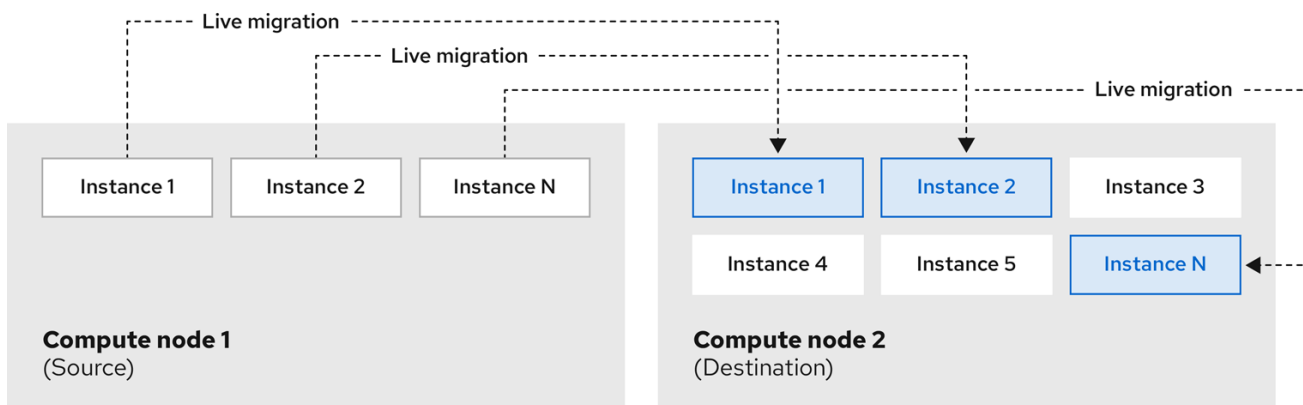


### 注意

冷迁移要求源和目标 Compute 节点都正在运行。

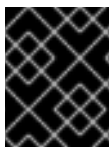
### 实时迁移

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。



273\_OpenStack\_0822

实时迁移实例涉及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，实例应该退出关键的路径。



### 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、内存带宽、存储 IO 或 CPU 速率降低。



### 注意

实时迁移要求源和目标 Compute 节点都正在运行。

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。

## 15.2. 迁移限制

迁移限制通常随块迁移、配置磁盘或一个或多个实例访问 Compute 节点上的物理硬件而出现。

### 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 模型之间迁移。在某些情况下，源和目标 Compute 节点的 CPU 必须完全匹配，如使用 CPU 主机透传的实例。在所有情况下，目标节点的 CPU 功能必须是源节点上 CPU 功能的超集。

### 内存限制

目标 Compute 节点必须有足够的可用 RAM。内存超额订阅可能会导致迁移失败。

### 块迁移限制

迁移使用 Compute 节点上本地存储的磁盘的实例要比迁移使用共享存储（如 Red Hat Ceph Storage）的卷支持的实例要长得多。这是因为 OpenStack Compute (nova) 默认在 control plane 网络上的 Compute 节点之间迁移本地磁盘块。相反，使用共享存储（如 Red Hat Ceph Storage）的卷支持的实例不必迁移卷，因为每个 Compute 节点已经能够访问共享存储。



#### 注意

通过迁移消耗大量 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（如 RabbitMQ）的性能。

### 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) 可以迁移同时具有读写功能的驱动器，包括驱动器格式（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。红帽不支持在实时迁移过程中增加数据包丢失、网络延迟、内存延迟或降低网络带宽、内存带宽、存储 IO 或 CPU 性能。

### 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阻止新的操作。否则，如果写入内存的速度比实时迁移复制内存状态要快，则实时迁移可能需要很长时间或可能永远不会结束。

### 使用 NUMA 的 CPU 固定

Compute 配置中的 **NovaSchedulerEnabledFilters** 参数必须包含值 **AggregateInstanceExtraSpecsFilter** 和 **NUMATopologyFilter**。

### 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，但不能在单元格间实时迁移。

### 浮动实例

当实时迁移浮动实例时，如果目标 Compute 节点上 **NovaComputeCpuSharedSet** 的配置与源 Compute 节点上的 **NovaComputeCpuSharedSet** 的配置不同，则实例不会分配给目标 Compute 节点上为共享（非固定）实例的 CPU 配置。因此，如果需要实时迁移浮动实例，您必须为所有计算节点配置与专用（固定）和共享（未固定）实例相同的 CPU 映射，或者为共享实例使用主机聚合。

### 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

带有基于 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/OVS 部署上的实时迁移

在实时迁移过程中，在目标主机中取消暂停虚拟机时，元数据服务可能无法使用，因为元数据服务器代理尚未生成。这种不可用是简明的。该服务很快就会可用，实时迁移可以成功。

### 阻止实时迁移的限制

您无法实时迁移使用以下功能的实例。

#### PCI passthrough

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 透传提供对 PCI 设备的实例专用访问权限，就像它们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，因此 QEMU/KVM 不支持使用 PCI 透传实时迁移实例。

#### 端口资源请求

您无法实时迁移使用具有资源请求的端口的实例，如保证最小带宽 QoS 策略。使用以下命令检查端口是否有资源请求：

```
$ openstack port show <port_name/port_id>
```

## 15.3. 准备迁移

在迁移一个或多个实例前，您需要确定 Compute 节点名称和要迁移的实例的 ID。

### 流程

1. 识别源 Compute 节点主机名和目标 Compute 节点主机名：

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

2. 列出源 Compute 节点上的实例，并找到您要迁移的实例或实例的 ID：

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

将 **<source>** 替换为源 Compute 节点的名称或 ID。

3. 可选：如果您要从源 Compute 节点迁移实例在节点上执行维护，则必须禁用该节点以防止调度程序在维护过程中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。

## 15.4. 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。冷迁移有助于迁移实时迁移无法促进的场景，例如迁移使用 PCI 透传的实例。调度程序自动选择目标 Compute 节点。如需更多信息，请参阅 [迁移限制](#)。

### 流程

1. 要冷迁移实例，请输入以下命令关闭并移动实例：

```
(overcloud)$ openstack server migrate <instance> --wait
```

- 将 **<instance>** 替换为要迁移的实例的名称或 ID。
- 如果迁移本地存储的卷，则指定 **--block-migration** 标记。

2. 等待迁移完成。在等待实例迁移完成时，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。状态"ACTIVE"表示实例已准备就绪。

- 如果迁移无法正常工作，请恢复它：

```
(overcloud)$ openstack server resize --revert <instance>
```

将 **<instance>** 替换为实例的名称或 ID。

4. 重启实例：

```
(overcloud)$ openstack server start <instance>
```

将 **<instance>** 替换为实例的名称或 ID。

5. 可选：如果您禁用了源 Compute 节点以进行维护，您必须重新启用该节点，以便可以为其分配新实例：

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

将 **<source>** 替换为源 Compute 节点的主机名。



## 15.5. 实时迁移实例

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

### 流程

1. 要实时迁移实例，请指定实例和目标 Compute 节点：

```
(overcloud)$ openstack server migrate <instance> --live-migration [--host <dest>] --wait
```

- 将 **<instance>** 替换为实例的名称或 ID。
- 将 **<dest>** 替换为目标 Compute 节点的名称或 ID。



### 注意

**openstack server migrate** 命令涵盖使用共享存储迁移实例，这是默认设置。指定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。在等待实例迁移完成时，您可以检查迁移状态。如需更多信息，请参阅 [检查迁移状态](#)。
4. 检查实例的状态，以确认迁移是否成功：

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

将 **<dest>** 替换为目标 Compute 节点的名称或 ID。

5. 可选：如果您禁用了源 Compute 节点以进行维护，您必须重新启用该节点，以便可以为其分配新实例：

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

将 **<source>** 替换为源 Compute 节点的主机名。

## 15.6. 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**：计算服务已接受迁移实例的请求，并且迁移为 pending 状态。
2. **Preparing**: Compute 服务正在准备迁移实例。
3. **Running**: Compute 服务正在迁移实例。
4. **Post-migrating**: Compute 服务已在目标 Compute 节点上构建实例，并释放源 Compute 节点上的资源。
5. **completed**：计算服务已完成迁移实例，并完成源 Compute 节点上的资源发布。

## 流程

1. 检索实例的迁移 ID 列表：

```
$ openstack server migration list --server <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：

```
$ openstack 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 **openstack 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

Compute 服务根据要复制的剩余内存字节数来测量迁移的进度。如果这个数字没有随着时间的推移减少，迁移可能无法完成，并且计算服务可能会中止。

有时，实例迁移可能需要很长时间或遇到错误。如需更多信息，[请参阅故障排除迁移](#)。

## 15.7. 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。

撤离过程会销毁原始实例，并使用原始镜像、实例名称、UUID、网络地址以及原始实例已分配给它的任何其他资源重建原始实例。

如果实例使用共享存储，则在撤离过程中不会重建实例根磁盘，因为目标 Compute 节点仍可访问磁盘。如果实例不使用共享存储，则在目标 Compute 节点上也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，API 会报告 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。

### 15.7.1. 清空一个实例

您可以一次撤离实例。

#### 流程

1. 确认实例没有运行：

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

- 将 **<node>** 替换为托管实例的 Compute 节点的名称或 UUID。

2. 确认主机 Compute 节点已被隔离或关闭：

```
(overcloud)[stack@director ~]$ openstack baremetal node show <node>
```

- 将 **<node>** 替换为托管要撤离的 Compute 节点的名称或 UUID。要执行撤离，Compute 节点必须处于 **down** 或 **forced-down** 状态。

3. 禁用 Compute 节点：

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

- 将 **<node>** 替换为要从中撤离实例的 Compute 节点的名称。
- 将 **<disable\_host\_reason>** 替换为您禁用 Compute 节点的详情。

4. 撤离实例：

```
(overcloud)[stack@director ~]$ nova evacuate [--password <pass>] <instance> [<dest>]
```

- 可选：将 **<pass>** 替换为访问撤离实例所需的管理密码。如果没有指定密码，则会生成一个随机密码，并在 evacuation 完成后输出。



### 注意

只有在临时实例磁盘存储在本地虚拟机监控程序磁盘上时，才会更改密码。如果实例托管在共享存储上，或者实例附加了 Block Storage 卷，且没有显示错误消息来告知您密码没有改变，则不会更改密码。

- 将 **<instance>** 替换为要撤离的实例的名称或 ID。
- 可选：将 **&lt;dest>** 替换为要撤离实例的 Compute 节点的名称。如果没有指定目标 Compute 节点，则计算调度程序会为您选择一个。您可以使用以下命令查找可能的 Compute 节点：

```
(overcloud)[stack@director ~]$ openstack hypervisor list
```

5. 可选：在恢复时启用 Compute 节点：

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

- 将 **<node>** 替换为要启用的 Compute 节点的名称。

## 15.7.2. 清空主机上的所有实例

您可以撤离指定 Compute 节点上的所有实例。

### 流程

1. 确认要撤离的实例没有运行：

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

- 将 **<node>** 替换为托管要撤离的 Compute 节点的名称或 UUID。

2. 确认主机 Compute 节点已被隔离或关闭：

```
(overcloud)[stack@director ~]$ openstack baremetal node show <node>
```

- 将 **<node>** 替换为托管要撤离的 Compute 节点的名称或 UUID。要执行撤离，Compute 节点必须处于 **down** 或 **forced-down** 状态。

3. 禁用 Compute 节点：

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

- 将 **<node>** 替换为要从中撤离实例的 Compute 节点的名称。
- 将 **<disable\_host\_reason>** 替换为您禁用 Compute 节点的详情。

## 4. 撤离指定 Compute 节点上的所有实例：

```
(overcloud)[stack@director ~]$ nova host-evacuate [--target_host <dest>] <node>
```

- 可选：将 **<dest>** 替换为目标 Compute 节点的名称，以撤离实例。如果没有指定目的地，Compute 调度程序会为您选择一个。您可以使用以下命令查找可能的 Compute 节点：

```
(overcloud)[stack@director ~]$ openstack hypervisor list
```

- 将 **<node>** 替换为要从中撤离实例的 Compute 节点的名称。

## 5. 可选：在恢复时启用 Compute 节点：

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

- 将 **<node>** 替换为要启用的 Compute 节点的名称。

## 15.8. 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。

### 15.8.1. 迁移过程中的错误

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 节点上的实例会在迁移到目标 Compute 节点之前被删除。

## 15.8.2. 永不结束实时迁移

实时迁移可能无法完成，这会使迁移保持 **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的变化速度快于计算服务可以将其复制到目标 Compute 节点。

使用以下方法之一解决这种情况：

- 中止实时迁移。
- 强制实时迁移完成。

### 中止实时迁移

如果实例状态变化比迁移步骤将其复制到目标节点更快，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：

```
$ openstack server migration list --server <instance>
```

将 **<instance>** 替换为实例的名称或 ID。

2. 中止实时迁移：

```
$ openstack server 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，并且希望临时挂起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：

```
$ openstack server migration list --server <instance>
```

将 **<instance>** 替换为实例的名称或 ID。

2. 强制实时迁移完成：

```
$ openstack server 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。

### 15.8.3. 迁移后实例性能降级

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可用资源。如果源和目标 Compute 节点之间 NUMA 配置不相同，则实时迁移可能会在实例性能下降的同时成功。例如，如果源 Compute 节点将 NIC 1 映射到 NUMA 节点 0，但目标 Compute 节点会将 NIC 1 映射到 NUMA 节点 5，在迁移实例后，可以将流量从第一个 CPU 路由到一个 NUMA 节点 5 的第二个 CPU，以将流量路由到 NIC 1。这可能导致预期的行为，但会降低性能。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](#)。